VMS

digital

Guide to Using VMS

Order Number AA–LA05A–TE

Guide to Using VMS

Order Number: AA–LA05A–TE

April 1988

This manual describes general user tasks that can be performed using the VMS operating system and is intended for users with all levels of experience. The information contained within is applicable to all members of the VAX and MicroVAX families of computers operating under the control of the VMS operating system.

Revision/Update Information:	This is a new manual.
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™

ZK3390

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's 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™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

™ PostScript is a trademark of Adobe Systems, Inc.

Contents

Preface

This manual provides an overview of the VMS operating system and is designed to support general users in their daily computing tasks.

Intended Audience

This manual is intended for all general users.

Document Structure

This manual includes the following chapters:

- Chapter 1—Introducing VMS and DCL
- Chapter 2—Working with Files and Directories
- Chapter 3—Working with Processes
- Chapter 4—Using Logical Names
- Chapter 5—Representing Data with Symbols
- Chapter 6—Writing and Using Command Procedures
- Chapter 7—Maintaining Accounts and System Security
- Chapter 8—Editing Files with the EVE and EDT Editors
- Chapter 9—Processing Files with DIGITAL Standard Runoff

Two appendixes contain the following information:

- ASCII character set
- ASCII and DEC multinational character set tables
- DEC Multinational Character Set
- Expressions

Conventions

Convention	Meaning
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.
$ 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 Introducing VMS and DCL

Your VAX computer operates under the control of the VMS (Virtual Memory System) operating system. VMS is an interactive system: while you are logged in to the computer, you and the system conduct a dialogue of command and response. You use DCL, the DIGITAL Command Language interpreter, to communicate with VMS. DCL provides you with over 200 commands and functions to use in communicating with VMS to accomplish your computing tasks.

This chapter describes basic interaction with the VMS operating system. In it, you will learn how to log in to the system, how to use DCL to communicate with the system, how to customize your computing environment, how to get online help, and how to use two system utilities that let you communicate with other users.

Some of the topics discussed in this chapter require you to be familiar with your terminal. For information on setting up or using your terminal, see the owner's manual supplied with your terminal.

1.1 Logging In to the System

In order to interact with the VMS operating system, you must log in to an account. Logging in consists of identifying yourself to the system as an authorized user. Your system manager or whoever authorizes system use at your installation usually sets up accounts. This person provides you with your user name and 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.

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. 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.
3. Enter your user name and press RETURN. The system displays your user name on the screen as you type it. (You have about 30 seconds to do this, otherwise the system "times out" and you must start the login procedure again.) After you enter your user name and press RETURN, the system prompts you for your password.
4. Enter your password and press RETURN. The system does not display your password as you type it; this preserves the secrecy of your password. The password you enter is compared with the encrypted password stored in a system file called the User Authorization File (UAF). (See Section 7.1 for more information about the UAF file.)

If you make a mistake entering your user name or password, or your password has expired, the system displays the message "User authorization failure," and 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. If you make a mistake entering your user name or password, press RETURN and try again. If your password has expired or you have any other problems logging in, get help from the person who set up your account.

Some accounts are set up to require two passwords. If you see a second password prompt, enter the second password required to access that account.

If none of these prompts ($, Password:, or Username:) appears when you press RETURN, a system password may be required to log in to your system. If you know the system password, type it and press RETURN. If you do not know it, see the person in charge of your system.

If your login is successful, a dollar sign symbol ($) is displayed in the left margin of your terminal. This dollar sign is a prompt that VMS uses to indicate you are at DIGITAL Command Language (DCL) level and can begin entering DCL commands. When you log in to the system and work interactively with DCL, you are at command level 0. When you execute a program interactively, you are placed at command level 1; when the program completes execution, you are returned to command level 0.

The following example shows a successful login:

```
RET
Username:  CASEY RET
Password:  RET
        Welcome to VAX/VMS version 5.0 on node MARS
    Last interactive login on Friday, 31-DEC-1988 08:41
    Last non-interactive login on Thursday, 30-DEC-1988 11:05
$
```

If your account was set up by someone else, immediately change your password after you log in for the first time. You should also change your password frequently to ensure system security. To change your password, enter the DCL command SET PASSWORD. 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. The following example shows what you see:

```
$ SET PASSWORD
Old password:
New password:
Verification:
```

(If you are managing your own system, see the *Guide to Setting Up a VMS System* for instructions on setting up a user account and establishing a password.)

Each time you log in, the system automatically executes up to two login command procedures. A *command procedure* is a file that contains a list of DCL commands. When a command procedure is executed, the DCL interpreter reads the file and executes the commands it contains.

If your system manager has set up a system login command procedure, it is executed when you log in. This command procedure allows your system manager to ensure that certain commands are always executed when you and other users on your system log in.

After executing the system login command procedure, the system executes your personal login command procedure, if one exists. Your personal login command procedure allows you to customize your computing environment. The commands contained in it are executed every time you log in. The person who set up your account may have placed a login command procedure in your top level directory. (Unless your account has been specially modified to do otherwise, the system automatically places you in your top level directory when you log in.) If a login command procedure is not there, you can create one yourself, name it LOGIN.COM, and place it in your top level directory unless your system manager tells you otherwise. DCL and DCL commands are discussed in Section 1.2. Directories, including your top level directory, are discussed in Chapter 2. A sample personal login command procedure is described in Section 6.3.

When you log in, an environment is created from which you can enter commands. This environment is called your *process*. The system obtains the characteristics that are unique to your process from the user authorization file (UAF). The UAF lists those users permitted to access the system and defines the characteristics for each user's process. The system manager usually maintains the UAF. See Chapter 3 for more information about processes.

1.1.1 Alternative Login Procedures

The standard login procedure described in the preceding section may not fit your needs if you must log in to a terminal assigned to a specific account, access a system other than your own, or dial in to your system by telephone. The following sections describe these procedures.

1.1.1.1 Automatic Login

You may need to log in to a terminal that is assigned to a particular account. This procedure, called automatic login, permits you to log in without specifying a user name. To log in, turn on the terminal and press the RETURN key. Either the DCL or password prompt appears. If the DCL prompt appears, you are logged in. If the password prompt appears, type the password of the account associated with the terminal and press RETURN. (The password is not displayed on the screen.) If your login is successful, you see the DCL dollar sign prompt that VMS uses to indicate you are at DCL command level and can begin entering commands.

1.1.1.2 Logging In over the Network

Your system may be part of a DECnet–VAX network. VMS systems linked together in a DECnet network are able to communicate with each other and share information and resources. Each system in the network is called a network *node* and is identified by a unique node name and address. When you are logged in to a network node, you can communicate with every other node in the network. The node at which you are logged in is called the *local node*; the other nodes on the network are called *remote nodes.*

If you have access to an account on a remote node, you can log in to that account from your local node and use the facilities of that remote node while remaining physically connected to your local node.

The following example shows how to access a remote node on the network using the DCL command SET HOST. HUBBUB is the name of the remote node.

```
$ SET HOST HUBBUB
```

You can then log in to your account on the remote node using the remote node's login procedure. When you use the SET HOST command to log in to a remote node, you can perform any operation on the remote node as though it were your local node. Note that the remote node need not be a VMS system. If the network link cannot be established, you receive an error message.

If you want to abort the login procedure, enter CTRL/Z at the user name or password prompt or enter CTRL/Y twice. The host system should respond with the question, "Are you repeating ^Y to abort the remote session?" Answering Y (uppercase or lowercase) aborts the remote session.

You can terminate a remote session in two ways:

- Use the remote system's logout procedure (for example, on a VMS system, use the LOGOUT command).
- Press CTRL/Y twice to obtain the host system's prompt that asks whether you want to abort the remote session. Answer Y if you want to abort the remote session. This method works regardless of the system running on the remote node.

When you terminate a remote session, the message "%REM-S-END, control returned to node _NODENAME::" is displayed, and you are returned to the system from which you made the remote node connection.

If the DECnet network has made intermediate connections for you and one of the intermediate systems goes down, DECnet either attempts to reroute the connection or waits a few seconds to determine whether the system will recover. If DECnet is able to recover the connection, the interruption may be so brief that you do not notice it, or it may last as long as 60 seconds. If DECnet cannot recover the connection, the remote session is terminated and the message "Path lost to partner" may be displayed.

See the *VMS DCL Dictionary* for more information about the DCL command SET HOST. For more information about networking and DECnet, see the *Guide to DECnet-VAX Networking.*

1.1.1.3 Dialing In

Dialing in allows you to communicate with your system by telephone. To dial in to your system, you need the following items:

- Modem (or data set)—A modem is a piece of hardware that is independent of the VMS system. The user's manual that comes with the modem should describe how to connect the modem to a telephone line and a terminal.
- Terminal—You cannot dial in unless the transmission speed (baud rate) of the terminal agrees with the baud rate of the modem and the modem terminal characteristic is set. To set the baud rate for VT200- and VT300-series terminals, select the "Comm" category in the Set-Up Directory. To set the baud rate for a VT100 series terminal, you must be in SET-UP B. To enter the Set-Up Directory, press the SET-UP key. Press the key labeled A/B to toggle to SET-UP B.

  To set the modem terminal characteristic, use the DCL command SET TERMINAL/MODEM. If your terminal has the modem terminal characteristic set (the DCL command SHOW TERMINAL lists the terminal characteristics set for your terminal), typing the SET TERMINAL/NOMODEM command causes the system to log you out.

See your terminal's installation manual for information on using the Set-Up Directory to set the baud rate and other terminal characteristics.

- Manual login account—If your account is set up for automatic login, you cannot dial in to it. Either change the account to a manual login account (one where you must type your user name) or use a different account.
- Telephone number for the system—To dial in, you must know the telephone number for your system. Get the telephone number from your system manager.

Once the terminal is receiving the signal through the telephone line, follow the conventions your site has instituted for remote login. When communication is established, your system should respond with the DCL dollar sign prompt.

If you are managing your own system, you can ensure system security by disallowing dialup accounts with null passwords.

1.1.2 Logging Out of the System

When you finish using the system, always log out. This prevents unauthorized users from accessing your account and the system at large. It is also a wise use of system resources: the resources you no longer need are freed for use by others.

To log out, enter the LOGOUT command, which can be abbreviated as LO. You see a display confirming that you are logged out of the system that looks similar to the following:

```
$ LOGOUT
HARRIS logged out at 31-DEC-1988  12:42:48.12
```

You can log out of the system only when you are at the DCL prompt. You cannot enter the LOGOUT command while you are compiling or executing a program, using an editor (such as EDT or EVE), or running a utility (such as MAIL). First you must exit the program, editor, or utility and receive the DCL prompt. It is possible to exit by entering CTRL/Y, but this is not always advisable. See Section 1.3.2.2 for more information about using CTRL/Y.

To find out how much time you spent at the terminal (elapsed time), how much computer time you used (charged CPU time), and other accounting information, include the /FULL qualifier to the LOGOUT command as follows:

```
$ LOGOUT/FULL
SIMPSON logged out at 31-DEC-1988  12:42:48.12

Accounting information:
 Buffered I/O count:      8005  Peak working set size:    212
 Direct I/O count:         504  Peak virtual size:        770
 Page faults:             1476  Mounted volumes:            0
 Charged CPU time:0 00:00:50.01  Elapsed time:0 02:27:43.06
```

If you are logging out from a dialup terminal, enter the LOGOUT command with the /HANGUP qualifier. This command causes the system to break the connection to the dialup line after you log out.

1.2 Using the DIGITAL Command Language

The DIGITAL Command Language (DCL) is the language you use to communicate with the VMS operating system. DCL commands let you do the following:

- Get information about the system
- Work with files
- Work with disks, magnetic tapes, and other devices
- Modify your work environment
- Develop and execute programs
- Provide security and ensure that resources are used efficiently

DCL commands are usually verbs that describe what you want the system to do. In response to the DCL dollar sign ($) prompt, you enter a command (in upper- or lowercase). The following example shows how to enter the DCL command SHOW TIME:

```
$ SHOW TIME RET
```

The system responds by displaying the current date and time and returns the DCL prompt to indicate it is ready to accept another command:

```
31-DEC-1988  15:41:43
$
```

You can use DCL in the following two modes:

- Interactive—In interactive mode, you enter commands from your terminal. One command has to finish executing before you can enter another.
- Batch—In batch mode, the system creates another *process* to execute commands on your behalf. *Batch* jobs and network processes use DCL in batch mode. A process is an environment created by the system that makes it possible for you to work with the system. A batch job is a command procedure or program that is submitted to the operating system for execution as a separate user process. After you submit the command procedure for batch execution, you can continue to use your terminal interactively. (See Chapter 3 for more information about processes and batch jobs.)

  When you type a command and press RETURN, it is read and translated by the DCL interpreter. The way the command interpreter responds to a command is determined by the type of command entered. The three types of commands are as follows:

  - Built-in commands—These commands, listed in Table 1–1, are built into the DCL interpreter. DCL executes a built-in command internally.
  - Commands that invoke programs—DCL calls another program to execute the command rather than executing it internally. The program invoked to execute a command is referred to as a *command image*. This command image can be either an interactive program like MAIL or a noninteractive program like COPY. Parameter and qualifier information (which modify the command) are passed to the program. Most commands not listed in Table 1–1 are in this category.

- Foreign commands—A symbol that executes an image is referred to as a *foreign command.* A foreign command executes an image whose name is not recognized by the command interpreter as a DCL command. The following example defines the symbol FUN as a foreign command. (No DCL command FUN exists.)

  ```
  $ FUN := $DISK1:[ROY.PROGRAMS]GAMES.EXE
  ```

  The request to execute the image GAMES.EXE is implied in the symbol definition by the presence of the dollar sign. (File names with a file type EXE are always executable images.) Once you equate the symbol FUN to the file specification shown, you can execute the image GAMES.EXE by typing FUN.

  See Chapter 5 and the *VMS DCL Concepts Manual* for information about defining symbols. See Chapter 3 for a description of images. Chapter 2 describes file names, file types, and file specifications.

Table 1–1 Built-In Commands

=, ==, :=, :==	ALLOCATE	ASSIGN
ATTACH	CALL	CANCEL
CLOSE	CONNECT	CONTINUE
CREATE/LOGICAL_NAME_TABLE	DEALLOCATE	DEASSIGN
DEBUG	DECK	DEFINE
DEFINE/KEY	DELETE/KEY	DELETE/SYMBOL
DEPOSIT	DISCONNECT	ENDSUBROUTINE
EOD	EXAMINE	EXIT
GOSUB	GOTO	IF
INQUIRE	ON	OPEN
READ	RECALL	RETURN
SET CONTROL	SET DEFAULT	SET KEY
SET ON	SET OUTPUT_RATE	SET PROMPT
SET PROTECTION/DEFAULT	SET UIC	SET VERIFY
SHOW DEFAULT	SHOW KEY	SHOW PROTECTION
SHOW QUOTA	SHOW STATUS	SHOW SYMBOL
SHOW TIME	SHOW TRANSLATION	SPAWN
STOP	SUBROUTINE	WAIT
WRITE		

1.2.1 DCL Command HELP

You can obtain online documentation for any DCL command by invoking the HELP facility. To use the HELP facility in its simplest form, enter the DCL command HELP. HELP displays a list of topics and the Topic? prompt. If you want to see information on one of the topics, type the topic name after the prompt. The system displays information on that topic. (Command and topic names can be abbreviated.)

If the topic has subtopics, HELP lists the subtopics and displays the Subtopic? prompt. If you want information on one of the subtopics, type the name after the prompt. If you want information on another topic, press RETURN. You can ask for information on another topic when HELP displays the Topic? prompt. If you want to exit the HELP facility, press RETURN again to return to DCL level. At any time, press CTRL/Z to exit.

If you know the command you need information about, type HELP and the command name.

If you need help but do not know what command or system topic to specify, enter the command HELP with the word HINTS as a parameter. Each task name listed in the HINTS text is associated with a list of related command names and system information topics.

Following is a sample HELP display for the DCL command SHOW USERS:

```
$ HELP SHOW USERS

SHOW

  USERS

     Displays the terminal name, username, and process
     identification code (PID)  of either specific interactive
     users or all interactive users on the sytem.

      Format:

      SHOW USERS [username]

  Additional information available:

  Parameters Command_Qualifiers
  /OUTPUT
  Examples

SHOW USERS Subtopic?
```

You can also obtain help while you are using an interactive utility. *Utilities* are programs that are invoked with DCL commands. To get help while you are using an interactive utility, type HELP at the utility prompt (and press RETURN) just as you would at DCL level. See Section 1.4 for more information about VMS utilities.

1.2.2 The DCL Command Line

DCL, like any language, has its own vocabulary and usage rules. The vocabulary consists of commands, parameters, and qualifiers, which are strung together in a way that DCL can interpret. The way in which the parts of a command line are put together is referred to as the *command line syntax.* DCL command line syntax follows the following general format shown. (Items in square brackets [ ] are optional and may not be required by a specific command.)

[$] [label:] command [/qualifier[=value]...] [parameter[/qualifier...]]

The DCL command line can contain the following components:

$	The dollar sign is the DCL prompt. When you work interactively with DCL, DCL displays the prompt when it is ready to accept a command. When you write a command procedure, you must type the dollar sign at the beginning of each line.
Label	Identifies a line in a command procedure. Use labels only within command procedures, which are described in Chapter 6.
Command	Specifies the name of the command.
Parameter	Specifies what the command acts upon. You must position parameters in a specified order within the command. The DCL command descriptions in the *VMS DCL Dictionary* describe what parameter values are allowed for each command and where they must be placed. Examples of parameter values include file specifications, queue names, and logical names.
Qualifier	Modifies the action taken by the command. Some qualifiers can modify parameters. Some can accept values. The DCL command descriptions in the *VMS DCL Dictionary* indicate whether a specific qualifier can accept a value and what kind of value is acceptable.
Value	Modifies a qualifier and is often preceded by an equal sign. A value can be a file specification, a character string, a number, or a DCL *keyword.* A keyword is a word reserved for use in certain specified syntax formats. You must use keywords exactly as listed in the description of the particular DCL command you want to specify. For example, SYSTEM, OWNER, GROUP, and WORLD are DCL keywords. A DCL keyword can also have a value.

Following is an example of a DCL command line:

```
$ PRINT/COPIES=5 LAUNDRY.LIS [RET]
```

In the previous example, the elements are as follows:

- *$* is the DCL prompt.
- *PRINT* is a command.
- */COPIES* is a qualifier that modifies the command.
- *5* is a value that modifies the qualifier.
- *LAUNDRY.LIS* is a parameter (in this case the parameter is a file specification).

In some cases (such as DELETE/ENTRY or SHOW QUEUE), a command is coupled with a parameter or qualifier. In these cases, the command and parameter or qualifier are used as a pair and cannot be separated. If you specify additional parameters or qualifiers, they must follow the command pair.

The following example shows a command pair that contains a command (SHOW) and a parameter (QUEUE). The additional parameter LN03_PRINT, which specifies the queue name whose jobs you want displayed, is specified after the command pair.

```
$ SHOW QUEUE LNO3_PRINT
```

Observe the following rules when entering DCL commands:

- You can use any combination of uppercase and lowercase letters. The DCL interpreter translates lowercase letters to uppercase. Upper- and lowercase characters in parameter and parameter qualifier values are equivalent unless enclosed in quotation marks.
- Separate the command name from the first parameter with at least one blank space. Separate each additional parameter from the previous parameter with at least one blank space. Begin each qualifier with a slash (/); the slash serves as a separator and need not be preceded by blank spaces or tabs.
- You may need more than one line on your terminal screen to type a command line. Continue the command line onto the next line by terminating it with a hyphen and pressing RETURN. The system responds to this combination of a hyphen and RETURN with an underscore (_) followed by the dollar sign prompt; you continue typing the command line after this prompt. (A single command line cannot exceed 256 characters.) A line beginning with an underscore means that the system is waiting for your response, as shown in the following example:

  ```
  $ COPY/LOG FORMAT.TXT,FIGURE.TXT,ART_WORK.TXT -
  _$ SAVE.TXT
  ```

  Note that you must include the appropriate spaces between command names, parameters, and so on. Pressing RETURN after the hyphen does not add a space.
- A command line can contain a maximum of 128 elements (for example, a file specification or qualifier). Each element in a command must not exceed 255 characters. The entire command must not exceed 1024 characters after all symbols and lexical functions are converted to their values. (You use *symbols,* described in Chapter 5, to pass information to the system in an abbreviated manner. A *lexical function,* described in Chapter 6, obtains information from the system, including information about system processes, batch and print queues, and user processes, and then substitutes the result of the operation for itself.)
- You can abbreviate any command name by typing only the first four characters. You can abbreviate a command name to fewer than four characters as long as the abbreviated name remains unique among all DCL command names.

  For example, the following commands are equivalent:

  ```
  $ PR/C=2 FORMAL_ART.TXT
  $ PRINT/COPIES=2 FORMAL_ART.TXT
  ```

  In interactive mode, you will work faster if you abbreviate. Do not abbreviate commands in command procedures because your command procedure will be difficult to read. Also, the abbreviations might not be valid if new DCL commands are added at a later date.

Other rules governing the format of commands apply mainly to their use in command procedures. See Chapter 6 for more information about using commands in command procedures.

1.2.3 Prompting and System Defaults

Some items must be entered on the command line. If you do not enter them, the system displays a prompt and asks you to supply the missing information. In the following example, the TYPE command expects a file specification. Because a file specification is a required parameter, if you do not enter one, the system requests it. A line beginning with an underscore (_) means the system is waiting for your response.

```
$ TYPE
_File:   WATER.TXT
```

When you are prompted for an optional parameter, press RETURN to omit it. At any prompt, you can enter one or more of the remaining parameters and any additional qualifiers.

If you press CTRL/Z after a command prompt, DCL ignores the command and redisplays the DCL prompt.

Some items need not be specified on the command line. These are called defaults. When DCL does something by *default*, it assumes that you want a command to use certain values or to take certain actions without your having to explicitly specify them. In general, the values and actions are those considered normal or expected by users.

For example, if you do not specify the number of copies as a qualifier for the PRINT command, DCL uses the default value of 1. Unless you specify otherwise, DCL assumes that you have chosen the default. You can override this default behavior and print multiple copies of a file by specifying the following:

```
$ PRINT/COPIES=4 MYFILE.TXT
```

DCL supplies default values in several areas, including command parameters and qualifiers. Parameter defaults are described in the following section; qualifier defaults are described in Section 1.2.5.2.

1.2.4 Entering Parameters

DCL supplies default values for some command parameters. The parameters accepted by a command as well as the specific command parameter defaults supplied by DCL are described in each command description in the *VMS DCL Dictionary*. The following rules apply when specifying parameters:

- Square brackets ([ ]) indicate optional items. In the following example, you do not have to enter a file specification:

  DIRECTORY [file-spec]

  Anything not enclosed in square brackets is required. In the following example, you must enter a device name:

  SHOW PRINTER device-name

- Place required parameters to the left of optional parameters.
- Precede an output file parameter with an input file parameter. In the following example, the input file, LISTS.TXT, is copied to the output file, FORMAT.TXT:

  ```
  $ COPY LISTS.TXT FORMAT.TXT
  ```

The following example reverses the order of the parameters, copying the input file, FORMAT.TXT, to the output file, LISTS.TXT:

```
$ COPY FORMAT.TXT LISTS.TXT
```

- A parameter can be one item or a series of items. If you enter a series of items, separate them with commas (,) or plus signs (+). Any number of spaces or tab characters can precede or follow a comma or a plus sign. Note that some commands regard the plus sign as a concatenator, not as a separator. The parameter section of each DCL command description in the *VMS DCL Dictionary* describes how each command interprets commas and plus signs.

  The following command syntax line shows that you can optionally enter a list of files as the parameter:

  ```
  DELETE file-spec[,...]
  ```

  The following example shows how to specify a list of parameters. Here, three files are copied to a fourth file. The three file specifications—PLUTO.TXT, SATURN.TXT, and EARTH.TXT—constitute the first parameter. PLANETS.TXT is the second parameter.

  ```
  $ COPY PLUTO.TXT,SATURN.TXT,EARTH.TXT PLANETS.TXT
  ```

1.2.5 Entering Qualifiers

The qualifiers accepted by a command are described in each command description in the *VMS DCL Dictionary*. The DCL command description also indicates whether a qualifier accepts a value and what kind of value is required.

You can abbreviate any qualifier name by typing only the first four characters (not counting the slash). You can use fewer than four characters to abbreviate a qualifier name as long as the abbreviated name remains unique among all qualifier names for the same command.

Although you are never required to specify a qualifier, commands have defaults automatically applied. You need to be aware of the defaults that apply for each qualifier. The following sections describe types of qualifiers and qualifier defaults.

1.2.5.1 Types of Qualifiers

The three types of qualifiers are as follows:

- Command qualifiers—A command qualifier modifies a command. Although it is a good practice to place the qualifier after the command name (or, if you are specifying multiple qualifiers, after other command qualifiers that follow the command name), a command qualifier can appear anywhere in the command line.

  In the following example, /QUEUE is a command qualifier. The files SATURN.TXT and EARTH.TXT are queued to the LN03_PRINT queue.

  ```
  $ PRINT/QUEUE=LN03_PRINT SATURN.TXT,EARTH.TXT
  ```

- Positional qualifiers—A positional qualifier can modify commands or parameters and has different meanings depending on where you place it in the command string. If you place a positional qualifier after the command but before the first parameter, it affects the entire command string. If you place a positional qualifier after a parameter, it affects only that parameter.

  In the following example, the first PRINT command requests two copies of the files SPRING.SUM and FALL.SUM. The second PRINT command requests two copies of the file SPRING.SUM, but only one copy of FALL.SUM.

  ```
  $ PRINT/COPIES=2 SPRING.SUM,FALL.SUM
  $ PRINT SPRING.SUM/COPIES=2,FALL.SUM
  ```

- Parameter qualifiers—A parameter qualifier can be used only with certain types of parameters, such as input and output files.

  For example, the BACKUP command accepts several parameter qualifiers that apply only to input and output file specifications. In the following example, the /CREATED and /BEFORE qualifiers, which can be specified only with input files, select specific input files for the backup operation. (For the purposes of this example, multiple copies of the file MYFILE.TXT exist. Only those versions that were created before December 31, 1988, are selected for the backup operation.)

  ```
  $ BACKUP MYFILE.TXT/CREATED/BEFORE=31-DEC-1988 NEWFILE.TXT
  ```

1.2.5.2 Qualifier Defaults

When you omit a specific qualifier from the command line, the system responds with default behavior. For example, when you delete a file with the DELETE command, the system by default does not request confirmation of each delete operation. However, by specifying the DELETE/CONFIRM command, you can override that default behavior and request that you be prompted for confirmation before each file is deleted.

You can specify qualifiers in several ways. The qualifier syntax required by a specific DCL command is given in the command descriptions in the *VMS DCL Dictionary*. The following paragraphs explain the syntax used to describe qualifiers and their defaults:

- Qualifiers with positive and negative forms—These qualifiers have a value of true or false. You do not specify a value, but indicate a true value by simply naming the qualifier. Negate the qualifier by inserting the prefix NO.

  For example, the /CONFIRM qualifier can be expressed positively or negatively. If you omit the qualifier from the command line, the default action is /NOCONFIRM. The syntax for the /CONFIRM qualifier is given in a DCL command description as follows:

  /CONFIRM
  /NOCONFIRM (default)

- Qualifiers that require values—If you use a qualifier that accepts a value, you must specify a value. If you omit the qualifier completely, the default value is applied. For example, if you use the /COPIES qualfier, you must provide a numeric value. If you omit the /COPIES qualfier, the default is /COPIES=1. The syntax for the /COPIES qualifier is given in a DCL command description as follows:

/COPIES=n

If the qualifier accepts a list of values, you must enclose the values in parentheses and separate them with commas as follows:

```
$ DELETE/ENTRY=(230,231) LN03_PRINT
```

The command deletes jobs 230 and 231 from the queue LN03_PRINT.

- Qualifiers that accept value and positive/negative combinations—Some qualifiers combine value and positive/negative characteristics so that the qualifier both accepts a value and allows you to negate the qualifier by inserting the prefix NO. For example, the SET TERMINAL command permits the following choices for the /PARITY qualifier:

```
$ SET TERMINAL/PARITY=EVEN
$ SET TERMINAL/PARITY=ODD
$ SET TERMINAL/NOPARITY
```

- Qualifiers that affect command execution only if specified—The qualifier has no corresponding default. For example, the /BY_OWNER qualifier does not affect the command if it is not specified. The syntax for the /BY_OWNER qualifier is given in a DCL command description as follows:

  /BY_OWNER

- Qualifiers that override other qualifiers—Sometimes a command has a qualifier that is automatically applied as a default. Other qualifiers are available to override the default qualifier.

  For example, the /BRIEF qualifier is applied by default when you specify the DIRECTORY command. That is, the DIRECTORY command generates a listing that includes only the file name, file type, and version number of each file in the directory. You must specify the /FULL qualifier to generate a listing that includes the file name, file type, and version number as well as the number of blocks used, the date of the file's creation, the date the file was last backed up, and so on.

Some commands contain conflicting qualifiers that cannot be specified in the same command line. If you use incompatible qualifiers, the command interpreter usually displays an error message. The command descriptions in the *VMS DCL Dictionary* indicate which qualifiers cannot be used together.

1.2.6 Entering Dates and Times as Values

Certain commands and qualifiers accept date and time values. You can specify these values in one of the following formats:

- Absolute time
- Delta time
- Combination time (combines absolute and delta time formats)

The command descriptions in the *VMS DCL Dictionary* indicate the time formats accepted by commands and qualifiers.

1.2.6.1 Absolute Time

Absolute time is a specific date or time of day. The format for an absolute time is as follows:

```
[dd-mmm-yyyy][:][hh:mm:ss.cc]
```

The fields are as follows:

Field	Meaning
dd	Day of the month; an integer in the range 1–31
mmm	Month; JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC
yyyy	Year; an integer
hh	Hour; an integer in the range 0–23
mm	Minute; an integer in the range 0–59
ss	Seconds; an integer in the range 0–59
cc	Hundredths of a second; an integer in the range 0–99

You can truncate the date or the time on the right. However, if you are specifying both date and time, you must include a colon between them. The date must contain at least one hyphen. You can omit any of the fields within the date and time as long as you include the punctuation marks that separate the fields. A truncated or omitted date field defaults to the corresponding fields for the current date. A truncated or omitted time field defaults to zero. If you specify a past time in a command that expects the current or a future time, the current time is used.

You can also specify an absolute time as one of the following keywords:

Keyword	Meaning
TODAY	The current day, month, and year at 00:00:00.0 o'clock
TOMORROW	00:00:00.00 o'clock tomorrow
YESTERDAY	00:00:00.00 o'clock yesterday

Some examples of absolute time specifications follow:

Time Specification	Result
31-DEC-1988:13	1 P.M. on December 31, 1988
31-DEC	Midnight at the beginning of December 31 this year
15:30	3:30 P.M. today
31-	The 31st day of the current year and month at midnight
31-::30	12:30 A.M. on the 31st of this month

1.2.6.2 Delta Time

Delta time is an offset (a time interval) from the current date and time to a time in the future. The general format of a delta time is as follows:

```
[dddd-][hh:mm:ss.cc]
```

The fields are as follows:

Field	Meaning
dddd	Number of days; an integer in the range 0–9999
hh	Number of hours; an integer in the range 0–23
mm	Number of minutes; an integer in the range 0–59
ss	Number of seconds; an integer in the range 0–59
cc	Number of hundredths of seconds; an integer in the range 0–99

You can truncate a delta time on the right. If you specify the number of days, include a hyphen. You can omit fields within the time as long as you include the punctuation that separates the fields. If you omit the time field, the default is zero.

Some examples of delta time specifications follow:

Time Specification	Result
3-	3 days from now (72 hours)
3	3 hours from now
:30	30 minutes from now
3-:30	3 days and 30 minutes from now
15:30	15 hours and 30 minutes from now

1.2.6.3 Combination Time

To combine absolute and delta time, specify an absolute time plus (+) or minus (–) a delta time. The format for combination time is as follows:

```
"[absolute time][+delta time]"
or
[absolute time][-delta time]
```

The variable fields and default fields for absolute and delta time values are the same as those described in the preceding sections. The delta time value must always be preceded by a plus or minus sign. (Note that the minus sign is the same keyboard key as the hyphen.) Whenever a plus sign precedes the delta time value, enclose the entire time specification in quotation marks. Also, you can omit the absolute time value. If you do, the delta time is offset from the current date and time.

Some examples of combination time specifications follow:

Time Specification	Result
"+5"	5 hours from now
"+:5"	5 minutes from now
–:5	Current time minus 5 minutes
–1-00	Current time minus 1 day. The minus sign (–) indicates a negative offset. The dash (-) separates the day from the time field.

If a qualifier is described as a value that may be expressed as an absolute time, a delta time, or a combination of the two, you must specify a delta time as if it were part of a combination time. For example, to specify a delta time value of five minutes from the current time, use "+:5" (not "0-0:5").

1.3 Entering and Editing DCL Commands

At the DCL level, you can use many individual keys and key combinations to change what you type, to recall commands, or to display information. Table 1–2 lists the keys that allow you to enter and edit DCL commands. Sections that follow describe these keys in greater detail.

DCL also provides you with shortcuts that simplify the typing of commands and command lines. You can establish symbols to use in place of command names and entire command strings. You can define keys, which enable you to enter commands with fewer keystrokes. These shortcuts are described in Section 1.3.7 and Section 1.3.8.

Table 1–2 Keys That Execute Terminal Functions

Key	Function
Keys That Enter DCL Commands	
CTRL/Z and F10[1]	Signals the end of the file for data entered from the terminal. CTRL/Z is displayed as "Exit."
RETURN	Sends the current line to the system for processing. (On some terminals, the RETURN key is labeled CR.) Before a terminal session, RETURN initiates a login sequence.
Keys That Interrupt DCL Commands	
CTRL/C and F6[1]	During command entry, cancels command processing. CTRL/C is displayed as "Cancel."
CTRL/T	Momentarily interrupts terminal output to display a line of statistical information about the current process. This display includes your node and user name, the time, the name of the image you are running, and information about system resources used during your current terminal session.

[1]This key is available only on an LK201 keyboard.

Table 1–2 (Cont.) Keys That Execute Terminal Functions

Key	Function
Keys That Interrupt DCL Commands	
	You can also use the CTRL/T key to determine whether the system is operating. CTRL/T does not return information if the system is temporarily unresponsive or if your terminal is set to NOBROADCAST. In order to use CTRL/T, SET CONTROL=T must be enabled either in the system login command procedure or by you, either interactively or in your login command procedure.
CTRL/Y	Interrupts command processing. CTRL/Y is displayed as "Interrupt." You can disable CTRL/Y with the command SET NOCONTROL=Y.
	Under most conditions, CTRL/Y returns you to the DCL prompt. The program running is still active. You can enter any built-in command (described in Section 1.3.2) and then continue the program with the CONTINUE command. (Press CTRL/W to refresh the screen after you enter the CONTINUE command.)
Keys That Recall Commands	
CTRL/B and Up arrow	Recalls up to 20 previously entered commands.
Down arrow	Displays the next line in the recall buffer.
Keys That Control Cursor Position	
⌫ , DELETE	Deletes the last character entered at the terminal. (On some terminals, the DELETE key is labeled RUBOUT.) The DELETE key also works when line editing is disabled.
CTRL/A, F14[1]	Switches between overstrike and insert mode. The default mode (as set with the SET TERMINAL/LINE_EDITING command) is reset at the beginning of each line.
CTRL/D and Left arrow	Moves the cursor one character to the left.
CTRL/E	Moves the cursor to the end of the line.
CTRL/F and Right arrow	Moves the cursor one character to the right.
CTRL/H, BACKSPACE, and F12[1]	Moves the cursor to the beginning of the line.
CTRL/I and TAB	Moves the cursor to the next tab stop on the terminal. The system provides tab stops at every eighth character position on a line. Tab settings are hardware terminal characteristics that, in general, you can modify. The TAB key also works when line editing is disabled.
CTRL/J, LINEFEED, and F13[1]	Deletes the word to the left of the cursor.
CTRL/K	Advances the current line to the next vertical tab stop.
CTRL/L	Causes the cursor to go to the beginning of the next page. This use of this key is ignored when line editing is enabled.
CTRL/R	Repeats the current command line and leaves the cursor positioned where it was when you pressed CTRL/R.
CTRL/U	Cancels the current input line.

[1]This key is available only on an LK201 keyboard.

Table 1–2 (Cont.) Keys That Execute Terminal Functions

Key	Function
Keys That Control Cursor Position	
CTRL/V	Turns off some of the line editing function keys. For example, if you press CTRL/V followed by CTRL/D, a CTRL/D is generated instead of the cursor moving left one character. CTRL/D is a line terminator at DCL level. When combined with CTRL/V, characters that are not line terminators have no effect. Examples are CTRL/H and CTRL/J. However, certain control keys, such as CTRL/U, retain their line editing functions.
CTRL/X	Cancels the current line and deletes data in the type-ahead buffer.
F7, F8, F9, F11	Reserved for DIGITAL.
Keys That Control Screen Display	
CTRL/O	Alternately suspends and continues display of output to the terminal. CTRL/O is displayed as "Output off" and "Output on."
CTRL/S	Suspends terminal output until CTRL/Q is pressed.
CTRL/Q	Resumes terminal output suspended by CTRL/S.
HOLD SCREEN[1]and NO SCROLL[2]	Suspends terminal output until the key is pressed again.

[1]This key is available only on an LK201 keyboard.

[2]This key is available only on a VT100 keyboard.

1.3.1 Entering a DCL Command

The RETURN key is recognized as a command line terminator. Once you type a command at the DCL prompt, press RETURN to terminate the line and send it to the DCL interpreter for execution. CTRL/Z is also recognized as a command line terminator.

When you enter a command at the terminal or execute an image that results in an error, the system displays an error message. Also, the system sometimes generates messages when a command has completed successfully. For interactive users, messages are normally displayed on the terminal; for batch job users, messages are written to the batch job log file.

Most system error messages have the following format:

```
%FACILITY-L-IDENT, text
```

The fields are as follows:

- FACILITY is a mnemonic for the program issuing the message.
- L is the first letter of the severity code; the severity level can be S (Success), I (Information), W (Warning), E (Error), or F (Fatal or severe error).
- IDENT is an abbreviation of the text.
- Text is an explanation of the error.

Suppress any component of the error message with the SET MESSAGE command. (See the *VMS DCL Dictionary* for the SET MESSAGE qualifiers you need to specify to suppress individual components of the error message.) A SET MESSAGE command remains in effect until you enter the SET MESSAGE command again or log out. The following command suppresses the abbreviation of the explanatory text of the message:

```
$ SET MESSAGE/NOIDENTIFICATION
```

1.3.2 Interrupting and Canceling a DCL Command

After you enter a DCL command, you can temporarily interrupt its execution, run other commands, and then return to executing the command that was interrupted. To interrupt the execution of a command, use CTRL/T, CTRL/Y, or CTRL/C. These keys perform in different ways depending on the type of DCL command currently executing.

As mentioned previously, there are built-in commands and command images. A built-in DCL command (listed in Table 1–1) is part of the command interpreter. A command image is a program that is called by the DCL interpreter. (For example, COPY is a command image.)

A command image can be privileged or nonprivileged. The VMS operating system, the system manager, or you may install a command image as privileged. Privileged command images may vary from system to system. Your system manager can tell you which command images on your system are privileged.

For more information on privileged command images, see the *VMS Install Utility Manual.*

1.3.2.1 Using CTRL/T

CTRL/T interrupts execution of the command, displays a line of statistical information about the current process (node name, process name, system time, currently running image, elapsed CPU time, page faults, direct and buffered I/O operations, and pages in physical memory), and resumes command execution. You can use CTRL/T to interrupt a built-in command, or a privileged or nonprivileged command image. The following example shows how pressing CTRL/T interrupts and then resumes the copy operation:

```
$ COPY [JONES.MEMOS]TODAY.LIS URGENT.LIS
CTRL/T
SATURN::JONES 16:54:17 COPY CPU=00:00:00.54 PF=241 IO=47 MEM=141
$
```

In order to use CTRL/T, SET CONTROL=T must be enabled either in the system login command procedure or by you. You can enable CTRL/T in your login command procedure, or you can enable it interactively by entering SET CONTROL=T at DCL level. (To enable CTRL/T for the current session, you must enable it interactively.) Section 1.1 describes your personal login command procedure; Section 6.3 shows a sample personal command procedure.

1.3.2.2 Using CTRL/Y

When you use CTRL/Y to interrupt a nonprivileged command image, the interrupted command is temporarily suspended and control returns to the DCL interpreter. You see the DCL prompt. To resume execution of the interrupted command, type CONTINUE. Only built-in commands can be entered after CTRL/Y and before CONTINUE without disturbing the interrupted command. Entering any other type of command effectively cancels the interrupted one.

If you are interrupting a privileged command image, you can press CTRL/Y and enter the built-in commands SPAWN, ATTACH, and CONTINUE only, followed by any other command. Entering a command after CTRL/Y other than SPAWN, ATTACH, and CONTINUE effectively cancels the interrupted one.

You can immediately terminate the privileged or nonprivileged command image that you interrupted with CTRL/Y by entering one of the following:

- The STOP or EXIT commands. STOP suppresses any cleanup activities such as the display of error messages. EXIT executes any cleanup procedures before terminating.
- A command that invokes another command image (that is, a nonbuilt-in command), which removes the interrupted command image from memory.

1.3.2.3 Using CTRL/C

CTRL/C works like CTRL/Y in many cases. CTRL/C interrupts the execution of a built-in command. Command images, however, can create different definitions for CTRL/C, in which case pressing CTRL/C does not necessarily interrupt the command and return you to DCL level. For example, the TYPE command (a command image) defines CTRL/C as "cancel." Pressing CTRL/C while typing a series of files to the terminal halts the display of the current file and begins the display of the next file in the series, but does not interrupt the command.

1.3.3 Redirecting the Output of Commands

Many commands allow you to specify the /OUTPUT qualifier to redirect output. The following example shows how the display produced by the DCL command DIRECTORY is redirected to a new text file named FULL.LIS in your default directory:

```
$ DIRECTORY/FULL/OUTPUT=FULL.LIS
```

1.3.4 Recalling Commands

At DCL level, you can recall previously typed command lines and avoid the inconvenience of retyping long command lines. The recall buffer holds up to 20 previously entered commands. Once a command is displayed, you can reexecute or edit it.

Each of the following lets you display the commands stored in the recall buffer:

- CTRL/B
- Up and down arrow keys

- RECALL command

Pressing CTRL/B once recalls the previous command line. Pressing CTRL/B again recalls the line before the previous line, and so on to the last saved command line.

Pressing the up and down arrow keys recalls the previous and successive command, respectively. Press the arrow keys repeatedly to move through the commands.

To examine up to 20 previously typed command lines, type RECALL/ALL. Following is a sample display generated by typing RECALL/ALL:

```
$ RECALL/ALL

 1 SET DEFAULT DISK2:[MARSHALL]
 2 EDIT ACCOUNTS.COM
 3 PURGE ACCOUNTS.COM
 4 DIRECTORY/FULL ACCOUNTS.COM
 5 COPY ACCOUNTS.COM [.ACCOUNTS]*
 6 SET DEFAULT [.ACCOUNTS]
```

Having reviewed the available commands, you can recall a particular command line by typing RECALL and the number of the desired command. The following example shows how to recall the fourth command line stored in the recall buffer:

```
$ RECALL 4
```

After you press RETURN, the fourth command in the list is displayed at the DCL prompt. (The RECALL command itself is not placed in the buffer.)

You can also follow RECALL with the first characters of the command line you want to display. RECALL scans the previous command lines (beginning with the most recent one) and returns the first command line that begins with the characters you typed. For example:

```
$ RECALL E
```

After you press RETURN, the following command line is displayed:

```
$ EDIT ACCOUNTS.COM
```

You can also perform command recall with CTRL/B and the up and down arrow keys. If you are running a utility or an application program that uses VMS screen management software, you can also use these keys to perform command recall. Line editing must be enabled. Some utilities that have this feature are MAIL, DEBUG, SHOW CLUSTER, the System Dump Analyzer (SDA), and the VAXTPU editor.

1.3.5 Editing a DCL Command

Your terminal has a set of keys that you can use to edit a DCL command line. Command-line editing is most useful for modifying long command lines. You can edit command lines that contain typographical errors or command lines that you have recalled and want to modify.

There are many types of terminals, each with its own operating characteristics. In general, they all have standard line editing keys. Line editing keys (keys that let you edit the DCL command line) allow you to control cursor position and are listed in Table 1–2.

For some of the line editing keys to work, the SET TERMINAL/ LINE_EDITING command must be in effect. To see whether or not line editing is enabled, enter the SHOW TERMINAL command, which displays your terminal's current characteristics. Use the SET TERMINAL command to change any of these characteristics. See the *VMS DCL Dictionary* for a description of SET TERMINAL.

For example, the following command line contains one mistake that can be corrected easily using the line editing keys:

```
$ DILETE SCHEDULE.TXT;3
```

If you are using an LK201 keyboard (VT200- and VT300-series terminals), press the F12 key. If you are using a VT100-series terminal, press the BACKSPACE key. Notice that the cursor moves to the beginning of the line. Press the right arrow key once to position the cursor over the "I" in "DILETE." Type the letter E and the typographical error is corrected.

The preceding example assumes that the SET TERMINAL/OVERSTRIKE attribute is in effect, as it is by default. The OVERSTRIKE attribute allows you to replace the incorrect character by typing the correct character over it.

To insert characters in the command line without simultaneously deleting others, change the OVERSTRIKE attribute to the INSERT attribute. While you are editing a command line, you can set the OVERSTRIKE or INSERT attributes temporarily by pressing F14 (or CTRL/A). Use the SET TERMINAL command to set either attribute for your current terminal session.

1.3.6 Controlling Screen Display

Your terminal has several keys that permit you to suspend and resume the display of output to the terminal screen. These keys—CTRL/O, CTRL/Q, and CTRL/S—are useful when a large file is scrolling on your screen and you want to stop the display temporarily.

To suspend output to your terminal, press CTRL/S. To resume the output suspended by CTRL/S, press CTRL/Q. To toggle between suspending and resuming the output, type CTRL/O, which is alternatively displayed as "Output off" and "Output on."

The VT200- and VT300-series terminal also have a HOLD SCREEN key that you can press to alternately hold and resume screen output. On VT100 terminals, the NO SCROLL key performs this same function.

1.3.7 Representing DCL Commands with Symbols

When you specify parameters, multiple qualifiers, and values, one DCL command line can make for much typing. You can simplify your interaction with DCL and save time by establishing *symbols* to use in place of command names and entire command strings you type frequently. A symbol is a name that represents a numeric, character, or logical value. When you use a symbol in a DCL command line, DCL uses the value you assign to the symbol. By defining a symbol as a command line, you can execute the command by typing only the symbol name.

The following example equates the symbol ME to the DCL command SHOW ENTRY:

```
$ ME == "SHOW ENTRY"
```

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 ME assumes a new identity as the DCL command SHOW ENTRY. Once the two are equated, use the symbol ME in place of the SHOW ENTRY command as follows:

```
$ ME

   Jobname    Username  Entry  Blocks  Status
   -------    --------  -----  ------  ------
   STAFF      JONES       202      38  Printing
On printer queue SYS$PRINT
```

You can also equate long command strings to symbols. The following example equates the symbol LN03 with the command string shown:

```
$ LN03 == "PRINT/QUEUE=HUBBUB_LN03A/NOBURST/NOFEED/NOTIFY"
```

By defining a symbol interactively, you create a symbol that is in effect for the current session only. If you want that symbol to be in effect each time you log in, place the symbol definition in your login command procedure. See Chapter 5 for more information about defining symbols. See Section 1.1 and Section 6.3 for more information about creating a personal login command procedure.

1.3.8 Defining Terminal Keys

Key definitions let you customize your keyboard so you can enter DCL commands with fewer keystrokes. A *key definition* is a string of characters that you assign to a particular terminal key. When a key is defined, you can press it instead of typing the string of characters. A key definition usually contains all or part of a command line. When you press a defined key, the command is either displayed on your terminal or executed.

Some definable keys are automatically enabled for definition (like keys PF1 through PF4 and keys F17 through F20 on VT200- and VT300-series terminals). However, before you can define other keys, including KP0 (keypad 0) through KP9 and the keypad keys PERIOD, COMMA, MINUS, and ENTER, you must enable them for definition by entering either the SET TERMINAL/APPLICATION_KEYPAD or the SET TERMINAL /NONUMERIC command. For a complete list of definable keys and for more information on how to create key definitions, see the description of the DCL command DEFINE/KEY in the *VMS DCL Dictionary*.

The following example shows how to equate the PF1 key to the PRINT command and the PF2 key to the qualifier /QUEUE=SATURN_LN03:

```
$ DEFINE/KEY PF1 "PRINT"
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$ DEFINE/KEY PF2 "/QUEUE=SATURN_LN03"
%DCL-I-DEFKEY, DEFAULT key PF2 has been defined
```

When you press the PF1 key and then the PF2 key, the words PRINT/ QUEUE=SATURN_LN03 are echoed and entered as if you had typed them. You need only supply the parameter, which is the file name of the file you want to print. When defining a command line with two or more keys, remember to include all the necessary spaces required in the command line syntax.

The informational message following the key definition indicates the key state for which the key is defined. Key states are described in Section 1.3.8.1. You can suppress the informational message using the /NOLOG qualifier of the DEFINE/KEY command.

A key definition remains in effect until you redefine the key, enter the DELETE/KEY command for that key, or terminate the session. If you want to use a key definition each time you log in, place the key definition in your login command procedure. See Section 6.3 for more information about creating your personal login command procedure.

1.3.8.1 Key States

The same key can be assigned multiple definitions, as long as each definition is associated with a different state. A *key state* is a name you invent to remind you of the types of key definitions grouped under it. If you do not create any key states, all keys are defined in the DEFAULT state.

Specify the /SET_STATE qualifier to the DEFINE/KEY command to change the key state temporarily (the key state remains in effect until you press a definable key or terminate the command line). Use the /IF_STATE qualifier to the DEFINE/KEY to define a key for the specified state.

In the following example, the PF1 key in the DEFAULT state is defined to enter the PRINT command and to change the key state to PRINTERS. The MINUS key is defined in the PRINTERS state to enter the /QUEUE=LN03_PRINT qualifier. The COMMA is defined in the PRINTERS state to enter the /QUEUE=LINE_PRINT qualifer. (Remember to enter the DCL command SET TERMINAL/APPLICATION_KEYPAD to enable keypad key definitions.) The /TERMINATE qualifier places a carriage return after the text; when you press the key, the system attempts to execute the command line.

```
$ DEFINE/KEY/SET_STATE=PRINTERS PF1 "PRINT"
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$ DEFINE/KEY/TERMINATE/IF_STATE=PRINTERS MINUS "/QUEUE=LN03_PRINT"
%DCL-I-DEFKEY, PRINTERS key MINUS has been defined
$ DEFINE/KEY/TERMINATE/IF_STATE=PRINTERS COMMA "/QUEUE=LINE_PRINT"
%DCL-I-DEFKEY, PRINTERS key COMMA has been defined
```

To change a key state permanently (until you log out or change the state again), specify the /LOCK and /SET_STATE qualifiers to the DEFINE/KEY command, or specify the /STATE qualifier to the SET KEY command. After permanently changing the key state, you can recall the DEFAULT key state. However, the system does not provide a mechanism that allows you to determine whether the DEFAULT state was the previous key state.

Because you cannot determine the previous key state after permanently changing the key state, you may want to use the following steps to extend the duration of a temporary state:

1 Use the /SET_STATE qualifier to the DEFINE/KEY command to change your key state temporarily.

2 Each time you define a key for that temporary state, use the /SET_STATE qualifier to reset the temporary state.

1.3.8.2 Examining and Deleting Keys

To examine the key definitions you have created, enter the SHOW KEY command. Specify the /DIRECTORY qualifier to display the states that you have defined as follows:

```
$ SHOW KEY/DIRECTORY
DEFAULT
GOLD
```

Specify the /ALL and /FULL qualifiers to list all the keys in the states specified by the /STATE qualifier. The following example shows that the PF1 key has been defined to enter the DIRECTORY command. The PF2 key has been defined to enter the SET DEFAULT command and change the key state from DEFAULT to DIRECTORIES.

```
$ SHOW KEY/ALL/FULL/STATE=DEFAULT
DEFAULT keypad definitions:
  PF1 = "DIRECTORY"  (echo,terminate,noerase,nolock)
  PF2 = "SET DEFAULT" (echo,noterminate,noerase,nolock,state=DIRECTORIES)
```

To delete a particular key definition, enter the DELETE/KEY command, as shown in the following example:

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

The following example shows how to delete all the keys defined in the GOLD state:

```
$ DELETE/KEY/ALL/STATE=GOLD
%DCL-I-DELKEY, GOLD key PF2 has been deleted
%DCL-I-DELKEY, GOLD key PF3 has been deleted
```

1.4 Utilities

A utility is a program that provides a service. Utilities are invoked with DCL commands. Some utilities—*interactive utilities*—provide a special environment from which you can perform a specific set of tasks. You work interactively with these utilities by entering subcommands and other information in response to the utility's prompt. For example, MAIL is an interactive utility; it has its own prompt and subcommands.

Other utilities are noninteractive. When you invoke a noninteractive utility, it occupies your terminal and executes a task. When the task is complete, you are returned to DCL level and your terminal is once again available. The SORT/MERGE and the LIBRARIAN utilities are two examples of noninteractive utilities.

Some utilities, both interactive and noninteractive, prompt you for a file name. When you are using such a utility (for example, BACKUP, MESSAGE, PATCH, and SORT/MERGE), you can add qualifiers to the DCL command line to tailor the utility to your specific needs, as shown in the following example:

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

To exit from a utility and return to DCL level, type EXIT (and press RETURN) or press CTRL/Z in response to the utility prompt.

The following sections describe the interactive VMS Mail Utility, the VMS Phone Utility, and the Sort/Merge Utility.

1.4.1 Using the Mail Utility

The interactive VMS Mail Utility (MAIL) allows you to send messages to and receive messages from other users on your system or on any other VAX computer that is connected to your system by means of DECnet–VAX. You can also file, forward, delete, reply to, and print messages that you have received.

To invoke the Mail Utility, enter the DCL command MAIL at the DCL prompt. The MAIL prompt appears, signaling that the utility is ready to accept subcommands as follows:

```
$ MAIL
MAIL>
```

For more information about MAIL commands and qualifiers, see the *VMS Mail Utility Manual* or type HELP at the MAIL prompt.

To exit from MAIL, enter the MAIL command EXIT or press CTRL/Z. Note, however, that if you are entering the text of a message, CTRL/Z sends the message. If you wish to cancel the send operation without exiting from MAIL, press CTRL/C.

1.4.1.1 Creating a Mail Subdirectory

When you receive mail messages, they are usually written to files named MAIL$xxxxxxxxxx.MAI located in your top level directory. To avoid the display of these MAI files in your top level directory, use the MAIL command SET MAIL_DIRECTORY, which creates a mail subdirectory and moves all your MAI files to that subdirectory. (The MAIL command SHOW MAIL_DIRECTORY displays the name of the subdirectory that contains all your MAI files.) To move the MAI files from a subdirectory back to your top level directory, use the SET NOMAIL_DIRECTORY command.

1.4.1.2 Sending Mail

You can create and send a mail message interactively to one user or many users with the Mail Utility. Also, you can send a file to other users from within MAIL or from DCL level.

Sending a Message

To send a mail message to any user on your system, invoke the Mail Utility and specify the MAIL command SEND. MAIL prompts you for the name of the user receiving the message, the subject of the message (optional), and the text of the message (optional). The following example sends a message to THOMPSON:

```
MAIL> SEND
To:     THOMPSON
Subj:   Meeting on January 9
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:
I have some new ideas for the Hubbub Cola account.  Let me know when
you're available to talk about them.

--Jeff
```

Press CTRL/Z to send the message. If you decide not to send the message, press CTRL/C. Doing so cancels the send operation without exiting from MAIL.

You can send the same message to several users. To do so, separate their user names with commas, as shown in the following example:

```
MAIL> SEND
To:      THOMPSON,JONES,BARNEY
Subj:    Meeting on January 9
```

If your computer system is part of a network, you can send mail to any other user on the network. If you are sending mail to someone not on your node, you must enter their node name and user name at the To: prompt. (See Section 1.1.1.2 for more information about nodes.) You can address the mail message to the intended recipient on the remote node using the following format:

nodename::username

The following example shows how to send a message to user HIGGINS on node CHEETA:

```
MAIL> SEND
To: CHEETA::HIGGINS
```

MAIL will notify you if the network connection to the remote node is not available.

You may want to use a VMS text editor to compose your message before you send it interactively. (A text editor allows you to enter text from the keyboard and use editing commands to modify that text. See Chapter 8 for a description of the EVE and EDT text editors.) To do so, specify the /EDIT qualifier with the SEND command as shown in the following example:

```
MAIL> SEND/EDIT
```

After you respond to the To: and Subj: prompts, MAIL invokes the text editor. By default, MAIL invokes the EDT editor. (Section 1.4.1.8 describes how to change the default editor.)

If you see an asterisk (*) after you enter the subject line and press RETURN, press the C key to enter the screen editor. To send the message, exit from the editor by pressing CTRL/Z and entering the EXIT command; to cancel the send operation, exit from the editor by pressing CTRL/Z and entering the QUIT command.

You can also use the /EDIT qualifier with the REPLY and FORWARD commands. By specifying /EDIT when you invoke MAIL, you can use the editor for send, reply, and forward operations during the ensuing mail session.

Sending a File

You can send a file to other users from within MAIL or from DCL level. The following example invokes MAIL and uses the MAIL command SEND to send a file:

```
$ MAIL
MAIL> SEND MEMO.TXT
To: EDGELL
Subj: Another memo
```

To send the file, press RETURN; to cancel the send operation, press CTRL/C or CTRL/Y. CTRL/C keeps you within the Mail Utility; CTRL/Y returns you to DCL level.

When you send a file from DCL level, MAIL is invoked, but you do not enter an interactive session, nor do you see the MAIL prompt. When the file is sent, you are automatically returned to DCL level. When you are sending a file in this way, the argument to the (optional) /SUBJECT qualifier must be enclosed in quotation marks if it contains any spaces or nonalphanumeric characters, as shown in the following example:

```
$ MAIL/SUBJECT="Another memo" MEMO.TXT CHEETA::EDGELL
```

To send the file, press RETURN; to cancel the send operation, press CTRL/C.

Sending a Message to a Distribution List

If you need to send one message to many users, you can create a file—called a *distribution list*—that contains a list of users. You then specify that file name rather than the individual user names when you send the message to those users. Use a text editor or the DCL command CREATE to create this file.

When you create a distribution list, type one user name per line. You can also include the names of other distribution lists by specifying an at sign (@) followed by the name of the distribution list. Exclamation points (!) delimit comments in programs and command procedures. DCL ignores everything to the right of the exclamation point when processing the line. For example:

```
! ALLBUDGET.DIS
!
! Budget Committee Members
@BUDGET    ! listed in BUDGET.DIS.
! Staff
HARRINGTON      ! me
BRUTUS::WILSON  ! Martha Wilson
PORTIA::RIPLEY  ! Roy Ripley
```

If the file BUDGET.DIS is not in the same directory as the new distribution list file you are creating, include the file specification for BUDGET.DIS in the new distribution file. (The file specification gives the system all the information necessary to locate a file. Depending on where you create ALLBUDGET.DIS, you may have to specify the device and directory in which BUDGET.DIS is located. See Chapter 2 for more information on file specifications.)

To send a message to a distribution list from within MAIL, type an at sign and the file name at the To: prompt. For example:

```
MAIL> SEND
To:      @ALLBUDGET
Subj:    Tomorrow's Meeting
```

By default, the system looks for a distribution list file with the file type DIS. If the file containing your distribution list has a different file type, you must specify the file name and file type at the To: prompt. If you invoke MAIL while in one directory and the file containing the distribution list is in another, enter the distribution list's file specification at the To: prompt.

1.4.1.3 Reading Mail

Invoke MAIL to read an old or new mail message. Messages you receive are stored in mail files, which have a default file type of MAI. Your default mail file, MAIL.MAI, is created in your top level directory the first time you receive a mail message.

By default, MAIL provides *folders.* New messages are automatically placed in a folder called NEWMAIL; old messages are held in a folder called MAIL. You can move between these folders to read old or new mail messages.

Reading New Messages

When you are logged in and receive a mail message, notice of the new message appears on your screen. (You can screen out notification of incoming messages by specifying the DCL commands SET TERMINAL/ NOBROADCAST or SET BROADCAST=NOMAIL.) For example, a message sent by user FELLINI appears as follows:

```
New mail from FELLINI
```

If you are part of a DEC-net VAX network and someone on a remote node sends you mail, the sender's node and name are indicated.

If you have new mail, you are notified when you log in and when you invoke MAIL. To read a new message, invoke MAIL. MAIL displays the number of mail messages received and prompts for a command, as shown in the following example:

```
$ MAIL

You have 1 new message.

MAIL>
```

To read the new message, press RETURN. The message appears on your screen as follows:

```
#1                  31-DEC-1988  14:12:27        NEWMAIL

From:  FELLINI
To:    JONES
Subj:  Sales presentation on January 9

The meeting to discuss the Hubbub Cola account has been moved
from our conference room to the auditorium.  Dress to impress.

MAIL>
```

You may have another new message. To read your next new message, press RETURN at the MAIL prompt. Pressing RETURN in MAIL is equivalent to specifying the READ command without parameters. When you have read all your new messages, MAIL issues the message "%MAIL-E-NOMOREMSG, no more messages," and continues to prompt for commands until you exit by entering EXIT or pressing CTRL/Z.

If you receive a mail message while you are in MAIL, enter the READ/NEW command to read the new message.

Reading Old Messages

If you have just read a new message and want to reread an old message, enter the following:

```
MAIL> SELECT MAIL
```

This command selects the MAIL folder. The SELECT command allows you to move between folders. Once you are in the MAIL folder, press RETURN at the MAIL prompt or use the READ command to read the old message. The first message (numbered 1) in your default mail file appears on your screen. Press RETURN to display the next message. If the message is too long to display on one screen, press RETURN to display the next part of the message. To skip part of a message and display the next message, type NEXT, which can be abbreviated to "N."

You can display a list of all messages within the current mail folder by entering the DIRECTORY command. You can then display a particular message by entering the READ command and the number of the message, as shown in the following example:

```
MAIL> DIRECTORY
                                                       MAIL
# From              Date             Subject
1 DOLCE::FELLINI    31-DEC-1988      Sales presentation on January 9
2 DOODAH::JONES     31-DEC-1988      status

MAIL> READ 2
```

You can also omit the READ command and enter just the number of the message.

If you have many messages, you can locate a particular message by using the SEARCH command to find a specified string. To search for a string, specify that string as a parameter to the SEARCH command, as shown in the following example:

```
MAIL> SEARCH "appointment"
```

The SEARCH command selects and displays the first message in the current folder that contains the specified string.

To search for a new string, specify the string as a parameter to the SEARCH command. Each time you specify a new string, the SEARCH command starts the search at message number 1. To continue searching the folder for messages that contain the specified string, use the SEARCH command without specifying a parameter.

1.4.1.4 Creating a File from a Mail Message

To copy a mail message to a text file, enter the EXTRACT command while you are reading the message. When you exit from MAIL, the file is listed in your current directory (unless you specify another directory). The following example shows how to create a file named JANUARY_MEETINGS.TXT containing the text of message number 3:

```
MAIL> READ 3
 .
 .
 .
MAIL> EXTRACT/NOHEADER JANUARY_MEETINGS.TXT
%MAIL-I-CREATED, DISK1:[JONES]JANUARY_MEETINGS.TXT;1 created
MAIL>
```

The mail header is composed of the From, To, and Subject lines. Specifying the /NOHEADER qualifier deletes the mail header and copies only the text of the message to the file. If the message has more than one header (as does, for example, a forwarded message), only the last header is deleted.

Use the /APPEND qualifier to the EXTRACT command to copy a message to the end of an existing file. Use the /ALL qualifier to copy all the files in the current folder to an existing file.

1.4.1.5 Deleting Mail

To delete a mail message, either enter the DELETE command while you are reading the message or enter the DELETE command followed by the number (or range of numbers) of the message you want to delete. The following example deletes messages 4, 5, 6, 11, 12, 14, 15, 16, and 17. You can use either the hyphen (-) or the colon (:) to define the range of messages to be deleted.

```
MAIL> DELETE 4-6,11,12,14:17
```

When you delete a message, the message is moved to a folder called WASTEBASKET. During your interactive MAIL session, you can recover any deleted message by moving the message out of the wastebasket folder. (See Section 1.4.1.6 for information on moving messages between folders.) Deleted messages collect in the WASTEBASKET folder until you exit from the current mail file (either by exiting from MAIL or by specifying a different mail file). Once you exit from the current mail file, WASTEBASKET is emptied and the folder itself is deleted. (See Section 1.4.1.6 for a discussion of mail files.)

1.4.1.6 Organizing Mail with Folders and Files

By default, each user account has one mail file (called MAIL.MAI). MAIL helps you organize your messages by providing the following folders as they are required:

- NEWMAIL—Contains all messages that have not been read. If you invoke MAIL when you have a new message, you are placed into the NEWMAIL folder. Once you leave the NEWMAIL folder (either by exiting MAIL or by changing to another folder), MAIL moves any messages that have been read but not deleted to your MAIL folder and deletes the NEWMAIL folder if it is empty.
- MAIL—Contains messages that have been read but not deleted. If you invoke MAIL and have no new messages, you are placed into the MAIL folder.
- WASTEBASKET–Contains messages that have been deleted. This folder and its contents are deleted when you exit MAIL or specify a different mail file.

You can extend this organizational scheme by creating your own folders. Each folder can contain any number of messages.

Like the default folders, the folders you create are normally stored in the mail file MAIL.MAI. You can also create your own mail files; each mail file can contain any number of folders. Although you can create any number of mail files, you usually organize your messages by creating folders rather than by creating mail files.

Creating and Modifying Folders

The following MAIL commands allow you to create and modify folders:

- FILE—Files the current message in the folder you specify. If the folder does not exist, you are asked whether you want to create it. After being filed, the message is automatically deleted from the current folder.
- COPY—Places a copy of the current message into the folder you specify. If the folder does not exist, you are asked whether you want to create it. The following commands copy all messages containing the word MEETING from the current folder to a folder named SCHEDULE. After the commands are executed, you have two copies of each message, one in the current folder and one in folder SCHEDULE. The first command selects and displays the first message containing the word "meeting":

  ```
  MAIL> SEARCH MEETING

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

  This command selects and displays the next message containing "meeting":

  ```
  MAIL> SEARCH

  MAIL> COPY MEETING
  MAIL> SEARCH
  %MAIL-E-NOTFOUND, no messages containing 'MEETING' found
  ```

- MOVE—Synonymous with the FILE command.

Selecting Folders

The name of the current folder is displayed in the top right corner of the screen each time you enter a READ or DIRECTORY command. You can work only with messages that are in your current folder.

To display a list of the folders in your current mail file, enter the DIRECTORY/FOLDER command, as shown in the following example:

```
MAIL> DIRECTORY/FOLDER
Listing of folders in SYS$LOGIN:[JONES]MAIL.MAI;1
     Press CTRL/C to cancel listing
MAIL                                  MEETING_MINUTES
MEMOS                                 PROJECT_NOTES
STAFF
```

To select a new folder as your current folder, use one of the following commands:

- SELECT—Selects the specified folder as the current folder.
- DIRECTORY—Selects the specified folder as the current folder and lists the messages in the folder.
- READ—Selects the specified folder as the current folder and displays the specified message (by default, the first message in the folder).

Deleting Folders

To delete a mail folder, delete all the messages in the folder or move them to another folder. The following example deletes the MUSIC folder:

```
MAIL> SELECT MUSIC
%MAIL-I-SELECTED, 2 messages selected
MAIL> DELETE/ALL
```

Creating and Accessing Mail Files

To create a mail file, move a message into the file by entering the COPY, MOVE, or FILE command as you would to create a folder. When MAIL prompts you for the name of the folder, specify the name of the mail file after the name of the folder.

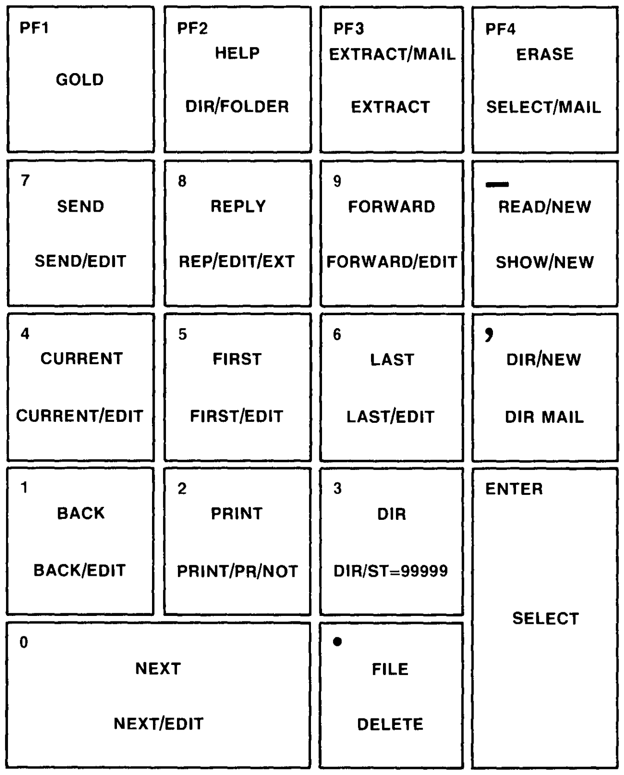
The following example creates the mail file ACCOUNTS.MAI, moves the current message into a folder named FEED in the file ACCOUNTS.MAI, and deletes the message from its current folder and file:

```
MAIL> MOVE
_Folder:  FEED [RET]
_File:  ACCOUNTS [RET]
```

To work within a mail file other than the default mail file, use the MAIL command SET FILE to specify the alternate file. (The MAIL command SHOW FILE displays the name of the current mail file.) When you change mail files, the WASTEBASKET folder of the current mail file is emptied and deleted, and the mail file is closed.

1.4.1.7 Using the Mail Keypad

You can use the keypad to execute commands in the Mail Utility. Most of the keypad keys can execute two commands. To enter the top command for each key shown in the following diagram, press the appropriate key. To enter the bottom command shown in the following diagram, press the PF1 key before you press the key.



ZK-1744-84

For example, to execute the MAIL command SEND, press the keypad key 7 (KP7). To execute the MAIL command SEND/EDIT, press the PF1 key first and then press KP7. (For more information on mail keypad commands, see the *VMS Mail Utility Manual.*)

You can redefine the keypad keys to execute MAIL commands when you are in the Mail Utility. Defining keypad keys in MAIL is similar to defining keypad keys to execute DCL commands; see the DEFINE/KEY command in the *VMS Mail Utility Manual* for more information.

1.4.1.8 Setting the Default Editor

By default, MAIL invokes the EDT editor when you specify the MAIL command SEND/EDIT. By entering the TPU parameter to the MAIL command SET EDITOR, you can specify that the TPU editor be invoked instead. (EVE is the default TPU editor.) The TPU editor remains your default MAIL editor (even if you log out of the system and log back in) until you enter the SET EDITOR EDT command.

The following example sets the default MAIL editor to TPU:

```
MAIL> SET EDITOR TPU
```

In the following example, the default MAIL editor has been set to TPU, and the MAIL command SEND/EDIT has been entered at the MAIL prompt. You see the following screen display:

```
Buffer   MAIN                                | Insert | Forward
```

Enter the text of your message, using EVE commands to move around in the buffer, which is a temporary storage area that exists only during an editing session. Send the message by pressing CTRL/Z. (See Chapter 8 and the *Guide to VMS Text Processing* for information about using EVE and EVE commands.)

You can display the default MAIL editor by entering the MAIL command SHOW EDITOR, as shown in the following example:

```
MAIL> SHOW EDITOR
Your editor is TPU.
```

1.4.2 Using the Phone Utility

The VMS Phone Utility (PHONE) allows you to "talk" by way of your terminal screen to other users on your system or on any other VAX computer connected to your system by means of DECnet–VAX. The Phone Utility simulates the functions and features of a telephone. To invoke the Phone Utility, type PHONE at the DCL prompt. Your screen display splits horizontally into two sections. Your name is in the top section. At the switchhook character (the % sign), type the name of the person you want to call. Type the following to reach user SMITH on node CHEETA:

```
% CHEETA::SMITH
```

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

PHONE rings the other party. If that person answers your call, their name appears in the bottom section. You can begin typing your conversation. If your call is not answered, you will be informed that the person is unavailable.

To answer a call from another user, invoke PHONE. Again, your terminal screen splits into two sections, with you in the top section. Enter the ANSWER command at the switchhook character. When you finish typing your conversation, enter the EXIT command or press CTRL/Z to exit from PHONE.

For more information about PHONE commands, see the *VMS Phone Utility Manual* or type HELP at the PHONE prompt.

1.4.3 Using the Sort/Merge Utility

The VMS Sort Utility (SORT), invoked with the DCL command SORT, sorts records from one or more input files according to the fields you select and generates one reordered output file. The Sort Utility reorders records in a file (or files) so that they are in alphabetic or numeric order, either low to high (ascending) or high to low (descending), according to a portion of each record called the *key*. By default, the Sort Utility sorts on the first character of the first field in each record contained in the input file.

The VMS Merge Utility (MERGE), invoked with the DCL command MERGE, combines up to ten previously sorted files into one ordered output file. By default, MERGE does sequence checking to ensure the input files are in order. The sequence check stops the merge if a record is found to be out of order. To prevent sequence checking during the merge, specify the /NOCHECK_SEQUENCE qualifier.

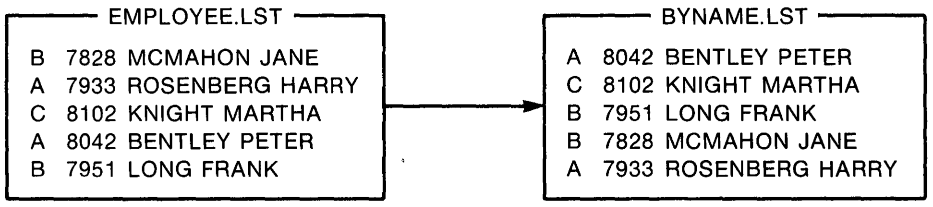
For more information about the SORT/MERGE parameters and qualifiers, see the *VMS Sort/Merge Utility Manual.*

1.4.3.1 Sorting Records

A file record can be thought of as a line of text in a file. Record sorting, the default sort operation, keeps records intact and produces an output file consisting of complete records. Records can be subdivided into fields, which describe individual segments of the record. A field is specified by the starting position of its first character in the record and the length, in characters, of the field. You can sort records based on the contents of certain fields by specifying the field as a sort key.

The following example illustrates an ascending (the default) record sort based on that portion of each record starting at character position 8 and extending to the end of the record (the name):

```
$ SORT/KEY=(POSITION=8,SIZE=15) EMPLOYEE.LST BYNAME.LST
```



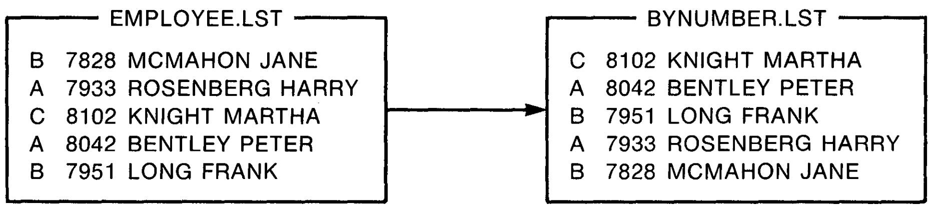
```
B 7828 MCMAHON JANE
A 7933 ROSENBERG HARRY
C 8102 KNIGHT MARTHA
A 8042 BENTLEY PETER
B 7951 LONG FRANK
```

```
A 8042 BENTLEY PETER
C 8102 KNIGHT MARTHA
B 7951 LONG FRANK
B 7828 MCMAHON JANE
A 7933 ROSENBERG HARRY
```

ZK-1748-84

The following example sorts the same file in descending order using the field in character positions 3 through 6 (the number) as the sort key:

```
$ SORT/KEY=(POSITION=3,SIZE=4,DESCENDING) EMPLOYEE.LST BYNUMBER.LST
```



ZK-1749-84

The first parameter of the SORT command names the file or files to be sorted. Multiple files are treated as one large file for sorting purposes. The second parameter provides a name for the ordered output file that the sort will create. The following example sorts the records in two files, EMPLOYEE.LST and EMPLOYER.LST, and creates the ordered output file BYNAME.LST:

```
$ SORT EMPLOYEE.LST,EMPLOYER.LST BYNAME.LST
```

Single Key

By default, the SORT command assumes that a key field in a record has the following characteristics:

- Begins in the first position of a record
- Includes the entire record
- Contains character data
- Will be sorted in ascending order

Use the /KEY qualifier to specify characteristics of the key field other than those assumed by default.

In the following example, the /KEY qualifier specifies that the key field starts in position 8 and is 15 characters long:

```
$ SORT/KEY=(POSITION=8,SIZE=15) EMPLOYEE.LST BYNAME.LST
```

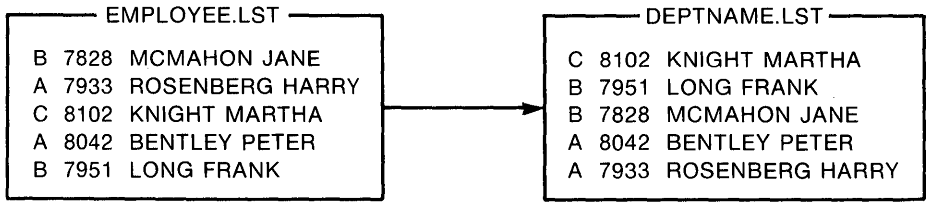
(If an actual key would have to extend beyond the end of the record to meet the size specification—for example, if the key is the last item in a variable-length format—the missing characters are treated as null characters.)

Multiple Keys

You can specify more than one key field, up to a limit of 255 characters. Each key can be ascending or descending. Specify multiple keys in the order of their priority in the sort. For example, the following command sorts records first on the value of position 1 in descending order, then on the value of positions 8 through 27 (or the end of the record) in ascending order:

```
$ SORT/KEY=(POSITION=1,SIZE=1,DESCENDING) -
_$ /KEY=(POSITION=8,SIZE=15) -
_$ EMPLOYEE.LST DEPTNAME.LST
```

The results of the sort specified in the preceding example are as follows:



```
EMPLOYEE.LST
B 7828 MCMAHON JANE
A 7933 ROSENBERG HARRY
C 8102 KNIGHT MARTHA
A 8042 BENTLEY PETER
B 7951 LONG FRANK
```

```
DEPTNAME.LST
C 8102 KNIGHT MARTHA
B 7951 LONG FRANK
B 7828 MCMAHON JANE
A 8042 BENTLEY PETER
A 7933 ROSENBERG HARRY
```

ZK-1764-84

By default, records with identical keys are kept but not sorted predictably. To retain identical keys and arrange them according to the input file order, specify the /STABLE qualifier. To eliminate duplicate keys, specify the /NODUPLICATES qualifier.

1.4.3.2 Other Types of Sorting

In addition to record sorting, you can perform the following types of sort operations:

- Tag sort—Sorts the keys only and then rereads the input file to produce an output file of complete records. The net result is the same as for a complete record sort. A tag sort is useful if disk space is at a premium, because it typically uses less scratch file space while sorting. Time may be saved if the records are large but the keys are relatively small. Specify the /PROCESS=TAG qualifier with the SORT command to generate a tag sort.
- Address sort—Sorts the keys only and produces an output file of record addresses (RFAs) in binary format. An address sort is faster than a record sort, but to take advantage of this feature, you must write a program to associate the record addresses with the records of the input file. Specify the /PROCESS=ADDRESS qualifier to generate an address sort.
- Indexed sort—Sorts the keys only and produces an output file of keys and record addresses (RFAs). The addresses are in binary format. An index sort is faster than a record sort, but, to take advantage of this feature, you must write a program to associate the record addresses with the records of the input file. Specify the /PROCESS=INDEX qualifier to generate an index sort.

1.4.3.3 Character Data Files

The SORT command assumes by default that the files to be sorted contain character data. Characters are sorted according to a collating sequence, which describes the order in which characters are arranged (A, B, C, and so on).

ASCII is the default collating sequence for character data. In general, ASCII orders numbers (0 through 9) first, then uppercase letters (A through Z), and then lowercase letters (a through z).

You can specify the EBCDIC collating sequence to generate an output file that is ordered in EBCDIC sequence (although it remains in ASCII representation). To use the EBCDIC collating sequence, specify the /COLLATING_SEQUENCE=EBCDIC qualifier.

The multinational collating sequence collates characters according to the international character set defined by DIGITAL (see Appendix A). The multinational collating sequence compares for different characters first, then for different diacritical forms of the same character (formed by using diacritical marks as part of "compose sequences" on VT200-series terminals), and then for different cases (uppercase or lowercase) of the same character. To use the multinational collating sequence, specify the /COLLATING_SEQUENCE=MULTINATIONAL qualifier.

Note: **Use caution when using the multinational collating sequence to sort or merge files for further processing. Sequence-checking procedures in most programming languages compare numeric characters. Because the multinational sequence is based on actual graphic characters (and not the codes representing those characters), normal sequence checking will not work.**

1.4.3.4 Noncharacter Data Files

If you sort files containing items other than character data, you must specify the data type of each key. Also, you must take care in calculating starting positions and sizes, because the items being compared may occupy more than one byte. For example, if you are sorting a file that contains 20 characters followed by 3 floating-point numbers in F_floating format, and the key is the last floating-point number, you must make the following specification:

```
$ SORT/KEY=(POSITION=29,F_FLOATING) STATS.RAW STATS.SOR
```

In the example, the character data occupies positions 1 through 20 (20 characters), the first F_floating-point number occupies position 21 through 24, the second F_floating-point number occupies positions 25 through 28, and the third F_floating-point number occupies positions 29 through 32. The size of the floating-point number is not specified (since it is fixed at 4 bytes).

1.4.3.5 Terminal Input

The records to be sorted or merged need not be in a file. You can enter the records directly from the terminal as you enter the SORT or MERGE command.

To enter the input records for a sort or merge operation from your terminal, specify SYS$INPUT as the input file parameter, qualifying it with the size of the longest record (in bytes) and the approximate size of the input file (in blocks). After you enter the command, enter the input records on successive terminal lines. Terminate each record by pressing RETURN. Terminate the file by pressing CTRL/Z.

The following example demonstrates a sort operation in which the input records to be sorted are entered directly from the terminal:

```
$ SORT/KEY=(POSITION=8,SIZE=15) -
_$ SYS$INPUT/FORMAT=(RECORD_SIZE=22,FILE_SIZE=10) BYNAME.LST
B 7828 MCMAHON JANE [RET]
A 7933 ROSENBERG HARRY[RET]
C 8102 KNIGHT MARTHA[RET]
A 8042 BENTLEY PETER[RET]
B 7951 LONG FRANK[RET]
[CTRL/Z]
```

1.4.3.6 Output File Organization

You must specify the file organization of the output file of a sort or merge operation if that organization differs from that of the input file. The following example assumes that EMPLOYEE.LST is an indexed file and you want the output file produced by the sort to be a sequential file (for more information on file organization, see Section 2.1.2):

```
$ SORT/KEY=(POSITION=8,SIZE=15) -
_$ EMPLOYEE.LST BYNAME.LST/SEQUENTIAL
```

If the organization of the output file is indexed, the file must already exist and must be empty. You must also qualify the output file parameter with /OVERLAY.

1.4.3.7 Batch Job Submission

If you are sorting large files, you should consider submitting the sort operation as a batch job, since the sort will require some time. Batch jobs are programs or DCL command procedures that run independently of your current session. See Sections 3.1.2 and 3.1.4 for more information about command procedures and batch jobs, respectively.

If the records to be sorted are in a file, the command procedure you submit as a batch job must contain the SORT command and explicitly set your default directory or include the directory in the command file specifications. The following example submits the DCL command procedure SORTJOB.COM as a batch job. The text of the command procedure is shown following the command line:

```
$ SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]    ! Set default to location of input files
$ SORT/KEY=(POSITION=8,SIZE=15) EMPLOYEE.LST BYNAME.LST
```

You can include the input records in the batch job by placing them after the SORT command, one record per line, as shown in the following example. As with terminal input of records, you specify the input file parameter as SYS$INPUT and qualify it with the record size (in bytes) and the approximate file size (in blocks):

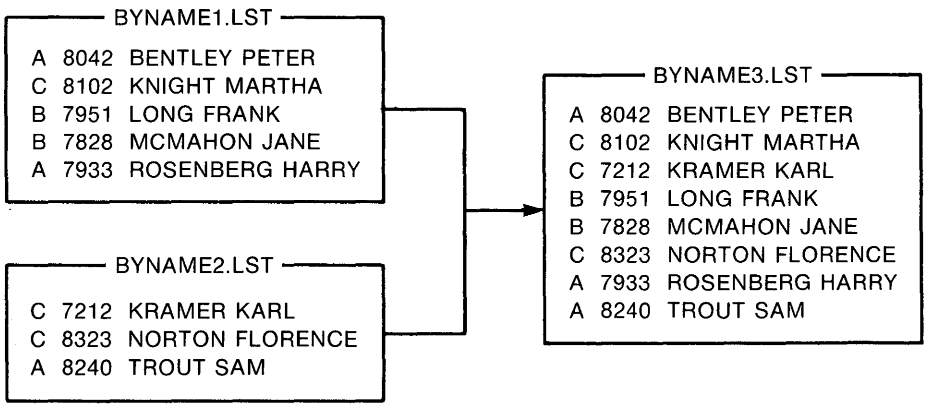
```
$ SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]
$ SORT/KEY=(POSITION=8,SIZE=15)-
SYS$INPUT-
/FORMAT=(RECORD_SIZE=22,FILE_SIZE=10)-
BYNAME.LST
B 7828 MCMAHON JANE
A 7933 ROSENBERG HARRY
C 8102 KNIGHT MARTHA
A 8042 BENTLEY PETER
B 7951 LONG FRANK
```

1.4.3.8 Merging Files

The MERGE command combines up to 10 sorted files into one ordered output file. The input files must all have the same format, and all must have been sorted on the same key fields.

The following example demonstrates the merging of two files based on the field in each record starting at position 8 and extending to the end of the record (the name field):

```
$ MERGE/KEY=(POSITION=8,SIZE=15) BYNAME1.LST,BYNAME2.LST BYNAME3.LST
```



ZK-1771-84

By default, MERGE does sequence checking to ensure the input files are in order. The sequence check stops the merge and reports an error if a record is found to be out of order. To prevent sequence checking during the merge, specify the /NOCHECK_SEQUENCE qualifier.

2 Working with Files and Directories

In the VMS operating system, information is hierarchically stored. At the top of this hierarchy is the *master file directory* (MFD). Your *user file directory* (UFD) is listed in this master file directory, along with the user file directories of other users. Your user file directory (usually called username.DIR) is a file that points to your *top level directory,* which is also called your *login directory* or *default directory* because the system places you there by default when you log in. This top level directory contains the files and subdirectories that you have created or that have been created for you. It is from your top level directory that you perform most of your daily online tasks.

An MFD and UFDs are stored on physical devices called *disks.* The access path to a *file* is through the node and device, through a top level directory, through any *subdirectories,* and then to the file.

Your directory structure resembles a family tree. At the top is your top level directory, which branches off to files and to subdirectories, which branch still further. You can ascend and descend the directory structure to access your files and subdirectories. You can also access other directory structures that have been set up to allow public access. With the correct *process privileges,* you can also access files and directories on remote systems. Process privileges control what commands and functions you are authorized to execute from your account. See the *Guide to Setting Up a VMS System* for more information about process privileges.

2.1 Files

A file contains information. This information can be machine-readable data that the computer understands. It can also be text you enter and manipulate. The text in the file might be the text of a document; a program that you can execute, written in a language such as C or Pascal; or a list of addresses. You can examine the data in these files by displaying the files on a terminal screen and printing them on paper.

Every file must have a file name or file type to identify it to both the system and you. A file also has a version number. This file information is specified using the following format:

```
filename.type;version
```

Taken together, these elements form a *file specification.* The following section describes the elements of a file specification and the rules for specifying these elements.

2.1.1 File Names, Types, and Versions

When you create a file, give it a name that is meaningful to you. The file name can be from 0 through 39 characters chosen from the letters A through Z (upper- or lowercase), the numbers 0 through 9, an underscore (_), a hyphen (-), or a dollar sign ($). Do not use a hyphen as the first or last character in the file name. Do not begin a file name with a dollar sign, although it is a legal character within the file name.

A file type identifies the nature of a file. The file type can be from 0 through 39 characters and must be preceded by a period. The rules for creating file names also apply to file types.

Including a file type is optional. With certain commands, if you omit the file type, the system applies a default value. Table 2–1 lists some of the more common default file types used by DCL commands. It also lists the default file types for some high-level language source programs.

Table 2–1 Default File Types

File Type	Contents
Default File Types for DCL Commands	
CLD	Command description file
COM	Command procedure file
DAT	Data file
DIS	Distribution list file for the MAIL command
DIR	Directory file
EDT	Startup command file for the EDT editor
EXE	Executable program image file created by the linker
HLP	Input source file for help libraries
JOU	Journal file created by the EDT editor
LIS	Listing file created by a language compiler or assembler; default input file for the PRINT and TYPE commands
LOG	Batch job output file
MAI	MAIL message file
MEM	Output file created by DIGITAL Standard Runoff (DSR)
OBJ	Object file created by a language compiler or assembler
RNO	Input source file for DIGITAL Standard Runoff
SIXEL	Sixel graphic file
SYS	System image
TJL	Journal file created by the VAXTPU and ACL editors
TMP	Temporary file
TPU	Command file for the VAXTPU editor
TXT	Input file for text libraries or MAIL command output

Table 2–1 (Cont.) Default File Types

File Type	Contents
Default File Types for Language Source Programs	
ADA	Input source file for the VAX Ada compiler
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
FOR	Input source file for the VAX FORTRAN compiler
MAR	Input source file for the VAX MACRO compiler
PAS	Input source file for the VAX Pascal compiler
PLI	Input source file for the VAX PL/I compiler

In addition to a file name and type, every file has a version number. Version numbers are decimal numbers from 1 to 32,767 that differentiate versions of a file. When you initially create a file, the system assigns it a version number of 1.

You may have several versions of a file. Unless you specify a version number, the system uses the highest existing version number of that file. When you modify that file, the system saves the original file and produces a modified output file. By default, this output file has the same name and type as the original, but the version number is incremented by one.

Version numbers must be preceded with a semicolon or a period. When the system displays file specifications, it generally displays a semicolon in front of the file version number.

The following example shows how to display the latest version of the file STAFF_VACATIONS.TXT. Because the system displays the latest version of a file by default, you can omit the version number from the file specification.

```
$ TYPE STAFF_VACATIONS.TXT
```

You can refer to versions of a file in a relative manner by specifying a zero or a negative version number. Specifying zero locates the latest (highest numbered) version of the file. Specifying –1 locates the next-most-recent version, –2 the version before that, and so on.

You can control the number of versions of a file by specifying the /VERSION_LIMIT qualifier to the DCL commands CREATE/DIRECTORY, SET DIRECTORY, and SET FILE.

2.1.2 File Characteristics

A file consists of records, each of which consists of a number of bytes of data. (Bytes are commonly used to represent characters.) A file's characteristics describe the physical layout of a file and determine how the file is treated during file operations. Specifically, file characteristics describe the following features of a file:

- File organization—Sequential, indexed, or relative.

  The records of a *sequential file* are arranged one after another in the order of creation. Records must be read from the file in order. The file must be rewritten (that is, another file or version of the file must be created) to update it.

  The records of an *indexed file* are arranged randomly and accessed through one or more indexes. An index contains a portion of each record called a key; the keys are arranged in sequence from lowest to highest (by binary, numeric, or ASCII value depending on data type); one key is called the primary key. You can read a record directly (randomly) by specifying an index and the value of one of its keys. You can read records sequentially by specifying an index—records are read in ascending sequence according to the key values for that index, starting with the current record. Update an indexed file in place by adding, deleting, or changing records. Indexed files require more space since, in addition to the data, the indexes must be stored.

  The records of a *relative file* are arranged in fixed-length, numbered cells. The cell numbers are used to determine the position of the record in the file. As with indexed files, you can read records sequentially or randomly. Typically, relative files are created and accessed by programs, rather than from DCL command level.

- Record format—Indicates the way all records in a file appear physically on the recording surface of the storage medium. Record format is defined in terms of record length and can be fixed length, variable length, variable length with fixed control area (VFC), or stream. All records in a fixed-length file are the same size. Records in a variable-length file vary in size. Records in a VFC file have a fixed-length header followed by a variable part. Note that VFC record format is not applicable for indexed files. Records with stream format are delimited with special control characters.

- Data type—Strictly speaking, a file does not have a data type, because programs processing a file must know how each item in the file is to be interpreted. However, a file whose records contain all character data (each item is one byte, interpreted according to ASCII conventions) is called a text, or character, file. A file whose data is formatted as integers, floating-point numbers, object code, or other non-ASCII data is called a binary file.

- Carriage control—New line (also known as "implied," "carriage return," or "CRLF"), FORTRAN carriage control, none, or print. New line places a carriage return and line feed at the end of each record when it is displayed or printed. FORTRAN carriage control uses the first character of each record to specify carriage-control information. "None" does not place carriage-control characters into a file; if you want to include control characters in the file, you must specify them as part of the data in the file. Note that the PRINT and TYPE commands interpret carriage-return, line-feed, and form-feed characters embedded in records. Print carriage control interprets the two bytes of each VFC record as prefix and postfix carriage-control information.

Files you create using the editor or the CREATE command use new-line carriage control. Each time you press RETURN, you create a new record. When the file is printed or typed, each record appears on a new line. Files you create using the OPEN, WRITE, and CLOSE commands use print carriage control. Each WRITE command adds a new record (in VFC format) to the file.

- File size—The size of a sequential file with fixed-length records can be calculated by multiplying the number of records and the size of each record. Variable-length records require two extra bytes per record, and indexed files require space for the indexes. In addition to the files themselves, the VMS system uses disk space to store directory entries, file headers, and other file-maintenance information.

At DCL level, you normally deal with sequential, variable-length text files, although some commands permit access to indexed files. You can examine a file's characteristics with the /FULL qualifier of the DIRECTORY command, as shown in the following example:

```
$ DIRECTORY/FULL RECEIPTS.DAT

Directory DISK1:[JONES.TAXES]

RECEIPTS.DAT;15                File ID: (103,75,0)
Size:      64/66           Owner:    [200,200]
Created:  02-JUN-1988 17:47:26.30
Revised: 31-DEC-1988 11:28:51.35 (2)
Expires:  <None specified>
Backup:  30-DEC-1987  22:48:08:23
File organization:   Sequential
File attributes:     Allocation=153, Extend=0 Global Buffer Count = 0
                     No version limit
Record format:       Variable length, maximum 82 bytes
Record attributes:   Carriage return carriage control
Journaling enabled:  None
File protection:     System:RWED, Owner:RWED, Group:RW, World:
Access Control List: None

Total of 1 file, 64/66 blocks.
```

The file size of the preceding example indicates that 64 blocks have been used out of the 66 allocated. (File size is the number of actual blocks used of the blocks that have been allocated; more will be allocated by the system as needed.) If you are only interested in the size of the file (or several files), use the /SIZE qualifier. The following example lists the number of blocks used by the files in one directory.

```
$ DIRECTORY/SIZE

Directory DISK1:[JONES.TAXES]

BILLING.DAT;31        62
LEGAL.TXT;9           20
LOCAL.DIS;2            4
PROPERTY.DIR;1         7
RECEIPTS.DAT;15       64
SALES.DIR;1            5

Total of 6 files, 162 blocks.
```

2.2 Directories

A directory is a special kind of file that catalogs (by name and location) a set of files. A directory file contains the following information for every file cataloged within it:

- The file name, type, and version number
- A pointer to the file header, which describes, among other things, the file's owner, protection code, and location

A directory file has the following format:

```
directory.DIR;1
```

For example, DOG.DIR;1 is a directory file. Because you cannot edit a directory file, all directory files have a version number of 1.

In addition to the file name, a *file specification* can include the directory in which the file is located. The following example shows the file specification used to display the file STAFF_VACATIONS.TXT located in the directory [JONES]:

```
$ TYPE [JONES]STAFF_VACATIONS.TXT
```

If you omit the directory name from the file specification, the current directory is assumed by default.

2.2.1 Directory Structure

Each disk contains a main directory that is set up by the system manager. This main directory is called the master file directory (MFD). The MFD contains a list of user file directories (UFDs). User file directories are files in the master file directory that point to top level directories. Your top level directory is also called your *login* or *default directory*. Unless your account has been specially modified to do otherwise, by default the system places you in your top level directory when you log in.

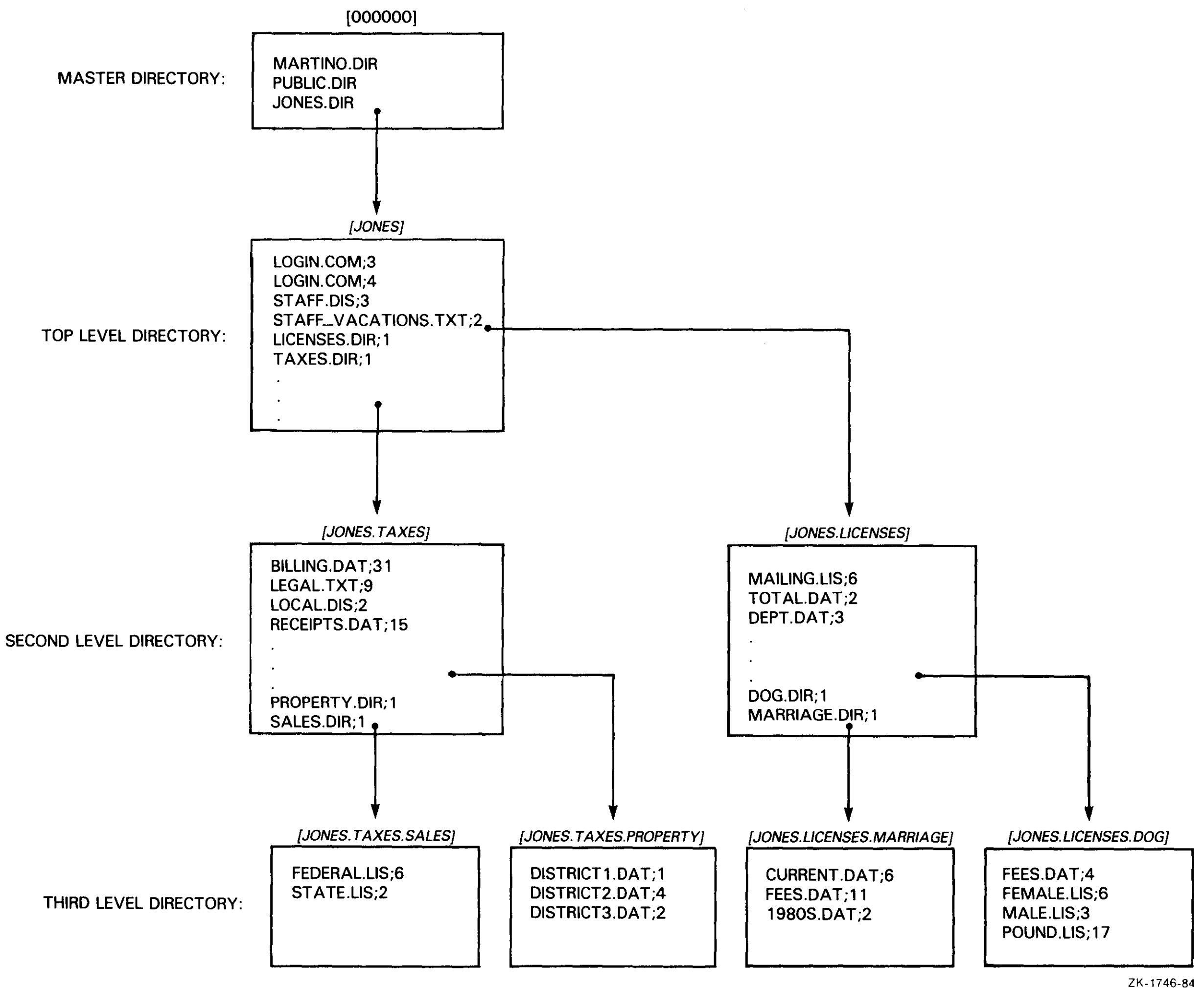
A UFD exists for each user on the system. It contains the names of and pointers to files cataloged in a user's directory. A *subdirectory* is any directory file that is not an MFD or a UFD. Subdirectories let you organize files into meaningful groups. Like a directory, a subdirectory contains names and pointers for the files cataloged within it. It can contain an entry for another subdirectory, which can contain an entry for another subdirectory, and so on to seven levels of subdirectories. This structure (a first level directory plus a maximum of seven levels of subdirectories) is called a *hierarchical directory structure.*

Figure 2–1 shows a sample directory hierarchy. At the top of the structure is the MFD. Its directory name is [000000]. (Directory names are always enclosed in either square brackets ([ ]) or angle brackets (< >).) Figure 2–1 contains entries for user file directories including MARTINO.DIR, PUBLIC.DIR, SCHULTZ.DIR, and JONES.DIR. The top level directory [JONES] exists as a user file directory named JONES.DIR;1 in [000000].

Assume that you are user JONES. At login, you are placed in [JONES], your default directory. [JONES] contains four nondirectory files and two directory files. The directory file TAXES.DIR;1 points to the [JONES.TAXES] subdirectory; LICENSES.DIR;1 points to the [JONES.LICENSES] subdirectory. (Subdirectories are specified by concatenating the subdirectory name to the name of the directory one level above it.) The [JONES.LICENSES] subdirectory contains three nondirectory files and two directory files. The directory file DOG.DIR;1 points to the [JONES.LICENSES.DOG] subdirectory; MARRIAGE.DIR points to the [JONES.LICENSES.MARRIAGE] subdirectory.

This sample directory structure is the basis for the examples in this chapter, which demonstrate how to ascend and descend the directory structure and how to access files within this structure.

Figure 2–1 Directory Structure



2.2.2 Directory Names

Use a named directory specification to refer to a directory. A named directory specification consists of a top level directory name that can be followed by a maximum of seven subdirectory names.

A named directory specification has the following format:

```
[directory.subdirectory[.subdirectory...]]
```

A directory name can contain up to 39 alphanumeric characters. Any characters valid for file names are also valid for directory names. Enclose the directory name in either square brackets ([]) or angle brackets (<>).

Default and *wildcard characters* can be applied. You use wildcard characters to apply DCL commands to multiple files rather than to one file at a time and to move around the directory structure. See Section 2.6.6.3 for more information about using wildcard characters in a named directory specification.

2.3 Devices

Files are stored on devices. In the VMS operating system, devices are classified as follows:

- Mass storage devices save the contents of files on a magnetic medium. Files saved this way can be accessed, updated, modified, or reused at any time. Disks and magnetic tapes are mass storage devices.
- Record-oriented devices read and write only single physical units of data at a time and do not provide online storage of the data. Terminals, printers, mailboxes, and card readers are record-oriented devices. (Printers and card readers are also called unit-record devices.)

A device name has the following three parts:

- The device type, which identifies the hardware device. (For example, an RP06 disk has the device type DB, and a TE16 magnetic tape has the device type 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.

The files you commonly access are stored on disks or magnetic tape. Your user file directory (UFD) and your default directory with all your files and subdirectories are located on a disk. You can use a file specification that contains directory information only if the file is located on a disk. Magnetic tapes do not have directory structures. To obtain a file stored on tape, use a file specification that contains only file information.

If you want to access a file that is not located on your default device, you must specify the device name. For files on disks, you must also specify the directory where the file is cataloged.

You can use physical, logical, or generic names, described in the following sections, to refer to devices.

2.3.1 Physical Device Names

Each physical device known to the system is uniquely identified by a *physical device name*. The physical device name identifies the kind of device, for example a storage disk or a terminal. A device name has the following format:

ddcu

The fields are as follows:

dd	Device code that represents a device type.
c	Controller designation. The controller designation, along with the unit number, identifies the location of the device within the hardware configuration of the system. Controllers are designated with alphabetic letters A through Z.
u	Unit number. The unit number, along with the controller designation, identifies the location of the device within the hardware configuration of the system. Unit numbers are decimal numbers from 0 through 65535.

The maximum length of the device name field, including the controller and the unit number, is 15 characters. When you specify a device name as part of a file specification, terminate it with a colon (:). If you do not specify a logical or physical device name, your default device name is supplied.

In addition to directory and file information, a *file specification* can include the device on which a directory and file are located. In the following example, the file STAFF_VACATIONS.TXT is located in the directory [JONES], which is located on a device with the logical name DISK2. To display the file from device DISK1, enter the following file specification:

```
$ TYPE DISK2:[JONES]STAFF_VACATIONS.TXT
```

A disk or tape must be mounted on a device in order to be recognized by the system as a *volume*. The system also recognizes *volume sets*. A volume set consists of two or more related volumes.

To access a file on a disk volume set, you have the following options:

- Specify the name of the device on which the first volume in the set is mounted. For example, if the disks DUA1 and DUA2 have been mounted as one volume set, access a file on that disk volume set by specifying DUA1 in the file specification.
- Specify the logical name assigned to the volume set when it was mounted. This is the preferred method because it allows system managers to move the volume to another device without disrupting users.

To access a file on a tape volume set, specify any device that has been allocated to that volume set. For example, if the tapes MUA1 and MUA2 have been mounted as one volume set, access a file on that tape volume set by specifying either MUA1 or MUA2 in the file specification.

2.3.2 Logical Device Names

Your system manager has probably set up logical names to represent the devices on your system. *Logical device names* can be used to equate a somewhat cryptic device name to a short, meaningful name. Use these logical names, rather than the physical device names, to refer to devices.

By using logical names, users can avoid making specific references to physical devices whose names may change. In daily system management, devices are sometimes shuffled about. You might not know when a storage disk is added to your system configuration and a frequently accessed file moved to that new disk. You continue to access the file with the same file specification because your system manager has redefined the logical name that previously pointed to one device to point to the new device.

Consequently, if your file specification contains a logical device name, you can access the file regardless of which physical device holds the disk or tape on which the file is stored. Your system manager will ensure that logical device names are always equated to the correct physical devices.

In the following example, a logical device name is used to specify the device containing the disk volume with the file STAFF_VACATIONS.TXT. Note that, like a physical device name, a logical device name must be terminated with a colon.

```
$ TYPE DISK1:[JONES]STAFF_VACATIONS.TXT
```

The VMS system also offers a special type of logical device name called a *concealed device name*. If a device has a concealed device name, the logical name (not the physical device name) will be displayed in system messages that refer to the device.

See Chapter 4 for a complete discussion of the use of logical names.

2.3.3 Generic Device Names

A *generic device name* consists of the device code and omits the specific controller or unit number. When you use a generic device name, the system locates the first available controller or device unit whose physical name satisfies the portions of the generic device name you specified.

When you use the DCL commands ALLOCATE and MOUNT, the system allows you to specify generic device names in which the controller, the unit number, or both is not specified. For example, if you enter the ALLOCATE command and specify only a device type, the ALLOCATE command locates the first available unit of that type.

For all other DCL commands, the system goes to controller A if you omit the controller designation, and to unit number 0 if you omit the unit number.

2.4 Full File Specification

As discussed in Chapter 1, a node is one of several VMS systems connected to form a computer network. If your VMS system is part of a network, the node that you access when you log in is your local node. Other nodes in the network are remote nodes. As a general user of the network, you can perform file operations on nodes other than the one at which you are logged in.

A node name can contain 1 to 6 alphanumeric characters and must contain at least one alphabetic character. A node name must always be followed by a double colon (::). You can also use a logical node name in place of the node name. For more information on logical node names, see Section 4.8.

When you add node information to the device, directory, and file information, you create a *full file specification*. A full file specification completely describes the access path the system uses to locate and identify a file. Because it describes the network node on which the file resides, a full file specification is also known as a *network file specification.*

The format for a full file specification follows:

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

Assuming the file protection is set to allow remote access, the following example shows the full file specification used to display the file STAFF_VACATIONS.TXT on node HUBBUB:

```
$ TYPE HUBBUB::DISK1:[JONES]STAFF_VACATIONS.TXT
```

If you specify your local node in the file specification, DECnet–VAX logs you in over the network to perform the file operation, even though the file exists on your local node. To save time and reduce system overhead when accessing a file on your current node, omit the node name in the file specification.

The full file specification can optionally include an access control string. To indicate that you are authorized to access a file protected against network access, include an *access control string* (a 0- to 42-character string that contains a user name and password). DECnet–VAX uses this access control string to log in at the remote node. The device, directory, and file information is passed to the remote node and interpreted there.

The usual format for a full file specification that contains an access control string is as follows:

```
node-name"username  password"::device:[directory]filename.type;version
```

Assume again that you are user JONES. The following example includes the access control string necessary for you to copy the file STAFF_SALARIES.TXT from your account on node HUBBUB to your default directory on another node. The asterisk at the end of the file specification is a wildcard character. Here, it instructs the system to duplicate the file name STAFF_SALARIES.TXT when that file is copied to the remote node.

```
$ COPY HUBBUB"JONES PANDEMONIUM"::DISK1:[JONES]STAFF_SALARIES.TXT *
```

If you omit the access control string, the login information sent to the remote node is determined as follows:

- If a proxy login account exists for you on the remote node, the system logs you in using that account. (A proxy login account gives access privileges on a remote node to selected users who do not have a private account on that node.)
- If no proxy login account exists, the system uses the default DECnet–VAX account for that node as specified by the local system manager.

If a file resides on a non-VMS system (that is, the file specification does not conform to VMS syntax), the name of the file as specified in this format is enclosed in a quoted string. The quotation marks prevent the local VMS system from performing syntax checking or logical name translation. In the following example, the file TEST?.DAT contains a question mark character, which is not recognized as a valid file name character in VMS:

```
$ COPY BOSTON::"TEST?.DAT" *
```

2.4.1 Using System Default Values When Specifying Files

When you enter a file specification, you can omit fields and let the system supply default values for these fields. Table 2–2 summarizes the defaults applied to each field in a file specification.

Note that the system supplies the defaults described in Table 2–2 for the first input file specification that you enter on a DCL command line.

Table 2–2 File Specification Defaults

Field	Defaults
Node	The system assumes that the default is the local system.
Device	The system uses the device (usually a disk) established at login or by the SET DEFAULT command. Devices are usually identified with logical names. If a physical device (ddcu) is used and a controller designation is omitted, the controller designation defaults to A. If a unit number is omitted, the unit number defaults to 0. (The ALLOCATE, MOUNT, and SHOW DEVICES commands, however, treat a device name that does not contain controller or unit numbers as a generic device name.)
Directory	The system uses the directory name established at login or by the SET DEFAULT command.
File name	No defaults are applied to the first file name in an input file specification. Most commands apply default output file names based on the file name of an input file.
File type	Various commands apply defaults for file types, based on the standard file type conventions summarized in Table 2–1.
File version	For input files, the system assumes the highest version number.

Table 2–2 (Cont.) File Specification Defaults

Field	Defaults
	For output files, if no file with the specified file name and file type exists in the current directory, the file is created with a version number of 1. However, if one or more versions do exist, the next highest version number is used.

When you enter more than one input file specification, the system applies temporary defaults for node, device, and directory names. The system uses the preceding file specification in the list that included this information. The following examples show how the system applies temporary defaults.

The following example copies the latest versions of DISK1:[JONES.TAXES.PROPERTY]DISTRICT1.DAT and DISK1:[JONES.TAXES.PROPERTY]DISTRICT2.DAT to the file AUDIT.DAT in the default directory. By default, the output (second) file specification parameter assumes the corresponding fields of the first file specification.

```
$ COPY DISK1:[JONES.TAXES.PROPERTY]DISTRICT1.DAT,DISTRICT2 AUDIT
```

When you want to specify the default file type, be sure to omit the period (which indicates a null file type).

The following example copies the files DISK1:[JONES.TAXES]BILLING.DAT and DISK1:[JONES]STAFF.DIS to DISK1:[JONES]ASSIGNMENTS.DAT. Note that the output (second) file specification parameter uses the default directory, not the directory in the first input file specification.

```
$ SET DEFAULT DISK1:[JONES]
$ COPY [.TAXES]BILLING.DAT,[]STAFF.DIS ASSIGNMENTS.DAT
```

The system applies defaults in different ways depending on the DCL command you specify. If, for example, you substitute the RENAME command for the COPY command in the previous example, you will produce one file [JONES.TAXES]ASSIGNMENTS.DAT and another [JONES]ASSIGNMENTS.DAT. See the *VMS DCL Dictionary* for more information on the defaults applied to specific DCL commands.

2.5 File Operations

File operations involve the creation, use, and deletion of files. File operations include the following:

- Displaying the contents of files
- Creating files
- Modifying files
- Copying files
- Renaming files
- Deleting files
- Printing files
- Purging files from directories

- Using wildcards

As a VMS user, you can also perform file operations over the DECnet network if you have sufficient privileges. You can display locally the contents of remote directories and files and copy files from node to node. You can print files at the remote node where they reside, copy them to a remote printing device, or copy them to the local node for printing. DCL commands permit you to access common or public directories or databases located on any node on the network. You can display their contents or print or copy the files.

See the descriptions of the DCL commands in the *VMS DCL Dictionary* for more information on specific file operations you can perform locally and over the network.

2.5.1 Using Wildcards with File Specifications

By using wildcard characters, you can apply a DCL command to multiple files rather than to one file at a time. The command applies to all files that match the portion of the file specification entered.

With many DCL commands, you can use an asterisk (*) and a percent sign (%) as a wildcard in directory names, file names, and file types. You can also use the asterisk, but not the percent sign, in version numbers.

The use of wildcard characters in DCL commands varies with the individual command. For more information on using wildcards with a particular DCL command, see the *VMS DCL Dictionary*.

2.5.1.1 The Asterisk (*) Wildcard Character

Use the asterisk wildcard character to match the following:

- An entire field, or a portion of it, in the directory, file name, and file type fields
- The entire version number field, but not a portion of it

The following example displays all versions of the file LOGIN.COM in the directory [JONES]:

```
$ TYPE [JONES]LOGIN.COM;*
```

The following example displays all versions and all file types of all files that begin with the word *STAFF* in the directory [JONES]. This would include STAFF_VACATIONS.TXT and STAFF.DIS.

```
$ TYPE [JONES]STAFF*.*;*
```

You can also use the asterisk wildcard character in a directory specification. The following example displays all versions of all files with the file type .LIS in all subdirectories one level down from [JONES]:

```
$ TYPE [JONES.*]*.LIS;*
```

You can use the asterisk in the name, type, and version fields in output file specifications. Use an asterisk in an output file specification when you want the output files to match the corresponding field in the input files.

The following example copies the latest versions of all DAT files in [JONES] to new files in [JONES] with the same name but a file type of SAV:

```
$ COPY *.DAT *.SAV
```

The following example copies the latest versions of all DAT files in [JONES] beginning with the characters 19 to new files with the same names but in the directory [SAVE]:

```
$ COPY 19*.DAT [SAVE]*.*
```

2.5.1.2 The Percent (%) Wildcard Character

The percent sign wildcard character can be used as a substitute for any single character in a file specification. You can use the percent sign in the directory, file name, and file type fields. You cannot, however, use the percent sign in the version number field.

The following example displays the latest versions of all DAT files whose names begin with DISTRICT:

```
$ TYPE [JONES.TAXES.PROPERTY]DISTRICT%.DAT
```

This display would include the files DISTRICT1.DAT, DISTRICT2.DAT, and DISTRICT3.DAT. The file DISTRICT4_5.DAT would not be displayed because it has more than one character after DISTRICT, nor would the file DISTRICT.DAT be displayed. The percent sign replaces one character position in a field, but there must be a character to replace.

2.5.2 Displaying the Contents of Files

You can display the contents of files on your terminal screen by using the TYPE command or by invoking an interactive text editor with the /READ_ONLY qualifier. The following example displays the file STAFF_VACATIONS.TXT:

```
$ TYPE STAFF_VACATIONS.TXT
```

The following example displays the file COMPANY_HOLIDAYS.TXT, which is located on remote node CHAOS:

```
$ TYPE CHAOS::DISK2:[PUBLIC]COMPANY_HOLIDAYS.TXT
```

If more than one file is listed in the TYPE command, the files are displayed in the order specified; if wildcard characters are used, the files are displayed in alphabetical order.

To stop the scrolling of the text on the screen temporarily, press the HOLD SCREEN key (F1 on VT200- and VT300-series terminals); to resume scrolling, press the HOLD SCREEN key again. To stop the display and return to DCL command level, press CTRL/Y or CTRL/O.

If you specify the /PAGE qualifier to the TYPE command, you can view one screen at a time. The system prompts you to press RETURN when you want to see the next screen.

By invoking an interactive text editor (for example, EVE or EDT) with the /READ_ONLY qualifier, you can use interactive editing commands to move around in a file and search for specific sequences of characters. The /READ_ONLY qualifier prevents you from modifying the file as you display it. Control characters are displayed rather than being interpreted when you use /READ_ONLY, however. For example, the form-feed character appears as <FF> rather than producing a form feed.

2.5.3 Creating and Modifying Files

The most versatile interactive tool for creating and modifying files is the interactive editor. EVE and EDT are two such editors; VMS supports several others. See Chapter 8 for a description of the EVE and EDT editors.

You can also create and modify files by using the DCL commands CREATE, COPY, and RENAME. The CREATE command creates a text file. You enter the CREATE command and then type lines of text, as shown in the following example:

```
$ CREATE POUND.LIS
Tag #23, Elmer Doolittle, notified
Tag #37, James Watson, notified
No tag, light brown, 30 lbs., looks part beagle
CTRL/Z
```

Pressing CTRL/Z signals the end of the file and returns you to DCL command level. You cannot modify a file with the CREATE command. Once you have pressed RETURN, you cannot return to a previous line to modify a word.

The COPY command duplicates the contents of the old file in a new file. The following example copies FEES.DAT to RECORDS.DAT in the default directory:

```
$ COPY FEES.DAT RECORDS
```

The COPY command can duplicate many files at a time. The following example copies all TXT files in the default directory to another directory:

```
$ COPY *.TXT;* [SAVETEXT]*.*;*
```

The COPY command can concatenate files. The following example appends FEES1.DAT to FEES.DAT (forming a new version of FEES.DAT) in your default directory:

```
$ COPY FEES.DAT,FEES1.DAT FEES.DAT
```

Use the COPY command to copy files from another node to your node. The following example copies the latest version of all files in DISK2:[PUBLIC] on node CHAOS to files with the same names in your default directory:

```
$ COPY CHAOS::DISK2:[PUBLIC]*.*  *
```

Use the COPY command to copy files from your node to another node. The following example copies the latest version of all files in your default directory to files with the same names in the directory DISK2:[STAFF_BACKUP] on node CHAOS:

```
$ COPY *.* CHAOS::DISK2:[STAFF_BACKUP]
```

If you receive a protection violation or DECnet-VAX error message when you attempt to copy a file across systems, you have two recourses:

- If you own the file, you can send it to a user account on the other node with the Mail Utility.
- You can follow the node name in the file specification with an access control string.

Use the /SINCE qualifier with the COPY command to select only those files that meet the specified criterion. The following example copies to the default directory only those files in the directory [JONES.LICENSES.DOG] that have been modified since December 31, 1988:

```
$ COPY/SINCE=31-DEC-1988/MODIFIED [JONES.LICENSES.DOG]*.* *
```

Use the RENAME command to give the file a new name and optionally locate it in a different directory. The following example gives the file FEES.DAT the new name RECORDS.DAT and moves it from the default directory to another directory:

```
$ RENAME FEES.DAT;4 [SAVETEXT]RECORDS.DAT
```

Note that after being renamed, the file FEES.DAT;4 no longer exists in the default directory. When you use the RENAME command, the input and output locations must be on the same device.

2.5.4 Deleting Files

The DELETE command removes files from directories and releases the disk space they occupy for use by other files. The DELETE command requires you to specify a version number or the asterisk wildcard character in each file specification. The following example deletes version 17 of POUND.LIS:

```
$ DELETE POUND.LIS;17
```

The following example deletes versions 16 and 17 of POUND.LIS:

```
$ DELETE POUND.LIS;16,;17
```

The following example deletes all versions of POUND.LIS:

```
$ DELETE POUND.LIS;*
```

When you delete many files with wildcard characters, you should confirm each deletion by specifying the /CONFIRM qualifier, as shown in the following example:

```
$ DELETE/CONFIRM *.*;*
DISK1:[JONES.LICENSES.DOG]FEES.DAT;4, delete? [N]:
DISK1:[JONES.LICENSES.DOG]FEMALE.LIS;6, delete? [N]:
DISK1:[JONES.LICENSES.DOG]MALE.LIS;3, delete? [N]:
DISK1:[JONES.LICENSES.DOG]POUND.LIS;17, delete? [N]:
```

Similarly, you may want to display the names of files as they are deleted. You can do this by specifying the /LOG qualifier with the DELETE command, as shown in the following example:

```
$ DELETE/LOG *.LIS;*
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]FEMALE.LIS;6 deleted (35 blocks)
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]MALE.LIS;3 deleted (5 blocks)
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]POUND.LIS;17 deleted (9 blocks)
```

The PURGE command deletes all but the latest version of the specified file (or all files) in the default directory or any other specified directory. Purging sequential files after updating them enables you to retain more free space on your disk volumes.

The following example deletes all but the latest two versions of each file in your default directory:

```
$ PURGE/KEEP=2
```

2.5.5 Printing Files

The PRINT command places your print job (all the files to be printed) in a list of jobs to be printed called a *print queue*. Print queues can be one of the following types of queues:

- Print queue—A queue assigned to a specific print device.
- Terminal queue—A print queue assigned to a hardcopy terminal that is being used solely as a printer (not interactively).
- Generic queue—A queue that distributes the processing of jobs to printers with similar characteristics. Jobs submitted to a generic queue are held in that queue until one of the assigned printer queues becomes available.

To print a file or files, use the PRINT command. The following example places a print job containing three files in the default print queue, SYS$PRINT.

```
$ PRINT POUND,MALE,FEES.DAT
Job POUND (queue SYS$PRINT, entry 202) started on SYS$PRINT
```

The file types of the files named in the PRINT command default to LIS or the last explicitly named file type; thus, the preceding example queues POUND.LIS, MALE.LIS, and FEES.DAT to SYS$PRINT. The system displays the job name (POUND), the queue name (SYS$PRINT), the job number (202), and indicates whether the job has started or is pending. By default, the job name is the name of the first (or only) file specification in the PRINT command. Once a job is submitted to a queue, you reference it using the job number. Once the job is queued, it will be printed when no other jobs precede it in the queue and when the printer is physically ready to print.

A print queue can execute only one job at a time. Print jobs are scheduled for printing according to their priority, and the job with the highest priority is printed first. If more than one job exists with the same priority, the smallest job is usually printed first. Jobs of equal size having the same priority are selected for printing according to their submission time.

The default print queue, SYS$PRINT, is usually initialized and started as part of the site-specific system startup procedures. The SHOW QUEUE command displays the queues that are initialized at your site. The SHOW ENTRY command displays the status of your print jobs, as shown in the following example:

```
$ SHOW ENTRY

   Jobname     Username   Entry   Blocks   Status
   -------     --------   -----   ------   ------
   POUND       JONES        202       38   Printing
On printer queue SYS$PRINT
```

Specify the USERNAME parameter to the SHOW ENTRY command to see jobs queued by other users. Use the ENTRY-NUMBER parameter to the DELETE/ENTRY command to delete your job from the queue, as shown in the following example:

```
$ DELETE/ENTRY=202
```

You can print a file on another system by copying that file to the remote node and specifying the /REMOTE qualifier to the PRINT command. The following example copies the file COMPANY_HOLIDAYS.TXT from your local node to the remote node CHAOS and queues the file for printing to the default system print queue (SYS$PRINT) on node CHAOS. The asterisk at the end of the file specification is a wildcard character. Here, it instructs the system to duplicate the file name COMPANY_HOLIDAYS.TXT when that file is copied to the remote node.

```
$ COPY COMPANY_HOLIDAYS.TXT CHAOS"JONES PANDEMONIUM"::DISK2:[JONES]*
$ PRINT/REMOTE CHAOS::DISK2:[JONES]COMPANY_HOLIDAYS.TXT
```

In the previous example, an access control string was specified to indicate that you are authorized to copy files to the directory [JONES] on node CHAOS. However, if you have a proxy account on that remote node, the access control string is unnecessary. (See Section 2.4 for more information about proxy accounts.)

Note that not all qualifiers to the PRINT command are compatible with the /REMOTE qualifier. For example, you cannot queue a job to a specific print queue; all jobs are queued to the default system print queue (SYS$PRINT). See the description of the /REMOTE qualifier to the DCL command PRINT in the *VMS DCL Dictionary* for a list of PRINT qualifiers compatible with /REMOTE.

DCL Commands That Control Print Jobs

The DCL commands listed in the following table allow you to control print jobs in various ways. For example, you can specify the number of copies printed or you can request that the system notify you when your print job is complete. For more information on any of these commands, see the *VMS DCL Dictionary*.

Print Operations	Print Job Commands and Qualifiers
Number of copies	
By job	PRINT/JOB_COUNT=n[1]
By file	PRINT/COPIES=n[1]
Specified file only	file-spec/COPIES=n[1]
Number of pages	PRINT/PAGES=[1]
Print features	
Flag pages	PRINT/FLAG=[1]
Type of forms (paper)	PRINT/FORM=[1]
Special features	PRINT/CHARACTERISTICS=[1]
Double-spacing	PRINT/SPACE[1]
Page heading	PRINT/HEADER[1]
Notification of job execution	PRINT/NOTIFY
Delay execution of a job	
For a specified time	PRINT/AFTER
Indefinitely	PRINT/HOLD
Release a delayed job	SET QUEUE/ENTRY/RELEASE
Display your print jobs	SHOW ENTRY

[1]Parallel qualifiers for the SET QUEUE/ENTRY command allow you to specify these operations for print jobs that are already queued but not yet printing.

Print Operations	Print Job Commands and Qualifiers
Stop a print job	
Delete job	DELETE/ENTRY=job-number
Stop currently printing job and begin printing the next job in the queue	STOP/ABORT
Stop currently printing job and requeue it for printing	STOP/REQUEUE

2.6 Device and Directory Operations

To access files on your system, you need to know how to navigate through many directory structures. Because directories reside on devices, you also need to know how to work with devices other than your default device.

Device and directory operations include the following:

- Displaying directories
- Creating directories
- Deleting directories
- Setting a default device
- Setting a default directory
- Searching the directory structure with wildcards

2.6.1 Displaying Directories

The DCL command DIRECTORY displays the names of the files in a directory. The following example lists the files in [JONES]:

```
$ DIRECTORY/COLUMNS=1

Directory DISK1:[JONES]

LICENSES.DIR;1
LOGIN.COM;3
LOGIN.COM;4
STAFF.DIS;3
STAFF_VACATIONS.TXT;2
TAXES.DIR;1

Total of 5 files.
```

The display shows us that [JONES] contains two subdirectories—[JONES.LICENSES] and [JONES.TAXES]—and four nondirectory files STAFF.DIS, STAFF_VACATIONS.TXT, and two versions of LOGIN.COM. The following example (assuming that the default directory remains [JONES]) lists the contents of the subdirectory [JONES.LICENSES]. Note that if you want to move one level down the directory structure, you need specify only the subdirectory name, preceded by a period, to which you want to move.

```
$ DIRECTORY/COLUMNS=1 [.LICENSES]

Directory DISK1:[JONES.LICENSES]

MAILING.LIS;6
TOTAL.DAT;2
DEPT.DAT;3
DOG.DIR;1
MARRIAGE.DIR;1

Total of 6 files.
```

If you have sufficient privileges, you can display the contents of the master file directory. To do so, specify [000000] as the *file-spec* parameter to the DIRECTORY command or search up one level from a top level directory using the [-] wildcard (described in Section 2.6.6.2). Note that nine of the files contained in [000000] are structure files, which are special files created and reserved by the system that must not be deleted. Note also that your system disk contains several directories with files that provide data required to run VMS and allow you to run command images and execute command procedures.

See the *Guide to Setting Up a VMS System* for more information about user privileges.

2.6.2 Creating Directories

The CREATE/DIRECTORY command creates a subdirectory, as shown in the following example.

```
$ CREATE/DIRECTORY [JONES.LICENSES]
```

If your current default directory is [JONES], specify the following:

```
$ CREATE/DIRECTORY [.LICENSES]
```

Note that you must have SYSPRV privilege to create a top level directory. See the *Guide to Setting Up a VMS System* for a discussion of user privileges.

2.6.3 Deleting Directories

You cannot delete a directory that contains files. Before deleting a directory or subdirectory, make sure it is empty by entering the DIRECTORY command, as shown in the following example:

```
$ SET DEFAULT [JONES.LICENSES]
$ DIRECTORY
  No files found.
```

If the directory contains any files, copy or rename them to another directory (if you want to save them) and delete them from the directory of interest. If the directory contains subdirectories, examine those subdirectories (copying and deleting their files) and delete the subdirectories.

To delete a directory, move to the directory one level above the directory you want to delete. This means that if you want to delete [JONES.LICENSES], you should set default to [JONES]. Remember that the subdirectory [JONES.LICENSES] exists as a file named LICENSES.DIR;1 in the directory [JONES]. You delete a directory by deleting the file that points to that directory.

Because a directory file is created without delete access to prevent accidental deletion of the directory, you must change the file protection to allow delete access before you can delete that directory file. (See Section 7.2 for more information about file protection.) The following example shows how to delete the subdirectory [JONES.LICENSES]:

```
$ SET DEFAULT [JONES]
$ SET PROTECTION=OWNER:D LICENSES.DIR
$ DELETE LICENSES.DIR;1
```

The directory files (for example, JONES.DIR;1) in the master file directory require SYSPRV privilege to delete. See the *Guide to Setting Up a VMS System* for a discussion of user privileges.

2.6.4 Setting a Default Directory

Ascend and descend the directory hierarchy by setting default to a different directory with the DCL command SET DEFAULT. The default remains in effect until you enter another SET DEFAULT command.

The following example sets default to the directory [JONES] and displays the file STAFF_VACATIONS.TXT:

```
$ SET DEFAULT [JONES]
$ TYPE STAFF_VACATIONS.TXT
```

Subdirectories are specified by concatenating the subdirectory name to the name of the directory one level above it. The following example displays the file BILLING.DAT located in the subdirectory [JONES.TAXES]:

```
$ SET DEFAULT [JONES.TAXES]
$ TYPE BILLING.DAT
```

When you move from your current default directory to a subdirectory one level below, you can omit the current directory's name in the file specification. By default, the system assumes the current directory. In the following example, the current default directory is [JONES]:

```
$ SET DEFAULT [.TAXES]
$ TYPE BILLING.DAT
```

You can display the current default directory by entering the command SHOW DEFAULT, as shown in the following example:

```
$ SHOW DEFAULT
  DISK1:[JONES.TAXES]
$ SET DEFAULT [PUBLIC]
$ SHOW DEFAULT
  DISK1:[PUBLIC]
```

2.6.5 Setting a Default Device

Section 2.6.4 describes how to set a default directory with the DCL command SET DEFAULT. You can also use the SET DEFAULT command to change the default device. The default remains in effect until you enter another SET DEFAULT command or log out.

The following example shows how to change the default device:

```
$ SHOW DEFAULT
  DISK1:[JONES]
$ SET DEFAULT DISK2:[GROUP]
$ SHOW DEFAULT
  DISK2:[GROUP]
```

You can specify the device to which you want to set default without including the directory in the command. In the following example, the directory [JONES] is assumed and exists on DISK1 and DISK2:

```
$ SHOW DEFAULT
  DISK1:[JONES]
$ SET DEFAULT DISK2:
$ SHOW DEFAULT
  DISK2:[JONES]
```

Note that VMS allows you to set default to a nonexistent disk or directory. If you find yourself in a nonexistent disk or directory and cannot carry out a desired operation, simply set default to an existing disk or directory and continue your task.

2.6.6 Searching the Directory Structure with Search Wildcards

From any point in a directory structure, you can refer to another directory or subdirectory in the structure. Do this by specifically naming the directory or subdirectory you want or by using the ellipsis (...) and hyphen (-) wildcard characters.

2.6.6.1 The Ellipsis (...) Wildcard Character

Use the ellipsis to search down into the directory hierarchy. To search the current directory and all the subdirectories below it, use the ellipsis by itself. The following command searches the current default directory and all subdirectories below it:

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

Assuming the current directory is [JONES], the following command displays the latest versions of all files named FEES.DAT in [JONES] and all subdirectories under [JONES]:

```
$ TYPE [JONES...]FEES.DAT
```

If you begin the directory specification with an ellipsis, the search begins from your current directory. Assuming the current default directory is [JONES], the following command searches all subdirectories that end in .SALES and displays the latest versions of the file FEDERAL.LIS:

```
$ TYPE [...SALES]FEDERAL.LIS
```

Assuming the current directory is [JONES], the following command displays the latest versions of all files named DEPT.DAT in [JONES] and all subdirectories under [JONES]:

```
$ TYPE [...]DEPT.DAT
```

However, if you begin the directory specification with a period, only the subdirectory that is one level lower than the current directory is searched. Assuming the current directory is [JONES], the following command searches only the [.LETTERS] subdirectory that is one level lower than [JONES] for the file INVITATION.TXT. The subdirectory [JONES.LETTERS] is searched, but [JONES.WORK.LETTERS] is not:

```
$ TYPE [.LETTERS]INVITATION.TXT
```

Assuming the current directory is [JONES], the following command displays the latest versions of all files named DEPT.DAT in the [.LICENSES] subdirectory under [JONES] and all subdirectories under the [.LICENSES] subdirectory:

```
$ TYPE [...LICENSES...]DEPT.DAT
```

To search all top level directories and their subdirectories from wherever you are in the directory structure, use an asterisk (*) followed by an ellipsis (...). The following command (which requires READALL privilege) searches as many as eight levels of directory names (the top level directory and seven subdirectories), if they exist. It does not search the MFD.

```
$ DIRECTORY [*...]
```

2.6.6.2 The Hyphen (–) Wildcard Character

The hyphen wildcard character permits you to move up through the directory structure. Each hyphen refers to the directory one level up from the current one. You can follow the hyphens with directory and subdirectory names to move down the directory structure on another path.

If the current directory is [JONES.LICENSES], the following command displays the latest version of STAFF.DIS in [JONES]:

```
$ TYPE [-]STAFF.DIS
```

If your current directory is [JONES.LICENSES], the following command displays the latest version of BILLING.DAT in [JONES.TAXES]:

```
$ TYPE [-.TAXES]BILLING.DAT
```

You can specify more than one hyphen. The following command moves you up two levels in the directory hierarchy.

```
$ SET DEFAULT [--]
```

If you enter so many hyphens that you point above the master file directory (MFD), the system displays an error message.

The following example uses the BACKUP command to copy all files in [JONES.TAXES] to a directory named [AUDIT], and all the files in any subdirectories under [JONES.TAXES] to corresponding subdirectories under [AUDIT]:

```
$ BACKUP [JONES.TAXES...]*.*.* [AUDIT...]*.*.*
```

The trailing ellipsis in the output specification lets you move the entire third level directory structure from the input directory to the second level of the output directory. Unlike the COPY command, the BACKUP command preserves the input directory structure in the output it creates.

2.6.6.3 Using Wildcards to Copy a Directory Structure

By including asterisk (*) and ellipsis (...) wildcards in output directory specifications, you can do the following:

- Duplicate an entire input directory structure
- Move files from one directory structure into another directory structure at the same or at a different level

Each wildcard character in an output directory specification refers to a corresponding directory level in the input specification. An output specification may contain only wildcards, or it may contain a combination of wildcards and directory names. If directory names are used, they must always precede any wildcards that are included.

Use the asterisk when you want a particular level in the output directory specification to match a level indicated by a wildcard in the input specification. For example:

```
$ BACKUP [JONES.*]*.*;* [SCHULTZ.SAVE.*]*.*;*
```

In the previous example, the BACKUP command copies all files from any subdirectories under [JONES] to any corresponding subdirectories under [SCHULTZ.SAVE]. For example, all files in the subdirectory [JONES.TAXES] are copied to the subdirectory [SCHULTZ.SAVE.TAXES]. Notice that the single asterisk in the output directory specification refers to the first subdirectory level in the input directory that contained a wildcard.

Use the ellipsis when you want the output directory specification to follow the same structure downwards as the input directory from the first level that contained a wildcard. For example:

```
$ BACKUP [JONES.TAXES...]*.*;* [AUDIT...]*.*;*
```

In the previous example, the BACKUP command copies all files in [JONES.TAXES] to a directory named [AUDIT], and all the files in all subdirectories under [JONES.TAXES] to corresponding subdirectories under [AUDIT]. The trailing ellipsis in the output specification lets you move the entire third level directory structure from the input directory to the second level of the output directory.

For output directory specifications, a trailing asterisk and ellipsis are mutually exclusive when they follow a specific directory name. Therefore, output directory specifications such as [USER.*...] and [USER...*] are invalid. However, [*...] is valid, because the asterisk wildcard is used in place of a directory name.

You can move an entire input directory structure to an output directory structure. The two ways to do this are as follows:

```
$ BACKUP DISK1:[JONES...]*.*;* DISK2[*]*.*;*
```

or

```
$ BACKUP DISK1:[JONES...]*.*;* DISK2[*...]*.*;*
```

These commands let you move all the files in the [JONES] directory structure on the disk DISK1 to the [JONES] directory structure on disk DISK2, from the top level directory down through the entire structure.

3 Working with Processes

The environment in which you interact with the system is called a *process*. A process contains identification and status information that the system needs to execute programs for you. Within a process, programs execute one at a time in the order in which they are invoked.

You can place your process into hibernation and create a second process called a *subprocess* under your user name. You can interact with the system and log out of that subprocess to return to the original process.

A program executes within the context of the process that invokes it. Some programs are system programs that control the flow of events within the process. For example, when you log in, your process is under the control of the system program SYS$SYSTEM:LOGINOUT.EXE. When you work at DCL level, your process is under the control of SYS$SYSTEM:DCL.EXE.

A command procedure is a file that contains a list of DCL commands. Complex command procedures resemble programs written in high-level programming languages. In this sense, command procedures provide a way to write programs in DCL.

You can submit programs and command procedures for execution as batch jobs, which you submit to the system as separate processes. Batch jobs allow you to continue to work interactively with the system while the program or procedure executes as another process under your user name.

3.1 Processes and the User Environment

Each user on the system is associated with a process, which is a special environment created by the system that makes interaction with the system possible. A process has a beginning and an end; for example, the system creates a process for you when you log in and deletes that process when you log out. A process contains all the information that the system needs to execute programs. It is within your process that the system executes your programs (also called images or executable images) one at a time.

A process can be a *detached process* (a process that is independent of other processes) or a *subprocess* (a process that is dependent on another process for its existence and resources). Your main process, also called your parent process, is a detached process.

The system creates a process for you when you do one of the following:

- Log in—The system creates a process for each interactive user.
- Submit a batch job—The system creates a process for each batch job. When the batch job is completed, the system deletes the process. Section 3.1.4 discusses batch jobs.
- Spawn a subprocess—The system creates a process when you use the SPAWN command. Section 3.1.3 describes subprocesses.
- Run a program using either the /DETACHED or the UIC=uic qualifiers. Section 3.1.1 describes programs.

The system also creates special system processes to perform various functions. The DCL command SHOW SYSTEM displays both user and system processes.

The following list summarizes the *process context*. Certain characteristics, such as the privileges, symbols, and logical names enabled in your process, collectively create the process context. Use the DCL command SHOW PROCESS/ALL to examine your process context.

```
31-DEC-1988 13:30:37.12 ❶                           User: CLEAVER ❷
Pid: 24E003DC ❸     Proc. name: CLEAVER_1 ❹     UIC: [DOC,CLEAVER] ❺
Priority:   4 ❻     Default file spec: DISK1:[CLEAVER] ❼

Process Quotas: ❽
 Account name: DOC
 CPU limit:                      Infinite  Direct I/O limit:        18
 Buffered I/O byte count quota:     31808  Buffered I/O limit:      25
 Timer queue entry quota:              10  Open file quota:         57
 Paging file quota:                 22276  Subprocess quota:         4
 Default page fault cluster:           64  AST quota:               38
 Enqueue quota:                       600  Shared file limit:        0
 Max detached processes:                0  Max active jobs:          0

Accounting information: ❾
 Buffered I/O count:       140  Peak working set size:       383
 Direct I/O count:           7  Peak virtual size:          2336
 Page faults:              304  Mounted volumes:               0
 Images activated:           1
 Elapsed CPU time:      0 00:00:00.55
 Connect time:          0 00:00:22.76

Process privileges: ❿
 GROUP                  may affect other processes in same group
 TMPMBX                 may create temporary mailbox
 OPER                   operator privilege
 NETMBX                 may create network device

Process rights identifiers: ⓫
 INTERACTIVE
 LOCAL
 SYS$NODE_AJAX

Process Dynamic Memory Area ⓬
    Current Size (bytes)         25600    Current Total Size (pages)      50
    Free Space (bytes)           19592    Space in Use (bytes)          6008
    Size of Largest Block        19520    Size of Smallest Block          24
    Number of Free Blocks            3    Free Blocks LEQU 32 Bytes        1

Processes in this tree: ⓭
CLEAVER
  CLEAVER_1 (*)
```

❶ Current date and time—The date and time when the SHOW PROCESS/ALL command is executed.

❷ User name—The user name assigned to the account that is associated with the process.

❸ Process identification number (PID)—A unique number assigned to the process by the system. The SHOW PROCESS command displays the PID as a hexadecimal number.

❹ Process name—The name assigned to the process. Since process names are unique, the first process logged in under an account is assigned the user name, and subsequent processes logged in under the same account are assigned the terminal name. You can change your process name with the DCL command SET PROCESS/NAME.

❺ User identification code (UIC)—The group and member numbers (or letters) assigned to the account that is associated with the process (for example, [PERSONNEL,RODGERS]). Part of your UIC identifies the group to which you belong. Within a group, users are allowed to share files or system resources more freely than between groups.

❻ Priority—The current priority of the process.

❼ Default file specification—The current device and directory. Change your current defaults with the DCL command SET DEFAULT.

❽ Process quotas—The quotas (limits) associated with the process. Examine these quotas with the /QUOTAS or /ALL qualifiers of the SHOW PROCESS command.

❾ Accounting information—The continuously updated account of the process's use of memory and CPU time. Examine this information with the /ACCOUNTING or /ALL qualifiers of the SHOW PROCESS command.

❿ Process privileges—The privileges granted to your processes. Privileges restrict the performance of certain system activities to certain users. Examine your privileges with the /PRIVILEGES or /ALL qualifiers of the SHOW PROCESS command.

⓫ Process rights identifiers—System-defined identifiers that are used in conjunction with access control list protection. Identifiers provide the means of specifying the users in an access control list. An access control list is a security tool that defines the kinds of access to be granted or denied to users of an object, such as a file, device, or mailbox. (See Chapter 7 for more information about identifiers and access control lists.)

⓬ Process dynamic memory area—The process's current use of dynamic memory. Dynamic memory is allocated by the system to an image when that image is executing. When that memory is no longer needed by one process, the system allocates it to another process. Examine this information with the /MEMORY or /ALL qualifiers of the SHOW PROCESS command.

⓭ Processes in this tree—A list of subprocesses belonging to the parent process. An asterisk appears after the current process. Examine this with the DCL SHOW PROCESS/SUBPROCESSES or /ALL command.

3.1.1 Programs

A program, also called an *image* or *executable image,* is a file that contains instructions and data in machine-readable format. Image files can be VMS- or user-supplied and usually have a file type of EXE. You cannot examine an image file with the DCL commands TYPE, PRINT, or EDIT because image files do not consist of ASCII characters. (Text files contain ASCII characters, which are a standard method of representing the alphabet, punctuation marks, numerals, and other special symbols.)

A program can be either a command image or a noncommand image as follows:

- Command image—A command image is a program associated with and invoked by a DCL command. For example, when you type the DCL command COPY, the system executes the program SYS$SYSTEM:COPY.EXE. COPY.EXE is a command image. A system directory named SYS$SYSTEM contains a number of command image files, most of which are VMS-supplied. Use the DCL command DIRECTORY SYS$SYSTEM to examine this system directory.
- Noncommand image—A noncommand image is a program not associated with a DCL command. To invoke a noncommand image, name the file containing the program as the parameter to the RUN command.

Executing Programs Across the Network

Because of support provided by DECnet–VAX, programs can execute across the network as if they were executing locally. Because DECnet–VAX is integrated within the VMS operating system, it is easy to write programs that access remote files. To access a remote file in an application program, you need only include in your file specification the name of the remote node and any required access control information.

Task-to-task communications, a feature common to all DECnet implementations, allows two application programs running on the same or different operating systems to communicate with each other regardless of the programming languages used. Examples of network applications are distributed processing applications, transaction processing applications, and applications providing connection to servers.

3.1.2 Command Procedures

A command procedure is a file that contains a list of DCL commands. When you execute a command procedure, DCL reads the command file and executes the commands it contains. Command procedures can be executed as interactive or batch processes. If you use command procedures that require lengthy processing time (for example, the compilation or assembly of large programs) submit these procedures as batch jobs so you can continue to use your terminal interactively.

When you submit a command procedure for batch execution, the system creates a detached process using your account and process characteristics. The system runs the job from that process and deletes the process when the job is completed.

You can use command procedures to automate sequences of commands that you enter frequently. For example, if you always examine the contents of a directory immediately after setting default to it, you can design a command procedure that issues the appropriate commands to display the directory's contents. A command procedure might contain the following commands to set default to the ACCOUNT subdirectory and display the subdirectory's contents. (Exclamation points delimit comments in command procedures; DCL ignores everything to the right of the exclamation point when processing the line.)

```
$ ! DISK1:[ADAMS]ACCOUNTD.COM
$ !
$ SET DEFAULT DISK1:[ADAMS.ACCOUNT]
$ DIRECTORY
```

To execute a command procedure interactively, type the @ command followed by the procedure's file specification. To execute the command procedure in the previous example, enter @DISK1:[ADAMS]ACCOUNTD (or @ACCOUNTD if your current disk and directory are DISK1:[ADAMS]).

Chapter 6 discusses command procedures in greater detail.

3.1.3 Subprocesses

The SPAWN command enables you to create a subprocess of your current process. Within this subprocess, you can interact with the system and log out of the subprocess to return to your parent process, or switch between your parent process and subprocesses. Only one of your processes is executing at any time.

Each user on the system is represented by a *job tree.* A job tree is a hierarchy of all your processes and subprocesses, with your main process at the top. A subprocess is dependent on the parent process and is deleted when the parent process exits. By default, the subprocess assumes the name of the parent process followed by an underscore and a unique number. For example, if the parent process name is DOUGLASS, the subprocesses are named DOUGLASS_1, DOUGLASS_2, and so on, forming a tree of subprocesses.

Typically, you use a subprocess in one of the following two ways:

- To interrupt a task, perform a second task, then return to the original task—Because SPAWN is a built-in command (listed in Chapter 1), you can use CTRL/Y to interrupt one task, spawn a subprocess to perform a second task, exit from the subprocess, and then enter the CONTINUE command to return to the original task. By default, when you create a subprocess, the parent process hibernates, and you are given control at DCL level within the subprocess. Your default directory is the current directory of the parent process. (If you interrupt the EDT editor, enter the CONTINUE command and press CTRL/W to refresh the screen.)
- To perform a second task while continuing to work on your original task—You can do so by creating the subprocess with the SPAWN/NOWAIT command. Use the SPAWN/NOWAIT command only to execute commands that do not require input; SPAWN/NOWAIT generates a noninteractive, batch-like subprocess.

  Because both the parent and the subprocess are executing concurrently, both attempt to control the terminal. To prevent conflicts, also specify the following:

  - /OUTPUT qualifier—Indicates that the subprocess should write output to a specified file rather than to the terminal.
  - SPAWN command parameter or /INPUT qualifier—Indicates that the subprocess should execute the specified commands rather than reading input from the terminal.

When you specify the /INPUT qualifier of the SPAWN command, the subprocess is created as a noninteractive process that exits upon encountering a severe error or an end-of-file indicator. At DCL level, CTRL/Z is treated as an end-of-file indicator.

In the following example, a command image (the TYPE command) is interrupted with CTRL/Y and a subprocess is spawned:

```
$ TYPE MICE.TXT
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
   .
   .
   .
CTRL/Y
$ SPAWN
%DCL-S-SPAWNED, process DOUGLASS_1 spawned
%DCL-S-ATTACHED, terminal now attached to process DOUGLASS_1
$ MAIL
MAIL>
   .
   .
   .
MAIL> EXIT
$ LOGOUT
Process DOUGLASS_1 logged out at 31-DEC-1988 12:42:12.46
  %DCL-S-RETURNED, control returned to process DOUGLASS
$ CONTINUE
Once inside, they may gnaw through electrical wires and raid
your food. Because mice reproduce so quickly, what started
as one or two mice can quickly become an invasion.  If you seal
the cracks and holes on the exterior of your foundation, you can
prevent these rodents from ever getting in.
```

Because each process you create is unique, commands executed in one process do not usually affect any other process. However, because control of the terminal passes between processes, commands that affect the terminal characteristics (for example, SET TERMINAL) affect any process controlling that terminal. For example, if one process inhibits echoing and exits without restoring it, echoing remains inhibited for the next process that gains control of the terminal. Reset any altered terminal characteristics with the SET TERMINAL command.

3.1.3.1 Exiting from a Subprocess

To exit from a subprocess created by SPAWN, use one of the following commands:

- LOGOUT—When you exit from a subprocess with the LOGOUT command, the subprocess is deleted (along with any subprocesses that it created), and you are returned to the parent process.
- ATTACH—When you exit from a subprocess with the ATTACH command, the subprocess hibernates, and control of your terminal is transferred to the specified process. (You must specify either a process name as a parameter to the ATTACH command or a process identification number (PID) as a value of the /IDENTIFIER qualifier of the ATTACH command.) The following example shows how to exit from the subprocess DOUGLASS_1 and attach to the process DOUGLASS:

```
$ ATTACH DOUGLASS
%DCL-S-RETURNED, control returned to process DOUGLASS
$ SHOW PROCESS
26-APR-1988 10:34:58.50   VTA303             User: DOUGLASS
Pid: 25C002B4   Proc. name: DOUGLASS         UIC: [200,200]
Priority:   4   Default file spec: SYS$SYSDEVICE:[DOUGLASS]
Devices allocated: $11$VTA303:
```

3.1.3.2 Subprocess Context

By default, a subprocess inherits the following items from the parent process: defaults, privileges, symbols, logical names, control characters, message format, verification state, and key definitions. The environment that these items collectively create is called the *process context*. The following items, however, are not inherited from the parent process:

- Process identification number (PID)—The system assigns each created subprocess a unique process identification number.
- Process name—By default, the subprocess name consists of the name of the parent process followed by an underscore and an integer. Use the /PROCESS qualifier of the SPAWN command to specify a process name other than the default. A process name must be unique.
- Created commands—Commands that are defined by a parent process using the SET COMMAND command are not copied to a subprocess. To use a created command in a subprocess, you must use SET COMMAND to create that command for the subprocess.
- Authorize privileges—When you spawn to a subprocess, the process context contains the privileges currently enabled, not the privileges that you may be authorized to enable. For example, if you spawn to a subprocess while in MAIL and want to perform a privileged operation, you need to have already set the proper privilege in the parent process.

You can use the following SPAWN qualifiers to prevent the subprocess from inheriting a number of these items:

Qualifier	Items Inhibited or Changed
/CARRIAGE_CONTROL, /PROMPT	DCL prompt
/NOCLI	CLI (command language interpreter; DCL by default)
/NOKEYPAD	Keypad definitions
/NOLOGICAL_NAMES	Logical names
/NOSYMBOL	Symbols

The /SYMBOL and /LOGICAL_NAMES qualifiers do not affect system-defined symbols (such as $SEVERITY and $STATUS) or system-defined logical names (such as SYS$COMMAND and SYS$OUTPUT). Symbols are described in Chapter 5. See Chapter 4 for more information about logical names.

Since copying logical names and symbols to a subprocess can be time-consuming (a few seconds), you may want to use the /NOLOGICAL_NAMES and /NOSYMBOL qualifiers to the SPAWN command unless you plan to use the logical names or symbols in the subprocess. If you use subprocesses frequently, the ATTACH command provides the most efficient way to enter and exit a subprocess. This method allows you to transfer control quickly between the parent process and subprocess rather than repeatedly waiting for the system to create a new subprocess for you.

3.1.4 Batch Jobs

Usually you use VMS in interactive mode. When you work interactively with VMS, the system must expend resources waiting for your input. If the system is heavily loaded with other jobs, your interaction with VMS is slowed. Also, when you run a long job interactively, your terminal is unavailable to you until the job is completed.

A batch job consists of one or more programs or command procedures that execute as a detached process without user interaction. A batch job allows you to use your terminal for other work while the system executes your program or command procedure.

When you submit a batch job, the system logs in under your user name and creates a detached process dedicated to executing the commands in that file. A batch job is also more efficient than an interactive session: it runs under a lower priority and can be run at times when the system is not overloaded with interactive users.

3.1.5 Submitting a Batch Job

To run a job in batch mode, submit your job to a batch queue (a list of batch jobs waiting to execute) by entering the DCL command SUBMIT. When you submit a job, it is directed to the default batch queue, SYS$BATCH, where it is added to the end of the queue of jobs waiting to be executed. When the jobs preceding yours are completed, your job is executed. (On a VMS system, the number of batch jobs that can execute simultaneously is specified when the batch queue is created by the system manager.)

By default, the SUBMIT command uses a file type of COM. The following command enters JOB1.COM into SYS$BATCH:

```
$ SUBMIT JOB1
Job JOB1 (queue SYS$BATCH, entry 651, started on SYS$BATCH)
```

The system displays the name of the job, the queue containing the job, and the entry number assigned to the job. You receive the DCL prompt once your job is submitted to the batch queue. If you need to reference your batch job in any DCL commands (DELETE/ENTRY, for example), do so by using the job entry number. (You can obtain the job entry number by using the SHOW ENTRY command.) Note that if multiple procedures are submitted in a batch job, the batch job terminates when any procedure exits with an error or fatal error status.

Your batch job does not necessarily have to start running at the time you submit it to the batch queue. To specify a different time, enter the SUBMIT/AFTER command. In the following example, the job is submitted after 11:30 p.m.:

```
$ SUBMIT/AFTER=23:30 JOB1.COM
```

3.1.6 Batch Job Output

By default, accumulated output from a batch job is written to a log file once each minute. (To specify a different time interval, include the SET OUTPUT_RATE command in your command procedure.) If you attempt to use the EDT editor to read the log file while the system is writing to it, you receive a message indicating that the file is locked by another user. Wait a few seconds and try again. The EVE editor, however, allows you to read the batch job's log file. By specifying EDIT/TPU/READ_ONLY and the name of the log file, you can use EVE commands to move around the log file and ensure that any changes you make to the file are not saved. If you omit the /READ_ONLY qualifier and modify the log file in any way, the batch job terminates.

Because your batch job is a process that logs in under your user name and executes your login command procedure, the output from a batch job includes the contents of your login command procedure. The output also includes everything written to the batch job log file (command procedure output, error messages, and so on) and the full logout message. To prevent your login command procedure from being written to the batch log file, add the following command to the beginning of your login command procedure:

```
$ IF F$MODE() .EQS. "BATCH" THEN SET NOVERIFY
```

By default, the log file name is the name under which you submitted the job. Also by default, the log file has a file type of LOG and assumes the device and directory specified by your login defaults. To specify a different log file name when you submit the job, use the /LOG_NAME qualifier to the SUBMIT command.

When the batch job completes, the log file is queued to the default system printer (SYS$PRINT), printed, and deleted. To save the log file after printing it, use the /KEEP qualifier to the SUBMIT command. To save the log file without printing it, use the /NOPRINT qualifier to the SUBMIT command.

3.1.7 Restarting Batch Jobs

If the system fails while your batch job is executing, your job does not complete. When the system recovers and the queue is restarted, your job is aborted, and the next job in the queue is executed. However, by specifying the /RESTART qualifier when you submit a batch job, you indicate that the system should reexecute your job if the system crashes before the job is finished.

By default, a batch job is reexecuted beginning with the first line. See Section 6.10 for more information about symbols you can add to your command procedures to specify a different restarting point.

4 Using Logical Names

When you define a logical name, you equate one character string to an *equivalence name,* which is usually a full or partial file specification, another logical name, or any other character string. Once you have equated a logical name to one or more equivalence names, you can use the logical name to refer to those equivalence names. For example, you might assign a logical name to your default disk and directory. Logical names serve two main functions:

- Shorthand and readability—You can define commonly used files, directories, and devices with short, meaningful logical names. Such names are easier to remember and type than the full file specifications. Names that you use frequently can be defined in your login command procedure. Names that most users on your system use frequently can be defined by a system manager in the site-specific system startup command procedure.
- File independence—You can use logical names to keep your programs and command procedures independent of physical file specifications. For example, if a command procedure references the logical name ACCOUNTS, you can equate ACCOUNTS to any file on any disk before executing the command procedure.

Logical names can be defined by you or by the system. Logical names and their definitions are kept in tables called *logical name tables.* The system provides the following logical name tables:

- Your process table
- The job table for your process
- Your group table
- The system table

When you enter a logical name as part of a command line, the system translates the logical name. It does this by searching the logical name tables in a certain order. Information about existing logical name tables and the order in which they are searched is stored in two *logical name directory tables.*

With DCL, you can also apply special attributes to logical names and define the order in which logical names tables are searched.

4.1 Creating Logical Names

You can create your own logical names with either the ASSIGN or the DEFINE command. This section uses the DEFINE command to create logical names. (Note that the syntax for the ASSIGN command differs from the syntax for the DEFINE command. For information on using the ASSIGN command, see the *VMS DCL Dictionary*.)

The syntax for defining a logical name is as follows:

```
DEFINE logical-name equivalence-name[,...]
```

The following example associates the logical name ACCOUNTS with the equivalence name DISK1:[JONES.ACCOUNTS]:

```
$ DEFINE ACCOUNTS DISK1:[JONES.ACCOUNTS]
```

Now you can use ACCOUNTS to refer to the directory DISK1:[JONES.ACCOUNTS].

Observe the following rules when creating a logical name with the DEFINE command:

- A logical name and its equivalence name can each have a maximum of 255 characters. A logical name can contain alphanumeric characters, as well as the underscore (_), dollar sign ($), and hyphen (-).
- The equivalence name must include the punctuation marks (colons, brackets, periods) that would be required if it were part of a file specification. For example, a device name is terminated by a colon, a directory specification is enclosed in square brackets, and a file type is preceded by a period.
- You can optionally terminate a logical name with a colon. If you do this, the ASSIGN command removes the colon before placing the logical name in a logical name table. The DEFINE command does not remove the colon before placing the name in a logical name table.

  In general, you should not specify a colon at the end of a logical name when you are creating it. However, if you do so and want to save the colon as part of the logical name, use the DEFINE command. (Note that when you delete a logical name ending with a colon, you need to specify two colons because the DEASSIGN command, like the ASSIGN command, removes one colon before it searches the logical name table for a match.)

If the logical name is part of a file specification, the logical name must be the leftmost component of the file specification and must be separated from the rest of the file specification by a colon. When you use a logical name to represent a complete file specification, the terminating colon is not needed. The following examples all display the file DISK1:[SALES_STAFF]PAYROLL.DAT:

```
$ DEFINE PAY DISK1:[SALES_STAFF]PAYROLL.DAT
$ TYPE PAY

$ DEFINE PAY_FILE DISK1:[SALES_STAFF]PAYROLL
$ TYPE PAY_FILE:.DAT

$ DEFINE PAY_DIR DISK1:[SALES_STAFF]
$ TYPE PAY_DIR:PAYROLL.DAT

$ DEFINE PAY_DISK DISK1:
$ TYPE PAY_DISK:[SALES_STAFF]PAYROLL.DAT
```

Note that if you combine a logical name with only an explicitly stated file type or version number, you must include the period or semicolon, respectively. For example, if PAY is equivalent to DUA1:[SALES_STAFF]PAYROLL.DAT, PAY:2 is an invalid file specification. (PAY:;2 is valid.) Defaults for the current directory, the file type (depending on the function being performed), and the version number are applied as usual after translation.

By default, the DEFINE command places logical names in your process logical name table, where the logical name is available only to your process and subprocesses. Section 4.2 describes logical name tables.

You can equate more than one logical name with an equivalence name. For example, you can equate the logical names $TERMINAL and CONSOLE to the physical name of a terminal so that both logical names translate to the same device. (If you equate a logical name to more than one equivalence string in a single command, you create a search list for the system to use to translate the names. See Section 4.7 for information about search list translation.)

If you equate a logical name to one equivalence string and then equate the same logical name to another equivalence string, the second definition supersedes the first. You can, however, equate the same logical name to different equivalence strings if the logical name definitions are in different tables (described in Section 4.2). You can equate the same logical name to different equivalence strings in the same table if they are defined in different access modes (described in Section 4.5).

If you cannot access a file, and the command you are specifying and the file specification seem in order, check the left-hand component of the file specification (with SHOW LOGICAL) to be sure that it is not defined as a logical name.

4.1.1 Displaying Logical Names

Display the definition of a logical name with either the SHOW LOGICAL or SHOW TRANSLATION command.

When you enter the SHOW LOGICAL command, the system searches the process, job, group, and system logical name tables (in that order) for the specified logical name. In the following example, the system found the logical name ACCOUNTS in both the process and job logical name tables:

```
$ SHOW LOGICAL ACCOUNTS
  "ACCOUNTS" = "DISK1:[ACCOUNTS]" (LNM$PROCESS_TABLE)
  "ACCOUNTS" = "DISK1:[ACCOUNTS]" (LNM$JOB_80891AE0)
```

Sometimes the definition of a logical name may include another logical name. The SHOW LOGICAL command continues to search the logical name tables until all levels of logical names in a definition have been found. This is referred to as *iterative translation.*

When iterative translation is performed, the SHOW LOGICAL command displays multiple lines. Each line has a number that shows the level of translation. For example:

```
$ SHOW LOGICAL MYDISK
  "MYDISK" = "DISK2" (LNM$PROCESS_TABLE)
1 "DISK2" = "$11$DUA4:" (LNM$SYSTEM_TABLE)
```

Level numbers are zero based; that is, 0 is the first level, 1 is the second, and so on. In the previous example, two translations were performed. The number 1 indicates the second level of translation. See Section 4.4.1 for more information about iterative translation.

Unless you have redefined the search order, you can display the contents of the process, job, group, and system logical name tables by entering the SHOW LOGICAL command without qualifiers or parameters. The following command displays the logical names and their definitions in all four tables:

```
$ SHOW LOGICAL
```

When you enter the SHOW TRANSLATION command, the system searches the process, job, group, and system logical name tables for the specified logical name. It displays the first definition it finds as well as the table in which it was found. In the following example, you see that the logical name ACCOUNTS is translated as DISK1:[ACCOUNTS] and exists in the process logical name table (LNM$PROCESS_TABLE):

```
$ SHOW TRANSLATION ACCOUNTS
 "ACCOUNTS" = "DISK1:[ACCOUNTS]"  (LNM$PROCESS_TABLE)
```

Some commands and lexical functions do not translate logical names iteratively. The SHOW TRANSLATION command, for example, provides only the immediate equivalence name, as shown in the following example:

```
$ DEFINE SALES_DISK WORKDISK:
$ DEFINE SALES SALES_DISK
$ SHOW TRANSLATION SALES
SALES = "SALES_DISK" (LNM$PROCESS_TABLE)
```

Although the SHOW LOGICAL and SHOW TRANSLATION commands both translate logical names, certain circumstances argue for the use of one command over the other, as follows:

- To ensure that all levels of logical name translation are performed, specify the SHOW LOGICAL command. If you are certain that a logical name does not require iterative translation, specify the SHOW TRANSLATION command.
- Because the SHOW TRANSLATION command is a built-in command, you can interrupt an image (with CTRL/Y, for example) and enter SHOW TRANSLATION without causing the interrupted image to exit. (Built-in commands are described in Section 1.2.) If you interrupt an image with the SHOW LOGICAL command, your interrupted image is forced to exit.

4.1.2 Deleting Logical Names

To delete a logical name defined interactively, use the DEASSIGN command. For example:

```
$ DEFINE STAFF [JONES.STAFF]
    .
    .
    .
$ DEASSIGN STAFF
```

Logical names in your process and job tables are automatically deleted when your process terminates. However, by specifying the /USER_MODE qualifier to the DEFINE command, you can place a logical name in the process logical name table and execute one command image before the logical name is deleted.

4.2 Logical Name Tables

The system stores logical names and their equivalence strings in four logical name tables called process, job, system, and group. Some logical name tables are available only to your process; these tables are called *process-private*. Other tables are *shareable*; that is, they are available to other users on the system.

Identical logical names can exist in more than one table. The logical name that is used depends on the order in which the logical name tables are searched. For example, when the system attempts to translate a logical name in order to identify the location of a file, it uses the logical name LNM$FILE_DEV to provide the list of tables in which to look for the name. The order in which the tables are listed is also the order in which they are searched. The precedence order defined by LNM$FILE_DEV is: (1) process table, (2) job table, (3) group table, (4) system table. Therefore, if a logical name exists in both the process and the group logical name tables, the logical name within the process table is used. See Section 4.3.2 for more information about LNM$FILE_DEV.

Within each table, the system defines some logical names for you. Each table and its system-defined logical names are described in the following sections.

4.2.1 The Process Table

Your process logical name table, named LNM$PROCESS_TABLE, contains logical names that are available only to your process and any subsequent subprocesses. Use the logical name LNM$PROCESS to refer to the process table.

Process logical names are recognized by the process they were created in and by any subsequent subprocesses. However, process logical names are not recognized by any parent process.

To display the logical names in your process table, use the following command:

```
$ SHOW LOGICAL/PROCESS
```

You can also specify the SHOW LOGICAL/TABLE=table_name command to display the contents of any logical name table.

By default, the DEFINE and DEASSIGN commands place names in and delete names from your process table.

Every process on the system has a process logical name table. When you log in, the system creates logical names for your process and places them in your process table. These names are listed in Table 4–1.

Table 4–1 Default Process Logical Names

Logical Name	Description
SYS$COMMAND	The initial file (usually your terminal) from which DCL reads input. (A file from which DCL reads input is called an *input stream*.) The command interpreter uses SYS$COMMAND to "remember" the original input stream.
SYS$DISK	Default device established at login or changed by the SET DEFAULT command.
SYS$ERROR	The default device or file to which DCL writes error messages generated by warnings, errors, and severe errors.
SYS$INPUT	The default file from which DCL reads input.
SYS$NET	The source process that invokes a target process in DECnet–VAX task-to-task communication. When opened by the target process, SYS$NET represents the logical link over which that process can exchange data with its partner. SYS$NET is defined only during task-to-task communication.
SYS$OUTPUT	The default file (usually your terminal) to which DCL writes output. (A file to which DCL writes output is called an *output stream*.)
TT	Default device name for terminals.

Note that the logical names SYS$INPUT, SYS$OUTPUT, SYS$ERROR, and SYS$COMMAND refer to files that remain open for the life of the process. They are referred to as *process-permanent files*. For more information on process-permanent files, see Section 4.9.1.

4.2.2 The Job Table

Your job logical name table contains logical names that are available to all processes in your job tree, no matter what process or subprocess you are currently in. Your job table is named LNM$JOB_xxx, where xxx is the Job Information Block address (defined by the system) for your job tree. Use the logical name LNM$JOB to refer to your job table.

When you log in, the system creates certain logical names and places them in the job logical name table. These names are listed in Table 4–2. In addition, the logical names created for mounted disks and tapes and temporary mailboxes are also placed in the job logical name table.

Table 4–2 Default Job Logical Names

Logical Name	Description
SYS$LOGIN	Your default device and directory when you log in.
SYS$LOGIN_DEVICE	Your default device when you log in.
SYS$REM_ID	For jobs initiated through a DECnet network connection, the identification of the process on the remote node from which the job was originated. On VMS operating systems, if proxy logins are enabled, this identification is the process's user name, or, if proxy logins are not enabled, this is the process identification number (PID). For more information about proxy logins, see the *Guide to VMS System Security*.
SYS$REM_NODE	For jobs initiated through a DECnet network connection, the name of the remote node from which the job was originated.
SYS$SCRATCH	Default device and directory to which temporary files are written.

There is one job table for each job tree in the system. All job tables are shareable so that all users may access them. However, to access a job logical name table other than your own, you must redefine LNM$JOB in your process directory logical name table. For more information about LNM$JOB, see Section 4.3.

4.2.3 The Group Table

The group logical name table contains logical names that are available to all users with the same user identification code (UIC) group number. The group table is named LNM$GROUP_xxx, where xxx represents your UIC group number. Use the logical name LNM$GROUP to refer to your group table. Every group on the system has a corresponding group logical name table.

To create or delete a name in your group table, you need GRPNAM, GRPPRV, or SYSPRV privilege. See the *Guide to Setting Up a VMS System* for a description of user privileges.

4.2.4 The System Table

The system logical name table contains logical names that are available to all users on the system. The system table is named LNM$SYSTEM_TABLE; use the logical name LNM$SYSTEM to refer to it. To create or delete a name in the system table, you must have a UIC group number between 0 and 10, or SYSNAM or SYSPRV privilege.

There is only one system logical name table for the system. It contains the names shown in Table 4–3.

Table 4–3 Default System Logical Names

Logical Name	Description
DBG$INPUT	Default input stream for the VMS Debugger; equated to SYS$INPUT
DBG$OUTPUT	Default output stream for the VMS Debugger; equated to SYS$OUTPUT
SYS$COMMON	Device and directory name for the common part of SYS$SYSROOT
SYS$ERRORLOG	Device and directory name of error log data files
SYS$EXAMPLES	Device and directory name of system examples
SYS$HELP	Device and directory name of system HELP files
SYS$INSTRUCTION	Device and directory name of system instruction data files
SYS$LIBRARY	Device and directory name of system libraries
SYS$LOADABLE_IMAGES	Device and directory of operating system executive loadable images, device drivers, and other executive loaded code
SYS$MAINTENANCE	Device and directory name of system maintenance files
SYS$MANAGER	Device and directory name of system manager files
SYS$MESSAGE	Device and directory name of system message files
SYS$NODE	Network node name for the local system if DECnet–VAX is active on the system
SYS$SHARE	Device and directory name of system shareable images
SYS$SPECIFIC	Device and directory name for node-specific part of SYS$SYSDEVICE
SYS$STARTUP	Device and directory name of system startup files
SYS$SYSDEVICE	VMS system disk containing system directories
SYS$SYSROOT	Device and root directory for system directories
SYS$SYSTEM	Device and directory of operating system programs and procedures
SYS$TEST	Device and directory name of User Environment Test Package (UETP) files
SYS$UPDATE	Device and directory name of system update files

4.3 Directory Logical Name Tables

The system provides the following two directory tables to catalog your logical name tables:

- LNM$PROCESS_DIRECTORY catalogs your process tables (LNM$PROCESS and LNM$JOB).
- LNM$SYSTEM_DIRECTORY catalogs your shareable tables (LNM$GROUP and LNM$SYSTEM).

Both of these directories contain logical names that translate iteratively to table names. The name of a logical name table must be recorded in one of these directory tables in order for the system to find it.

You can see the relationship of directory tables to logical name tables with the SHOW LOGICAL/STRUCTURE command, as shown in the following example:

```
$ SHOW LOGICAL/STRUCTURE
(LNM$PROCESS_DIRECTORY)
    (LNM$PROCESS_TABLE)
(LNM$SYSTEM_DIRECTORY)
    (LNM$GROUP_000360)
    (LNM$JOB_806E98E0)
    (LNM$SYSTEM_TABLE)
```

4.3.1 The Process Directory Table

Each process on the system has its own process directory logical name table. When you log in, the VMS operating system places certain logical names in your process directory table. These names are listed in Table 4–4.

Table 4–4 Default Process Directory Logical Names

Logical Name	Description
LNM$GROUP	A logical name that is defined as LNM$GROUP_xxx, where xxx represents your group number. LNM$GROUP_xxx is the logical name table used by your UIC group. (The table LNM$GROUP_xxx is cataloged in the system directory table.) Therefore, LNM$GROUP is a logical name that translates iteratively to the name of your group logical name table.
LNM$JOB	A logical name that is defined as LNM$JOB_xxx, where xxx represents a number unique to your job tree. LNM$JOB_xxx is the logical name table used by your job. (The table LNM$JOB_xxx is cataloged in the system directory table.) Therefore, LNM$JOB is a logical name that translates iteratively to the name of your job logical name table.

Table 4–4 (Cont.) Default Process Directory Logical Names

Logical Name	Description
LNM$PROCESS	A logical name that translates iteratively to LNM$PROCESS_TABLE, which is the name of your process logical name table.
LNM$PROCESS_DIRECTORY	The name of your process directory logical name table.
LNM$PROCESS_TABLE	The name of your process logical name table.

4.3.2 The System Directory Table

There is one system directory logical name table. The VMS operating system places certain logical names in the system directory table. These names are listed in Table 4–5.

Table 4–5 Default System Directory Logical Names

Logical Name	Description
LNM$DCL_LOGICAL	A logical name that is defined as LNM$FILE_DEV. This logical name iteratively translates into the list of logical name tables searched and displayed by the SHOW LOGICAL and SHOW TRANSLATION commands and the F$TRNLNM lexical function. By default, these commands search and display the process, job, group, and system logical name tables, in that order.
LNM$DIRECTORIES	A logical name that is defined as LNM$PROCESS_DIRECTORY and LNM$SYSTEM_DIRECTORY.
LNM$FILE_DEV	A logical name that is defined as the list of logical name tables searched by the system when processing a file specification. By default, it is defined as LNM$PROCESS, LNM$JOB, LNM$GROUP, and LNM$SYSTEM. This means that the process, job, group, and system logical name tables are searched, in that order.
LNM$GROUP_xxx	The name of a group logical name table, where xxx is a group number. There is an LNM$GROUP_xxx logical name table for each group in the system.
LNM$JOB_xxx	The name of a job logical name table, where xxx is a number unique to this job tree. There is an LNM$JOB_xxx logical name table for each job in the system.

Table 4–5 (Cont.) Default System Directory Logical Names

Logical Name	Description
LNM$SYSTEM	A logical name that translates iteratively to LNM$SYSTEM_TABLE, which is the name of the system logical name table.
LNN$SYSTEM_DIRECTORY	The name of the system directory logical name table.
LNM$SYSTEM_TABLE	The name of the system logical name table.

Generally, you do not need to change the default logical name table definitions set up in the directory tables, LNM$PROCESS_DIRECTORY and LNM$SYSTEM_DIRECTORY. Two reasons for changing the entries in the directory tables are: (1) to create another logical name table, and (2) to change the search order for file specification logical names by redefining LNM$FILE_DEV. See Section 4.6 for information about creating your own logical name table and changing the order in which the system searches the logical name tables.

Multiple tables with the same name may exist. For example, there may exist both a process-private and a shareable table called MY_TABLE. The process-private version always takes precedence over the shareable table in all logical name table processing. When a logical name, such as LNM$FILE_DEV, is used as a table name, the logical name is iteratively translated until a list of table names is formed. During this iterative translation, each name is first translated in the process directory. If this translation fails, it is then translated in the system directory. This order of precedence cannot be changed. As a consequence of this ordering, a logical name placed in the process directory table for use as a table name will always take precedence over any identical name residing in the system directory.

4.4 Logical Name Translation

When the system reads a file specification or device name in a DCL command line, it examines the file specification or device name to see whether the leftmost component is a logical name. If the leftmost component ends with a colon, space, comma, or a line terminator (for example, RETURN), the system attempts to translate it as a logical name. If the leftmost component ends with any other character, the system does not attempt to translate it as a logical name.

After you enter the command shown in the following example, the system checks to see whether PUP is a logical name because PUP is the leftmost component of the file specification. Because the leftmost component is terminated with RETURN, the system attempts to translate PUP.

```
$ TYPE PUP
```

After you enter the command shown in the next example, the system checks whether DISK is a logical name. The system attempts to translate DISK because it is the leftmost component and ends with a colon. (The system does not check PUP.)

```
$ TYPE DISK:PUP
```

In the third example, the system does not try to translate [DRYSDALE]PUP because the leftmost component ends with a square bracket (]):

```
$ TYPE [DRYSDALE]PUP
```

By default, when the system translates logical names in file specifications, it searches the process, job, group, and system tables in that order, and uses the first match it finds.

4.4.1 Iterative Translation

Logical name translation can be iterative. This means that after the system translates a logical name, it repeats the translation process for any logical names it finds contained within the first logical name. For example:

```
$ DEFINE DISK DUA1:
$ DEFINE MEMO DISK:[JEFF.MEMOS]COMPLAINT.TXT
```

In this example, the first DEFINE command equates the logical name DISK to the device name DUA1. The second DEFINE command equates the logical name MEMO to the file specification DISK:[JEFF.MEMOS]COMPLAINT.TXT. When the system translates the logical name MEMO, it finds the equivalence name DISK:[JEFF.MEMOS]COMPLAINT.TXT. It then checks to see whether the leftmost component in this file specification ends in a colon, a space, a comma, or an end-of-line terminator. It finds a colon after DISK. The system translates that logical name also. The final translation of the file specification is as follows:

```
DUA1:[JEFF.MEMOS]COMPLAINT.TXT
```

The system limits the number of levels to which it performs logical name translation. The number of levels varies among system facilities, but it is at least nine. If you define more than the system-determined number of levels, or if you create a circular definition, an error occurs when the logical name is used.

4.4.2 Modifying Logical Name Translation

When you create a logical name, you can specify translation attributes that modify how the system interprets the equivalence name. Use the /TRANSLATION_ATTRIBUTES qualifier to the DEFINE command. (This is a positional qualifier: depending on where you place it on a command line, it can apply translation attributes to all equivalence names or only to certain ones.) Two translation attributes can be specified as values to the /TRANSLATION_ATTRIBUTES qualifier: CONCEALED and TERMINAL.

The CONCEALED attribute causes the logical name of a device, rather than the physical name, to be displayed in system messages (except for the SHOW LOGICAL display). The CONCEALED attribute is usually specified when defining logical names for devices. Using concealed devices allows you to write programs and command procedures and perform other operations without being concerned about which physical device actually holds the disk or tape. It also lets you use names that are more meaningful than the physical device names.

The following example shows how to create a concealed device name:

```
$ DEFINE/TRANSLATION_ATTRIBUTES=CONCEALED DISK DJA3:
$ SHOW DEFAULT
  DISK:[SAM.PUP]
$ SHOW LOGICAL DISK
  "DISK" = "DJA3" (LNM$PROCESS_TABLE)
```

The logical name DISK represents the physical device DJA3. Thus, the SHOW DEFAULT command displays the logical name DISK rather than the actual physical device name, DJA3. The SHOW LOGICAL command reveals the translation of DISK.

The TERMINAL attribute prevents iterative translation of a logical name. That is, the equivalence name is not examined to see if it is also a logical name. The translation is "terminal" (final, or completed) after the first translation.

4.4.3 System Defaults During Logical Name Translation

When the system translates a logical name, it fills in any missing fields in a file specification. It fills them in with the current default device, directory, and version number. When you use a logical name to specify the input file for a command, the command uses the logical name to assign a file specification to the output file as well.

If the equivalence name contains a file name and file type, the output file is given the same file name and file type. If the equivalence name does not contain a file type, a default file type is supplied. The file type supplied depends on the command you are using.

4.5 Logical Name Access Modes

The four access modes in the VMS operating system are as follows:

- User-mode (the outermost and least privileged mode)
- Supervisor-mode
- Executive-mode
- Kernel-mode (the innermost and most privileged mode)

When you create a logical name with DCL commands, it has an access mode of user, supervisor, or executive. By default, logical names are created in supervisor mode; you must have SYSNAM privilege to create an executive mode logical name. To see the access mode for a logical name, use the SHOW LOGICAL/FULL command, as follows:

```
$ SHOW LOGICAL/FULL PROJECT
  "PROJECT" [super] = "DISK1:[JONES]" (LNM$PROCESS_TABLE)
```

This shows that the logical name PROJECT was created in supervisor mode.

You can equate the same logical name to different equivalence strings in the same logical name table by specifying different access modes for each definition. The following example equates the logical name ACCOUNTS to two different equivalence names in the process logical name table—one in supervisor-mode and one in executive-mode:

```
$ DEFINE ACCOUNTS DISK1:[ACCOUNTS]CURRENT.DAT
$ DEFINE/EXECUTIVE_MODE ACCOUNTS DISK1:[JANE.ACCOUNTS]OBSOLETE.DAT
```

Logical names created in user mode are temporary. Define a logical name in user mode when you want to define it only for the execution of the next image. In the following example, the logical name ADDRESSES is automatically deleted after the execution of the program PAYABLE:

```
$ DEFINE/USER_MODE ADDRESSES DISK1:[SAM.ACCOUNTS]OVERDUE.LIS
$ RUN PAYABLE
```

In looking up logical names, all privileged images and utilities, such as LOGINOUT and MAIL, bypass the user- and supervisor-mode portions of the system logical name table. Therefore, DIGITAL recommends that logical names for important system components (public disks and directories, for example) be defined in executive mode, using the DCL command DEFINE/SYSTEM/EXECUTIVE. (Only the operating system and privileged programs can create logical names in kernel-mode.) This operation requires either the SYSPRV or SYSNAM privilege.

4.6 Creating a Logical Name Table

The CREATE/NAME_TABLE command creates a logical name table and catalogs it in one of the directory logical name tables. (Logical names that identify logical name tables or that translate iteratively to logical name tables must always be entered into one of the directory logical name tables.) To create a logical name table that is private to your process, create the table in LNM$PROCESS_DIRECTORY (the default). If you want the table to be shareable, specify /PARENT_TABLE=LNM$SYSTEM_DIRECTORY with the CREATE/NAME_TABLE command. Creating shareable name tables requires SYSPRV privilege or ENABLE access to the parent table.

The following example creates a process-private logical name table named TAX, places the definition for the logical name CREDIT in the table, and verifies the table's creation. (You must specify the /TABLE qualifier with the SHOW LOGICAL command to display a logical name in any table other than LNM$SYSTEM or LNM$PROCESS.)

```
$ CREATE/NAME_TABLE TAX
$ DEFINE/TABLE=TAX CREDIT [ACCOUNTS.CURRENT]CREDIT.DAT
$ SHOW LOGICAL/TABLE=TAX CREDIT
 "CREDIT" = "[ACCOUNTS.CURRENT]CREDIT.DAT"  (TAX)
```

To make the system search a user-created logical name table automatically when processing file specifications, you must create a process-private version of the default search list (LNM$FILE_DEV) in LNM$PROCESS_DIRECTORY. To add the created table's name to the default search list, you can define LNM$FILE_DEV as follows:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
_$ TAX,LNM$PROCESS,LNM$JOB,LNM$GROUP,LNM$SYSTEM
```

Placing the table's name first specifies that the system search that table first, and so on in the order of specification.

To delete a logical name table, specify the table that contains it (the system or process directory logical name table) and the name of the table. Deleting a shareable logical name table requires DELETE access to the table or SYSPRV privilege. For example, to delete the logical name table TAX of the preceding example, specify the following command line:

```
$ DEASSIGN/TABLE=LMN$PROCESS_DIRECTORY TAX
```

Note that all logical names in descendant tables (and the descendant tables themselves) are deleted when a parent logical name table is deleted.

4.7 Search Lists

A search list is a logical name that has more than one equivalence name. You can use a search list in any place you can use a logical name. For example:

```
$ DEFINE GETTYSBURG [JONES.HISTORY],[JONES.WORKFILES]
$ SHOW LOGICAL GETTYSBURG

  "GETTYSBURG" = "[JONES.HISTORY]" (LNM$PROCESS_TABLE)
       = "[JONES.WORKFILES]"
```

The logical name GETTYSBURG is a search list because it has more than one equivalence name.

When you use a logical name that is a search list, the system translates the logical name until it finds a match. The order in which you specify the equivalence strings determines the order in which the system translates the names. It uses each equivalence name listed in the definition until a match is found.

A search list is not a wildcard. It is a list of places to look. Once a file is found, the search is ended. For example:

```
$ TYPE GETTYSBURG:SPEECH.TXT

DISK1:[JONES.HISTORY]SPEECH.TXT;2

Fourscore and seven years ago, our fathers brought forth on
this continent a new nation, conceived in liberty, and
dedicated to the proposition that all men are created equal.
   .
   .
   .
```

In the previous example, the TYPE command searches the equivalence names [JONES.HISTORY] and [JONES.WORKFILES] in the order they were listed when GETTYSBURG was defined. Once it finds a file named SPEECH.TXT, the search is halted and the file is displayed.

You can use a search list with a command that accepts wildcards. When you use wildcards, the system forms file specifications using each equivalence name in the search list. The command operates on each file specification that identifies an existing file.

For example, if you specify the DIRECTORY command with a wildcard character in the version field, it finds all versions of SPEECH.TXT in the search list defined by GETTYSBURG, as shown in the following example:

```
$ DIRECTORY GETTYSBURG:SPEECH.TXT;*

Directory DISK1:[JONES.HISTORY]

SPEECH.TXT;2      SPEECH.TXT;1

Total of 2 files.

Directory DISK1:[JONES.WORKFILES]
```

```
SPEECH.TXT;1

Total of 1 file.

Grand total of 2 directories, 3 files.
```

The DIRECTORY command searches the equivalence names [JONES.HISTORY] and [JONES.WORKFILES] in the order they were listed when GETTYSBURG was defined. It finds a file named SPEECH.TXT in each directory. If SPEECH.TXT exists in only one of the directories, only one directory listing is displayed. If SPEECH.TXT does not exist in either directory, an error message is displayed indicating that the file was not found.

When you use a search list with a command that does not accept wildcards in a file specification, the system forms a file specification using each equivalence name in the search list until a file specification for an existing file is found. The command affects only the first file found. For example:

```
$ DEFINE DECEMBER DISK1:[FRED],WORK2:[BARNEY]
$ EDIT/EDT DECEMBER:QUOTAS.TXT
```

First, the system forms the file specification DISK1:[FRED]QUOTAS.TXT and searches for that file. If QUOTAS.TXT is found in DISK1:[FRED], it is opened for editing. No other files are subsequently opened. If QUOTAS.TXT is not found in DISK1:[FRED], the system searches for it in WORK2:[BARNEY]. If QUOTAS.TXT is found there, it is opened. If it is not found, an error message is displayed. The system displays an error message only after it checks all equivalence names in a search list. Then the system reports an error only on the last file it attempted to find.

The RUN command is an exception. When the RUN command is followed by a search list, the system forms file specifications as described previously. However, the system then checks to see whether any of the files in the list are installed images. It runs the first file in the search list that is an installed image. Then the RUN command terminates.

If none of the file specifications are installed images, the system repeats the process of forming file specifications. This time it looks for each file specification on the disk. It runs the first file it finds there. An error message is displayed if none of the specified files is found in either the known file list or on the disk.

4.8 Logical Node Names

A logical node name is a special type of logical name that can be used in place of a network node name or in place of a node name and an access control string. For example:

```
$ DEFINE BOS "BOSTON""ADAMS JOHN""::"
```

The logical name BOS is equated to the node name BOSTON and an access control string, where ADAMS is the user name and JOHN is the password. Use the logical name BOS to avoid typing (and displaying) your user name and password on the terminal screen.

Note: **Do not place a DEFINE command that includes a password in a file (your login command procedure, for example). If others read the file, they will see the password.**

4.9 System-Created Logical Names

The system creates a number of logical names for you when you start the system and log in. By default, DCL creates and assigns logical names to four process-permanent files. When you redefine these logical names, only your process is affected. The system defines other logical names that you can reassign only with special privileges.

4.9.1 Process-Permanent Logical Names

Process-permanent logical names are created by DCL when you log in and remain defined for the life of your process. You cannot deassign these logical names. You can redefine them (by specifying the same name in a DEFINE command), but if the redefined name is later deassigned, the process-permanent name is reestablished. These process-permanent logical names, as follows, are available to each user of the system at the process level:

- SYS$INPUT—Logical name that refers to the default input device or file
- SYS$OUTPUT—Logical name that refers to the default output device or file
- SYS$ERROR—Logical name that refers to the default device or file to which the system writes messages
- SYS$COMMAND—Logical name that refers to the value of SYS$INPUT when you log in

Table 4–6 shows what these logical names are equated to by default.

Table 4–6 Equivalence Names for Process-Permanent Logical Names

Logical Name	Interactive Mode	Batch Mode	Command Procedure
SYS$COMMAND	Terminal[1]	Disk[2]	Terminal
SYS$INPUT	Terminal	Disk	Disk
SYS$ERROR	Terminal	Log file[3]	Terminal
SYS$OUTPUT	Terminal	Log file	Terminal

[1]Device name of your terminal

[2]Device name of the initial default device

[3]Batch job log file

The following sections describe how to use process-permanent logical names as file specifications.

4.9.1.1 Redefining SYS$INPUT

You can redefine SYS$INPUT so that a command procedure reads input from the terminal or another file. For example, to edit a file from a command procedure, include the following lines in the command procedure:

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ EDIT/TPU MYFILE.DAT
     .
     .
     .
```

In the previous example, SYS$INPUT is redefined as SYS$COMMAND so that the editor obtains input from the terminal rather than from the command procedure file (the default). SYS$COMMAND refers to the terminal, the initial input stream when you logged in. The /USER_MODE qualifier tells the command procedure that SYS$INPUT is redefined only for the duration of the next image. In this example, the next image is the editor. When the editor is finished, SYS$INPUT resumes its default value; in this case, the default value is the command procedure file.

Note that if you redefine SYS$INPUT, DCL ignores your definition. DCL always obtains input from the default input stream. However, images, such as command procedures, can use your definition for SYS$INPUT.

4.9.1.2 Redefining SYS$OUTPUT

You can redefine SYS$OUTPUT to redirect output from your default device to another file. When you redefine SYS$OUTPUT, the system opens a file with the name you specify in the logical name assignment. When you define SYS$OUTPUT, all subsequent output is directed to the new file.

In the following example, SYS$OUTPUT is defined as MYFILE.LIS before the SHOW DEVICES command is entered. The display produced by SHOW DEVICES is directed to MYFILE.LIS in your current directory rather than to your terminal. You can manipulate this data as you would any other text file.

```
$ DEFINE SYS$OUTPUT MYFILE.LIS
$ SHOW DEVICES
```

Remember to deassign SYS$OUTPUT, or output will continue to be written to the file you specify. Note that you can redefine SYS$OUTPUT in user mode (with DEFINE/USER_MODE) to redirect output from an image. This definition is in effect only until the next command image is executed. Once the command image is executed (that is, the output is captured in a file), the logical name SYS$OUTPUT resumes its default value.

When you log in, the system creates two logical names called SYS$OUTPUT. One name is created in executive mode; the other name is created in supervisor mode. You can supersede the supervisor mode logical name by redefining SYS$OUTPUT. If you deassign the supervisor mode name, the system redefines SYS$OUTPUT in supervisor mode, using the executive mode equivalence name. You cannot deassign the executive mode name.

In the following example, SYS$OUTPUT is redefined to the file TEMP.DAT. When SYS$OUTPUT is redefined, output from DCL and from images is directed to the file TEMP.DAT. The output from the SHOW LOGICAL command and from the SHOW TIME command is also sent to TEMP.DAT. When you deassign SYS$OUTPUT, the system closes the file TEMP.DAT and redefines SYS$OUTPUT to your terminal. When you enter the TYPE command, the output collected in TEMP.DAT is displayed on your terminal.

```
$ DEFINE SYS$OUTPUT TEMP.DAT
$ SHOW LOGICAL SYS$OUTPUT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
$ TYPE TEMP.DAT
   "SYS$OUTPUT" = "DISK1:" (LNM$PROCESS_TABLE)
  31-DEC-JAN-1988 13:26:53
```

When you redefine SYS$OUTPUT to a file, the logical name contains only the device portion of the file specification, even though the output is directed to the file you specify. In the previous example, when SYS$OUTPUT was redefined, the equivalence name contained the device name DISK1:, not the full file specification.

If the system cannot open the file you specify when you redefine SYS$OUTPUT, an error message is displayed.

After you redefine SYS$OUTPUT, most commands direct output to the existing version of the file. However, certain commands create a new version of the file before they write output.

4.9.1.3 Redefining SYS$ERROR

You can redefine SYS$ERROR to direct error messages to a specified file. However, if you redefine SYS$ERROR so it is different from SYS$OUTPUT (or if you redefine SYS$OUTPUT without also redefining SYS$ERROR), DCL commands send informational, warning, error, and severe error messages to both SYS$ERROR and SYS$OUTPUT. Therefore, you receive these messages twice—once in the file indicated by the definition of SYS$ERROR and once in the file indicated by SYS$OUTPUT. Success messages are sent only to the file indicated by SYS$OUTPUT.

If you redefine SYS$ERROR and then run an image that references SYS$ERROR, the image sends error messages only to the file indicated by SYS$ERROR even if SYS$ERROR is different from SYS$OUTPUT. Only DCL commands and images using standard VMS error display mechanisms send error messages to both SYS$ERROR and SYS$OUTPUT when these files are different.

4.9.1.4 Redefining SYS$COMMAND

Although you can redefine SYS$COMMAND, DCL ignores your definition. DCL always uses the default definition for your initial input stream. However, if you execute an image that references SYS$COMMAND, the image can use your new definition.

4.9.2 System-Permanent Logical Names

The following table lists the logical names automatically defined when the system starts up. These names are available to all users of the system at the system level.

Logical Name	Equivalence Name
DBG$INPUT	SYS$INPUT at the process level
DBG$OUTPUT	SYS$OUTPUT at the process level
SYS$COMMON	SYS$SYSDEVICE:[SYS*n*.SYSCOMMON.], where *n* is the root directory number of your processor
SYS$ERRORLOG	SYS$SYSROOT:[SYSERR]
SYS$EXAMPLES	SYS$SYSROOT:[SYSHLP.EXAMPLES]
SYS$HELP	SYS$SYSROOT:[SYSHLP]
SYS$INSTRUCTION	SYS$SYSROOT:[SYSCBI]
SYS$LIBRARY	SYS$SYSROOT:[SYSLIB]
SYS$LOADABLE_IMAGES	SYS$SYSROOT:[SYS$LDR]
SYS$MAINTENANCE	SYS$SYSROOT:[SYSMAINT]
SYS$MANAGER	SYS$SYSROOT:[SYSMGR]
SYS$MESSAGE	SYS$SYSROOT:[SYSMSG]
SYS$NODE	Name of your node if you are on a network
SYS$SHARE	SYS$SYSROOT:[SYSLIB]
SYS$SPECIFIC	SYS$SYSDEVICE:[SYS*n*.], where *n* is the root directory number of your processor
SYS$STARTUP	As a search list, points first to SYS$SYSROOT:[SYS$STARTUP], then to SYS$MANAGER
SYS$SYSDEVICE	System disk (usually SYS$DISK)
SYS$SYSROOT	As a search list, points first to SYS$SYSDEVICE:[SYS*n*.], where *n* is the root directory number of your processor; then to SYS$COMMON
SYS$SYSTEM	SYS$SYSROOT:[SYSEXE]
SYS$TEST	SYS$SYSROOT:[SYSTEST]
SYS$UPDATE	SYS$SYSROOT:[SYSUPD]

5 Representing Data with Symbols

As you carry out your computing tasks with the support of DCL and VMS, you may need to store and manipulate data, such as numbers and strings of characters, through the use of symbols. Like files, symbols store data. Yet, unlike files, the symbols you create are temporary and have no means of physical storage—they exist only for the life of your computing session or for the life of a program's execution.

This chapter describes the kinds of data you can use in DCL, how you can use symbols to represent that data, and how you can combine symbols into expressions to manipulate the data that the symbols represent.

5.1 Data Storage

With the VMS operating system, the most common units in which data can be stored are the following:

- Bit—The most basic unit of storage, a bit has a value of 0 or 1.
- Byte—Equal to 8 bits, a byte can represent an unsigned integer value of 0 through 255 and a signed value of –128 through 127. Characters are stored one per byte.
- Word—Equal to 2 bytes, a word can represent an unsigned integer value of 0 through 65,353 and a signed value of –32,768 through 32,767.
- Longword—Equal to 4 bytes (32 bits), a longword can represent an unsigned integer value of 0 through 4,294,967,295 and a signed value of –2,147,483,648 through 2,147,483,647.

The first unit in any series of these units is called the *low-order* unit. In numeric values, the low-order unit is the least-significant unit in the number. For example, in a binary number composed of the series of bits 11111110, the 0 is the low-order bit.

5.2 Creating and Using Symbols

A *symbol* is a name that represents a character value (for example, "DOG"), a numeric value (for example, 17), or a logical value (for example, True). When you use a symbol in a DCL command, DCL replaces the symbol with its value before executing the command. Symbols are useful for representing data in commands and command procedures and as shortcuts for entering commands you use frequently.

For example, you may define a symbol as any of the following:

- Foreign command—Defining a symbol as a foreign command allows you to execute an image by entering only the symbol name. (The command is "foreign" because it is unknown to DCL.) In the following example, the symbol FIX is defined to execute the image NUMFIX.EXE in the [BILLS] directory on the disk ACCOUNTS:

  ```
  $ FIX == "$ACCOUNTS:[BILLS]NUMFIX"
  ```

- Command line—Defining a symbol as a command line allows you to execute the command by entering only the symbol name. In the following example, the symbol HUBBUB is defined to establish a network connection to the node HUBBUB:

  ```
  $ HUBBUB == "SET HOST HUBBUB"
  ```

  Setting a symbol equal to a command line that executes a command procedure allows you to execute the procedure by typing only the symbol name. In the following example, COUNT is defined to execute the command procedure CENSUS:

  ```
  $ COUNT == "@DISK1:[JONES.PROCEDURES]CENSUS"
  ```

  When you enter COUNT to execute CENSUS, place any parameters for CENSUS after the symbol as if you had entered @CENSUS.

- Character string—Defining a symbol as a character string allows you to insert that string in a command line by typing the symbol (with surrounding apostrophes to force substitution, as described in Section 5.5). In the following example, the symbol FILE is first defined as a complete file specification and then used in the TYPE command:

  ```
  $ FILE == "DISK1:[JONES.TAXES]CORPORATE.DAT"
  $ TYPE 'FILE'
  ```

  The string can be a directory you often access. In the following example, whenever the symbol TAXES occurs in a command line, the literal value replaces the symbol before the line is executed.

  ```
  $ TAXES == "[JONES.TAXES]"
  $ COPY 'TAXES'OVERDUE.DAT OVERDUE.TMP
  ```

Symbols can also hold variables, which are values that you calculate or that you assign as something other than a literal. For example, you might assign the value of a lexical function to a variable or read the value of a file record into a variable. As variables, symbols are most often used in command procedures (see Chapter 6).

To create a symbol, assign a value to a symbolic name using the following format:

```
symbol-name =[=] value
```

The symbol name can be 1 through 255 characters long and must begin with a letter, an underscore (_), or a dollar sign ($). In a symbol name, both lowercase and uppercase letters are treated as uppercase.

The value you assign to a symbol can be made either locally or globally available to the command interpreter:

- Local—A local symbol is available to the command level that defined it, to any command procedure executed from that level, and to lower command levels. (By convention, DCL level—command level 0—is the highest command level and command level 31 the lowest command level. Thus, when you move from command level 3 to command level 2, you are moving to the next higher command level. If you execute a command procedure interactively, the commands in the procedure are executed at command level 1. You can create a maximum of 32 command levels.)

- Global—A global symbol is available to any command level regardless of the level at which it was defined.

To create a local symbol, use a single equal sign in the assignment statement. To create a global symbol, use two equal signs. The following commands define the local symbol FILE as the character string DISK2:[BOLIVAR]PRICES.CUR and the global symbol MAX_VALUE as the number 24.

```
$ FILE = "DISK2:[BOLIVAR]PRICES.CUR"
$ MAX_VALUE == 24
```

You can omit the quotation marks around character strings in assignment statements if you precede the equal sign or signs with a colon. Symbol assignments that omit quotation marks automatically change the character string to uppercase letters and compress multiple spaces and tabs to a single space. The following example again creates the local symbol FILE, this time omitting the quotation marks because of the included colon:

```
$ FILE := ACCOUNTS:[BOLIVAR]PRICES.84
```

The result of DCL's evaluation of a symbol is either a character string or an integer value. The data type (character or integer) of a symbol is determined by the data type of the value currently assigned. The type is not permanent: if the value changes type, as in the following example, the symbol changes type. In the following example, the local symbol NUM is first assigned a character value and then converted to an integer value when used in an expression with an integer:

```
$ NUM = "12"
$ RESULT = NUM + 10
```

Local symbols take precedence over global symbols with the same name. Symbols take precedence over identical command names. This means that if you define a symbol with the same name as a DCL command, your definition overrides the command name. For example, if you define the symbol HELP as the command TYPE HELP.LST, you can no longer invoke the system's HELP facility by typing HELP.

Symbols are stored in the following tables, which are maintained by the operating system:

- Local symbol table—DCL maintains a local symbol table for your main process and for every command level that you create when you execute a command procedure, use the CALL command, or submit a batch job. A local table is deleted when its associated command level terminates. (See Chapter 3 for more information about processes, command procedures, and batch jobs.)

  In addition to the local symbols you create, a local symbol table contains eight symbols that are maintained by DCL. These symbols, named P1, P2, and so on through P8, are used for passing parameters to a command procedure. Parameters passed to a command procedure are regarded as character strings. Otherwise, P1 through P8 are defined as null character strings (""). They are stored in the local symbol table.

- Global symbol table—DCL maintains only one global symbol table for the duration of a process. In addition to the global symbols you create, the global symbol table contains the reserved global symbols described in the following table. These global symbols give you status information on your programs and command procedures as well as on system commands and utilties.

Reserved Global Symbols	Definition
$STATUS	The condition code returned by the most recently executed command. $STATUS conforms to the format of a VMS message code. User programs can set the value of the global symbol $STATUS by including a parameter value to the EXIT command. The system uses the value of $STATUS to determine which message, if any, to display and whether to continue execution at the next-higher command level. The value of the lower three bits in $STATUS is placed in the global symbol $SEVERITY.
$SEVERITY	The severity level of the condition code returned by the most recently executed command. $SEVERITY, which is equal to the lower three bits of $STATUS, can have the following values: 0 Warning 1 Success 2 Error 3 Information 4 Severe (fatal) error
$RESTART	Has the value TRUE if a batch job was restarted after it was interrupted by a system crash. Otherwise, $RESTART has the value FALSE.

5.3 Abbreviating Symbol Names

You can use abbreviated forms of symbols if you define them with the asterisk. The following example shows how to create a local symbol that can be abbreviated:

```
$ M*AIL = "MAIL"
```

Once this symbol is established, the VMS Mail Utility is invoked whenever you specify any of the following versions of the symbolic name:

```
$ M
$ MA
$ MAI
$ MAIL
```

Generally, you can use abbreviated symbol definitions in any situation that allows a symbol to be used. However, there are some restrictions as follows:

- You cannot abbreviate symbols that involve substring replacement.
- When you define a symbol that includes an asterisk, existing symbols may possibly be deleted. If an existing symbol exactly matches the new symbol at or past the asterisk, the new symbol replaces the existing symbol.

- If you define a symbol with an asterisk, you cannot define another symbol whose name partly matches the existing symbol at or past the asterisk.

5.4 DCL Commands to Use with Symbols

Table 5–1 shows the DCL commands you can use with symbols.

Table 5–1 DCL Commands to Use with Symbols

Command	Function
SHOW SYMBOL	Displays the value of the specified symbol. By default, the SHOW SYMBOL command searches the local symbol tables and then the global symbol table to locate a specified symbol name.
DELETE/SYMBOL	Deletes a symbol. By default, the DELETE/SYMBOL command searches for symbols only in the local symbol table. To delete a global symbol, use the /GLOBAL qualifier.
SET SYMBOL/SCOPE	You can mask global or local symbols at the specified command level.
INQUIRE	Reads a value from SYS$COMMAND and assigns it to a symbol.
READ	Reads a record from a file and assigns its contents to a symbol.

The SHOW SYMBOL command displays symbol values. Specify the name of the symbol to display the value of a particular local symbol. Specify the name of the symbol and /GLOBAL to display the value of a particular global symbol. Specify /ALL to display all local symbols and /ALL/GLOBAL to display all global symbols.

The DELETE/SYMBOL command deletes a symbol. You must include the /GLOBAL qualifier to delete a global symbol. In the following example, the global symbol TEMP is deleted:

```
$ DELETE/SYMBOL/GLOBAL TEMP
```

The SET SYMBOL/SCOPE=(keyword,...) command controls access to local and global symbols in command procedures. This allows you to treat symbols as undefined without deleting them. Symbol scoping works differently for local and global symbols.

If you specify /SCOPE=NOLOCAL, all local symbols defined in an outer procedure level are treated as undefined by the current procedure and any inner levels. Specifying LOCAL removes any symbol translation limit set by the current procedure level.

For example, if SET SYMBOL/SCOPE=NOLOCAL was specified at procedure levels 2 and 4, procedure level 2 can access only procedure level 2 local symbols. Procedure level 3 can access procedure levels 2 and 3 local symbols; procedure level 4 can access procedure level 4 local symbols and any local symbols in inner procedure levels.

Global symbols are not procedure level dependent. The global symbol scoping state (GLOBAL or NOGLOBAL) that is in effect when a new procedure level is invoked is propagated to the new procedure level. Specifying /SCOPE=NOGLOBAL makes all global symbols inaccessible for all subsequent commands until you either specify /SCOPE=GLOBAL or exit to a previous level at which global symbols were accessible.

In the following example, the SET SYMBOL command denies access to all global symbols:

```
$ SET SYMBOL/SCOPE=NOGLOBAL
```

Exiting a procedure level back to an outer procedure level causes the symbol scope-state to be restored for both local and global symbols.

The INQUIRE and READ commands are most often used within DCL command procedures and are therefore discussed in Chapter 6.

5.5 Symbol Substitution

When a command line is executed, symbols in the following positions are automatically substituted:

- On the right side of an [:]= or [:]== assignment statement
- In a lexical function
- In the brackets on the left side of an assignment statement when you are performing substring substitution or number overlays (see Section 5.6.2.4)
- In a DEPOSIT, EXAMINE, IF, or WRITE command
- At the beginning of the command line

To force substitution of a symbol that is not in one of the positions listed, enclose the symbol with apostrophes as follows:

```
$ TYPE 'B'
```

To force substitution of a symbol within a quoted character string, precede that symbol with double apostrophes and follow it with a single apostrophe as follows:

```
$ T = "TYPE ''B'"
```

When processing a command line, DCL replaces symbols with their values in the following order:

- Forced substitution—From left to right, DCL replaces all strings delimited by apostrophes (or double apostrophes for strings within quotation marks). Symbols preceded by single apostrophes are translated iteratively; symbols preceded by double apostrophes are not.
- Automatic substitution—From left to right, DCL evaluates each value in the command line, executing it if it is a command and evaluating it if it is an expression. Symbols in expressions are replaced by their assigned values; this substitution is not iterative.

The following example demonstrates the effect of the order in which DCL substitutes symbols. Assume the following symbol definitions:

```
$ PN = "PRINT/NOTIFY"
$ FILE1 = "[BOLIVAR]TEST_CASE.TXT"
$ NUM = 1
```

Given the preceding symbol definitions, the following commands print the file named [BOLIVAR]TEST_CASE.TXT:

```
$ FILE = "'FILE''NUM''"
$ PN 'FILE'
```

In the first command, forced substitution causes NUM to become 1, making FILE''NUM' become FILE1. If you enter the command SHOW SYMBOL FILE, you will see that FILE = "'FILE1'".

The second command performs two substitutions. First, 'FILE' is substituted with 'FILE1'. 'FILE1' also requires substitution because it is enclosed in single quotation marks. Automatic substitution causes FILE1 to become [BOLIVAR]TEST_CASE.TXT. This file name is then appended to the value of PN, which is PRINT/NOTIFY. The resulting string is as follows:

```
$ PRINT/NOTIFY [BOLIVAR]TEST_CASE.TXT
```

5.6 Storing and Manipulating Data with Symbols

You can use symbols to store and manipulate a variety of values. This section describes the values that can be stored in symbols. It also describes how symbols can be combined in expressions to manipulate the values the symbols contain.

5.6.1 Symbol Values

A symbol can be defined as a character string, a number, a lexical function, a logical value, or another symbol. The following sections describe these values.

5.6.1.1 Character String Values

A character string can contain any characters that can be printed. Appendix A, which includes tables of the ASCII character set and the DEC Multinational Character Set, shows those characters that you can include in a character string.

Characters fall into the following three main categories:

- Alphanumeric characters—The uppercase letters A through Z, the lowercase letters a through z, the digits 1 through 9, the dollar sign ($), the underscore (_), and the hyphen (-).
- Special characters—All other characters that can be displayed or printed: the exclamation point (!), quotation marks ("), number sign (#), and so on.
- Nonprintable characters—All characters that cannot be printed or displayed. In general, nonprintable characters are ignored for display

and print purposes. However, several nonprintable characters serve control functions as follows:

Character	Function
HT	Starts printing or typing at the next horizontal tab
LF	Starts printing or typing on the next line
FF	Starts printing or typing at the top of the next page
CR	Starts printing or typing at the first space on the same line
ESC	Introduces a terminal escape sequence
SP	Inserts one space

You can define a character string by enclosing it in quotation marks. In this way, alphabetic case and spaces are preserved when the symbol assignment is made.

5.6.1.2 Numeric Values

A number can have the following values:

- Decimal—The ASCII characters 0 through 9
- Hexadecimal—The ASCII characters 0 through 9 and A through F
- Octal—The ASCII characters 0 through 7

The number you assign to a symbol must be in the range -2147483648 through 2147483647 (decimal). (An error is not reported if a number outside this range is specified or calculated, but an incorrect value results.)

At DCL command level and within command procedures, specify a number as follows:

- Positive numbers—Specify a positive number by typing the appropriate digits. The following example assigns the number 13 to the symbol DOG_COUNT:

```
$ DOG_COUNT = 13
$ SHOW SYMBOL DOG_COUNT
  DOG_COUNT = 13   Hex = 0000000D  Octal = 00000000015
```

- Negative numbers—Precede a negative number with a minus sign, as in the following example:

```
$ BALANCE = -15237
$ SHOW SYMBOL BALANCE
  BALANCE = -15237   Hex = FFFFC47B  Octal = 37777742173
```

- Radix—Specify a number in a radix other than decimal by preceding the number (but not the minus sign) with %X for hexadecimal numbers and %0 for octal numbers. For example:

```
$ DOG_COUNT = %XD
$ SHOW SYMBOL DOG_COUNT
  DOG_COUNT = 13   Hex = 0000000D  Octal = 00000000015

$ BALANCE = -%X3B85
$ SHOW SYMBOL BALANCE
  BALANCE = -15237   Hex = FFFFC47B  Octal = 37777742173
```

- Fractions—A number cannot include a decimal point. In calculations, fractions are truncated; for example, 8 divided by 3 equals 2.

Numbers are stored internally as signed 4-byte integers, called longwords; positive numbers have values of 0 through 2147483647 and negative numbers have values of 4294967296 minus the absolute value of the number. The number –15237, for example, is stored as 4294952059. Negative numbers are converted back to minus-sign format for ASCII or decimal displays; however, they are not converted back for hexadecimal and octal displays. For example, the number –15237 appears in displays as hexadecimal FFFFC47B (decimal 4294952059) rather than hexadecimal –00003B85.

Numbers are stored in text files as a series of digits using ASCII conventions (for example, the digit 1 has a storage value of 49).

5.6.1.3 Values Returned by Lexical Functions

Typically used in command procedures, lexical functions provide users with a means to obtain information from the system, including information about system processes, batch and print queues, and user processes. You can also use lexical functions to manipulate character strings and translate logical names. When you assign a lexical function to a symbol, the symbol is equated to the information returned by the lexical function (for example, a number or character string). At DCL level, you can then display that information with the DCL command SHOW SYMBOL. In a command procedure, the information stored in the symbol can be used later in the procedure. See the *VMS DCL Dictionary* for a description of each lexical function.

To use a lexical function, type the name of the lexical function (which always begins with F$) and its argument list. Use the following syntax:

```
F$function-name(args[,...])
```

The argument list follows the function name with any number of intervening spaces and tabs. When using a lexical function, observe the following rules:

- Enclose the argument list in parentheses.
- Within the list, specify arguments in exact order and separate them with commas; even if you omit an optional argument, do not omit the comma.
- If no arguments are required, type an empty set of parentheses.
- Do not enclose lexical functions in quotation marks.
- If an argument contains a character string, enclose the character string in quotation marks.
- If an argument contains an integer, a symbol, or another lexical function, do not enclose these values in quotation marks.

Use lexical functions the same way you would use character strings, integers, and symbols. The following example uses the F$LENGTH function. F$LENGTH returns an integer that specifies the length of the string. The returned value is assigned to the symbol LEN.

```
$ LEN = F$LENGTH("The cow jumped over the moon.")
$ SHOW SYMBOL LEN
  LEN = 29   Hex = 0000001D  Octal = 00000000035
```

You can use a lexical function in any position that you can use a symbol. In positions where symbol substitution must be forced by enclosing the symbol in apostrophes, lexical function evaluation must be forced by placing the lexical function within apostrophes. Lexical functions can also be used as argument values in other lexical functions.

The following example equates the length of the character symbol LINE to a numeric symbol named L:

```
$ L = F$LENGTH (LINE)
```

The following example strips the last two characters from the character string that is the value of the symbol LINE:

```
$ LINE = F$EXTRACT (0,F$LENGTH(LINE)-2,LINE)
```

5.6.1.4 Logical Values

Some operations interpret numbers and character strings as logical data with values as follows:

- True—A number has a logical value of true if it is odd (that is, the low-order bit is 1). A character string has a logical value of true if the first character is an uppercase or lowercase T or Y.
- False—A number has a logical value of false if it is even (that is, the low-order bit is 0). A character string has a logical value of false if the first character is not an uppercase or lowercase T or Y.

In both of the following examples, DOG_COUNT is assigned the value 13. IF STATUS means if the logical value of STATUS is true.

```
$ STATUS = 1
$ IF STATUS THEN DOG_COUNT = 13
```

```
$ STATUS = "TRUE"
$ IF STATUS THEN DOG_COUNT = 13
```

5.6.1.5 Using a Symbol as a Value for Another Symbol

After a symbol is defined, it can be used as a value for another symbol. It can be interpreted as a character string or a number, depending on the context in which it is used. For example, suppose a symbol, COUNT, is assigned the integer value 3 as follows:

```
$ COUNT = 3
```

Then the value of COUNT can be used in other assignment statements. In the following example, the value of COUNT is added to 1:

```
$ TOTAL = COUNT + 1
```

The result, 4, is equated to the symbol TOTAL. You can confirm the assignment of the value to TOTAL by entering the SHOW SYMBOL command as follows:

```
$ SHOW SYMBOL TOTAL
  TOTAL = 4   Hex = 00000004  Octal = 00000000004
```

You can include the symbol COUNT in a string assignment statement as follows:

```
$ BARK := P'COUNT'
```

COUNT is converted to a string value and appended to the character P. BARK now has the value P3.

To include a symbol in a string assignment, use either a colon and an equal sign (:=) or a colon and two equal signs (:==), and enclose the symbol in apostrophes. Otherwise, DCL will not recognize it as a symbol.

If you define a null character string for a symbol, that symbol has a value of 0, as shown in the following example:

```
$ A = ""
$ B = 2
$ C = A + B
$ SHOW SYMBOL C
   C = 2  Hex = 00000002  Octal = 00000000002
```

5.6.2 Using Symbols in Expressions

An *expression* is a combination of values. Each value in an expression is connected to another value by an *operator*. Operators are denoted in the following ways:

- Special characters—Asterisk (*), slash (/), plus sign (+), and minus sign (-).
- Special names—.EQ., .GE., .GT., .LE., .LT., .NE., .NOT., .AND., and .OR.; the names can be uppercase or lowercase.

Data entities and operators can be adjacent or can be separated by any number of spaces or tabs. The values in the expression can be symbols or literal values (such as 3 or "DOG"). Expressions take the following two forms:

- Operations—An operation combines two data entities or alters a data entity. The following example combines the literal values 10 and 3 by adding them:

  ```
  $ DOG_COUNT = 10 + 3
  $ SHOW SYMBOL DOG_COUNT
    DOG_COUNT = 13   Hex = 0000000D  Octal = 00000000015
  ```

- Comparisons—A comparison evaluates a relationship between two entities as true or false. A true comparison evaluates to a numeric value of 1, and a false comparison evaluates to a numeric value of 0. The following example compares the value of the symbol DOG_COUNT with the literal value 13 and finds them to be equal:

  ```
  $ DOG_CHECK = DOG_COUNT .EQ. 13
  $ SHOW SYMBOL DOG_CHECK
    DOG_CHECK = 1   Hex = 00000001  Octal = 00000000001
  ```

You can create character string expressions, numeric expressions, and logical expressions. These are described in the following sections.

5.6.2.1 Character String Expressions

A character string expression can contain character strings, lexical functions that evaluate to character strings, or symbols that have character string values. Attempting an operation or comparison between a character string and a number causes the character string to be converted to a number.

You can specify the following character string operations:

- Concatenation—The plus sign concatenates two character strings. For example:

  ```
  $ COLOR = "light brown"
  $ WEIGHT = "30 lbs."
  $ DOG2 = "No tag, " + COLOR + ", " + WEIGHT
  $ SHOW SYMBOL DOG2
    DOG2 = "No tag, light brown, 30 lbs."
  ```

- Reduction—The minus sign removes the second character string from the first character string. For example:

  ```
  $ SHOW SYMBOL DOG2
    DOG2 = "No tag, light brown, 30 lbs."
  $ DOG2 = DOG2 - ", 30 lbs."
  $ SHOW SYMBOL DOG2
    DOG2 = "No tag, light brown"
  ```

  If the second character string occurs more than once in the first character string, only the first occurrence of the string is removed.

When you compare two character strings, the strings are compared character by character; strings of different lengths are not equal (for example, "dogs" is greater than "dog").

The comparison criteria are the ASCII values of the characters. Under this criterion, the digits 0 through 9 are less than the letters A through Z, and the uppercase letters A through Z are less than the lowercase letters a through z. A character string comparison terminates when either of the following conditions is true:

1. All the characters have been compared, in which case the strings are equal.
2. The first mismatch occurs.

You can specify the following varieties of string comparisons. In the examples, assume that the symbol LAST_NAME has the value "WHITFIELD."

- Equal to—The operator .EQS. compares one character string to another for equality. The following comparison evaluates to 0 to indicate that the value of the symbol LAST_NAME does not equal the literal "NORMAN":

  ```
  $ TEST_NAME = LAST_NAME .EQS. "NORMAN"
  $ SHOW SYMBOL TEST_NAME
    TEST_NAME = 0   Hex = 00000000  Octal = 00000000000
  ```

- Greater than or equal to—The operator .GES. compares one character string to another for a greater or equal value in the first specified string. The following comparison evaluates to 1 to indicate that the value of the symbol LAST_NAME is greater than or equal to the literal "NORMAN":

```
$ TEST_NAME = LAST_NAME .GES. "NORMAN"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 1   Hex = 00000001  Octal = 00000000001
```

- Greater than—The operator .GTS. compares one character string to another for a greater value in the first specified string. The following comparison evaluates to 1 to indicate that the value of the symbol LAST_NAME is greater than the literal "NORMAN":

```
$ TEST_NAME = LAST_NAME .GTS. "NORMAN"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 1   Hex = 00000001  Octal = 00000000001
```

- Less than or equal to—The operator .LES. compares one character string to another for a lesser or equal value in the first specified string. The following comparison evaluates to 0 to indicate that the value of the symbol LAST_NAME is not less than or equal to the literal "NORMAN":

```
$ TEST_NAME = LAST_NAME .LES. "NORMAN"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 0   Hex = 00000000  Octal = 00000000000
```

- Less than—The operator .LTS. compares one character string to another for a lesser value in the first specified string. The following comparison evaluates to 0 to indicate that the value of the symbol LAST_NAME is not less than the literal "NORMAN":

```
$ TEST_NAME = LAST_NAME .LTS. "NORMAN"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 0   Hex = 00000000  Octal = 00000000000
```

- Not equal—The operator .NES. compares one character string to another for inequality. The following comparison evaluates to 1 to indicate that the value of the symbol LAST_NAME does not equal the literal "NORMAN":

```
$ TEST_NAME = LAST_NAME .NES. "NORMAN"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 1   Hex = 00000001  Octal = 00000000001
```

5.6.2.2 Numeric Expressions

In a numeric expression, the values involved must be literal numbers (such as 3) or symbols with numberic values. In addition, you can use a character string that represents a number (for example, "23" or "–51"). Attempting an operation or comparison between a number and a character string causes the character string to be converted to a number.

You can specify the following numeric operations:

- Multiplication—The asterisk multiplies two numbers. For example:

```
$ BALANCE = 142 * 14
$ SHOW SYMBOL BALANCE
  BALANCE = 1988   Hex = 000007C4  Octal = 00000003704
```

- Division—The slash divides the first specified number by the second specified number. For example:

```
$ BALANCE = BALANCE / 14
$ SHOW SYMBOL BALANCE
  BALANCE = 142   Hex = 0000008E  Octal = 00000000216
```

  If a number does not divide evenly, the remainder is lost. (No rounding takes place.)

- Addition—The plus sign adds two numbers. For example:

```
$ BALANCE = BALANCE + 37
$ SHOW SYMBOL BALANCE
  BALANCE = 179   Hex = 000000B3  Octal = 00000000263
```

- Subtraction—The minus sign subtracts the second specified number from the first specified number. For example:

```
$ BALANCE = BALANCE - 15416
$ SHOW SYMBOL BALANCE
  BALANCE = -15237   Hex = FFFFC47B  Octal = 00000142173
```

- Unary plus and minus—The plus and minus signs change the sign of the number they precede. For example:

```
$ BALANCE = -(-142)
$ SHOW SYMBOL BALANCE
  BALANCE = 142   Hex = 0000008E   Octal = 00000000216
```

You can specify the following numeric comparisons:

- Equal to—The operator .EQ. compares one number to another for equality. The following comparison evaluates to 1 to indicate that BALANCE equals -15237:

```
$ TEST_BALANCE = BALANCE .EQ. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 1   Hex = 00000001  Octal = 00000000001
```

- Greater than or equal to—The operator .GE. compares one number to another for a greater or equal value in the first number. The following comparison evaluates to 1 to indicate that BALANCE is greater than or equal to -15237:

```
$ TEST_BALANCE = BALANCE .GE. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 1   Hex = 00000001  Octal = 00000000001
```

- Greater than—The operator .GT. compares one number to another for a greater value in the first number. The following comparison evaluates to 0 to indicate that BALANCE is not greater than -15237:

```
$ TEST_BALANCE = BALANCE .GT. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 0   Hex = 00000000  Octal = 00000000000
```

- Less than or equal to—The operator .LE. compares one number to another for a lesser or equal value in the first number. The following comparison evaluates to 1 to indicate that BALANCE is less than or equal to -15237:

```
$ TEST_BALANCE = BALANCE .LE. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 1   Hex = 00000001  Octal = 00000000001
```

- Less than—The operator .LT. compares one number to another for a lesser value in the first number. The following comparison evaluates to 0 to indicate that BALANCE is not less than -15237:

```
$ TEST_BALANCE = BALANCE .LT. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 0   Hex = 00000000  Octal = 00000000000
```

- Not equal to—The operator .NE. compares one number to another for inequality. The following comparison evaluates to 0 to indicate that BALANCE equals −15237:

```
$ TEST_BALANCE = BALANCE .NE. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 0   Hex = 00000000  Octal = 00000000000
```

5.6.2.3 Logical Expressions

A logical operation affects all the bits in the number being acted upon. The values for logical expressions are integers, and the result of the expression is an integer as well. If you specify a character string value in a logical expression, the string is converted to an integer before the expression is evaluated.

String and integer values are evaluated as follows:

- If the first character is T, t, Y, or y, a character string has a logical value of true (1).
- If the first character is not T, t, Y, or y, a character string has a logical value of false (0).
- If an integer is odd (the low-order bit is 1), it has a logical value of true (1).
- If an integer is even (the low-order bit is 0), it has a logical value of false (0).

Typically, you use logical expressions to evaluate the low-order bit of a logical value; that is, to determine whether the value is true or false. You can specify the following logical operations:

- Not—The operator .NOT. reverses the bit configuration of a logical value. A true value becomes false and a false value becomes true. The following example reverses a true value to false. The expression evaluates to −2; the value is even and is therefore false:

```
$ SHOW SYMBOL STATUS
  STATUS = 1   Hex = 00000001  Octal = 00000000001
$ STATUS = .NOT. STATUS
$ SHOW SYMBOL STATUS
  STATUS = -2   Hex = FFFFFFFE  Octal = 37777777776
```

- And—The operator .AND. combines two logical values as follows:

Bit Level	Entity Level
1 .AND. 1 = 1	true .AND. true = true
1 .AND. 0 = 0	true .AND. false = false
0 .AND. 1 = 0	false .AND. true = false
0 .AND. 0 = 0	false .AND. false = false

The following example combines a true value and a false value to produce a false value:

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .AND. STAT2
$ SHOW SYMBOL STATUS
  STATUS = 0   Hex = 00000000  Octal = 00000000000
```

- Or—The operator .OR. combines two logical values as follows:

Bit Level	Entity Level
1 .OR. 1 = 1	true .OR. true = true
1 .OR. 0 = 1	true .OR. false = true
0 .OR. 1 = 1	false .OR. true = true
0 .OR. 0 = 0	false .OR. false = false

The following example combines a true value and a false value to produce a true value:

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .OR. STAT2
$ SHOW SYMBOL STATUS
  STATUS = 1   Hex = 00000001  Octal = 00000000001
```

5.6.2.4 Substring Replacement and Numeric Overlays

You can replace a part of a character string with another character string. The assignment statement has the following format:

symbol-name[offset,size] := replacement-string

or

symbol-name[offset,size] := = replacement-string

The fields are as follows:

- *Offset* is an integer that indicates the position of the replacement-string relative to the first character in the original string. An offset of 0 means the first character in the symbol, an offset of 1 means the second character, and so on.
- *Size* is an integer that indicates the length of the replacement-string.

To replace substrings, observe the following rules:

- The square brackets are required notation. No spaces are allowed between the symbol name and the left bracket.
- Integer values can be in the range of 0 through 768.
- The replacement-string must be a character string.

In the following example, the first assignment statement gives the symbol A the value PACKRAT. The second statement specifies that MUSK replace the first four characters in the value of A. The result is that the value of A becomes MUSKRAT.

```
$ A := PACKRAT
$ A[0,4] := MUSK
$ SHOW SYMBOL A
  A = "MUSKRAT"
```

The symbol name you specify can be undefined initially. The assignment statement creates the symbol name and, if necessary, provides leading or trailing spaces in the symbol value. For example:

```
$ B[4,3] := RAT
```

If the symbol B does not have a previous value, it is given a value of four leading spaces followed by RAT. This format creates a blank line of any length. The following example gives the symbol LINE a value of 80 blank spaces:

```
$ LINE[0,80]:= " "
```

Lining up records in columns makes a list easier to read and sort. You can use this format to specify how you want data to be stored. For example:

```
$ DATA[0,15] := 'NAME'
$ DATA[17,1] := 'GRADE'
```

The first statement fills in the first 15 columns of DATA with whatever value NAME has. The second statement fills in column 18 with whatever value GRADE has. Columns 16 and 17 contain blanks.

A special format of the assignment statement can also be used to perform binary (bit-level) overlays of the current symbol value. This format is as follows:

$ symbol-name[bit-position,size] = replacement-expression

or

$ symbol-name[bit-position,size] = = replacement-expression

where:

- *bit-position* is an integer that indicates the location relative to bit 0 at which the overlay is to occur.
- *size* is an integer that indicates the number of bits to be overlaid.

When using numeric overlays, observe the following rules:

- The square brackets are required notation. No spaces are allowed between the symbol name and the left bracket.
- Literal values are assumed to be decimal.
- The maximum length for both *bit-position* and *size* is 32 bits.
- The *replacement-expression* must be a numeric expression.
- When *symbol-name* is either undefined or defined as a string, the result of the overlay is a string. Otherwise the result is an integer.

The following example defines the symbol BELL as the value 7. The low-order byte of BELL has the binary value 00000111. By changing the 0 at offset 5 to 1 (beginning with 0, count bits from right to left), you create the binary value 00100111 (decimal value 39).

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
 BELL = 39   Hex = 00000027   Octal = 00000000047
```

5.6.2.5 Order of Operations and the Results of Evaluations

An expression can contain any number of operations and comparisons. You can indicate precedence (the order in which operation and comparison should be evaluated) by placing operations to be performed first in parentheses. (Parentheses can be nested.) Otherwise, operations within an expression are evaluated in the following order:

1 Unary plus (+) and minus (−)

2 Multiplication and division

3 All other numeric and character operations

4 All numeric and character comparisons

5 Logical NOT operations

6 Logical AND operations

7 Logical OR operations

Operations and comparisons that have the same precedence are evaluated from left to right. The following examples illustrate precedence of operations in expressions:

```
$ BALANCE = 150 + 20 * 4
      (BALANCE = 150 + 80)
$ SHOW SYMBOL BALANCE
  BALANCE = 230   Hex = 000000E6  Octal = 00000000346

$ BALANCE = (150 + 20) * 4
  (BALANCE = 170 * 4)
$ SHOW SYMBOL BALANCE
  BALANCE = 680  Hex = 000002A8  Octal = 00000001250

$ STATUS = 150 * 4 .GT. 80 * 2
      (STATUS = 600 .GT. 160)
$ SHOW SYMBOL STATUS
  STATUS = 1   Hex = 00000001  Octal = 00000000001
```

An expression has either an integer or a string value, depending on the types of values and the operators used. The following table summarizes how DCL evaluates expressions. The first column lists the different values and operators that an expression might contain. The second column tells, for each case, what the entire expression is equated to. Within the table *any value* stands for a string or an integer.

Table 5–2 Determining the Value of an Expression

Expression	Resulting Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer

Table 5–2 (Cont.) Determining the Value of an Expression

Expression	Resulting Value Type
String lexical function	String
Integer symbol	Integer
String symbol	String
+, –, or .NOT. any value	Integer
Any value .AND. or .OR. any value	Integer
String + or - string	String
Integer + or - any value	Integer
Any value + or - integer	Integer
Any value * or / any value	Integer
Any value (string comparison) any value	Integer
Any value (numeric comparison) any value	Integer

6 Writing and Using Command Procedures

A command procedure is a file that contains DCL commands and data lines used by the DCL commands. You can write both simple and complex command procedures. A simple command procedure executes a series of DCL commands in the order in which they are written. A complex command procedure performs program-like functions.

6.1 Format

Follow these instructions when formatting a command procedure:

- Begin each line containing a command, comment, or label with a dollar sign.
- Do not begin data lines with a dollar sign.
- Use comments to explain the command procedure to anyone who must maintain it. Place comments at the beginning of a procedure to describe the procedure and the parameters passed to it; place them at the beginning of each block of commands to describe that section of the procedure. The command interpreter ignores comments when the command procedure executes. Precede a comment with an exclamation point; the comment is all text to the right of an exclamation point. (To include a literal exclamation point in a command line, precede and follow it with quotation marks.)
- Use complete names for commands and qualifiers. Commands and qualifiers are usually self-explanatory when they are not abbreviated. Abbreviated commands and qualifiers may no longer be unique when new commands and qualifiers are added to the VMS operating system.
- Put labels on separate lines to make loops, subroutines, and conditional code easier to understand. (Labels mark the beginning of loops, subroutines, and conditional code.) You may choose to differentiate labels from commands by placing labels immediately after the dollar sign and preceding commands by a space. A label can have up to 255 characters, cannot contain embedded blanks, and must be terminated by a colon. (The GOTO, GOSUB, and CALL commands transfer control to labels, which mark the beginning of a loop, a section of code, or a subroutine.)
- Separate command sequences with lines containing a dollar sign and an exclamation point ($!). This makes it easier to see the outline of the command procedure. (If you insert blank lines, the command interpreter interprets them as data lines and produces a message warning you that the data lines were ignored. If you insert lines containing only a dollar sign, the command interpreter searches the whole line for a command.)

6.2 Execution

Command procedures can be executed interactively from DCL level, from within another command procedure, on a remote node using the TYPE command, or in batch mode. To execute a command procedure interactively, type an at sign (@) followed by the file specification of the procedure. The file type defaults to COM. The following command executes the procedure SETD.COM in the directory [MAINT.PROCEDURES] on the disk WORKDISK:

```
$ @WORKDISK:[MAINT.PROCEDURES]SETD
```

To simplify the invocation of a procedure, create a global symbol or a logical name, and place the symbol or logical name in your login command procedure. (Section 6.3 describes how to create a login command procedure. Symbols are described in Chapter 5. See Chapter 4 for more information about logical names.) Equating the command line to a global symbol allows you to invoke the command procedure from any directory by entering the global symbol name as follows:

```
$ SETD == "@WORKDISK:[MAINT.PROCEDURES]SETD"
$ SETD
```

Equating the file specification to a logical name allows you to invoke the command procedure from any directory by entering an at sign followed by the logical name as follows:

```
$ DEFINE SETD WORKDISK:[MAINT.PROCEDURES]SETD.COM
$ @SETD
```

To invoke a command procedure from within another command procedure, use the at sign (@) followed by the file specification of the procedure. In the following example, the command procedure WRITEDATE.COM invokes the command procedure GETDATE.COM:

```
$! WRITEDATE.COM
$ INQUIRE TIME "What is the current time, in hh:mm format?"
$ @GETDATE
```

Use the TYPE command to execute a command procedure interactively on a remote node. The TYPE command lets you execute command procedures to list the users logged on to the remote node or to display the status of services in the local cluster not provided clusterwide. The output of the command procedure is displayed on the user's terminal at the local node.

To execute a command procedure in the default DECnet account of the remote node, specify the command procedure as a parameter to the TYPE command as follows:

```
$ TYPE node_name::"TASK=command_procedure"
```

The variable *node_name* is the name of the remote node on which the command procedure resides; *command_procedure* is the name of the command procedure.

To execute a command procedure in the top level directory of another account on the remote node, use an access control string in the command as follows:

```
$ TYPE node_name"user_name password"::"TASK=command_procedure"
```

The variable *user_name* is the user name of the account on the remote node, *password* is the password of the account on the remote node, and *command_procedure* is the name of the command procedure.

The following command procedure, SHOWUSERS.COM, displays the users logged in at the remote node on which the command procedure resides:

```
$! SHOWUSERS.COM
$ IF F$MODE() .EQS. "NETWORK" THEN DEFINE/USER SYS$OUTPUT SYS$NET
$ SHOW USERS
```

The following command executes the command procedure SHOWUSERS.COM and displays the output from this command procedure on the user's terminal. The command procedure resides in the top level directory of account BIRD on node ORIOLE.

```
$ TYPE ORIOLE"BIRD FLIESFAST"::"TASK=SHOWUSERS"

          VAX/VMS Interactive Users
           09-DEC-1988 17:20:13.30
    Total number of interactive users = 4
 Username     Process Name      PID     Terminal
 FLICKER      Freddie         00536278  TXA1:
 ROBIN        Red             00892674  VTA2:
 DOVE         Whitie          00847326  TXA3:
 DUCK         Donna           02643859  RTA1:
```

You can also submit a command procedure to a batch queue to execute as a batch job. If your system is part of a network, you can submit a command procedure to execute as a batch job on a remote node. Within a command procedure, you can use DCL commands to open and close files on a remote node and read and write records in these files, using the same commands and qualifiers as for local files. Section 3.1.4 contains more information about batch jobs.

6.2.1 Changing Command Levels

A command level is an input stream for the DCL command interpreter. You can create a maximum of 32 command levels. There are two ways to create new command levels. You can either use the CALL command to call a subroutine that exists within the command procedure, or you can nest command procedures by using an execute procedure (@) command inside one command procedure to invoke another command procedure. When you use the CALL command or nest a command procedure, the command level increases by 1.

When you invoke a command procedure, the command level increases by 1. For example, if you invoke procedure SUB from DCL command level (level 0), SUB executes at command level 1. If SUB then invokes SUB1, which invokes SUBSUB1, SUB1 executes at command level 2, and SUBSUB1 executes at command level 3.

By convention, DCL level (command level 0) is the highest command level and command level 31 the lowest command level. Thus, when you move from command level 3 to command level 2, you are moving to the next higher command level.

6.2.2 Exiting from Command Procedures

A command procedure exits when it reaches the end of the procedure, an EXIT command, or a STOP command. If the exit is caused by the end of the procedure or an EXIT command, control returns to the next higher command level. If the exit is caused by the EXIT command, you can return a status value to the next higher command level by specifying the value as the parameter of the EXIT command. (A status value is a hexadecimal representation of a VMS message code.) This status value is placed in the global symbol $STATUS. If you return the status value 44 with the EXIT command, control returns to DCL command level and the following error message is displayed:

```
%SYSTEM-F-ABORT, abort
```

See Section 5.2 for more information about the global symbols $STATUS and $SEVERITY. For example, if you invoke SUB at DCL command level, and SUB calls SUB1, the following sequence of actions occurs:

1 Exiting from SUB1 returns you to SUB at the command line following the call to SUB1.

2 Exiting from SUB returns you to DCL command level.

If the exit is caused by the STOP command, control always returns to DCL command level, regardless of the command level in which the STOP command executes.

6.3 Designing a Login Command Procedure

You can create a command procedure, called a login command procedure, to execute the same commands each time you log in. Name your login command procedure LOGIN.COM, and place it in your top level directory, unless your system manager tells you otherwise.

The following sample LOGIN.COM procedure illustrates some commands you may want to include in your login command procedure:

```
$! Sample LOGIN.COM for user MARCIA with
$!  default disk of DISK3
$!
$! Exit if this is a batch job or another
$! type of noninteractive process
$!
$ IF F$MODE() .NES. "INTERACTIVE" THEN EXIT  ❶
$!
$! Tailor the default behavior of
$! certain DCL commands
$!
$ PUR*GE :== PURGE/LOG
$ SUB*MIT :== SUBMIT/NOLOG_FILE/NOTIFY
$ M*AIL :== MAIL/EDIT=(SEND,FORWARD,REPLY)
$!
$! Define global symbols
$!
$ DISPLAY :== MONITOR PROCESSES/TOPCPU
$ GO :== SET DEFAULT
$ LP :== SHOW QUEUE/ALL SYS$PRINT
$ SS :== SHOW SYMBOL
$ SQ :== SHOW QUEUE/ALL
$ REM :== @DISK3:[MARCIA.PROG]REMINDER
$ MAIN :== SET DEFAULT DISK3:[MARCIA]
$!
$! Define logical names for:
$!   Directories
$ DEFINE HOME DISK3:[MARCIA]
$ DEFINE REV DISK3:[MARCIA.REVIEWS]
$ DEFINE TOOLS DISK3:[MARCIA.TOOLS]
$!   Files
$ DEFINE EQUIP DISK3:[MARCIA.LISTS]EQUIPMENT.DAT
$ DEFINE ACCOMP DISK3:[MARCIA]ACCOMPLISHMENTS.DAT
$!   Users
$ DEFINE JON DAISY::HARRIS
$ DEFINE JANE DAISY::MOORE
$!
$! Define keys to execute commands
$!
$ DEFINE/KEY PF3 "SHOW USERS" /TERMINATE
$ DEFINE/KEY KP7 "SPAWN" /TERMINATE
$ DEFINE/KEY KP8 "ATTACH "
$ DEFINE/KEY KP4 "SET HOST "
$!
$! Change the prompt string to a three-character
$! abbreviation of the node name
$!
$ NODE = F$GETSYI("NODENAME")  ❷
$ PROMPT = F$EXTRACT(0,3,NODE)
$ SET PROMPT = "''PROMPT'> "
$!
$! Type the system notices   ❸
$!
$ TYPE SYS$SYSTEM:NOTICE.TXT
$!
$! Run a program that displays today's appointments   ❹
$!
$ RUN DISK3:[MARCIA.PROG]REMINDER
```

❶ The F$MODE lexical function returns the mode (interactive, batch, network or other) that the process is in when the LOGIN.COM procedure is executing. This statement causes the procedure to exit unless you are using the system interactively. You should test the mode at the beginning of your LOGIN.COM procedure to ensure that commands used only in interactive mode are not executed in any other mode; in some cases these commands can abort noninteractive processes.

❷ This group of commands changes the DCL prompt to the first three characters of the node name. The F$GETSYI lexical function determines the node name. The F$EXTRACT lexical function extracts the first three characters of the name. The SET PROMPT command changes the prompt from a dollar sign to the first three characters of the node name followed by the right-angle bracket character (>) and a space.

❸ This command displays the system notices that your system manager keeps in the file SYS$SYSTEM:NOTICE.TXT.

❹ This command runs a user-written program that displays your daily appointments. If you have written programs that you always run after you log in, you may prefer to execute them directly from your LOGIN.COM file.

The system manager assigns the file specification for your login command procedure in the LGICMD field for your account. In most installations, the login command procedure is called LOGIN.COM. However, if you want to execute a file other than the one named in the LGICMD field for your account, use the /COMMAND qualifier when you log in.

6.4 Passing Data

Command procedures frequently require data provided by a user. To specify the same data each time the command procedure is executed, place the data on data lines following the command that requires the data. (A data line is a line that does not begin with a dollar sign. To include a data line that begins with a dollar sign, use the DCL commands DECK and EOD, which are described in the *VMS DCL Dictionary*.) The following command procedure executes the image CENSUS.EXE, which reads the data 1981, 1982, and 1983 each time the procedure is executed:

```
$ ! CENSUS.COM
$ !
$ RUN CENSUS
1981
1982
1983
$ EXIT
```

The text on a data line is passed directly to the image; DCL does not process data lines. Therefore, DCL does not translate symbols or evaluate arithmetic expressions on data lines before passing the symbols or arithmetic expressions to the image. Logical names are not translated by DCL; therefore, a logical name included on a data line is translated before it is passed to an image.

To specify different data each time a command procedure executes, use one of the following mechanisms, which are described in Sections 6.4.1 through 6.4.4:

- Pass the data as one or more parameter values.
- Use the INQUIRE or READ command within the command procedure to prompt for data.
- Specify a device or file from which to read the data by redefining the logical name SYS$INPUT.

6.4.1 Using Parameters to Pass Data

When you invoke a command procedure, you can pass it up to eight parameters. Place the parameters after the file specification of the command procedure. Separate the parameters with one or more spaces or tabs. For example, the following command invokes SUM.COM and passes eight parameters to the procedure:

```
$ @SUM 34 52 664 89 2 7 87 3
```

To pass parameters to a command procedure executed in batch mode, use the /PARAMETERS qualifier of the SUBMIT command. If you pass more than one parameter, place the parameters in parentheses and separate them with commas. If you execute more than one command procedure using a single SUBMIT command, the specified parameters are used for each command procedure in the batch job. The following command passes three parameters to the command procedures ASK.COM and GO.COM, which are executed as batch jobs:

```
$ SUBMIT/PARAMETERS=(TODAY,TOMORROW,YESTERDAY) ASK.COM, GO.COM
```

DCL places parameters passed to a command procedure in the local symbols P1 through P8; P1 is assigned the first parameter value; P2 the second; P3 the third, and so on. If you pass more than eight values, you receive the following error message and the command procedure does not execute:

```
%DCL-W-DEFOVF, too many command procedure parameters - limit to eight
```

If you pass fewer than eight values, the extra symbols are assigned null values.

Specify a parameter value as one of the following:

- Integer—When you specify an integer, it is converted to a string as follows:

  ```
  $ @ADDER 24 25
  ```

  In this example, P1 is the string value 24; P2 is the string value 25. (You can, however, use the symbols P1 and P2 in both integer and character string expressions; DCL performs the necessary conversions automatically.)

- String—Specify character strings as follows:

  ```
  $ @DATA Paul Cramer
  ```

  In this example, the strings Paul and Cramer are converted to uppercase letters; P1 is PAUL and P2 is CRAMER.

  To preserve spaces, tabs, or lowercase characters, place quotation marks before and after the string as follows:

  ```
  $ @DATA "Paul Cramer"
  ```

  In this example, P1 is Paul Cramer and P2 is null.

- Symbol—To pass the value of a symbol, place an apostrophe character before and after the symbol, as shown in the following example. When passing a symbol, DCL removes quotation marks that enclose a string. (To preserve spaces, tabs, and lowercase characters in a symbol value, surround the symbol with quotation marks.)

  ```
  $ NAME = "Paul Cramer"
  $ @DATA 'NAME'
  ```

  In this example, P1 is Paul and P2 is Cramer.

  To include a quotation mark as part of a string, enter three quotation marks as follows:

  ```
  $ NEW_NAME = """Paul Cramer"""
  $ @DATA 'NEW_NAME'
  ```

  In this example, P1 is "Paul Cramer" and P2 is null.

- Null—To pass a null parameter, use a set of quotation marks as a placeholder in the command string. In the following example, the first parameter passed to DATA.COM is a null parameter:

  ```
  $ @DATA "" "Paul Cramer"
  ```

In the preceding example, P1 is null, and P2 is Paul Cramer.

For example, when DATA.COM is invoked with the following command, P1 through P8 are defined in DATA.COM as follows:

P1 = Paul Cramer
P2 = 24
P3 = (555) 111-1111
P4–P8 = null

```
$ @DATA  "Paul Cramer" 24 "(555) 111-1111"
```

You can pass up to eight parameters to a nested command procedure. The local symbols P1 through P8 in the nested procedure are not related to the local symbols P1 through P8 in the invoking procedure. In the following example, DATA.COM invokes the nested command procedure NAME.COM:

```
$ ! DATA.COM
$ @NAME 'P1' Joe Cooper
```

Because P1 in DATA.COM is the string Paul Cramer, which contains no quotation marks, it is passed to NAME.COM as two parameters. In NAME.COM, P1 through P8 are defined as follows:

P1 = PAUL
P2 = CRAMER
P3 = JOE
P4 = COOPER
P5–P8 = null

Because DCL removes quotation marks when passing a symbol, you must enclose the value in three sets of quotation marks to preserve spaces, tabs, and lowercase characters in the symbol value. In the following example, the literal value in P1 is enclosed in three sets of quotation marks and passed to NAME.COM. If P1 originally contained the value "Paul Cramer", the value "Paul Cramer" is passed to NAME.COM.

```
$ ! DATA.COM
$ QUOTE = """
$ P1 = QUOTE + P1 + QUOTE
$ @NAME 'P1' "Joe Cooper"
```

In this example, P1 is Paul Cramer and P2 is Joe Cooper in the command procedure NAME.COM.

An alternative is to enclose the text in quotation marks and, where a symbol appears, precede the symbol with two apostrophes and follow it with one apostrophe as follows:

```
$ ! DATA.COM
$ @NAME "''P1'"
```

Passing Data and Parameters to a Batch Job

To specify parameters for a job submitted in batch mode, use the /PARAMETERS qualifier of the SUBMIT command. Note that you can also pass data to a batch job by including the data in a command procedure or by defining SYS$INPUT to be a file. The specified parameters are used for each command procedure in the batch job. The following SUBMIT command passes two parameters to the command procedures LIBRARY.COM and SORT.COM:

```
$ SUBMIT-
_$ /PARAMETERS=(DISK:[ACCOUNT.BILLS]DATA.DAT,DISK:[ACCOUNT]NAME.DAT) -
_$ LIBRARY.COM, SORT.COM
```

Your batch job executes as if you had logged in and executed each of the submitted command procedures. For example, the previous SUBMIT command executes a batch job that logs in under your account, executes your login command procedure, and then executes the following commands:

```
$ @LIBRARY DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT
$ @SORT DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT
```

6.4.2 The INQUIRE Command

You can use the INQUIRE command to obtain data for command procedures that you execute interactively. The INQUIRE command prompts for a value, reads the value from the terminal, and assigns it to a symbol. The response to the prompt is interpreted as a character string. By default, the response is converted to uppercase, multiple blanks and tabs are replaced by a single space, and leading and trailing spaces are removed. To preserve lowercase characters, multiple spaces, and tabs, enclose your response in quotation marks. The following command procedure writes the prompt *Filename:* and puts your response into the local symbol FILE:

```
$ INQUIRE FILE "Filename"
```

To suppress the colon and space automatically added to the end of the prompt, use the /NOPUNCTUATION qualifier. To make the symbol global instead of local, use the /GLOBAL qualifier. The following command procedure writes the prompt *Do you want to use defaults?* and puts the response into the global symbol DEFAULT:

```
$ INQUIRE/NOPUNCTUATION/GLOBAL DEFAULT-
_$ "Do you want to use defaults?"
```

When a command procedure is submitted as a batch job, the value for a symbol specified in an INQUIRE command is read from the data line following the INQUIRE command. If you do not include a data line, the symbol is assigned a null value.

6.4.3 The READ Command

You can use the READ command to obtain data for command procedures that you execute interactively. The READ command prompts for a value, reads the value from the source specified by the first parameter, and assigns it to the symbol named as the second parameter. If you do not specify a prompt, the READ command outputs *DATA:* as the default prompt. To specify a different prompt, use the /PROMPT qualifier. All characters typed on the terminal in response to the prompt are taken as an exact character string value (case, spaces, and tabs are preserved). The following command writes the prompt *Filename:*, reads the response from the source specified by the logical name SYS$COMMAND (by default, the terminal), and assigns the response to the symbol FILE:

```
$ READ/PROMPT="Filename: "  SYS$COMMAND FILE
```

6.4.4 Obtaining Data from SYS$INPUT

Commands, utilities, and other system images usually take their input from the source specified by the logical name SYS$INPUT. SYS$INPUT is a process-permanent logical name that the system defines automatically. You can specify SYS$INPUT as any one of the following:

- Data line—In a command procedure, the default value of SYS$INPUT is the data lines of the procedure. In the following command procedure, the image CENSUS.EXE uses the default value of SYS$INPUT to take input (1986, 1987, and 1988) from the data lines:

```
$ ! CENSUS.COM
$ !
$ ! Execute CENSUS
$ RUN CENSUS
1986
1987
1988
$
```

- Terminal—A command procedure can get input from a terminal by defining SYS$INPUT as the terminal. This allows you to perform interactive tasks from a command procedure. The following command procedure defines SYS$INPUT as SYS$COMMAND, which is, by default, the terminal. The command procedure then invokes the EDT editor, beginning an interactive editing session. (The /USER_MODE qualifier redefines SYS$INPUT for a single image; you should use this qualifier whenever you redefine a process-permanent logical name.)

```
$ ! EDIT.COM
$ !
$ ! Edit the file STATS.DAT
$ WRITE SYS$OUTPUT "Edit STATS.DAT:"
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$ EDIT STATS.DAT
```

- File—A command procedure can get input from a file by defining SYS$INPUT as a file. The following command procedure defines SYS$INPUT as the file YEARS.DAT, then invokes the program CENSUS. CENSUS reads its input from the file YEARS.DAT.

```
$ ! CENSUS.COM
$ !
$ ! Execute CENSUS
$ DEFINE/USER_MODE SYS$INPUT YEARS.DAT
$ RUN CENSUS
```

6.5 Returning Data

To return a value from a command procedure (either to a calling procedure or to DCL command level), you must assign the value to a global symbol. The global symbol can be read at any command level. Use comments to explain the use of any global symbols.

To create a global symbol, specify the value to be passed on the right side of a global assignment statement. In the following example, the command procedure DATA.COM invokes the command procedure NAME.COM, passing NAME.COM a full name. NAME.COM places the last name in the global symbol LAST_NAME. When NAME.COM completes, DCL continues executing DATA.COM, which reads the last name by specifying the global symbol LAST_NAME. (The command procedure NAME.COM would be in a separate file; it is indented here for clarity.)

```
$ @DATA "Paul Cramer"

$ ! DATA.COM
$ !
$ ! P1 is a full name
$ ! NAME.COM returns the last name in the
$ ! global symbol LAST_NAME
$ !
$ @NAME 'P1'
       $ ! NAME.COM
       $ ! P1 is a first name
       $ ! P2 is a last name
       $ ! return P2 in the global symbol LAST_NAME
       $ LAST_NAME == P2
       $ EXIT
$ ! write LAST_NAME to the terminal
$ WRITE SYS$OUTPUT "LAST_NAME = ''LAST_NAME'"

LAST_NAME = CRAMER
```

6.6 Displaying Data

Commands, utilities, and other system images normally write their output to the source specified by the logical name SYS$OUTPUT. By default, SYS$OUTPUT is equated to the terminal. However, you can redirect the output of a command procedure to a file by using the /OUTPUT qualifier. In the following example, output from the command procedure SETD.COM is written to the file RESULTS.TXT instead of to the terminal:

```
$ @SETD/OUTPUT=RESULTS.TXT
```

DCL commands that accept the /OUTPUT qualifier include: ACCOUNTING, CALL, DIRECTORY, HELP, LIBRARY, RUN (process), SPAWN, and TYPE.

6.6.1 Displaying Character Strings and Symbols

To display character strings and symbols on the terminal, use the WRITE command as follows:

- Character string—Enclose the text to be displayed in quotation marks. The following example displays the text: *Two files were written.*

  ```
  $ WRITE SYS$OUTPUT "Two files were written."
  ```

- Symbol value—The WRITE command automatically substitutes symbols and lexical functions. The following example displays the text *STAT1.DAT,* which is the translation of the symbol FILE:

  ```
  $ FILE =  "STAT1.DAT"
  $ WRITE SYS$OUTPUT FILE
  ```

- Combination of character strings and symbol values—Enclose the text to be displayed in quotation marks. Preface a symbol with two apostrophes, and follow it with one apostrophe. The following example displays the text: *STAT1.DAT and STAT2.DAT were written.* STAT1.DAT is the translation of the symbol AFILE; STAT2.DAT is the translation of the symbol BFILE.

  ```
  $ AFILE = "STAT1.DAT"
  $ BFILE = "STAT2.DAT"
  $ WRITE SYS$OUTPUT "''AFILE' and ''BFILE' were written."
  ```

  You can also use commas and quotation marks to display a combination of character strings and symbol values. The following example displays the same text as the previous example:

  ```
  $ AFILE = "STAT1.DAT"
  $ BFILE = "STAT2.DAT"
  $ WRITE SYS$OUTPUT AFILE, " and " ,BFILE, " were written."
  ```

6.6.2 Displaying Text

To display text that is more than one line long, use the TYPE command. TYPE writes data to SYS$OUTPUT (the terminal, by default). Using SYS$INPUT as the parameter causes TYPE to read the data from the command procedure. When the following command procedure is executed, the text on the data lines is displayed on the terminal:

```
$ ! CLEAN.COM
$ !
$ TYPE SYS$INPUT

This command procedure executes a command that allows you
to clean up a directory.

Please enter one of the following commands after the prompt:
  EXIT, DIRECTORY, TYPE, PURGE, DELETE, COPY

$ INQUIRE COMMAND "Command"
   .
   .
   .
```

6.6.3 Displaying Files

To display the contents of a file, use the TYPE command. The following example displays the file STAT1.DAT on the terminal:

```
$ TYPE DUAO:[HORACE]STAT1.DAT
```

6.7 Reading and Writing Files (File I/O)

To move data to and from files, use the OPEN, CLOSE, READ, and WRITE commands. The logical name you specify in the OPEN command is used to refer to the file in the WRITE, READ, and CLOSE commands.

6.7.1 Specifying Files in Batch Job Command Procedures

A batch job command procedure executes as if you had logged in and executed the command procedure interactively. Since your login default directory is not usually the default directory needed to access files mentioned in a command procedure, command procedures that will be executed in batch mode should use one of the following mechanisms to ensure that the correct files are accessed:

- Use complete file specifications—When specifying a file in a command procedure or passing a file to a command procedure, include the device and directory names as part of the file specification, as shown in the previous example.
- Use the SET DEFAULT command—Before accessing a file in a command procedure, use the SET DEFAULT command to specify the proper device and directory.

6.7.2 Writing to a File

To write data to a file, take the following steps:

1. Open the file—The OPEN command assigns to the logical name specified in the first parameter the file name specified in the second parameter.

   Use the /APPEND qualifier of the OPEN command to write data to the end of an existing file. If you use the /APPEND qualifier to open a nonexistent file, an error occurs and no file is opened.

   Use the /WRITE qualifier of the OPEN command to create a new file and to open this file for write access. If you use the /WRITE qualifier to open an existing file, a new version of that file is created.
2. Begin the write loop with a label—File I/O is always done in a loop unless you are writing or reading a single record.
3. Read the data to be written—Use the INQUIRE command or the READ command to read data into a symbol.
4. Test the data—Check the symbol containing the data. If the symbol is null (you pressed RETURN and entered no data on the line), you have reached the end of the data to be written to the file and should go to the end of the loop. Otherwise, continue.

5 Write the data to the file—Use the WRITE command to write the value of the symbol (one record) to the file.

6 Return to the beginning of the loop—You remain in the loop until there is no more data to be written to the file.

7 End the loop and close the file—The CLOSE command disassociates the file name from the logical name and closes the file. (Files opened by the OPEN command remain open until you log out unless you explicitly close them.)

The following command procedure writes data to the new file STAT.DAT. If a file of that name exists, a new version is created.

```
$ ! Write a file
$ ON ERROR THEN EXIT                     !EXIT if the command procedure
$                                        ! cannot open the file
$ OPEN/WRITE IN_FILE STAT.DAT            !Open the file
$ ON CONTROL_Y THEN GOTO END_WRITE       !Close the file if you abort
$                                        ! execution with a CTRL/Y
$ ON ERROR THEN GOTO END_WRITE           !Close the file if an error
$                                        ! occurs
$WRITE:                                  !Begin loop
$ INQUIRE STUFF "Input data"             !Get input
$ IF STUFF .EQS. "" THEN GOTO END_WRITE  !Test for end of file
$ WRITE IN_FILE STUFF                    !Write to the file
$ GOTO WRITE                             !Goto beginning
$END_WRITE:                              !End loop
$ !
$ CLOSE IN_FILE                          !Close the file
```

Note: **The logical name in the OPEN command must be unique. If the OPEN command does not work and your commands seem correct, change the logical name in the OPEN command. Use the SHOW LOGICAL command to display logical name definitions.**

If you want to create a file with a unique name, use the F$SEARCH lexical function to see whether the name is already in the directory. (See the lexical function descriptions in the DCL Commands section for more information about F$SEARCH.) The following command procedure prompts the user for a file name, then uses the F$SEARCH lexical function to search the default directory for the name. If a file with that name already exists, control is passed to ERROR_1, the procedure prints the message *File already exists,* and control returns to the label GET_NAME. You are again prompted for a file name.

```
$ ! FILES.COM
$ !
$GET_NAME:
$ INQUIRE FILE "File"         ! Get a file name
$ CHECK = F$SEARCH (FILE)     ! Make sure the file name is unique
$ IF CHECK .NES. "" THEN GOTO ERROR_1
$ OPEN/WRITE IN_FILE 'FILE'   ! Open and write to the file
   .
   .
   .
$ EXIT
$ERROR_1:
$ WRITE SYS$OUTPUT "File already exists"
$ GOTO GET_NAME
```

6.7.3 Reading from a File

To read data from a file, take the following steps:

1 Open the file—The OPEN/READ command opens the file for read access and associates the file name with a logical name.

2 Begin the read loop—File I/O is always done in a loop unless you are reading or writing a single record.

3 Read the data from the file—Use the READ command with the /END_OF_FILE qualifier to read the next record in the file to a symbol. The /END_OF_FILE qualifier causes the VMS system to pass control to the label specified by the /END_OF_FILE qualifier when you reach the end of the file. Generally, you specify the label that marks the end of the read loop.

4 Process the data—When you read a file sequentially, process the current record before reading the next one.

5 Return to the beginning of the loop—You remain in the loop until you reach the end of the file.

6 End the loop and close the file—The CLOSE command disassociates the file name from the logical name and closes the file.

The following command procedure reads and processes each record in the file STAT.DAT:

```
$ OPEN/READ OUT_F STAT.DAT              !Open the file
$ !
$READ_DATA:                             !Begin the loop
$ READ/END_OF_FILE=END_READ OUT_F STUFF !Read a record; test for
$                                       !  end of file
$                                       ! Process the data
   .
   .
   .
$ GOTO READ_DATA                        !Go to the beginning
$                                       ! of the loop
$END_READ:                              !End of loop
$ !
$ CLOSE OUT_F                           !Close the file
```

6.7.4 Modifying a File

You can modify a file in the following ways:

- Rewrite records—This method allows you to make minor changes to a small number of records in a file. You cannot change the size of a record or the number of records in the file.
- Rewrite the file—This method allows you to change, delete, and insert records. You create a new file using the old file as the main source of input.
- Append records to a file—This method allows you to add new records to the end of the file.

6.7.4.1 Minor Modifications

To make minor changes to the records in a file, take the following steps:

1 Open the file for both read and write access.

2 Use the READ command to read through the file until you reach the record that you want to modify.

3 Create a symbol containing the modified record. The modified record must be exactly the same size as the original record. If the text of the modified record is shorter, pad the record with spaces. If the text of the modified record is longer, you cannot use this method to modify the file.

4 Use the WRITE/UPDATE command to write the modified record back to the file.

5 Repeat steps 2 through 4 until you have changed all records you intend to change.

6 Close the file.

Since this method does not allow you to modify the size of the record, use it only if you have formatted the records in a file (for example, in a data file).

The following command procedure reads each record in a data file. The record is displayed on the terminal, and you are asked whether the record is to be modified. If you choose to modify the record, a new record is read from the terminal, and its length is compared to the length of the original record. If the original record is longer, the new record is padded with spaces. If the original record is shorter, an error message is displayed, and you are again prompted for a new record. If you choose not to modify the record, the next record is read from the file.

```
$ ! MODIFY.COM
$ !
$ SPACES = "          "          ! Initialize string of spaces
$                                !  for padding
$ !
$ OPEN/READ/WRITE FILE STATS.DAT ! Open the file
$ !
$BEGIN_LOOP:                     ! Begin the loop
$ !
$ READ/END_OF_FILE=END_LOOP FILE RECORD ! Read and display a record
$PROMPT:
$ WRITE SYS$OUTPUT RECORD
$ !
$ ! Does the user want to change the record?
$ INQUIRE/NOPUNCTUATION YN "Change? [Y] "
$ IF YN .EQS. "N" THEN GOTO BEGIN_LOOP  ! If not, get next record
$ INQUIRE NEW_RECORD "New record"       ! Otherwise, get the new record
$ !
$ OLD_LEN = F$LENGTH (RECORD)          ! Compare the old and new records
$ IF OLD_LEN .GE. F$LENGTH(NEW_RECORD) THEN GOTO NO_ERROR
$  ! New record longer than old record
$  WRITE SYS$OUTPUT "ERROR -- New record is too long"
$  GOTO PROMPT
$ !
$NO_ERROR:
$ IF OLD_LEN .EQ. F$LENGTH(NEW_RECORD) THEN GOTO WRITE_RECORD
$ ! New record shorter than old record
$ PAD = F$EXTRACT(0,OLD_LEN-F$LENGTH(NEW_RECORD),SPACES)
$ NEW_RECORD = NEW_RECORD + PAD
$ !
$WRITE_RECORD:                         ! Write the new record
$ WRITE/UPDATE FILE NEW_RECORD
$ GOTO BEGIN_LOOP
$ !
$END_LOOP:
$ CLOSE FILE
$ EXIT
```

6.7.4.2 Major Modifications

To make extensive changes to a file, open that file for read access and open a new file for write access. Since the /WRITE qualifier opens a new file for write access, the new file can have the same name as the original file. The new file has a version number one greater than the version number of the old file.

Note: **You must open the existing file for read access before you open the new version for write access to ensure that the correct file is opened for reading.**

To make major modifications to a file, take the following steps:

1 Open the file for read access. This is the file you are modifying.

2 Open a new file for write access.

3 Use the READ command to read each record from the file you are modifying.

   As you read each record from the original file, decide how the record is to be treated. In the following examples, the symbol RECORD contains the record read from the original file:

   - No change—Write the same symbol to the new file.

     ```
     $ ! No change
     $ WRITE NEW_FILE RECORD
     ```

   - Change—Use the INQUIRE command to read a different record into the symbol, then write the modified symbol to the new file.

     ```
     $ ! Change
     $ INQUIRE NEW_RECORD "New record"
     $ WRITE NEW_FILE NEW_RECORD
     ```

   - Delete—Do not write the symbol to the new file.

   - Insert—Use a loop to read records into the symbol and to write the symbol to the new file, as shown in the following example:

```
$ ! Insertion
$LOOP:
$ !Get new records to insert
$ INQUIRE NEW_RECORD "New record"
$ IF RECORD .EQS. "" THEN GOTO END_LOOP
$ WRITE NEW_FILE NEW_RECORD
$ GOTO LOOP
$END_LOOP:
```

4 Continue reading and processing records until you have finished.

5 Use the CLOSE command to close both the input and the output files.

6.7.4.3 Appending Records to a File

The OPEN/APPEND command allows you to append records to the end of an existing file. Use the following steps to append records to a file:

1 Use the OPEN command with the /APPEND qualifier to position the record pointer at the end of the file. The /APPEND qualifier does not create a new version of the file.

2 Use the WRITE command to write new data records. Continue adding records until you are through.

3 Use the CLOSE command to close the file.

6.7.5 Handling Input/Output (I/O) Errors

Use the /ERROR qualifier with the OPEN, READ, or WRITE command to suppress error messages and to pass control to a specified label if an error occurs during an input or output operation. This qualifier overrides all other error-control mechanisms (except the /END_OF_FILE qualifier on the READ command). In the following command procedure, if an error occurs during execution of the OPEN command, the message *Error opening STAT.DAT* is printed and the procedure exits:

```
$ OPEN/READ/ERROR=READ_ERR OUT_F STAT.DAT
   .
   .
   .
$ EXIT
$READ_ERR:
$ WRITE SYS$OUTPUT "Error opening STAT.DAT"
$ EXIT
```

6.8 Complex Command Procedures

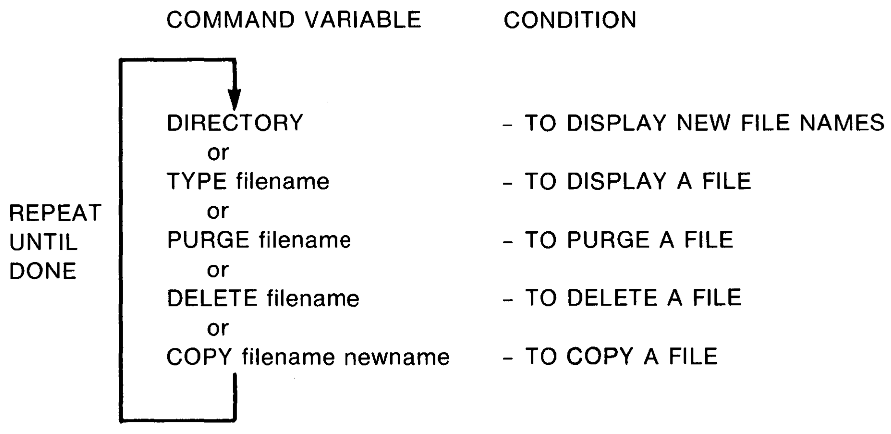
Complex command procedures perform programlike functions. You can use variable input in a complex command procedure, execute sections of the procedure only if certain conditions are true, execute subroutines, or invoke other command procedures. The following sections describe how to design, code, and test complex command procedures.

6.8.1 Designing Complex Command Procedures

Before writing a complex command procedure, perform the tasks interactively that the command procedure will execute. As you type the necessary commands, note the following:

- Variables—Data that changes each time you perform the task.
- Conditionals—Any command or set of commands that may vary each time you perform the task. Note the commands and the conditions under which you would execute them.
- Iteration—Any command or set of commands that you repeat. Note the commands and the factor that controls how often you repeat them.

The following example shows the commands needed to clean up a directory:



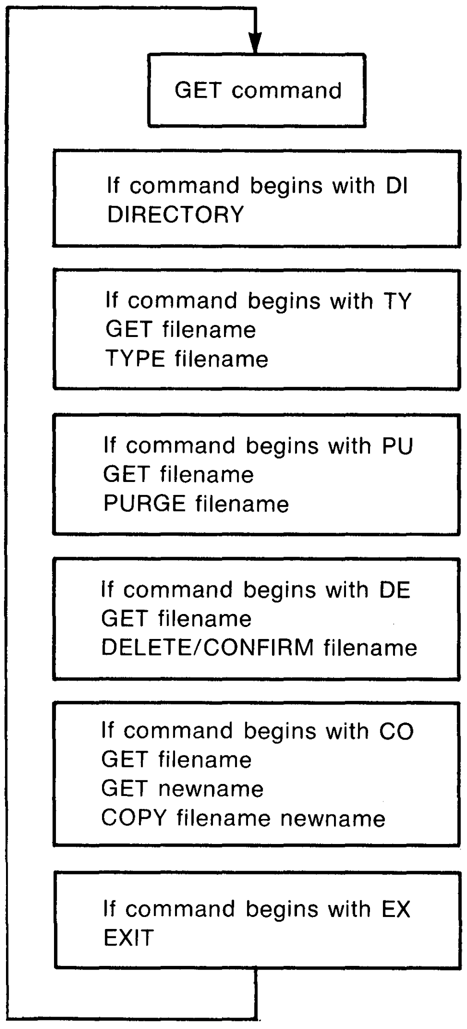
ZK-1750-84

The file names change each time you clean your directory; therefore, they are variables. Any or all of the commands may be executed depending on the operation you need to perform; therefore, each command is conditional. The entire process is repeated until the directory is clean; therefore, it is iterative.

You must decide how to load the variables, test the conditionals, and exit from the loop. For the directory cleaning procedure, the following design decisions were made:

- Load variables—The command procedure gets the file names from the terminal.
- Test conditionals—The command procedure gets a command name from the terminal and executes the appropriate statements based on the command name. The first two characters of each command must be read to differentiate between DELETE and DIRECTORY.
- Exit from loop—You must enter the EXIT command to exit from the loop.

Complete the design as follows:



ZK-1751-84

6.8.2 Coding Complex Command Procedures

To make the command procedure easier to understand and to maintain, try to write the statements so that the procedure executes in a linear fashion, from the first command to the last command. The following sections describe how to execute conditional code and loops. (See Section 5.6.2 for information about the logical operators used in condition expressions.)

6.8.2.1 The IF Command

The IF command tests the value of an expression and causes different commands to execute when the expression is true and when it is false. DCL provides two distinct formats for the IF command. The first format executes a single command when the condition specified to the IF command is true as follows:

```
$ IF condition THEN command
```

DCL also provides a block-structured IF format. The block-structured IF command executes more than one command if the condition is true and accepts an optional ELSE statement that executes one or more commands if the condition is false. To execute more than one command upon a true condition, specify the THEN statement as a verb (a DCL command preceded by a dollar sign) and terminate the resulting block-structured statement with an ENDIF statement as follows:

```
$ IF condition
$ THEN  command
$       command
    .
    .
    .
$ ENDIF
```

To execute one or more commands upon a false condition, specify the ELSE statement as a verb and terminate the resulting block-structured statement with an ENDIF statement as follows:

```
$ IF condition
$ THEN  command
$       command
    .
    .
    .
$ ELSE  command
$       command
    .
    .
    .
$ ENDIF
```

Command blocks can be executed in several ways, depending on whether you leave the commands in the same command procedure or put them in another command procedure and execute them there:

- If you leave the commands in the command procedure, place them after the THEN statement.

  ```
  $ IF condition
  $ THEN  command
          command
      .
      .
      .
  $ ENDIF
  ```

- If you place the commands in a separate procedure, make the call to that command procedure as part of the THEN statement.

  ```
  $ IF condition
  $ THEN @command_procedure
  $ ELSE command
  $      command
  $ ENDIF
  ```

You can continue to specify the nonblock structured IF format and direct flow to a labeled region when the condition specified is met as follows:

```
$ IF not condition THEN GOTO END_LABEL
     .
     .
     .
$END_LABEL:
```

In the following example, a specified file is purged if COMMAND equals "PU". If COMMAND does not equal "PU", a specified file is printed.

```
$! Purge a file.  If no file exists, print the requested file.
$ IF COMMAND .EQS. "PU"
$ THEN
$    INQUIRE FILESPEC "File to purge"
$    PURGE 'FILESPEC'
$ ELSE
$    INQUIRE FILESPEC "File to print"
$    PRINT 'FILESPEC'
$ ENDIF
$! Type a file.  If no file exists, exit"
$ IF COMMAND .EQS. "TY"
     .
     .
     .
$ EXIT
```

In the following example, the command procedure SCREEN_SETUP.COM is executed if F$MODE() equals "INTERACTIVE". If F$MODE() does not equal "INTERACTIVE", the procedure exits. (The command procedure SCREEN_SETUP.COM would be in a separate file; the commands it contains are indented here for clarity.)

```
$ IF F$MODE() .EQS. "INTERACTIVE"
$ THEN
$    @SCREEN_SETUP
         $ ! SCREEN_SETUP.COM
         $ ! Set terminal characteristics
         $ SET TERMINAL/DEVICE=VT200
         $ SET TERMINAL/WIDTH=132
$ ! Invoke Editor
$    EVE :== EDIT/TPU
$ ELSE
$    EXIT
$ ENDIF
     .
     .
     .
```

6.8.2.2 Case Statements

A case statement is a special form of conditional code that executes one out of a set of command blocks, depending on the value of a variable or expression. Typically, the valid values for the case statement are labels at the beginning of each command block. The case statement passes control to the appropriate block of code by using the specified value as the target label in a GOTO statement.

To write a case statement:

1 List the labels—Equate a symbol to a string that contains a list of the labels delimited by slashes (or any character you choose to act as a delimiter). This symbol definition should precede the command blocks.

```
$ COMMAND_LIST = "/PURGE/DELETE/EXIT/"
```

2 Write the "case statement"—First, use the INQUIRE command to get the value of the case variable. Next, use the IF command with F$LOCATE and F$LENGTH to determine whether the value of the case variable is valid. If the case variable is valid, execute the case statement (a GOTO command) to pass control to the appropriate block of code. Otherwise, display a message and exit or request a different case value.

In the following example, the label is equated to the full command name. Therefore, F$LOCATE includes the delimiters in its search for the command name to ensure that the command is not abbreviated.

```
$GET_COMMAND:
$ INQUIRE COMMAND -
  "Command (EXIT,PURGE,DELETE)"
$ IF F$LOCATE ("/"+COMMAND+"/",COMMAND_LIST) .EQ. -
   F$LENGTH (COMMAND_LIST) THEN GOTO ERROR_1
$ GOTO 'COMMAND'
   .
   .
   .
$ERROR_1:
$ WRITE SYS$OUTPUT "No such command as ''COMMAND'"
$ GOTO GET_COMMAND
```

3 Write the command blocks—Each block of commands may contain one or more commands. Begin each command block with a unique label. End each command block by passing control to a label outside the list of command blocks.

```
$GET_COMMAND:
   .
   .
   .
$PURGE:
$ INQUIRE FILE
$ PURGE 'FILE'
$ GOTO GET_COMMAND
$ !
$DELETE:
$ INQUIRE FILE
$ DELETE 'FILE'
$ GOTO GET_COMMAND
$ !
$EXIT:
```

6.8.2.3 Loops

A loop is a group of commands that executes repeatedly until a condition is met. The following arrangement is recommended for statements that form a loop:

1 Begin the loop.

2 Change the termination variable.

3 Test the termination variable. If the condition is met, go to the end of the loop.

4 Perform the commands in the body of the loop.

5 Return to the beginning of the loop.

6 End the loop.

You can also write loops that test the termination variable at the end of the loop rather than at the beginning as follows:

1 Begin the loop.

2 Perform the commands in the body of the loop.

3 Change the termination variable.

4 Test the termination variable. If the condition is not met, go to the beginning of the loop.

5 End the loop.

Note that when you test the termination variable at the end of the loop, the commands in the body of the loop execute at least once, regardless of the value in the termination variable.

Both of the following examples execute a loop that terminates when COMMAND equals "EX" (EXIT). (F$EXTRACT truncates COMMAND to its first two characters.) In the first example, COMMAND, the termination variable, is tested at the beginning of the loop; in the second, it is tested at the end of the loop.

```
$ ! EXAMPLE 1
$ !
$GET_COMMAND:
$ INQUIRE COMMAND-
  "Command (EXIT,DIRECTORY,TYPE,PURGE,DELETE,COPY)"
$ COMMAND = F$EXTRACT(0,2,COMMAND)
$ IF COMMAND .EQS. "EX" THEN GOTO END_LOOP
     .
     .
     .
$ GOTO GET_COMMAND
$END_LOOP:
```

```
$ ! EXAMPLE 2
$ !
$GET_COMMAND:
$ INQUIRE COMMAND-
  "Command (EXIT,DIRECTORY,TYPE,PURGE,DELETE,COPY)"
$ COMMAND = F$EXTRACT(0,2,COMMAND)
    .
    .
    .
$ IF COMMAND .NES. "EX" THEN GOTO GET_COMMAND
$ ! End of loop
```

To perform a loop a specific number of times, use a counter as the termination variable. In the following example, 10 file names are input by the user and placed into the local symbols FIL1, FIL2, ..., FIL10:

```
$ NUM = 1                        ! Set counter
$LOOP:                           ! Begin loop
$ INQUIRE FIL'NUM' "File"        ! Get file name
$ NUM = NUM + 1                  ! Update counter
$ IF NUM .LT. 11 THEN GOTO LOOP  ! Test for termination
$END_LOOP:                       ! End loop
    .
    .
    .
```

To perform a loop for a known sequence of values, use F$ELEMENT. In the following example, the files CHAP1, CHAP2, CHAP3, CHAPA, CHAPB, and CHAPC are processed in order.

```
$ FILE_LIST = "1,2,3,A,B,C"
$ INDEX = 0
$PROCESS:
$ NUM = F$ELEMENT(INDEX,",",FILE_LIST)
$ IF NUM .EQS. "," THEN GOTO END_LOOP
$ FILE = "CHAP''NUM'"
$ ! process file named by FILE
    .
    .
    .
$ INDEX = INDEX + 1
$ GOTO PROCESS
$END_LOOP:
$ EXIT
```

6.8.2.4 Subroutines

Use the GOSUB command or the CALL command to transfer control to a subroutine within a command procedure. The GOSUB command transfers control to a labeled subroutine in a command procedure without creating a new procedure level. Since the GOSUB command does not create a new command level, it is referred to as a *local* subroutine call. The RETURN command terminates the GOSUB subroutine procedure, returning control to the command following the calling GOSUB statement.

The following command procedure shows how to use the GOSUB command to transfer control to labeled subroutines:

```
$!
$! GOSUB.COM
$!
$ SHOW TIME
$ GOSUB TEST1
$ WRITE SYS$OUTPUT "success completion"
$ EXIT
$!
$! TEST1 GOSUB definition
$!
$ TEST1:
$     WRITE SYS$OUTPUT "This is GOSUB level 1."
$     GOSUB TEST2
$     RETURN
$!
$! TEST2 GOSUB definition
$!
$ TEST2:
$     WRITE SYS$OUTPUT "This is GOSUB level 2."
$     RETURN
```

The CALL command transfers control to a labeled subroutine in a command procedure and creates a new command level. The CALL command allows you to keep more than one related command procedure in a single file, making the procedures easier to manage. You can pass up to eight parameters to the subroutine; you can create up to 32 command levels with the CALL command.

By default, the CALL command sends output to SYS$OUTPUT. The optional /OUTPUT qualifier allows you to direct output from the subroutine to a file. Do not use wildcard characters in the output file specification. The default file type for the output file is LIS.

Unless they are masked using the SET SYMBOL command, local symbols defined in an outer level are available to any inner procedure or subroutine levels and global symbols are available at any command level. Labels are valid only for the level in which they are defined.

The SUBROUTINE and ENDSUBROUTINE commands define the beginning and end of a subroutine invoked with the CALL command. The label defining the entry point to the subroutine immediately precedes the SUBROUTINE command. The ENDSUBROUTINE command functions as an EXIT command if an EXIT command is not specified in the procedure. The ENDSUBROUTINE command terminates the subroutine and transfers control to the command line immediately following the CALL command.

Command lines in a CALL subroutine execute only when the subroutine is called with the CALL command. During the line-by-line execution of the command procedure, the command language interpreter skips all commands between the SUBROUTINE and the ENDSUBROUTINE commands.

The following procedure shows how to use CALL to transfer control to a labeled subroutine. The example also shows that you can call another command procedure from within a subroutine. (You can also call another subroutine from a subroutine.) The CALL command invokes the subroutine SUB1, directing output to the file NAMES.LOG and allowing other users write access to the file.

```
$!
$! CALL.COM
$!
$! Define subroutine SUB1
$!
$ SUB1: SUBROUTINE
    .
    .
    .
$ @FILE           !Invoke another command procedure
    .
    .
    .
$ EXIT
$ ENDSUBROUTINE   !End of SUB1 definition
$!
$! Start of main routine. At this point, SUB1 has
$! been defined, but none of the commands in the
$! subroutine have executed.
$!
$ START:
$  CALL/OUTPUT=NAMES.LOG SUB1 "THIS IS P1"
    .
    .
    .
$  EXIT !Exit this command procedure file
```

6.8.3 Testing and Debugging

For lengthy or complex command procedures, write the logic for the main procedure, but use stubs for the nested procedures and subroutine-type pieces of code. A stub is a command that writes a message stating the function it is replacing. For example, the following stub replaces the purge routine:

```
    .
    .
    .
$ ! Purge a file
$ IF COMMAND .NES. "PU" THEN GOTO END_PURGE
$ WRITE SYS$OUTPUT "Purge routine"   ! stub
$END_PURGE:
    .
    .
    .
```

If you have a number of places that need stubs, you can use one nested command procedure to insert the stub logic as follows. (The command procedure STUB.COM would be in a separate file; it is indented here for clarity.)

```
    .
    .
    .
$ ! Purge a file
$ IF COMMAND .NES. "PU" THEN GOTO END_PURGE
$ @STUB "Purge"
$        ! STUB.COM
$        ! Procedure STUB
$         WRITE SYS$OUTPUT "''P1' routine"
$END_PURGE:
    .
    .
    .
```

Once you have written the code using stubs, you can test the overall logic of the command procedure as follows. Test all possible paths of execution.

```
$ ! CLEAN.COM
$ !
$GET_COMMAND:
$ ! Read a command from the terminal
$ INQUIRE COMMAND-
  "Command (EXIT, DIRECTORY, TYPE, PURGE, DELETE, COPY)"
$ COMMAND = F$EXTRACT(0,2,COMMAND)
$ !
$ IF COMMAND .EQS. "EX" THEN GOTO END_COMMAND
$ !
$ ! Purge a file
$ IF COMMAND .NES. "PU" THEN GOTO END_PURGE
$ WRITE SYS$OUTPUT "Purge routine."
$END_PURGE:
$ !
$ ! Delete a file
$ IF COMMAND .NES. "DE" THEN GOTO END_COMMAND
$ WRITE SYS$OUTPUT "Delete routine."
$ END_DELETE:
    .
    .
    .
$ GOTO GET_COMMAND
$END_COMMAND:
```

Once the overall logic of the procedure works, you can begin filling in the stubs. Fill in the first stub, test it, and debug it if necessary. When that stub works, move on to the next one.

The following commands are useful for debugging command procedures:

- SET VERIFY—SET VERIFY prints each line before it is executed. When an error occurs with verification set, you see the error and the line that generated the error. In the following example, seeing the command line that generated the error explains the error message.

```
$ SET VERIFY
$ @CDIR
$ ! Read a command from the terminal
$ INQUIRE COMMAND-
  "Command (EXIT, DIRECTORY, TYPE, PURGE, DELETE, COPY)"
Command (EXIT, DIRECTORY, TYPE, PURGE, DELETE, COPY):  DELETE
$ COMMAND = F$EXTRACT(0,2,COMMAND)
$GET_COMMAND:
$ IF COMMAND .EQ. "EX" THEN GOTO END_COMMAND
%DCL-W-IVCHAR, non-numeric character in value string
 \E\ EXIT
$ IF COMMAND .NES. "DI" THEN GOTO END_DIR
    .
    .
    .
```

The logical operator .EQ. is used to compare numbers, not strings (see Section 5.6.2 for information about logical operators). To correct the error, change .EQ. to .EQS.. Note that you can use keywords with SET VERIFY to indicate that only command lines or data lines are to be verified.

- SHOW SYMBOL—Use the SHOW SYMBOL command to print the values of the symbols involved in an error. In the following procedure, the IF statements are not passing control to the expected procedures. Putting the command SHOW SYMBOL COMMAND before the IF statements allows you to check the value of COMMAND.

```
$ SET VERIFY
$ @CDIR
$GET_COMMAND:
$ ! Read a command from the terminal
$ INQUIRE COMMAND-
  "Command (EXIT, DIRECTORY, TYPE, PURGE, DELETE, COPY)"
Command (EXIT, DIRECTORY, TYPE, PURGE, DELETE, COPY):  DELETE
$ COMMAND = F$EXTRACT(1,2,COMMAND)
$ SHOW SYMBOL COMMAND
  COMMAND = "EL"
$GET_COMMAND:
$ IF COMMAND .EQS. "EX" THEN GOTO END_COMMAND
    .
    .
    .
```

The F$EXTRACT lexical function is extracting two characters beginning at character 1 (the second character) rather than at character 0 (the first character). To correct the error, change F$EXTRACT(1,2,COMMAND) to F$EXTRACT(0,2,COMMAND). Note that INQUIRE automatically converts input to uppercase; therefore, the quoted string in the IF statement must be written in uppercase for DCL to evaluate the strings as equal.

6.9 Handling Errors and CTRL/Y Interrupts

The following table describes the default action taken when an error occurs or when you press CTRL/Y. These default actions can be overridden with the ON, SET [NO]ON, and SET [NO]CONTROL=Y commands.

Interrupt	Default Action
Error or severe error	Procedure exits to the next command level.
CTRL/Y at DCL command level or command level 1	Interrupts procedure: procedure can continue if no other image forces it to exit.
CTRL/Y at command level lower than level 1	Procedure exits to the next higher command level.

6.9.1 The ON Command

The ON command specifies an action to be performed if an error of a certain severity or greater severity occurs. (See Section 5.2 for discussion of error conditions and severity levels.) When such an error occurs, the system takes the following actions:

1. The error message is displayed.
2. The action specified by the ON command is performed.
3. The default error action (exit to the next higher command level) is reset.

When an error of less than the specified severity occurs, the error message is displayed, and the command procedure continues executing. Assume a command procedure executes the following command:

```
$ ON ERROR THEN GOTO ERR1
```

The command procedure continues to execute unless an error or severe error occurs. When such an error occurs, the error message is displayed; the default error action (exit to the next higher command level) is reset; and the command procedure continues execution at ERR1. If a second error occurs before another ON or SET NOON command is executed, the procedure exits to the next higher level.

The action specified by the ON command applies only within the command level in which the command is executed. Therefore, if you execute an ON command in a procedure that calls another procedure, the ON command action does not apply to the nested command procedure.

The execution of an ON statement performs an implicit SET ON function, thus nullifying any SET NOON condition that may be in effect.

Note: **Only one ON statement of each type can be in effect at any one time. For example, if the command procedure includes more than one ON ERROR statement, the ON ERROR statement executed most recently is in effect.**

6.9.2 The SET [NO]ON Command

The SET ON and SET NOON commands enable and disable error checking for the current command level. The SET NOON command overrides the ON command. If an error (regardless of severity) occurs after a SET NOON command is executed, the system takes the following actions:

1 Displays the error message

2 Continues executing the procedure

SET NOON remains in effect until either an ON or SET ON command is executed. The SET ON command reenables error checking for the current command level. The action specified by the last ON command, if one exists, is reestablished. Otherwise, the default error action is reenabled.

In the following command procedure, if an error or severe error occurs while copying the file, the procedure continues to execute without going to GET_COMMAND as specified by the ON command.

```
$ ON ERROR THEN GOTO GET_COMMAND
$GET_COMMAND:
     .
     .
     .
$ ! Type a file
$ IF COMMAND .NES. "CO" THEN GOTO END_COPY
$ INQUIRE FILESPEC "File to move"
$ INQUIRE COPYSPEC "New file specification"
$ SET NOON
$ COPY 'FILESPEC' 'COPYSPEC'
$ SET ON
$END_COPY:
     .
     .
     .
```

6.9.3 CTRL/Y Interrupts

The ON CONTROL_Y command specifies an action to be performed when CTRL/Y is pressed at the current command level. (By default, pressing CTRL/Y causes the system to prompt for command input at the CTRL/Y command level.) In the following command procedure, pressing CTRL/Y while a file is being typed passes control to the label END_TYPE.

```
     .
     .
     .
$ ! Type a file
$ IF COMMAND .NES. "TY" THEN GOTO END_TYPE
$ ON CONTROL_Y THEN GOTO END_TYPE
$ TYPE 'FILESPEC'
$END_TYPE:
$ !
$ !Reset default
$ SET NOCONTROL=Y
$ SET CONTROL=Y
     .
     .
     .
```

An ON CONTROL_Y command remains in effect until another ON CONTROL_Y or a SET NOCONTROL=Y command executes or the command procedure exits.

See Section 6.11 for another example of using the ON CONTROL_Y command.

To exit from a nonterminating loop when CTRL/Y is disabled, you must delete your process from another terminal using the DCL command STOP. If you disable the default CTRL/Y action, reset it as soon as possible. To reset the default CTRL/Y action, execute the SET NOCONTROL=Y command followed by the SET CONTROL=Y command, as shown in the previous example.

6.10 Restarting Batch Jobs

Chapter 3 describes how to specify that your batch job be reexecuted if the system crashes before the job is finished. By default, a batch job is reexecuted beginning with the first line. However, you can use the following symbols in your command procedures to specify a different restarting point:

- $RESTART—A global symbol whose value is true if the batch job has been started at least once before this execution. Do not specify a value for $RESTART; the system will assign the appropriate value.
- BATCH$RESTART—A global symbol whose value you specify using the SET RESTART_VALUE command.

The following steps describe how to use these symbols in a command procedure:

- Begin each possible starting point of the procedure with a label.
- As the first step in each section, equate the value of BATCH$RESTART to the label using the SET RESTART_VALUE command.
- At the beginning of the procedure, test $RESTART. If $RESTART is true, issue a GOTO statement using BATCH$RESTART as the transfer label.

The following command procedure extracts a number of modules from a library, concatenates those modules, and then sorts the resulting file. If aborted, the command procedure reexecutes from the beginning of the file, the statement labeled CONCATENATE_LIBRARIES or the statement labeled SORT_FILE, depending on the value of BATCH$RESTART. (If you were extracting a number of separate modules, you could make each extraction a separate section.)

```
$ ! SORT_MODULES.COM
$ !
$ ! set default to the directory containing
$ ! the library whose modules are to be sorted
$ SET DEFAULT WORKDISK:[ACCOUNTS.DATA83]
$
$ ! check for restarting
$ IF $RESTART THEN GOTO BATCH$RESTART
$
```

```
$ EXTRACT_LIBRARIES:
$ SET RESTART_VALUE=EXTRACT_LIBRARIES
                .
                .
                .
$ CONCATENATE_LIBRARIES:
$ SET RESTART_VALUE=CONCATENATE_LIBRARIES
                .
                .
                .
$ SORT_FILE:
$ SET RESTART_VALUE=SORT_FILE
                .
                .
                .
$ EXIT
```

6.11 Cleanup Operations

In general, execution of a command procedure should not change the user's process state. Therefore, a command procedure should include a set of commands that returns the process to its original state. Common cleanup operations include the following (see the lexical function descriptions in the DCL Commands section for lexical function specifications):

- Closing files—If you have opened any files, make sure that they are closed before the command procedure exits. You can use the lexical function F$GETJPI to examine the remaining open file quota (FILCNT) for the process. If FILCNT is the same at the beginning and end of the command procedure, you know that no files have been left open. In the following example, a warning message is displayed if a file is left open:

  ```
  $ FIL_COUNT = F$GETJPI("","FILCNT")
      .
      .
      .
  $ IF FIL_COUNT .NE. F$GETJPI("","FILCNT") THEN-
     WRITE SYS$OUTPUT "WARNING -- file left open"
  ```

- Deleting temporary or extraneous files—If you have created temporary files, delete them. In general, if you have updated any files, you should purge them to delete the previous copies. Take care in deleting files that you have not created. For example, if you have updated a file that contains crucial data, you may wish to make the purging operation optional.

- Resetting default device and directory—If you change the default device and/or directory, reset the original defaults before the command procedure exits.

  To save the name of the original default directory, use the DEFAULT keyword of the F$ENVIRONMENT lexical function. At the end of the command procedure, include a SET DEFAULT command that restores the saved device and directory.

The following example saves and restores device and directory defaults:

```
$ SAV_DEFAULT = F$ENVIRONMENT("DEFAULT")
   .
   .
   .
$ SET DEFAULT 'SAV_DEFAULT'
```

The following table lists other commonly changed process characteristics as well as the lexical functions and commands used to save and restore the original settings:

Characteristic	To save...	To restore...
DCL prompt	F$ENVIRONMENT	SET PROMPT
Default protection	F$ENVIRONMENT	SET PROTECTION/DEFAULT
Privileges	F$SETPRV	F$SETPRV or SET PROCESS /PRIVILEGES
Control characters	F$ENVIRONMENT	SET CONTROL
Verification	F$VERIFY	F$VERIFY
Message format	F$ENVIRONMENT	SET MESSAGE
Key state	F$ENVIRONMENT	SET KEY

To ensure that cleanup operations are performed even if the command procedure is aborted, begin each command level in the command procedure with the following statement:

```
$ ON CONTROL_Y THEN GOTO CLEAN_UP
```

In each command level of the command procedure, place cleanup operations after the CLEAN_UP label.

7 Maintaining Accounts and System Security

By being aware of system security, you can take steps to protect your files from unauthorized access. This chapter discusses security features of the VMS operating system and describes how to protect your files and use the system securely.

7.1 User Accounts

User accounts are maintained in a file named SYS$SYSTEM:SYSUAF.DAT, referred to as the user authorization file, or UAF. A VMS system uses the UAF to validate login requests and to set up processes for users who successfully log in. You can examine and modify this file with the Authorize Utility (AUTHORIZE).

The UAF contains a record for each account. Each record consists of fields providing information for the following areas: identification, login characteristics, login restrictions, priority, limits, and privileges. The user name field is specified as a parameter to Authorize Utility commands; the other fields are specified as qualifier values of Authorize Utility commands. See the *VMS Authorize Utility Manual* for descriptions of the qualifiers and information about invoking the Authorize Utility.

7.2 Protection

The VMS operating system provides two related mechanisms to control the access that users have to system objects as follows:

- UIC-based protection—Each user process in the system is assigned a user identification code (UIC) in the user authorization file (UAF) with the Authorize Utility. Each object on the system, such as a file, is also assigned a UIC (typically the UIC of its creator). Each object also maintains a protection mask, a structure which defines the type of access allowed to users, based upon the relationship between the user UIC and the object UIC.
- ACL-based protection—An access control list (ACL) that specifies the type of access to be granted or denied to a particular user or group of users may be associated with a system object. An ACL is an optional form of protection that is typically created by the object owner using the ACL editor (invoked with the DCL command EDIT/ACL) or the SET ACL command.

  The system objects for which ACL-based protection can be specified are: files, directories, devices, batch and print queues, logical name tables, and global sections. Users are specified by identifiers in the rights database that are assigned with the Authorize Utility. (The rights database is described in Section 7.2.2.2.)

The system determines whether to grant a user access to an object, as follows:

1 ACL—If the user matches an identifier in the object's ACL, the system grants or denies access based on the ACL. However, even if an entry in the ACL denies access, the system may still grant access based on the SYSTEM and OWNER fields of the UIC-based protection (see Section 7.2.1.2).

2 UIC—If the user does not match an identifier in the object's ACL or the object has no ACL, the system grants or denies access based on the relationship between the user's UIC and the object's UIC as specified in the object's protection mask.

3 Privileges—If the system denies the user access, the user may be granted access by using one of the following privileges: BYPASS, GRPPRV, READALL, or SYSPRV.

UIC-based protection is useful for denying or granting access to a specified group of users or to all users on the system. The optional ACL-based protection allows further control over the protection of an object. You can grant or deny access to individual users, and you can further identify users by certain aspects of their usage (such as whether they are interactive, batch, local, remote, or dialup users). The combination of UIC and ACL protection provides a way to specify multiple subsets and overlapping groups of users.

7.2.1 UIC-Based Protection

Typically, you use UIC-based protection if the object is to be accessed by: (1) only the owner, (2) all users on the system, or (3) a specific group of users.

7.2.1.1 UIC Format

UICs are a subset of the identifiers that the VMS system uses to identify users and groups of users (see Section 7.2.2.2 for other identifiers). Each user on the system is assigned a unique UIC when the account is created. A UIC consists of two parts, group and member, specified in the following format:

```
[group,member]
```

A UIC can be either numeric or alphanumeric.

- Numeric UIC—Consists of a group number in the range 0 through 37776 (octal) and a member number in the range 0 through 177776 (octal).
- Alphanumeric UIC—Consists of a member name (the user name parameter specified with the Authorize Utility command ADD) and, optionally, a group name. The UIC group name is taken from the account name specified with the /ACCOUNT qualifier. Member and group names must contain at least one alphabetic character and up to a maximum of 31 alphanumeric characters (including A through Z, 0 through 9, underscore, and dollar sign characters). An alphanumeric UIC is equated to a numeric UIC in the rights database by default once the rights database has been created. You can generally specify a numeric UIC and its equivalent alphanumeric UIC interchangeably.

The member component of a UIC must be unique to the system. By default, the member component of an alphanumeric UIC is equated to both the group and member components of a numeric UIC in the rights database (so that specifying just the member part of an alphanumeric UIC is sufficient). The following examples illustrate several UICs in proper UIC notation:

UIC	Meaning
[200,10]	Group 200, member 10
[3777,3777]	Group 3777, member 3777
[USER,FRED]	Group USER, member FRED
[EXEC,JONES]	Group EXEC, member JONES
[JONES]	Group EXEC, member JONES

When you log in to a VMS system, the UIC of your process is the UIC specified in your UAF account. Typically, your process UIC does not change, although it can be changed with the SET UIC command (which requires CMKRNL privilege). By default, detached processes (created by the DCL command SUBMIT or RUN) and subprocesses (created by the DCL command SPAWN) take the same UICs as their creators. If you have DETACH privilege, you can create a detached process with a different UIC (by using the /UIC qualifier of the RUN command).

By default, an object (such as a file) receives the UIC of the process creating it. You can change the UIC of a file with the /OWNER_UIC qualifier of the BACKUP, CREATE, SET DIRECTORY, and SET FILE commands. With SYSPRV privilege, you can specify any UIC; with GRPPRV privilege, you can specify any member within your current group; otherwise, you can specify only your own UIC.

You can specify the UIC of a disk volume with the /OWNER_UIC qualifier of the INITIALIZE and MOUNT commands (except for the system disk, which retains the UIC specified when it was initialized). You can also change the UIC of a disk (including the system disk) with the SET VOLUME command. You must have the VOLPRO privilege to specify a UIC other than your own. In addition, when you initialize a system disk (/SYSTEM qualifier), it receives a UIC of [1,1] and a group disk (/GROUP qualifier) receives a UIC of [n,0], where *n* is the group number of the owner. You can specify a UIC for a device such as a terminal (by default, devices are not owned) with the /OWNER_UIC qualifier of the SET PROTECTION/DEVICE command.

7.2.1.2 Ownership and Access Categories

The relationships between the UIC of a process and the UIC of an object fall into the following four ownership categories:

- System—The UIC of the process is in the range 1 through 10 (octal) or the process has SYSPRV privilege. (The range of system UICs is determined by the SYSGEN parameter MAXSYSGROUP, which defaults to 10 octal.)
- Owner—The UIC of the process and the UIC of the object are identical.
- Group—The group number of the process and the group number of the object are identical or the process has GRPPRV privilege.

- World—All users on the system.

A process may be able to access an object through more than one of these ownership categories. For example, a user with a UIC of [CS102,MARTIN] can attempt access to an object with a UIC of [CS102,PROF] through both the group and world categories.

A process can access an object in the following ways:

- Read (allocate)—Read a file; read from a disk volume; allocate nonfile devices.
- Write—Write a file; write to a disk volume.
- Execute (create)—Execute an image file; look up entries in a directory if you explicitly specify the file name (without using wildcard characters); create files on a disk volume.
- Delete—Delete files.

7.2.1.3 Protection Masks

A protection mask is a structure created by the system for each system object defining the type of UIC access allowed to the object. A protection mask consists of four fields, each with four indicators. Each field applies to one category of ownership. Each indicator within a field applies to one category of access. The fields and indicators are as follows:

Ownership Fields	Access Indicators			
SYSTEM	READ	WRITE	EXECUTE	DELETE
OWNER	READ	WRITE	EXECUTE	DELETE
GROUP	READ	WRITE	EXECUTE	DELETE
WORLD	READ	WRITE	EXECUTE	DELETE

Protection for an object must be specified in the following format:

```
(ownership[:access],...)
```

Specify ownership as one of the following (each may be abbreviated to one character): SYSTEM, OWNER, GROUP, or WORLD. Specify access as one or more of the following indicators: R (read), W (write), E (execute), D (delete). Omission of the colon and access indicators disallows access for that category of ownership. The following protection specification allows system users full access to an object, the owner full access except delete, and group and world users no access:

```
(S:RWED,O:RWE,G,W)
```

Omission of an ownership category generally results in no access being granted to that category. However, the SET PROTECTION command retains the existing protection of the object for omitted fields of the ownership category.

7.2.1.4 Securing User Data and Devices

The suggestions for establishing UIC-based protection of data and devices belonging to individual and application accounts are as follows:

- Default protection—Make sure your default protection is adequate. In general, you do not want to grant write or delete access to world users. You may or may not want to grant write access to group users. You may or may not want to grant read access to world users.

  By default, the system assigns each file the following protection mask:

  ```
  (S:RWED,O:RWED,G:RE,W)
  ```

  You can redefine this default for all files you create using the SET PROTECTION/DEFAULT command, or you can modify the protection on individual files using the /PROTECTION qualifier to the SET FILE command.

- Sensitive files—Protect sensitive files by specifying extra protection, for example, with the SET PROTECTION command. You can also protect sensitive files by maintaining them in a subdirectory on which extra protection is set. However, to protect sensitive files completely, directory protection alone is not adequate. You must also protect each individual file contained within the directory. You can add a default protection ACE to the subdirectory file that defines the UIC protection mask for newly created files in the directories (see Section 7.2.2.5).
- Individual access—Use access control lists (ACLs) to grant or deny individual users access to a file (see Section 7.2.2).
- Copied files—In general, do not copy a file into someone else's directory (for example, if you have write access to a group member's directory), as it will have your UIC instead of the UIC of the directory's owner. Use the MAIL command to send the file, or have the owner of the directory copy the file.
- Private volumes—A private volume is one that is mounted on a device allocated to your process. No other users on the system can access the volume. If you mount a private volume, use the DEALLOCATE command to deallocate the device when you are done.

7.2.2 ACL-Based Protection

Typically, you use access control lists (ACLs) on system objects to grant or deny access to individual users, groups of users, and to subsets of user groups. ACLs contain entries (ACEs) that specify the access to be granted or denied a user or group of users. The user or group of users is designated by identifiers, as described in Section 7.2.2.2.

7.2.2.1 Object Types

You can establish ACLs for various system objects: files, directory files, devices, global sections, queues, and logical name tables. In general, you need not be concerned about the object type when establishing or changing an ACL; however, ACLs set up on devices, logical name tables, and global sections (except those backed by files) are not saved and must be reestablished every time the system is booted.

ACLs on system objects are set up and modified with the ACL editor (see Section 7.3.2) or with the DCL command SET ACL in the following format:

```
SET ACL/OBJECT_TYPE=object-type object-name ...
```

For example, the following command adds an ACL to the file PERSONNEL.DAT containing a single ACE that denies all network access to the file:

```
$ SET ACL/OBJECT_TYPE=FILE PERSONNEL.DAT -
_$ /ACL=(IDENTIFIER=NETWORK,ACCESS=NONE)
```

7.2.2.2 Identifiers

An identifier is a value that represents an individual user, or a group of users, for an aspect of the user's environment. Following are the three types of identifiers:

- UIC identifiers
- General identifiers
- System-defined identifiers

A UIC identifier is created each time a new user account is added with the Authorize Utility. The UIC identifier matches the UIC specified for the user. A second UIC identifier is created matching the name of the UIC group when a UIC group is defined for the first time. UICs can be in both numeric and alphanumeric form (see Section 7.2.1.1) and are useful in identifying individual users in ACLs. UIC identifiers must be enclosed in brackets and can have wildcard characters in either the group or member fields (for example, [EXEC,*]).

Identifiers include other general identifiers that you explicitly associate with users in the rights database. These general identifiers are useful in identifying multiple groups of users outside the bounds of UIC groups. For example, you could create the identifier SECRET and assign it in the rights database to a selected group of users, some of whom could be in different UIC groups.

A third type of identifier includes the following system-defined identifiers, which you can use to identify users by their mode of using the system:

System-Defined Identifiers	Type of User
BATCH	Batch user
DIALUP	User logged in on a dialup terminal
INTERACTIVE	Interactive user
LOCAL	User logged in on local terminal
NETWORK	Network process
REMOTE	User logged in over the network

Generally, you should treat the preceding system-defined identifiers as being mutually exclusive. However, you can combine them with UIC identifiers or general identifiers by connecting them with plus signs (for example, [FRED]+BATCH). Access is granted only if both identifiers are true. (In the example [FRED]+BATCH, the user identified as [FRED] must be running a batch job for the system to grant the specified access.)

Following are examples of user-defined identifiers that are valid for ACL-based protection:

- PAYROLL—Specifies all users holding the identifier PAYROLL.
- [USER,JONES]—Specifies the user whose alphanumeric UIC is group USER and member JONES.
- [200,10]—Specifies the user whose numeric UIC is group 200, member 10.
- [FRED]+BATCH—Specifies the batch user whose alphanumeric UIC is FRED.
- DIALUP—Specifies all users logged in on a dialup terminal.

Rights List

The system determines protection by checking identifiers in the object's ACL against the list of identifiers held by the accessor to find a matching entry. The list of identifiers held by the accessor is called a *rights list*.

The file containing all the identifiers defined in the system is called the *rights database*. (Identifiers are defined and granted to specific users with the AUTHORIZE commands ADD/IDENTIFIER and GRANT/IDENTIFIER.) The rights list, created for each process at login, is the portion of the rights database containing all the identifiers and attributes held by the user.

7.2.2.3 Access Control List Entries (ACE)

An entry in an access control list specifies the access to a system object that is to be granted or denied a user. This access is specified by the identifier. Different kinds of access are: NONE, READ, WRITE, EXECUTE, DELETE, and/or CONTROL.

Following are the three types of ACEs:

- Identifier ACE
- Default_protection ACE
- Alarm_journal ACE

7.2.2.4 IDENTIFIER ACEs

An identifier ACE controls the type of access allowed to a particular user or group of users. Identifier ACEs have the following format:

```
(IDENTIFIER=identifier[,OPTIONS=options+...],ACCESS=access+...)
```

The following are examples of ACEs:

```
(IDENTIFIER=[200,201],ACCESS=READ+WRITE+EXECUTE)
```

Grants the user identified by UIC identifier [200,201] read, write, and execute access to the system object.

```
(IDENTIFIER=[FRED]+BATCH,ACCESS=WRITE+EXECUTE)
```

Grants batch user with the alphanumeric UIC [FRED] write and execute access to the system object.

```
(IDENTIFIER=PAYROLL,ACCESS=READ)
```

Grants users who hold the identifier PAYROLL read access to the system object.

```
(IDENTIFIER=DIALUP,ACCESS=NONE)
```

Denies holders of the system-defined identifier DIALUP any access to the system object.

The preceding ACEs could be specified in a single ACL for a system object as follows:

```
(IDENTIFIER=[200,201],ACCESS=READ+WRITE+EXECUTE)
(IDENTIFIER=[FRED]+BATCH,ACCESS=WRITE+EXECUTE)
(IDENTIFIER=PAYROLL,ACCESS=READ)
(IDENTIFIER=DIALUP,ACCESS=NONE)
```

To specify one or more default ACEs for inclusion in the ACLs of files subsequently created in a directory, use the OPTIONS=DEFAULT option of an identifier ACE. The following ACE, when placed in the ACL of the directory file, grants all users holding the SECRET identifier read, write, and execute access to new files in the directory:

```
(IDENTIFIER=SECRET,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE)
```

Note: **Default protection is associated only with newly created files, not existing ones. If you add a default protection ACE to a directory, you must also change the protection on files already in the directory.**

Because the system determines access at the first matching entry, the order of the entries is critical. For example, if the last entry in the previous example—(IDENTIFIER=DIALUP,ACCESS=NONE)—were placed at the top of the list, all dialup users (including those specified in the remaining entries) would be denied access to the associated file. Placing it last in the access control list allows users holding identifiers [200,201], [FRED]+BATCH, and [PAYROLL] the specified access, even when they are dialup users.

7.2.2.5 DEFAULT_PROTECTION ACEs

To specify default protection for new files in a particular directory, place a default_protection ACE in the ACL of the directory file. (The directory file is the file with the directory name as file name and DIR as file type in the parent directory). The default_protection ACE affects files that are subsequently created in the directory and in any subdirectories under that directory unless protection is specified for one of those files individually. Default_protection ACEs apply UIC-based protection. Specify a default_protection ACE in the following format, where *protection-mask* is the same mask used in UIC protection (see Section 7.2.1.3):

```
(DEFAULT_PROTECTION[,options],protection-mask)
```

The following default_protection ACE specifies that by default the system and owner have read, write, execute, and delete access to any files subsequently created for the directory and that group and world users have no access.

```
(DEFAULT_PROTECTION,S:RWED,O:RWED,G,W)
```

7.2.2.6 ALARM_JOURNAL ACEs

The alarm_journal ACE allows you to specify that an alarm message be sent to the security operator's terminal if a certain type of access takes place. Alarms are supported only for files and global sections. The alarm_journal ACE functions only when alarms to the security operator's terminal have been enabled through the DCL command SET AUDIT, security messages to the operator's terminal have been enabled with the DCL command REPLY/ENABLE=SECURITY, and the OPCOM process is executing.

Following is the format of an alarm_journal ACE:

```
(ALARM_JOURNAL=SECURITY[,options+...][,access+...])
```

The following alarm_journal ACE specifies that an alarm will be sent to the security operator's terminal if an accessor attempts to read the object, and that this ACE will be preserved even when an attempt is made to delete the entire ACL:

```
(ALARM_JOURNAL=SECURITY,OPTIONS=PROTECTED,ACCESS=READ+SUCCESS+FAILURE)
```

For an alarm to have any effect, you must include either SUCCESS or FAILURE or both in the ACCESS field.

7.2.3 File Protection

File protection is usually transparent. To set protection or modify the ACL of a file, you must own the file, have control access to the file, or have GRPPRV, SYSPRV, BYPASS, or READALL privilege.

Note: **To completely protect a file, you must apply the same or greater protection to the directory in which the file resides. See Section 7.2.3.3 for information on directory protection.**

7.2.3.1 Default File Protection

A new file receives default UIC-based protection and the default ACEs (if any) of its parent directory. A renamed file's protection is unchanged. A new version of an existing file receives the UIC-based protection and ACL of the previous version. (Use the /PROTECTION qualifier of the BACKUP, COPY, CREATE, and SET FILE commands to override the default UIC-based protection.)

You can use either of the following methods to override the default UIC-based protection given to new files:

- Default UIC protection—The operating system provides each process with a default UIC-based protection of (S:RWED,O:RWED,G:RE,W). This indicates that SYSTEM users and the owners of objects have full access to the object, users in the same UIC group as the object owner have read and execute access to the object, and all other users are denied access to the object. To change the default protection, invoke the SET PROTECTION command with the /DEFAULT qualifier. For example, if you place the following command in your login command procedure, you grant all processes read and execute access to any files that you create. (Remember that you must execute the login command procedure for this command to execute.)

  ```
  $ SET PROTECTION = (S:RWED,O:RWED,G:RE,W:RE)/DEFAULT
  ```

- Default ACL protection—You can override default UIC protection for specified directories or subdirectories by placing a default_protection ACE in the ACL of the appropriate directory file. The default protection specified in the ACE is applied to any new file created in the specified directory or subdirectory of the directory. The following ACE, which must be in the ACL of a directory file, specifies that the default protection for that directory and the directory's subdirectories allow system and owner processes full access, group processes read and execute access, and world users no access.

  ```
  (DEFAULT_PROTECTION,S:RWED,O:RWED,G:RE,W:)
  ```

  To specify a default identifier ACE to be copied to the ACL of any file subsequently created in the directory, specify the DEFAULT option in the directory file's identifier ACL.

7.2.3.2 Explicit File Protection

You can explicitly specify UIC-based protection for a new file with the /PROTECTION qualifier (valid with the BACKUP, COPY, and CREATE commands) as shown in the following example:

```
$ CREATE MAST12.TXT/PROTECTION=(S:RWED,O:RWED,G,W)
```

You can change the UIC-based protection on an existing file with the SET PROTECTION command as shown in the following example:

```
$ SET PROTECTION=(S:RWED,O:RWED,G,W) MAST12.TXT
```

After a file is created and you have created an ACL for the file, you can modify the ACL and add as many ACEs to the ACL as you want. The protection specified by the ACL overrides the file's UIC protection.

7.2.3.3 Directory Protection

You cannot completely protect a file without applying at least the same protection to the directory in which the file resides. For example, if you deny a user all access to a file but allow that user read access to the file's directory, the user cannot access the contents of the file but can see that it exists. Conversely, a user allowed access to a file and denied access to the file's directory (or one of the parent directories) cannot see that the file exists.

Note: **To protect sensitive files, directory protection alone is not adequate. You must also protect each file within the directory.**

By default, top level directories receive UIC-based protection (S:RWE,O:RWE,G:RE,W:E) and no ACL. Subdirectories receive UIC-based protection from the parent directory.

To specify UIC-based protection explicitly when creating a directory, use the /PROTECTION qualifier of the CREATE/DIRECTORY command. You cannot specify an ACL for the directory until the directory is created. To change the UIC-based protection of an existing directory, use the SET PROTECTION command (apply this command to the directory file). To specify or change the ACL of an existing directory, edit the directory file's ACL (see Section 7.2.3.1).

You can limit but not prohibit directory access by specifying execute access but not read access. Execute access on a directory permits you to examine and read files that you know are contained in the directory (that is, you know the file specifications), but prevents you from displaying a list of the files in the directory.

7.2.3.4 Mail File Protection

Mail files receive the protection (S:RWD,O:RW,G,W). Files of type MAI created with the EXTRACT command of the Mail Utility receive the protection (S:RWD,O:RWD,G,W).

7.2.4 Disk Volume Protection

By default, no protection is applied to newly initialized disk volumes. You can specify protection with the /PROTECTION qualifier of the INITIALIZE command and you can specify an ACL for a disk volume. The following example specifies UIC-based protection for the disk volume ACCOUNT1:

```
$ INITIALIZE WORKDISK: ACCOUNT1 -
_$ /PROTECTION=(S:RWED,O:RWED,G:R,W:R)
```

You can respecify the protection each time you mount the volume with the /PROTECTION qualifier of the MOUNT command. You must own the volume or have VOLPRO privilege to change protection.

You can also limit access to a disk volume with the following qualifiers to the INITIALIZE and MOUNT commands:

- /SYSTEM—All processes have RWED access to the volume, but only system processes (or processes with SYSNAM and SYSPRV privileges) can create first-level directories. (The volume is owned by [1,1].)
- /GROUP and /NOSHARE—System, owner, and group processes have RWED access to the volume. World users have no access.

- /NOSHARE—System and owner processes have RWED access to the volume. Group and world users have no access.

At initialization time, the preceding qualifiers override any protection mask specified. At mount time, however, the protection mask overrides the qualifiers. When mounting a volume, you must have GRPNAM privilege to specify /GROUP, and SYSNAM privilege to specify /SYSTEM.

7.2.5 Device Protection

In general, device protection controls the ability to allocate the device and is specified by granting read access in an ACL. To specify an ACL for a disk device, use the SET ACL/OBJECT_TYPE=DEVICE command. For example, to grant a user with the alphanumeric UIC [FRED] read access to the disk device WORKDISK, type:

```
$ SET ACL/OBJECT_TYPE=DEVICE/ACL=(IDENTIFIER=[FRED],ACCESS=READ) -
_$ WORKDISK
```

Note that when you specify an ACL for a disk device, the ACL is associated with the device, not with the disk volume. For example, when you mount a disk volume, specify an ACL for the device. Then when you dismount the volume, the ACL protection remains on the device.

The only protection that applies to a nonfile device is the ability to allocate it, specified by read access. By default, nonfile devices such as mailboxes are unprotected. Interactive terminals are set up to provide complete access to system users and no access to all other users. (Note that access here refers to access via an application program. The device protection on a terminal does not control who can log in to it.)

You can change the protection of a nonfile device through the use of ACLs or by changing the standard UIC protection. Modify the ACL with the DCL command SET ACL/OBJECT_TYPE=DEVICE. Modify the UIC protection with the DCL command SET PROTECTION/DEVICE (requires OPER privilege). For example, the following command allows users holding the PAYROLL identifier read access to TERMINAL3:

```
$ SET ACL/OBJECT_TYPE=DEVICE/ACL=(IDENTIFIER=PAYROLL,ACCESS=READ) -
_$ TERMINAL3
```

7.2.6 Displays of Ownership and Protection

You can display ownership and protection information with the following commands and qualifiers:

Command	Display
DIRECTORY/ACL file-spec	File's ACL
DIRECTORY/OWNER_UIC file-spec	File's UIC
DIRECTORY/PROTECTION file-spec	File's UIC-based protection
DIRECTORY/SECURITY	All of the above
DIRECTORY/FULL file-spec	All of the above

Command	Display
SHOW ACL	Device, file, batch or print queue, logical name table, or global section's ACL
SHOW PROCESS/ALL	Process UIC
SHOW PROTECTION	Default file protection
SHOW DEVICES/FULL device-name	Device UIC and protection

7.3 Creating and Deleting ACLs

Use the ACL editor (invoked with the DCL command EDIT/ACL) to create an ACL or to make major changes to it. Use the DCL command SET ACL with the /OBJECT_TYPE qualifier to make minor changes to an ACL or to set an ACL on more than one object.

7.3.1 Using the SET ACL Command

The DCL command SET ACL allows you to specify an ACL for a file, directory, device, batch or print queue, system logical name table, or global section. The /OBJECT_TYPE qualifier is required for all objects except files. To specify an ACL for a device, for example, specify the SET ACL command with the /OBJECT_TYPE=DEVICE qualifier. The following command specifies an ACL that grants users who hold the identifier PAYROLL read access to the disk device WORKDISK:

```
$ SET ACL/OBJECT_TYPE=DEVICE/ACL=(IDENTIFIER=PAYROLL,ACCESS=READ) -
_$ WORKDISK
```

To set the same ACL on all files with the file type DAT in the default directory, enter the following:

```
$ SET ACL/OBJECT_TYPE=FILE/ACL=(IDENTIFIER=PAYROLL,ACCESS=READ) -
_$ *.DAT;*
```

To select a subset of the files specified by the wildcard character, use one or more of the following qualifiers:

Qualifier	Meaning
/BY_OWNER	Selects files with a specified UIC
/EXCLUDE	Selects all but the specified files
/BEFORE	Selects files created or modified before a specified time
/SINCE	Selects files created or modified since a specified time
/CREATED	Specifies that /BEFORE or /SINCE check for the creation date

You can also use the /CONFIRM qualifier with the preceding qualifiers to elicit a confirmation prompt for each file; the /CONFIRM qualifier allows you to specify whether the file is to be excluded.

By default, the DCL commands that create ACLs add the entries to the top of the ACL in the order specified. (If the object does not have an ACL, an ACL is created.) Use the /AFTER qualifier to specify a position for the entry (other than the top of the ACL). For example, the following command places a new entry after an existing entry—the ACE specified with the /AFTER qualifier—in the access control list of the file DOGS83.DAT:

```
$ SET ACL/AFTER=(IDENTIFIER=[JONES],ACCESS=READ)-
_$ /ACL=(IDENTIFIER=SECRET,ACCESS=READ+EXECUTE)  DOGS83.DAT
```

The /AFTER qualifier is especially useful because the ACEs are processed from first to last. For example, if users holding the identifier PAYROLL are denied access in an ACE that precedes one granting access to user FRED, and if FRED holds both identifiers, FRED will be denied access.

The SET ACL command also permits the following ACL operations:

- Delete an ACE—Use the /DELETE qualifier to delete ACEs (specified on the /ACL qualifier) from an object's ACL. If no value is specified with the /ACL qualifier, all the entries in the ACL of the specified objects are deleted (except those entries protected by the PROTECTED option). The following example deletes the ACE specified with the /ACL qualifier from the ACL of the latest version of each file with type FOR in the default directory:

  ```
  $ SET ACL/OBJECT_TYPE=FILE -
  _$ /ACL=(IDENTIFIER=PAYROLL,ACCESS=READ+WRITE+EXECUTE)-
  _$ /DELETE *.FOR
  ```

- Replace an ACE—Use the /REPLACE qualifier to specify an ACE to replace the ACE specified by the /ACL qualifier. If no value is specified with the /ACL qualifier, the ACEs specified by the /REPLACE qualifier are added to the ACLs of the specified objects. The following example changes the ACE specified with /ACL to the one specified by /REPLACE in the ACL of the device WORKDISK:

  ```
  $ SET ACL/ACL=(IDENTIFIER=[200,200],ACCESS=READ)-
  _$ /REPLACE=(IDENTIFIER=NONEXEC,ACCESS=NONE)-
  _$ /OBJECT_TYPE=DEVICE WORKDISK
  ```

- Copy an ACL—Use the /LIKE qualifier to copy an ACL from one object to another. Specify the name of the object whose ACL is to be copied as the value of the /LIKE qualifier and specify the name of the objects to receive the ACL as the parameter of the SET ACL command. The following example copies the ACL from [USER]X.X to all versions of all files in the [USER] directory tree. The /LOG qualifier displays each file specification as the file is modified. (You might also want to include the /CONFIRM qualifier, which issues a request for confirmation before each modification.)

  ```
  $ SET ACL/LOG/LIKE=[USER.TESTS]X.X  [USER...]*.*;*
  ```

- Propagate the directory ACL—Use the /DEFAULT qualifier to copy the ACL on a directory file to files in the directory. For subdirectory (DIR) files in the directory, all ACEs except those with the NOPROPAGATE option are copied. For all other files, only ACEs with the DEFAULT option are copied from the parent directory. Any ACL that already exists on the file is deleted.

Use wildcards to copy the directory ACL to multiple files or to copy the directory ACL throughout the directory tree. The following example propagates the ACL on the [JMARTIN.WORK] directory to all files in the subdirectory:

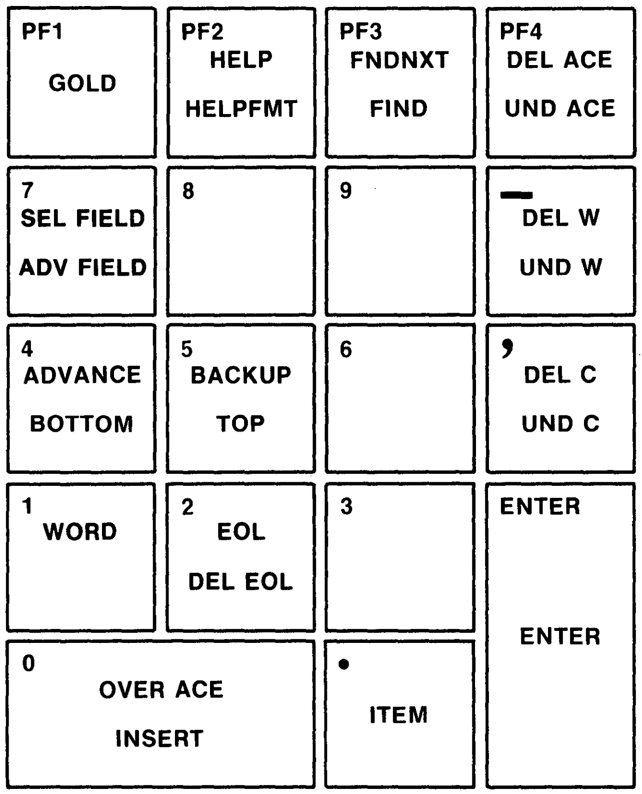
```
$ SET ACL/DEFAULT [JMARTIN.WORK]*.*;*
```

7.3.2 ACL Editor

It is often more convenient to use the ACL editor instead of the SET ACL command (for example, when creating or modifying long ACLs). To invoke the ACL editor, type EDIT/ACL followed by the name of the object whose ACL you want to edit and press the RETURN key as follows:

```
$ EDIT/ACL PROTA.TXT
```

If the object has an ACL, the ACL appears on the screen. Otherwise, you are creating an ACL and will begin the editing session by entering an ACE. Enter ACL editor commands using the keypad keys shown in the following diagram.



ZK-1740-84

By default, the ACL editor operates as if the object whose ACL is being edited is a file. To specify an object other than a file, use the /OBJECT_TYPE qualifier. For example, to edit the ACL of a device named TTA1, enter the following command line:

```
$ EDIT/ACL/OBJECT_TYPE=DEVICE TTA1:
```

You can specify any one of the following objects with the /OBJECT_TYPE qualifier:

FILE	Specifies that the object type is a file or a directory file
DEVICE	Specifies that the object type is a device
SYSTEM_GLOBAL_SECTION	Specifies that the object type is a system global section
GROUP_GLOBAL_SECTION	Specifies that the object type is a group global section
LOGICAL_NAME_TABLE	Specifies that the object type is a logical name table
QUEUE	Specifies that the object type is a queue

To execute a command, press the key. To execute the lower command shown on a key in the diagram, press the GOLD key first, and then press the key. You can display information on each command by pressing the HELP key followed by the command. To display information on the proper format of an ACE, press GOLD and then HELP.

To exit from the editor and create or modify the ACL, press CTRL/Z. To exit without creating or modifying the ACL, press GOLD and CTRL/Z. To refresh the screen while in the editor (for example, after receiving a broadcast message), enter CTRL/W.

The ACL editor edits a copy of the ACL. If your ACL editing session is terminated by entering CTRL/Y or if the system fails, the existing ACL remains unchanged or, if no ACL exists, no ACL is created. To recover an ACL editing session interrupted by CTRL/Y or system failure, specify the /RECOVER qualifier of the EDIT/ACL command.

7.3.2.1 Using Prompts

By default, the ACL editor prompts for each ACE and provides values in the various fields within an ACE whenever possible. The /MODE qualifier controls the choice of mode. To disable prompting, specify /MODE=NOPROMPT.

The FIELD, ITEM, and ENTER commands on the keypad enable you to take full advantage of prompt mode in the ACL editor.

- FIELD—Completes the current ACE field and moves the cursor to the next ACE field or subfield, inserting text as needed. If the ACL editor is not in prompt mode, the ACL editor advances to the next field in the current existing ACE.
- ITEM—Selects the next item for the current ACE field. If the ACL editor is not in prompt mode, this key is ignored.
- ENTER—Indicates that the current ACE is complete. This key ends the insertion. You can press it while the cursor is located at any position within the ACE. (Pressing the RETURN key produces the same results.)

7.3.2.2 Moving the Cursor

To position the cursor within the current line, use the following commands:

Command	Action
EOL	Moves the cursor to the end of the line
WORD	Moves the cursor one word in the current direction
ADVANCE	Sets the cursor direction to forward
BACKUP	Sets the cursor direction to backward
LEFT ARROW	Moves the cursor left one character
RIGHT ARROW	Moves the cursor right one character
FIELD	Completes the current ACE field and moves the cursor to the next ACE field or subfield
ITEM	Selects the next item for the current ACE field

The following commands move the cursor to a different line of the ACL:

Command	Action
BOTTOM	Moves the cursor to the end of the ACL
TOP	Moves the cursor to the beginning of the ACL
UP ARROW	Moves the cursor up a line
DOWN ARROW	Moves the cursor down a line
OVER ACE	Moves the cursor to the next ACE

To locate a specified text string or locate the next occurrence of a previously specified text string, use FIND and FIND NEXT.

If the command moves the cursor out of an unprocessed ACE, the editor attempts to process the current ACE. If the ACE is improperly formatted, an error occurs and the cursor remains in place.

7.3.2.3 Entering and Deleting Data

Unless you disable prompting, the following text appears on the screen when you invoke the ACL editor:

```
(IDENTIFIER=
```

If you want to create an identifier ACE, specify the identifier value by typing in the appropriate value. You can also delete the text and type a default_protection or alarm_journal ACE.

To delete text, use the following commands:

Command	Action
DEL C	Deletes the current character
DELETE key	Deletes the previous character
DEL W	Deletes the current word
LINE FEED	Deletes the previous word
DEL EOL	Deletes from cursor to end of line
DEL ACE	Deletes the current ACE
CTRL/U	Deletes the text from the cursor to the beginning of the line

If you delete text by mistake, you can restore it by using the following commands:

Command	Action
UND C	Restores the most recently deleted character
UND W	Restores the most recently deleted word
UND ACE	Restores the most recently deleted ACE
GOLD CTRL/U	Restores the most recently deleted portion of a line

7.3.2.4 Processing an ACE

To process an ACE, press ENTER or carriage return. You must process the current ACE before you can begin editing a second ACE. If the ACE is in the proper format, the editor puts the ACE into the ACL and moves the cursor to the next line. If the cursor is positioned at the end of the ACL and if prompting is enabled, the editor displays the text IDENTIFIER=. (Moving the cursor off a line containing an unprocessed ACE implicitly processes the ACE.)

8 Editing Files with the EVE and EDT Editors

Text editors are computer programs that allow you to enter text from a keyboard into computer memory. Once the text is in memory, you can modify the text using text editing commands. For example, you can type in data for a report and then rearrange sections, duplicate information, or substitute phrases. Text editors also format text so that the report or article can be ready for distribution. Different formatting commands set margins, insert white space, and paste in figures and tables.

VMS supports several text editors. This chapter discusses the EVE and the EDT editors.

8.1 The EVE Editor

EVE, the Extensible VAX Editor, is an editor built on the VAX Text Processing Utility (VAXTPU), a high performance, programmable, text processing utility. Using EVE, you can create and edit new files or edit existing files. You can add text to a file and modify or format that text. EVE is interactive, so you see the changes to a file as you make them.

Unlike the EDT editor, EVE lets you display more than one buffer on the screen at a time and to edit more than one file during the same editing session. EVE is easy to customize or extend using EVE commands and VAXTPU procedures.

8.1.1 Beginning and Ending an Editing Session

To begin an editing session, invoke EVE with the DCL command EDIT/TPU. In an editing session, you can create and edit a new file, or you can edit an existing file. The session ends when you enter the EXIT or QUIT command. Exiting from EVE typically produces a new file or a new version of an existing file.

8.1.1.1 Invoking EVE

You can start an editing session by creating a new file and inserting text into it during the session. You can also begin by specifying an existing file when you invoke EVE.

To begin an EVE editing session, enter the DCL command EDIT/TPU. Specify a file name on the command line if you want to edit an existing file or assign a name to a new file.

For example, to invoke EVE to create a new file named NEWFILE.DAT, enter the following command:

```
$ EDIT/TPU MYFILE.DAT
```

This command produces the following screen output:

```
[End of file]


Buffer NEWFILE.DAT                                       | Insert | Forward
Editing new file: could not find WORKDISK:[USER]NEWFILE.DAT
```

EVE inserts the text of the file you are editing into a temporary holding area called a *buffer*. A buffer is a temporary storage area and exists only during an editing session. The contents of the buffer are shown in an area of your screen that is called a *window*. When you end an editing session, you direct EVE to save or discard the contents of a buffer.

The end-of-file marker defines the end of an EVE buffer. It is only visible on the screen and does not become part of your file. When you add text to the buffer, the end-of-file marker moves downward. The marker may not be visible when you are viewing the beginning of a buffer that contains many lines of text.

A highlighted status line appears at the bottom of the EVE window and provides information about the EVE buffer. The status line shows the buffer name, current mode (insert or overstrike), and current direction (forward or reverse).

If you invoke EVE with a file name, an informational message appears in the message buffer beneath the highlighted status line stating either that the file is a new file or that a certain number of lines were read from an existing file. EVE communicates with you throughout the editing session by displaying messages in the Message window.

To invoke EVE to edit an existing file named OLDFILE.DAT, enter the EDIT/TPU command in the following format:

```
$ EDIT/TPU OLDFILE.DAT

Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                       | Insert | Forward
4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

When you invoke EVE to edit an existing file, you can use the asterisk wildcard character (*) as a substitute for all or some of the characters in the file name and file type. You can use the percent wildcard character (%) as a substitute for one character in the file name and file type, and you can use the ellipsis wildcard ([. . .]) as a substitute for a directory specification. When using wildcards in EVE, follow the same rules as using wildcards in DCL. If only one match is made, the file is displayed on your screen. If more than one match is made, EVE displays a list of matching files and prompts you to provide a more complete file specification. If no match is made, the file name that you typed appears in the highlighted status line. The message *Editing new file. Could not find:* is displayed in the message buffer.

You may want to use a command symbol to invoke the EVE editor. For example, if you place the following statement in your login command file and execute your login command file, you only need to type *EVE* to invoke the EVE editor.

```
$ EVE == EDIT/TPU
```

Rather than name a file at the beginning of an editing session, you can invoke EVE with the command EDIT/TPU and then enter text into the buffer. You can save the text by writing it to a file using the WRITE FILE command described in Section 8.1.7. Alternately, when you finish creating your file, EVE prompts for a file name as follows:

```
Enter a file name to write buffer MAIN; else press RETURN:
```

Type the name of the file, and press RETURN to write out the buffer to a file.

8.1.1.2 Ending an Editing Session

Two different commands can end an EVE editing session. EVE produces a new version of the edited file when you end the session with the EXIT command. EVE discards your edits when you end a session with the QUIT command. Any existing versions of the files remain unchanged regardless of how the editing session is ended.

To save your edited text, use the EXIT command. Enter the EXIT command by pressing the F10 key (on VT200-series or VT300-series terminals) or by pressing CTRL/Z.

If you have modified the current buffer, EVE creates a new version of the file with the same file name and file type as the original version, with the version number incremented by 1. For example, if you use the EXIT command after modifying a file named FUN.DAT;1, the output file is named FUN.DAT;2, as follows:

```
Buffer FUN.DAT                                      | Insert | Forward
4 lines written to file WORKDISK:[USER]FUN.DAT;2
```

To exit from a session without saving your edits, use the QUIT command. To execute the QUIT command, press the DO key (PF4 on VT100-series terminals), type QUIT at the *Command:* prompt, and press RETURN. For example, if you have modified a file named FUN.DAT and enter the QUIT command, the following display appears on your terminal screen:

```
Buffer FUN.DAT                                      | Insert | Forward
Buffer modifications will not be saved, continue quitting (Y or N)?
```

Type Y and press RETURN if you want to quit without saving the edits. If you change your mind and decide to save your edits, type N, press RETURN, and exit from the file using the EXIT command.

If you have modified buffers other than the current one, EVE asks if you want to save the contents of those buffers. If you type Y, EVE creates new versions of any existing files, incrementing the version number by 1. EVE prompts for a file name if no file currently exists.

8.1.2 Entering EVE Commands

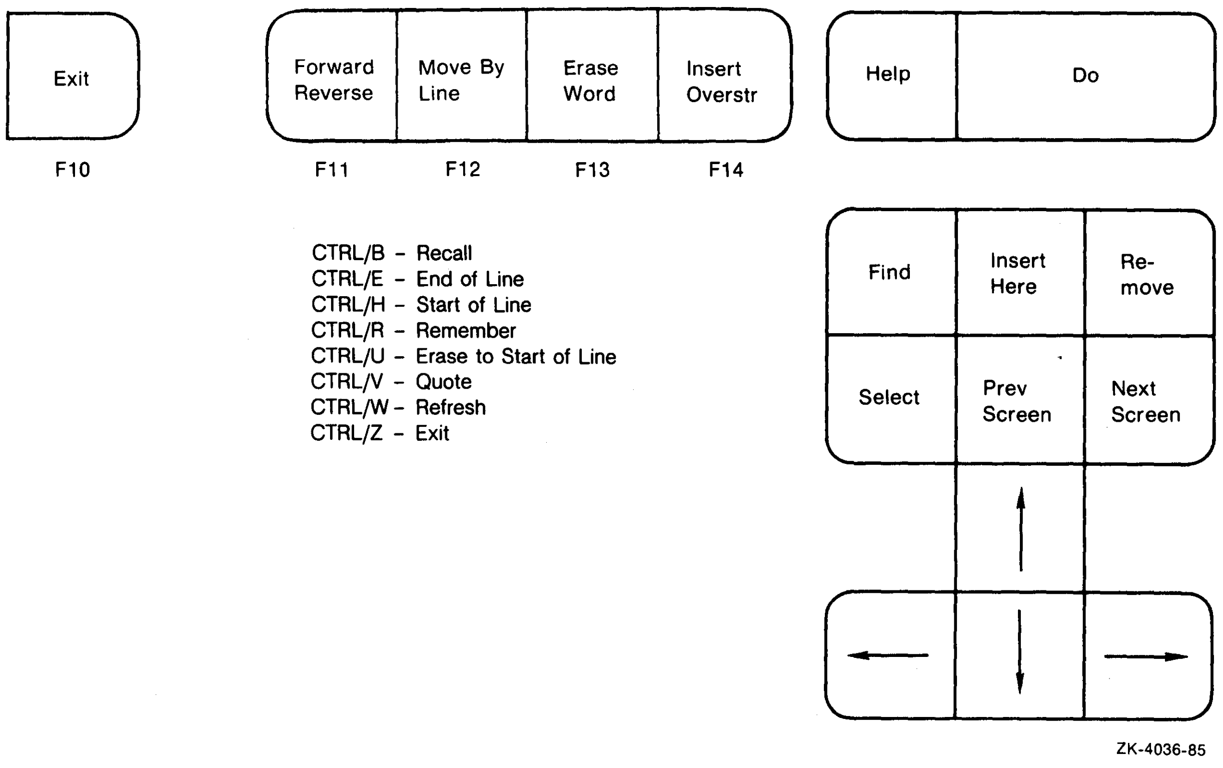
Once you have invoked EVE, enter EVE commands to edit and manipulate text. There are two ways to enter EVE commands: by pressing predefined editing keys and by typing the actual commands.

8.1.2.1 Using Defined Keys to Enter EVE Commands

EVE defines some editing keys by default. You can define additional editing keys to enter commands or to perform editing operations that you use frequently (see Section 8.1.9). The predefined editing keys on VT200-series and VT300-series terminals include the minikeypad (located between the main keypad and the numeric keypad), certain function keys, and certain control key sequences. On VT100-series terminals, EVE automatically defines most of the numeric keypad keys, the arrow keys, and certain control keys. Each predefined editing key performs one editing command.

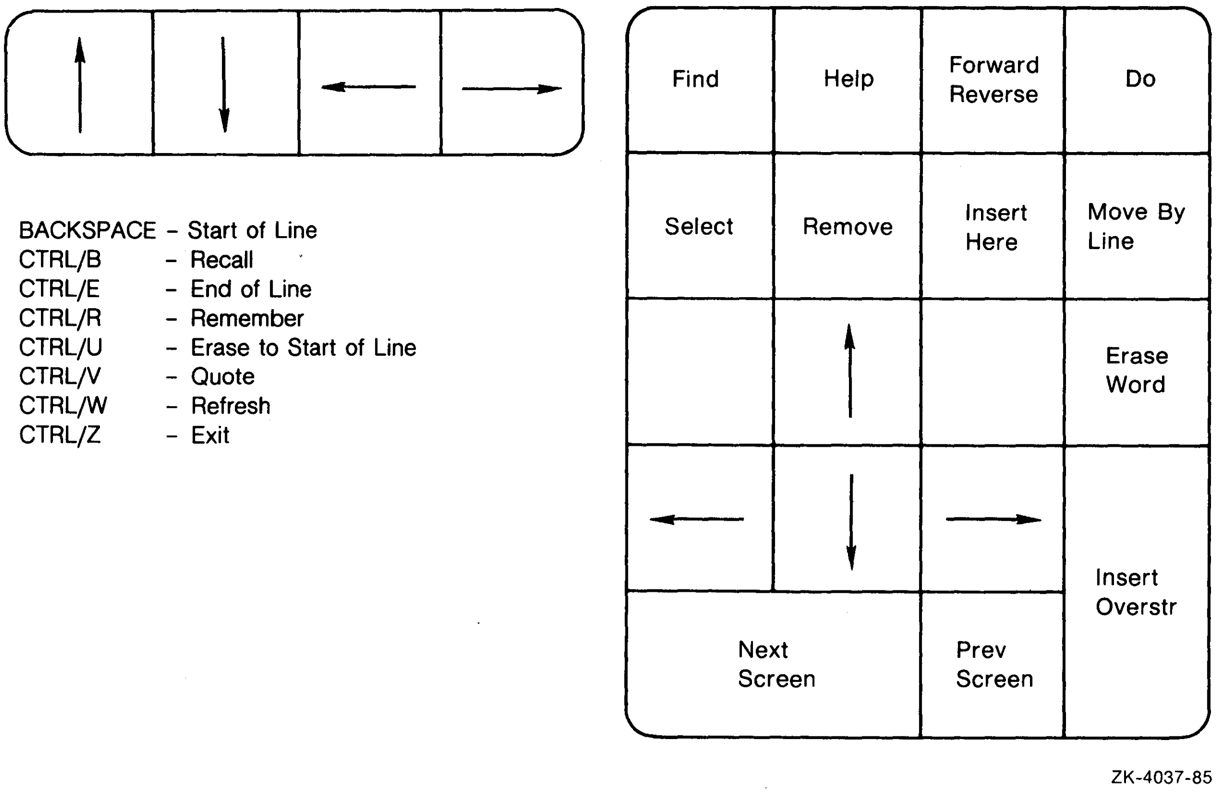
Throughout this chapter, EVE editing keys are referred to by their names, rather than by their location on the VT200-series, VT300-series, or VT100-series keyboards. For example, on a VT200-series or VT300-series terminal, the DO key is located at the top of the editing keypad and is labeled DO. On a VT100-series terminal, the DO key is located at the upper right of the numeric keypad and is labeled PF4. Figure 8–1 shows the predefined keys on the VT200-series and VT300-series terminal. Figure 8–2 shows the predefined keys on the VT100-series terminal.

Figure 8–1 Editing Keys—VT200-Series and VT300-Series Terminals



CTRL/B	- Recall
CTRL/E	- End of Line
CTRL/H	- Start of Line
CTRL/R	- Remember
CTRL/U	- Erase to Start of Line
CTRL/V	- Quote
CTRL/W	- Refresh
CTRL/Z	- Exit

Figure 8–2 Editing Keys—VT100-Series Terminals



Note that EVE uses the numeric keypad differently on the VT100-series terminals than on the VT200-series and VT300-series. EVE automatically defines 16 of the 18 numeric keypad keys on the VT100 as editing keys, but it does not automatically define the numeric keypad keys on the VT200-series and VT300-series terminals. On these later model terminals, you can use the numeric keypad keys to enter numeric data, or you can define them to enter EVE commands (see Section 8.1.9).

EVE offers two default keypads in addition to the default EVE keypad. You can select an EDT keypad or a WPS keypad. Although neither fully implements EDT or WPS editing functions, each provides most keypad functions.

8.1.2.2 Entering EVE Commands

In addition to using defined keys, you can enter EVE commands by typing them at the *Command:* prompt. When you enter EVE commands, you always perform the following three steps:

1. Press the DO key. EVE displays the *Command:* prompt.
2. Type the EVE command after the prompt.
3. Press either RETURN or the DO key to enter the command.

You can correct typing mistakes on the EVE command line by pressing DCL line-editing keys. Use CTRL/U to erase to the beginning of the line, CTRL/E to move to the end of the line, and CTRL/B to recall the last command entered. By default, the editing mode of the EVE command line is the same as the editing mode of your terminal. (You can change the default prior to invoking EVE with the DCL command SET TERMINAL. Once in EVE, you can change the editing mode with CTRL/A.)

To save keystrokes when typing EVE commands, you can use CTRL/B, abbreviate commands, use the REPEAT command, or press the DO key. Each method of saving keystrokes is described as follows:

- Press CTRL/B to recall the last EVE command you entered. Pressing CTRL/B again recalls the previous command that you entered. Continue pressing CTRL/B until the command you wish to execute appears on your screen, and press RETURN to enter the command.
- Abbreviate EVE command names, making sure the abbreviation is unambiguous. If you enter an abbreviation that is not unique, EVE displays a list of matching commands and prompts you for a choice. Type enough additional characters to ensure that the abbreviation is unique, and press RETURN to enter the command. You can also abbreviate buffer names and, file names. Again, EVE provides a list of choices if you do not provide a unique abbreviation.
- Use the REPEAT command to repeat an EVE command or keystroke. Press the DO key, type REPEAT and the number of times it is to be repeated, and press RETURN. EVE repeats the next character or command you enter the specified number of times. For example, to insert the character *p* in your editing buffer 20 times, press the DO key, type REPEAT 20, and press RETURN. Then type *p*. EVE inserts a p into the current buffer 20 times.
- Press the DO key twice to activate the last command entered.

8.1.3 Editing Text

Once you know how to invoke the EVE editor and how to enter commands, you use EVE commands to edit new and existing files. Editing keys and commands allows you to position the cursor and perform such text editing operations as moving, erasing, and restoring text.

8.1.3.1 Moving the Cursor

When editing files with EVE, first you move the cursor to the place in the text where you want to perform an editing function. Therefore, the more quickly and efficiently you move the cursor through the text, the more time you save in your editing session.

The following tables show the EVE editing keys and commands that move the cursor:

Editing Key	Moves the Cursor to the Following Position
Up arrow	Up one line.
Down arrow	Down one line.
Left arrow	One character or column to the left.
Right arrow	One character or column to the right.
CTRL/E	End of the current line.
CTRL/H	Beginning of the current line.
Move By Line	In forward direction: end of the current line or if the cursor is already at the end of a line, to the end of the next line. In reverse direction: beginning of the current line or if the cursor is already at the beginning of a line, to the beginning of the previous line.
Next Screen	Forward in the current buffer. The number of lines the cursor moves depends on the size of the current window.
Previous Screen	Backward in the current buffer. The number of lines the cursor moves depends on the size of the current window.

Editing Command	Moves the Cursor to the Following Position
BOTTOM	End of the current buffer.
BUFFER	Puts the specified buffer in the current window and moves the cursor to the end of the buffer. Creates a new buffer if the specified buffer does not exist.
END OF LINE	End of the current line.
GET FILE	Creates a new buffer containing text of the specified file (or an empty buffer if the file does not exist) and puts the cursor at the beginning of the buffer. If entered a second time with the same file name during an editing session, EVE puts the existing buffer in the current window and positions the cursor at its last location in the buffer.
LINE	Beginning of the specified line in the current buffer.
MOVE BY PAGE	Next or previous page break, depending on the current direction.
MOVE BY WORD	Beginning of the next word, if direction is forward. In reverse direction, cursor moves to the beginning of the current word; if already there, EVE positions cursor at the beginning of the previous word.
NEXT WINDOW	Next window on your screen, assuming another exists. The cursor appears in the last location it occupied in that editing window.

Editing Command	Moves the Cursor to the Following Position
PREVIOUS WINDOW	Previous window on screen, assuming another window exists. The cursor appears in the last location it occupied in that editing window.
SET CURSOR BOUND	Changes cursor mode. Cursor follows the flow of text and cannot be put into an unused portion of the buffer. Similar to cursor behavior in EDT and other editors.
SET CURSOR FREE	The default mode. Cursor is not bound to the flow of the text but can be put anywhere on the screen and text can be entered.
START OF LINE	Beginning of the current line.
TOP	Beginning of the current buffer.

The following example shows how to move the cursor through a buffer. The example assumes that you created a file named OLDFILE.DAT in an earlier editing session.

Invoke EVE to edit the file OLDFILE.DAT using the command EDIT/TPU. EVE reads the contents of OLDFILE.DAT into a buffer and places the cursor at the beginning of the first line of text.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                | Insert | Forward
4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

Press CTRL/E to move the cursor to the end of the first line of text. CTRL/E works the same way in EVE as it does in DCL.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                | Insert | Forward
4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

Use the BOTTOM command to move the cursor to the end of the buffer. Press the DO key, type BOTTOM, and press RETURN.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                | Insert | Forward
Command:  BOTTOM
```

Press the up arrow to move the cursor to the beginning of the fourth line of text.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                   | Insert | Forward
```

Press the Forward Reverse key to change the current buffer direction to reverse. Press the Move By Line key to move the cursor to the beginning of the third line of text.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                   | Insert | Reverse
```

Press the DO key, type the command LINE 1, and press RETURN to move the cursor to the beginning of the first line of text.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                   | Insert | Forward
Command:  LINE 1
```

8.1.3.2 Inserting Text

You can insert sections of text, entire files, and special nonprinting characters (such as control characters) into the buffer you are currently editing. The following tables show the editing keys and EVE commands that you use while inserting text:

Editing Key	What It Does
Insert Overstrike	Changes the current editing mode as displayed on the highlighted status line. In insert mode, EVE inserts text at the current character position, moving existing text to accommodate the insertion. In overstrike mode, EVE overwrites text at the current position.
CTRL/A	Same as the Insert Overstrike key.
CTRL/V	Lets you insert nonprinting characters (or control codes) in a buffer. You can search for special characters using the Find key. First, press the Find key, then press CTRL/V and the special character to be found, and activate the search by pressing RETURN.

Command	What It Does
INCLUDE FILE	Inserts the entire contents of the specified file into a buffer at the line before the current cursor location.

Before you begin inserting text into a buffer, look at the highlighted status line to determine whether EVE is in insert or overstrike mode. If EVE is in insert mode, text is inserted at the cursor position, and text that already appears in the file moves to accommodate your insertions. If EVE is in overstrike mode, text that you type at the keyboard is inserted at the cursor position, and the text that already appears in the file is overwritten as the cursor moves through it.

Press CTRL/A or the Insert Overstrike key to change from one mode to the other.

You can add text to your buffer in the following ways:

- Text — Type characters that EVE adds to the buffer at the current cursor position. The characters are added according to the current mode of the buffer (insert or overstrike).
- Files — Add entire files by pressing the DO key and entering the EVE command INCLUDE FILE. Type the file specification at the *File to include:* prompt and press RETURN. EVE disregards the current mode (insert or overstrike) of the buffer and inserts the entire contents of the specified file into the buffer just before the line in which the cursor currently appears.

  Wildcards are allowed in the file specification. If there is more than one match for a file specification with a wildcard, EVE displays a list of choices and prompts you to provide a more complete file specification. If the specified file does not exist, EVE displays a message stating that it could not include the file.
- Special Nonprinting Characters — Add special nonprinting characters by pressing CTRL/V followed by the special character. For example, to insert an escape character into the buffer on a VT200-series or VT300-series terminal, press CTRL/V followed by CTRL/3. (On a VT100-series terminal, press CTRL/V and then press CTRL/[.) The special

characters are added according to the current mode of the buffer (insert or overstrike).

The following example shows how to insert text into a file, first in insert mode and then in overstrike mode. Invoke EVE to edit the existing file OLDFILE.DAT.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer  OLDFILE.DAT                                  | Insert | Forward
4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

Check the highlighted status line to ensure that EVE is in insert mode. Press the Insert Overstrike key (or CTRL/A) to change to insert mode, if necessary. Move the cursor to the first *s* in the word supervisor, type *Engineering,* and press the space bar.

The word Engineering is inserted in your text buffer, and the rest of the text on the line shifts to the right.

```
Schedule for 1 July
10:00 AM meeting with Engineering supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer  OLDFILE.DAT                                  | Insert | Forward
4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

Now press the Insert Overstrike key to change to overstrike mode. Move the cursor to the letter *D* in the word Donna and type *Andrea.*

The word Andrea is placed in the buffer, overwriting the word Donna.

```
Schedule for 1 July
10:00 AM meeting with Engineering supervisor
Read and review memo from Andrea
Work on Pascal program
[End of file]


Buffer  OLDFILE.DAT                                  | Overstrike | Forward
4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

8.1.3.3 Erasing and Restoring Text

With the EVE editor, you can easily delete text from a file or correct mistakes made during an editing session. If you erase text by mistake, you can restore the most recently erased text to its former location or, by moving the cursor, to another location. The following table shows the editing keys and EVE commands that erase and restore text:

Editing Key	What It Erases
⌫	Character to the left of the cursor.
Erase Word	Current word or, if the cursor is not on a word, erases the next word.
CTRL/U	All characters from the current cursor position to the beginning of the line.

Command	What It Does
ERASE CHARACTER	Erases the current character.
ERASE LINE	Erases from the current cursor position to the end of the current line, appending the next line to the end of the current line.
ERASE PREVIOUS WORD	Erases the previous word or the word the text cursor is on. If you are at the start of a line, you erase the carriage return at the end of the previous line, and the current line moves up. If you are between words or on the first character of a word, you erase the previous word. If you are in the middle of a word, you erase all of that word (same as ERASE WORD).
RESTORE	Restores, at the current cursor position, the word, sentence, or line that you have erased most recently with an EVE command or editing key. RESTORE does not restore single characters.
RESTORE CHARACTER	Restores, at the current cursor position, the character you have erased most recently with an EVE command or editing key.
RESTORE LINE	Restores, at the current cursor position, the line that you have erased most recently with an EVE command or editing key.
RESTORE WORD	Restores, at the current cursor position, the word that you have erased most recently with an EVE command or editing key.

To erase text from your buffer, move the cursor to the location of the text that you want to erase, and press the appropriate editing key, or type the appropriate EVE command.

The following example shows how to erase and restore text. Invoke EVE to edit the existing file RHYMES.DAT. (This example assumes that you created this file in an earlier editing session.)

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
```

Move the cursor to the letter *l* in the word *also*. Press the DO key, type the command ERASE LINE, and press RETURN.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
Commmand:  ERASE LINE
```

EVE erases all characters from the letter *l* to the end of the line and appends the next line to the current line.

```
She rhymes with tree,
a and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
```

Move the cursor to the letter *y* in the word *rhymes*. Press the DO key and enter the command ERASE WORD.

```
She rhymes with tree,
a and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
Command:  ERASE WORD
```

EVE erases the word *rhymes* and shifts the remaining text to the left.

```
She with tree,
a and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
```

Move the cursor to the space between the letter *a* and the word *and* on the second line. Press the DO key and enter the command RESTORE LINE.

```
She with tree,
a and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
Command:  RESTORE LINE
```

EVE restores the last line that was erased, in this case, *lso with bee,*.

Move the cursor to the letter *w* in the word *with* on the first line. Press the DO key and enter the command RESTORE WORD.

```
She with tree,
also with bee,
and this one makes three.
[End of file]


Buffer RHYMES.DAT                               | Insert | Forward
Command:  RESTORE WORD
```

EVE restores the last word that was erased, in this case, *rhymes*.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]


Buffer RHYMES.DAT                               | Insert | Forward
```

Section 8.1.3.4 describes the functions of the Select and Remove keys, which can be used together to erase text from a buffer.

8.1.3.4 Moving Text from One Location to Another

The following tables describe the functions of the Select, Remove, and Insert Here keys as well as the STORE TEXT command, which are used to erase text, to move text from one location to another within a buffer in "cut and paste" operations, and to duplicate text. For information on how to move text from one buffer to another, see Section 8.1.7.

Editing Key	What It Does
Select	Marks text (highlighting it in reverse video) from the cursor location to wherever you move the cursor. To cancel the selection, press the Select key again or use RESET.
Remove	Removes the text that was marked with SELECT, and places it in the Insert Here buffer.
Insert Here	Inserts the text from the Insert Here buffer at the current cursor location.

Command	What it Does
STORE TEXT	Copies text that was marked with SELECT or highlighted by FIND, placing it in the Insert Here buffer. Text that is copied is not removed from its original position.

To mark text when the buffer is set in a forward direction, place the cursor on the first character that you wish to erase. Press the Select key, and then move the cursor to one character beyond the last character that you wish to erase. (In reverse direction, move the cursor to the last character, not one beyond.) The text that will be erased is highlighted in reverse video. (If you decide not to remove text from the buffer, press the Select key again to cancel the selection.) Press the Remove key. EVE deletes the highlighted text from your screen and places it in the Insert Here buffer.

You can insert the text at any cursor location by pressing the Insert Here key, or you can erase text permanently from your buffer by leaving it in the Insert Here buffer. You can insert the text contained in the Insert Here buffer any number of times at any cursor location until you select a new section of text and put that new text in the Insert Here buffer using the Remove key or the STORE TEXT command. The Insert Here buffer contains whatever text was last copied or removed.

The following example shows how to erase and move text from one location to another using the Select, Remove, and Insert Here keys. Invoke EVE to edit the file RHYMES.DAT.

Move the cursor to the beginning of the second line of RHYMES.DAT and press the Select key.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                  | Insert | Forward
Move the cursor to select text.
```

Press the down arrow key once. The second line of text is highlighted. Press the Remove key. The second line of text is removed from the current buffer.

```
She rhymes with tree,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                  | Insert | Forward
Remove completed.
```

Press the down arrow key once. Press RETURN twice and then press the Insert Here key. The text in the Insert Here buffer is inserted at the current cursor location.

```
She rhymes with tree,
and this one makes three.

also with bee,
[End of file]

Buffer RHYMES.DAT                                  | Insert | Forward
```

The STORE TEXT command allows you to duplicate text in a file. Move the cursor to the first line of text and press the Select key. Press CTRL/E to move the cursor to the end of the first line and enter the STORE TEXT command. (Press DO, type STORE TEXT, and press RETURN.) The Insert Here buffer now contains a copy of the selected text. Now move the cursor to the line above *also with bee* and press the Insert Here key.

```
She rhymes with tree,
and this one makes three.

She rhymes with tree,
and also with bee,
[End of file]

Buffer RHYMES.DAT                                | Insert | Forward
Store text completed.
```

8.1.3.5 Locating Text

Use the Find key to locate specified strings of text in your buffer. Press the Find key (PF1 on VT100-series terminals). Then type the text string that you wish to locate, called the *search string,* and press RETURN.

EVE attempts to move the cursor to the beginning of the specified string.

If the search string contains all lowercase letters, EVE disregards the case of letters and locates any occurrence of the string. Thus, *the, THe,* and *thE* all match the search string *the.* If the search string contains one or more uppercase letters, EVE locates only the occurrences of the string in which the case of letters is exactly the same. Therefore, the only match for the search string *tHis* is tHis.

EVE is sensitive to diacritical (accent) marks and locates only those occurrences of the string in which diacritical marks are exactly the same. For example, in searching for ë, EVE does not locate occurrences of e, é, è, or ê.

The current direction of the buffer determines whether EVE first searches in a forward or a reverse direction.

If the editor cannot find the string in the current direction but finds it in the opposite direction, EVE prompts you to change direction. To search in the opposite direction, type Y. EVE moves the cursor to the first occurrence of the string in the opposite direction. The current direction in the highlighted status line is not changed, however.

When EVE finds the search string, the editor highlights it and moves the cursor to the first letter of the string. You can use any one of the following commands to modify a highlighted search string: REMOVE, FILL RANGE, STORE TEXT, LOWERCASE, UPPERCASE, and CAPITALIZE. To cancel the highlighting, move the cursor off the search string or enter the RESET command.

If you press the Find key twice, EVE tries to find the next occurrence of the search string.

The following example uses the existing file RHYMES.DAT to illustrate the use of the Find key. When you invoke EVE to edit RHYMES.DAT, the cursor appears on the first letter of the first line of the buffer, and the current direction is forward.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
3 lines read from file WORKDISK:[USER]RHYMES.DAT
```

Press the Find key, type the letters *ree,* and press RETURN. The cursor moves to the letter *r* in the word tree and highlights the letters *ree.*

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
Forward Find: ree
```

Press Find twice to find the next occurrence of the string *ree.* The cursor moves to the letter *r* in the word three and highlights the letters *ree.*

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
Finding previous target: ree
```

When a search string is found and highlighted, you can use any command that works on a selected range. For example, enter the UPPERCASE command.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
Command:  UPPERCASE
```

The UPPERCASE command changes the case of the highlighted letters from lowercase to uppercase.

```
She rhymes with tree,
also with bee,
and this one makes thREE.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
```

You can also use wildcards to search for a text string. Wildcard characters in a search string refer to a group of strings, rather than a specific string. By default, EVE searches for text using the VMS wildcard patterns, which include the asterisk (*) and percent sign (%). (To display all wildcard patterns for VMS, enter the SHOW WILDCARD command.) EVE can also search for text using ULTRIX patterns.

Move the cursor to the *s* in *She.*

To search for text strings ending in *ee,* enter the command WILDCARD FIND *ee.

```
She rhymes with tree,
also with bee,
and this one makes thREE.
[End of file]


Buffer RHYMES.DAT                                   | Insert | Forward
Command:  WILDCARD FIND *ee
```

EVE positions the cursor at the start of the line containing the *r* in *tree.*

The WILDCARD FIND command can also search for a string terminated by the Forward Reverse key. The key resets the initial direction of the search, overriding the current direction for the buffer.

You can specify how the FIND command treats the blank spaces between words such as spaces, tabs, and line breaks, which is called white space. By default EVE treats the white space in a search string literally and does not match across a line break. However, if you use the SET FIND WHITESPACE command before the search, the FIND command ignores white space when searching for a text string,

8.1.3.6 Marking Locations in Text

The MARK and GO TO commands are very useful when you are editing a large file and know that you want to return to a specific cursor location later in the editing session. The following table describes the MARK and GO TO commands:

Command	What It Does
MARK	Associates a unique and invisible label, consisting of one or more alphanumeric characters, with the current cursor location. The mark exists for the rest of an editing session.
GO TO	Returns the cursor to the location labeled by the MARK command. If the labeled location is contained in another buffer, EVE moves the cursor to the other buffer and places the buffer in the current window.

To mark a cursor location, press the DO key, type MARK label-name, and press RETURN. The label name can be one or more printable characters, including alphanumeric and punctuation characters. To return the cursor to the marked location, press the DO key, type GO TO label-name, and press RETURN.

The following example shows you how to use the MARK and GO TO commands to mark a cursor position with the label name FIRST and how to return to that cursor position.

Move the cursor to the letter *b* in the word bee. Press the DO key, type MARK and press RETURN. To mark the cursor location with the label FIRST, type FIRST at the *Mark name:* prompt.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]


 Buffer RHYMES.DAT                                    | Insert | Forward
Mark name: FIRST
```

Move the cursor to the letter *t* of the word three.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]


 Buffer RHYMES.DAT                                    | Insert | Forward
 Current position marked as FIRST
```

Press the DO key, type GO TO FIRST, and press RETURN to return the cursor to the position labeled FIRST.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]


 Buffer RHYMES.DAT                                    | Insert | Forward
 Going to mark: FIRST
```

8.1.3.7 Replacing Text

The REPLACE command allows you to replace a text string that appears in the current buffer with another text string. This is especially useful if you have spelled a word incorrectly throughout a long file, and you want to fix every occurrence.

To use the REPLACE command, press the DO key, type REPLACE, and press RETURN. Type the string that you wish to replace at the *Old string:* prompt and press RETURN. Type the new string at the *New string:* prompt and press RETURN.

If EVE finds the old string in the current direction, it moves the cursor to the first occurrence of the old string, highlights the string, and provides the following prompt: *Replace? Type yes, no, all, last, or quit.*

If EVE does not find the string in the current direction but finds it in the opposite direction, EVE provides the following prompt: *Found in 'reverse /forward' direction. Go there? [Y].* If you type Y, EVE moves the cursor to the first occurrence of the string in the new current direction, highlights the string, and provides the following prompt: *Replace? Type yes, no, all, last, or quit:* The current direction of the buffer in the highlighted status line is not changed.

Respond to the prompt by typing a single-character abbreviation of the response and pressing RETURN. Simply pressing RETURN is equivalent to typing Y and pressing RETURN. The following table explains the possible responses and actions when you type these responses:

Response	Replacement Procedure
Yes	Replaces the string and attempts to locate another occurrence of the string in the current direction. If found, the cursor moves to the next occurrence of the string, the string is highlighted, and EVE prompts: *Replace? Type yes, no, all, last, or quit:.* If the string is not found in the current direction but is found in the opposite direction, EVE prompts: *Found in 'reverse/forward' direction. Go there? [Y].*
No	Skips this occurrence and searches for another occurrence of the string in the current direction. If found, the cursor moves to the next occurrence of the string, the string is highlighted, and EVE prompts: *Replace? Type yes, no, all, last, or quit:.* If the string is not found in the current direction but is found in the opposite direction, EVE prompts: *Found in 'reverse/forward' direction. Go there? [Y].*
All	Replaces the string and all other occurrences of the string in the current direction. EVE leaves the cursor at the position where the first replacement occurred. After all occurrences of the string in the current direction have been replaced, EVE may inform you: *Found in 'forward/reverse' direction. Go there? [Y].* If you type yes, EVE replaces all occurrences of the string in the new direction.
Last	Replaces this occurrence of the string and stops the REPLACE procedure; the cursor does not move.
Quit	Does not replace this occurrence of the string and stops the REPLACE procedure; the cursor does not move. Pressing CTRL/Z has the same effect.

The REPLACE command is case sensitive. If the old string and the new string are given in all lowercase characters, then EVE matches the case appropriately for each replacement. For example, in replacing the string *parsley* with the string *dill,* EVE replaces a capitalized version of parsley with a capitalized version of dill. If the old string is lowercase, the search is general. However, if the old string is uppercase or mixed case, the search is case-exact. If the old and the new strings are lowercase, the replacement mirrors the case of the occurrence. Whereas, if the new string is uppercase or mixed case, the replacement is exact.

After finding the old string and prompting for the replacement of all occurrences of the string in both directions, EVE continues to search the buffer for the string. If EVE finds more occurrences, it informs you: *Found in forward/reverse direction. (May have already been replaced.) Go there [N]?* The default answer is N to prevent you from replacing the occurrences a second time.

The following example shows how to use the REPLACE command to replace every occurrence of the string *ee* with the string *oo.* Move the cursor to the top of the buffer. Press the DO key, type REPLACE, and press RETURN. Type *ee* at the highlighted *Old string:* prompt, press RETURN, and type *oo* at the highlighted *New string:* prompt.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]
Buffer RHYMES.DAT                                   | Insert | Forward
Old string: ee
```

The cursor moves to the highlighted string *ee* in the word tree. Type *all,* and press RETURN. All occurrences of the string *ee* are replaced with the string *oo.*

```
She rhymes with troo,
also with boo,
and this one makes throo.
[End of file]
Buffer RHYMES.DAT                                   | Insert | Forward
Replace? Type yes, no, all, last, or quit: all
```

8.1.4 Using the HELP Facility

EVE has an online HELP Facility that quickly supplies information on editing commands and keys during your editing session without disturbing your work. You can obtain help by entering the HELP command or by pressing the Help key.

To view a list of EVE commands, press the DO key and enter HELP. Use the Previous Screen and Next Screen keys (up and down arrow keys on a VT100-series terminal) to scroll through the entire list of EVE commands. To get information on a particular command, type a command name after the help prompt and press RETURN. The help text appears on the screen.

If you know the name of a specific command for which you need help, press the DO key, type HELP followed by the name of the command, and press RETURN. The help text for that command appears on the screen. For example, to receive help on the MOVE BY LINE command, enter the command HELP MOVE BY LINE. The following help text appears on your screen:

```
MOVE BY LINE

   Moves the text cursor a line at a time in the current direction (shown in
   the status line).

   o  In FORWARD direction, the cursor moves to the end of the current line
      or if already there, to the end of the next line (if any).

   o  In REVERSE direction, the cursor moves to the start of the current line
      or if already there, to the start of the previous line (if any).

   Keys:  By default, key F12 is defined as MOVE BY LINE.  The VT100 keypad
          defines MINUS on the keypad.

   Related topics:

       END OF LINE   LINE    START OF LINE   WHAT LINE


 Buffer HELP
Type topic name or ? for list. Press Return if done:
```

The HELP Facility also provides information on general topics. For example, if you choose to use the EDT or the WPS keypad, use the command HELP SET KEYPAD EDT or the command HELP SET KEYPAD WPS to get information on changing your default keypad. To display a list of all defined keys for the keypad you are using, enter the command HELP KEYS. There is also help on the difference between an EDT keypad within EVE and the original EDT keypad (HELP EDT DIFFERENCES).

The Help key produces a keypad diagram for the keypad you are using. The diagram shows both the default editing keys and the keys you have defined for the minikeypad (LK201 keyboard), the main keypad, the keys F10-F14 (LK201 keyboard), and the GOLD key (described in Section 8.1.9.3).

You can get help on particular editing keys after you display the keypad by pressing the desired key. If you press a key to which you have assigned an EVE command, EVE provides the help text for that EVE command.

8.1.5 Recovering from System Interruptions

EVE has recovery procedures for two types of system interruptions. You can remove extraneous characters that appear on your screen. You can also recover edits from an interrupted editing session with the journaling facility.

8.1.5.1 Refreshing the Screen

If extraneous characters, such as a message from the operator, appear on your terminal screen while you are editing or inserting text, press CTRL/W to refresh your screen. The screen becomes blank, and then all characters are redrawn, minus any extraneous characters.

8.1.5.2 Using the Journal File

If you are editing a file and a system interruption (that is, a break in communication between your terminal and the computer) occurs, you can recover your lost editing session. By default, EVE records the keystrokes you enter during an editing session in a journal file that has the same file name as the file you are editing and a file type of TJL.

Typically, an editing session ends without interruption, so the system deletes the journal file. When you experience a system interruption, however, the journal file is saved. EVE can use the journal file to reconstruct your editing session so that only the last few keystrokes of your editing session are lost.

To recover an editing session, enter the command you used to invoke EVE with the /RECOVER qualifier. For example, to recover an editing session you began with the command EDIT/TPU LETTER.RNO, type the following command and file name and press RETURN:

```
$ EDIT/TPU/RECOVER LETTER.RNO
```

You must recover an editing session at a terminal of the same type as the one you used for your editing session. When EVE finishes recovering the session, check to ensure that the last few keystrokes of your editing session were recovered and continue editing the file. If another system interruption occurs before you exit, a journal file containing the keystrokes from both editing sessions is saved.

The journal file is saved in the current default editing directory. However, you can create a journal file in another directory using the /JOURNAL qualifier:

```
$ EDIT/TPU/JOURNAL=[ALEXIS.JOURNEYS]LETTER.TJL LETTER.RNO
```

If you use the /JOURNAL qualifier to create a journal file with a different file name or a different directory, you must use the /JOURNAL qualifier and the file name when you recover the file. For example, to recover the file LETTER.RNO when the journal file is in directory [ALEXIS.JOURNEYS], enter the following command:

```
$ EDIT/TPU/JOURNAL=[ALEXIS.JOURNEYS]LETTER.TJL/RECOVER LETTER.RNO
```

The journaling facility has the following restrictions:

- If you use the WRITE FILE command during your editing session to copy the contents of the buffer to another file, you need to recover the original version of the file that you were editing. That is, you must specify the original version number in order to recover your file. For example, if you are editing an existing file called LETTER.RNO;1 and use the WRITE FILE command, EVE creates LETTER.RNO;2. If you experience a system interruption, you must enter the original version of LETTER.RNO on the EDIT/TPU/RECOVER command line. In this example it would be LETTER.RNO;1. See Section 8.1.7.4 for more information on the WRITE FILE command.
- EVE is usually unable to recover keystrokes entered after you press CTRL/C. If you press CTRL/C, end the editing session immediately with the EXIT command and invoke EVE again.

8.1.6 Formatting Text

EVE provides commands that enable you to format your text by setting margins, tabs, screen width, and word wrap. It allows you to center lines, take extra white space out of text, and insert page breaks. The following table lists text formatting commands and describes their functions:

Command	What It Does
CAPITALIZE WORD	Capitalizes a single word or each word in the text highlighted by FIND or SELECT.
CENTER LINE	Centers the line of text marked by the cursor between the current left and right margins. The text cursor moves with the line, remaining on the same character as the line moves.
FILL	Reformats the current paragraph or selected range according to the margins of the buffer, so the maximum number of words fits on a line.
FILL PARAGRAPH	Reformats the paragraph the text cursor identifies according to the margins set for the buffer.
FILL RANGE	Reformats the currently selected range of text (or the current FIND range) according to the current margin settings.

Command	What It Does
INSERT PAGE BREAK	Inserts a form feed character at the current editing position to mark the beginning of a new page. A page break appears as a small double-F and is always on a line by itself. By default, CTRL/L inserts a page break.
LOWERCASE WORD	Makes a single word or the text highlighted with FIND or SELECT all lowercase.
SET LEFT MARGIN	Sets the left margin in current buffer. The left margin must be greater than 0 but less than the right margin. By default, the left margin is 1.
SET RIGHT MARGIN	Sets the right margin for the current buffer. The right margin must be greater than the left margin. By default, the right margin is one less than the screen width. The width is typically 80, so the default margin is typically 79.
SET TABS AT	Sets tab stops at the columns that you specify. Columns are specified as a sequence of positive integers separated by spaces. By default, tab stops are set in every eighth column. This command does not affect the hardware tab settings of your terminal.
SET TABS EVERY	Sets tab stops at the specified interval. By default, tab stops are set in every eighth column. This command does not affect the hardware tab settings of your terminal.
SET TABS INSERT	Turns on tab insert mode, so that EVE sets a tab stop at the column where you press the Tab key, moving over text currently on the screen. SET TABS INSERT is the default mode.
SET TABS SPACES	Changes tab mode to insert an appropriate number of spaces rather than a tab character when the Tab key is pressed. Previously existing tab characters are not affected.
SET TABS MOVEMENT	Changes the tab mode so the Tab key becomes a cursor-control key. Pressing the Tab key moves the cursor to the next tab stop but does not insert a tab character.
SET TABS VISIBLE	Displays tabs as visible characters on the screen.
SET TABS INVISIBLE	Does not display tabs as visible characters on the screen. SET TABS INVISIBLE is the default mode.
SET WIDTH	Sets the width of lines displayed on the screen. Specify width as a positive integer *n*. By default, screen width is your terminal setting, typically 80 columns. If *n* is greater than 80, EVE sets the terminal to 132-column mode for the current editing session. When the EVE session is terminated, the terminal is restored to the default setting. Setting the width changes the display of text in all windows.

Command	What It Does
SET WRAP	Enables word wrapping at the right margin of the buffer so EVE starts new lines without a RETURN command or use of the FILL command. SET WORD WRAP is the default setting.
SET NOWRAP	Disables word wrapping at the right margin of the buffer. You must start new lines by pressing RETURN or by using the FILL command.
SHIFT LEFT	Moves the current window to the left a specified number of columns. The SHIFT LEFT command can be used only to reverse the effect of the SHIFT RIGHT command.
SHIFT RIGHT	Moves the current window to the right a specified number of columns, allowing you to view columns of characters that do not currently appear on the terminal screen.
UPPERCASE WORD	Makes a single word or the text highlighted with FIND or SELECT all uppercase.

The following example shows how to use EVE commands to set margins and screen width and to shift the current window: Invoke EVE to edit the existing file RHYMES.DAT. Press the DO key, type SET LEFT MARGIN 20, and press RETURN to set a left margin of 20. The text that currently appears in the buffer does not change.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
Command:  SET LEFT MARGIN 20
```

Move the cursor to the end of the buffer and type the following new text: *Also with thee, and me.* The new text that you enter is inserted at the left margin of 20.

```
She rhymes with tree,
also with bee,
and this one makes three.
                   Also with thee, and
                   me.
[End of file]

Buffer RHYMES.DAT                                      | Insert | Forward
Left margin set to 20
```

Reset the left margin to 1. Press the DO key, type SET LEFT MARGIN 1, and press RETURN. Again, the text that currently appears in the buffer does not change. When you insert new text, it is inserted at a left margin setting of 1.

```
She rhymes with tree,
also with bee,
and this one makes three.
                   Also with thee, and
                   me.
[End of file]

Buffer RHYMES.DAT                                 | Insert | Forward
Command:  SET LEFT MARGIN 1
```

Next, set the right margin to 25 by pressing the DO key, typing SET RIGHT MARGIN 25, and pressing RETURN.

```
She rhymes with tree,
also with bee,
and this one makes three.
                   Also with thee, and
                   me.
[End of file]

Buffer RHYMES.DAT                                 | Insert | Forward
Command:  SET RIGHT MARGIN 25
```

Enter the following new text in your file and notice that it wraps automatically to the next line at a right margin of 25.

```
She rhymes with tree,
also with bee,
and this one makes three.
                   Also with thee, and
                   me.
And free, and fee, and
see, and brie, and any
number of other words.

[End of file]

Buffer RHYMES.DAT                                 | Insert | Forward

Right margin set to 25
```

To reset the right margin to 79, press the DO key, type SET RIGHT MARGIN 79, and press RETURN.

```
She rhymes with tree,
also with bee,
and this one makes three.
                   Also with thee, and
                   me.
And free, and fee, and
see, and brie, and any
number of other words.

[End of file]

Buffer RHYMES.DAT                                 | Insert | Forward

Right margin set to 79
```

Now, set the width of your text window to 20 by pressing the DO key, typing SET WIDTH 20, and pressing RETURN.

```
 She rhymes with tre◆
 also with bee,◆
 and this one makes◆
 And free, and fee,◆
 see, and brie, and◆
 number of other wor▒

 [End of file]

 Buffer RHYMES.DAT                                  | Insert | Forward
Command:  SET WIDTH 20
```

The appearance of the current text window changes; all text beyond the twentieth column disappears from the screen. Press the DO key, type SHIFT RIGHT 5, and press RETURN to view five columns of text beyond the right boundary of the window.

The window shifts five columns to the right, and you can see characters that were not visible before the shift operation.

```
 ymes with tree,
 with bee,
 his one makes three.◆
               Also  ◆
               me.
 ree, and fee, and
 and brie, and any
 r of other words.▒

 [End of file]

 Buffer RHYMES.DAT                                 | Insert | Forward
Command:  SHIFT RIGHT 5
```

Shift the window to its original location by pressing the DO key, typing SHIFT LEFT 5, and pressing RETURN.

```
 She rhymes with tre ◆
 also with bee,
 and this one makes  ◆
 And free, and fee,  ◆
 see, and brie, and  ◆
 number of other wor ▒

 [End of file]

 Buffer RHYMES.DAT                                | Insert | Forward
Command:  SHIFT LEFT 5
```

Set the screen width to 80 by pressing the DO key, typing SET WIDTH 80, and pressing RETURN.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
                    Also with thee, and
                    me.
 And free, and fee, and
 see, and brie, and any
 number of other words.▒
 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward
Command:  SET WIDTH 80
```

EVE sets the width of lines on your screen to 80.

The following example shows how to fill a highlighted section of text and how to fill a paragraph. Using the existing file RHYMES.DAT, set the left margin to 5 and the right margin to 55.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
                    Also with thee, and
                    me.
 And free, and fee, and
 see, and brie, and any
 number of other words.

 [End of file]

 Buffer RHYMES.DAT                                    | Insert | Forward
Command: SET RIGHT MARGIN 55
```

Next, fill a highlighted section of text by selecting the first three lines of text (with the Select key) and entering the FILL command. Press the DO key, type the command FILL, and press RETURN.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
                    Also with thee, and
                    me.
 And free, and fee, and
 see, and brie, and any
 number of other words.

 [End of file]

 Buffer RHYMES.DAT                                    | Insert | Forward
Command:  FILL
```

EVE fills the highlighted text between the left margin of 5 and the right margin of 55.

```
    She rhymes with tree, also with bee, and this one
    makes three.
                    Also with thee, and
                    me.
 And free, and fee, and
 see, and brie, and any
 number of other words.

 [End of file]

 Buffer RHYMES.DAT                                    | Insert | Forward
```

For EVE, a paragraph is defined by blank lines, the top or bottom of the buffer, or page breaks. To compress text in a paragraph, put the cursor anywhere in the text of a paragraph, press the DO key, type the command FILL PARAGRAPH, and press RETURN.

```
   She rhymes with tree, also with bee, and this one
   makes three.
                   Also with thee, and
                   me.
And free, and fee, and
see, and brie, and any
number of other words.

 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward
Command:  FILL PARAGRAPH
```

EVE fills the paragraph according to the margins set for the buffer, in this case 5 and 55.

```
   She rhymes with tree, also with bee, and this one
   makes three. Also with thee, and me. And free, and
   fee, and see, and brie, and any number of other
   words.

 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward
```

To center a line of text, put the cursor anywhere on the line you want to center. For example, to center the text *words* in the last line of RHYMES.DAT, put the cursor on the *w*. Press the DO key, type the command CENTER LINE, and press RETURN.

```
   She rhymes with tree, also with bee, and this one
   makes three. Also with thee, and me. And free, and
   fee, and see, and brie, and any number of other
   words.

 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward
Command: CENTER LINE
```

The command centers the line of text between the current left and right margins; in this example, the left margin is set at 1 and the right margin is set at 55. The cursor moves with the line, remaining on the character w as the line is centered.

The EVE commands that change the case of text—CAPITALIZE, UPPERCASE, and LOWERCASE—work either on a selected range of text or on the word the cursor is on.

To change the case of the first line of text, move the cursor to the word *she* and press the Select key. To mark the end of the selection, move the cursor to the end of the line, press the DO key, and enter the command UPPERCASE.

```
   SHE RHYMES WITH TREE, ALSO WITH BEE, AND THIS ONE
   makes three. Also with thee, and me. And free, and
   fee, and see, and brie, and any number of other
                        words.

 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward
Command: UPPERCASE
```

To change the case of a particular word, position the cursor on the word, press the DO key, and enter the appropriate case-changing command. Once EVE changes the case of the word, the cursor moves to the next word or to the end of the line. If the cursor is between two words, the case of the word to the right of the cursor is changed, and the cursor moves to the end of the word.

8.1.7 Using Buffers

Buffers are storage areas that exist only during an editing session. The following table describes EVE commands used to create, manipulate, and delete buffers:

Command	What It Does
BUFFER	Puts the specified buffer in the current window and moves the cursor to the last location it occupied in that buffer. Creates a new buffer if the specified buffer does not exist.
DELETE BUFFER	Deletes the buffer you specify.
GET FILE	Creates a new buffer that contains the text of the specified file (or an empty buffer if you specify a file that does not exist); places the new buffer in the current window; and places the cursor at the beginning of the new buffer. If you specify the same file again during an editing session, GET FILE places the buffer in the current window. If you specify the same file name and file type with a different device or directory name during an editing session, EVE prompts you for a different buffer name into which to read the file.
GO TO	Returns the cursor to the location labeled by the MARK command. If the labeled location is contained in another buffer, EVE moves the cursor to the other buffer and places the buffer with the label in the current window.
SHOW	Displays information about the buffers you have created during the editing session. If more than one buffer is active in your editing session, EVE displays information about the buffer you are currently editing. For information on other buffers, press the DO key. To resume editing, press any other key.
SHOW BUFFERS	Lists the buffers you have created during an editing session. You can move the cursor through the list and specify a particular buffer for viewing using the Select key.
SHOW SYSTEM BUFFERS	Lists the system buffers created by EVE. You can move the cursor through the list and specify a buffer for viewing using the Select key.
WRITE FILE	Writes the contents of the current buffer to a file. If you do not specify a file name, EVE uses the buffer name as the file name. If you created the current buffer with the BUFFER command, EVE prompts you for a file specification.

When you invoke EVE to edit an existing file, EVE reads the contents of the file into a buffer. The highlighted status line contains the buffer name, editing mode, and current direction of the buffer.

To display more information about the current buffer, enter the SHOW command. The information displayed includes buffer name, name of the input and output files, whether the buffer has been modified, current mode and direction, number of lines, margin and screen-width settings, tab-stop settings, and marks that have been defined in the buffer. If more than one buffer is active during an editing session, you are prompted to press the DO key to receive information about the other buffers.

To delete a buffer, enter the DELETE BUFFER command, specifying the name of the buffer you wish to delete. For example, the command DELETE BUFFER MYFILE.TXT deletes the buffer called MYFILE.TXT. The buffer name must be typed in full; no abbreviations are allowed.

If the buffer (in this example, MYFILE.TXT) has been modified, EVE issues the following prompt to confirm that you want to delete the buffer:. *That's a modified buffer. Type delete_only, write_first, or quit.*

If you are viewing a buffer that you want to delete, EVE replaces the buffer with the oldest buffer existing in the editing session.

8.1.7.1 Listing Buffers

To display a list of all buffers you have created during an editing session, enter the SHOW BUFFERS command. You can scroll through the list and specify a buffer you want to view or any buffers you want to delete. To display a buffer in your current window, move the cursor to the buffer name and press the Select key. To delete a buffer, move the cursor to the buffer name and press the Remove key.

These applications of the Select and Remove keys apply only when you are viewing a list of buffers.

To display a list of all buffers that EVE has created, enter the SHOW SYSTEM BUFFERS command. You can scroll through the list and specify a buffer you want to view by moving the cursor to the buffer name and pressing the Select key. EVE puts the buffer in your current window.

Note: **Do not delete system buffers because these buffers are necessary for some commands to work properly.**

8.1.7.2 Displaying the Contents of the Messages Buffer

EVE uses the Messages window, which appears at the bottom of the screen, to communicate error and informational messages during an editing session. The Messages window displays the last message in the Messages buffer.

You can display these messages with the BUFFER command. To display the contents of the Messages buffer, enter the command BUFFER MESSAGES. To return to the buffer you were editing, enter the BUFFER command followed by the name of the appropriate buffer. For example, to return to the buffer named RHYMES.DAT, you would enter the command BUFFER RHYMES.DAT. Alternately, you can enter the SHOW BUFFERS command to display the buffers you have created. Use the Select key to specify the appropriate buffer.

8.1.7.3 Editing Two Buffers

During an editing session, you can use several buffers if you want to edit more than one file or if you want temporary storage areas for manipulating blocks of text. Multiple buffers are especially useful if you want to copy text from one file to another.

To create a new buffer, enter the GET FILE command and the name of the file you want to copy to the new buffer. You can use the asterisk wildcard character (*) as a substitute for all or some of the characters in the file name and file type. You can use the percent wildcard character (%) as a substitute for one character in the file name and file type. You can use the ellipsis wildcard ([. . .]) as a substitute for a directory specification.

If the specified file exists, EVE reads the contents of the file into a new buffer and displays the buffer in the current window. If there is more than one match for a wildcarded file specification, EVE displays a list of choices and prompts you to provide a more complete file specification. Otherwise, EVE creates an empty buffer and displays the buffer in the current window.

To change the buffer in the current window, press the DO key, type BUFFER and the name of the buffer you want to display on the screen, and press RETURN. If you forget a buffer name, enter the SHOW BUFFERS command to display the names of active buffers in your editing session, and specify a buffer with the Select key.

The following example shows how to use two buffers to edit two files during an EVE editing session. This example assumes that the original versions of files RHYMES.DAT and OLDFILE.DAT exist in your current default directory. Invoke EVE to edit the file RHYMES.DAT.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
3 lines read from WORKDISK:[USER]RHYMES.DAT
```

Press the DO key, type the command GET FILE OLDFILE.DAT, and press RETURN to create a new buffer that contains the most recent version of OLDFILE.DAT.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                  | Insert | Forward
```

Now that you have two buffers, practice copying from one buffer to another. Place the cursor on the letter *R* in the word *Read*, and press the Select key. Press the down arrow once and the third line of OLDFILE.DAT is highlighted. Press the Remove key to place the selected line in the Insert Here buffer.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]


Buffer OLDFILE.DAT                                  | Insert | Forward
Move the text cursor to select text.
```

The text from OLDFILE.DAT remains in the Insert Here buffer until you overwrite it with other selected text. Now switch to the other buffer to resume editing RHYMES.DAT. Press the DO key, type the command BUFFER RHYMES.DAT, and press RETURN.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
```

Press the Insert Here key. The text from the Insert Here buffer that was removed from the buffer OLDFILE.DAT is inserted in the top of the buffer RHYMES.DAT.

```
Read and review memo from Donna
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]

Buffer RHYMES.DAT                                   | Insert | Forward
```

If you exit from an editing session in which you have modified multiple buffers, EVE writes the contents of the current buffer to a file and then asks you whether you want to write each of the other modified buffers to files.

8.1.7.4 Reading and Writing Files

There are three ways to read a file into an EVE buffer:

- Invoke EVE with a file specification.
- Enter the INCLUDE FILE command and the name of the file you want to include. EVE reads the entire contents of a file into a buffer just before the line where the cursor is located. Using the INCLUDE FILE command does not change the name of the buffer on the status line.
- Enter the GET FILE command and the name of the file you want to use. This command creates a new buffer and reads the contents of an existing file into the buffer. The name of the buffer on the status line is the same as the file name you specified with the GET FILE command. (See Section 8.1.7.3.)

To write the contents of the current buffer to a file, enter the WRITE FILE command. You can include a file specification with the WRITE FILE command. If you do not include a file specification, EVE writes the file using the input file specification. If you created the current buffer with the BUFFER command, EVE prompts you for a file specification to which it writes the file.

If you have used the WRITE FILE command in an editing session and you experience a system interruption, you can recover the editing session. See Section 8.1.5 for information.

8.1.8 Using Windows

During an EVE editing session, the text buffer you are editing is displayed on the screen in a window. A highlighted status line appears at the bottom of a window identifying the name, current editing mode, and current direction of the buffer.

EVE allows you to view more than one window on your terminal screen at the same time. For example, you can have two windows in order to view and edit different sections of the same buffer.

The following table describes EVE commands used to create and manipulate windows:

Command	Effect in a Window Environment
SPLIT WINDOW	Splits the window that the cursor is in, forming two smaller windows. Adding an argument to the command allows you to divide the window into more than two parts. For example, SPLIT WINDOW 3 splits the window into 3 windows.
TWO WINDOWS	Synonymous with the SPLIT WINDOW command.
NEXT WINDOW	Puts the text cursor in the next (or other) window.
PREVIOUS WINDOW	Puts the text cursor in the previous (or other) window.
OTHER WINDOW	Synonymous with the NEXT WINDOW command.
ONE WINDOW	Restores the current window as a single, large window.
DELETE WINDOW	Deletes the window the text cursor is in, assuming you are using more than one window.
ENLARGE WINDOW	Enlarges the current window by a specified number of lines. For example, ENLARGE WINDOW 5 enlarges the window the text cursor is in by 5 lines. The adjacent window shrinks accordingly.
SHRINK WINDOW	Shrinks the current window by a specified number of lines. For example, SHRINK WINDOW 5 shrinks the window the text cursor is in by 5 lines. The adjacent window is enlarged accordingly.

8.1.8.1 Editing One Buffer

To view or edit two sections of a file at the same time, use the SPLIT WINDOW command. EVE splits your screen and creates two identical windows. The cursor maintains its position in the buffer but appears only in the bottom window. Notice that the buffer name in each of the status lines is the same.

Now you can edit different sections of a single file. Any edits that you make in one window are made simultaneously in the other window. Unless you are viewing two different sections of a file, you can see EVE incorporate edits simultaneously in the two windows.

Displaying two sections of a long file makes moving text within a file very efficient. You can select and remove text from one part of the file and insert it into the other. To move the cursor from one window to the other, enter the NEXT WINDOW command.

To remove the second window from the screen and expand the current window to occupy the whole editing area, press the DO key, type ONE WINDOW, and press RETURN.

8.1.8.2 Editing Two Buffers

The following steps describe how to edit two buffers containing different files:

1 Invoke EVE to edit a file.

2 Create two windows by entering the command SPLIT WINDOW. EVE splits your screen and creates two windows. The cursor maintains its position in the buffer but appears only in the bottom window. Notice that the buffer name in each of the highlighted status lines is the same.

3 Put a different buffer in the current window using either the GET FILE or the BUFFER command.

   To create a new buffer in the current window, use the GET FILE command with a file specification.

   Or, to display a buffer that you created earlier in the editing session in the current window, enter the BUFFER command and the name of the buffer you want to display.

   EVE replaces the current buffer with the buffer named with the GET FILE command or the BUFFER command.

4 Your terminal screen now displays two different buffers. You can select and remove text from one buffer and insert it into the other buffer. To move the cursor from one window to the other, enter the command NEXT WINDOW.

The following example shows you how to edit two files and move text from one file to another using two windows. First, invoke EVE to edit the file RHYMES.DAT.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

Buffer RHYMES.DAT                                     | Insert | Forward

3 lines read from file WORKDISK:[USER]RHYMES.DAT
```

Press the DO key, type the command SPLIT WINDOW, and press RETURN to create two windows on your screen.

```
She rhymes with tree,
also with bee,
and this one makes three.

 [End of file]

 Buffer RHYMES.DAT                                    | Insert | Forward

She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

 Buffer RHYMES.DAT                                    | Insert | Forward
Command:  SPLIT WINDOW
```

Press the DO key, type the command GET FILE OLDFILE.DAT, and press RETURN to create a new buffer containing the text of OLDFILE.DAT in the bottom window of your screen.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

Buffer RHYMES.DAT                                    | Insert | Forward

 Schedule for 1 July
 10:00 AM meeting with supervisor
 Read and review memo from Donna
 Work on Pascal program
 [End of file]


 Buffer OLDFILE.DAT                                   | Insert | Forward

 4 lines read from file WORKDISK:[USER]OLDFILE.DAT
```

Move the cursor to the letter *R* in the word *Read,* press the Select key, and press the down arrow twice. The last two lines in OLDFILE.DAT are highlighted.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

Buffer RHYMES.DAT                                    | Insert | Forward

Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Donna
Work on Pascal program
[End of file]

Buffer OLDFILE.DAT                                    | Insert | Forward

Move the text cursor to select text.
```

Press the Remove key to place the highlighted text in the Insert Here buffer.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]


Buffer RHYMES.DAT                                    | Insert | Forward

Schedule for 1 July
10:00 AM meeting with supervisor
[End of file]


Buffer OLDFILE.DAT                                   | Insert | Forward
Remove completed.
```

Enter the NEXT WINDOW command to move the cursor to the other window. Move the cursor to the bottom of the buffer and press the Insert Here key. The text that you removed from OLDFILE.DAT is inserted into RHYMES.DAT.

```
She rhymes with tree,
also with bee,
and this one makes three.
Read and review memo from Donna
Work on Pascal program
[End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward

 Schedule for 1 July
 10:00 AM meeting with supervisor
 [End of file]


 Buffer OLDFILE.DAT                                  | Insert | Forward
```

Enter the ONE WINDOW command to remove all other windows from the screen and expand the window containing the cursor to occupy the whole editing area of the screen. Press the DO key, type ONE WINDOW, and press RETURN.

If you exit from the editing session, EVE writes the contents of the current buffer to a file. EVE prompts *Write buffer?* if another modified buffer exists. To write the contents of the other buffer to a file, type Y.

8.1.9 Defining Keys

You can define keys to execute EVE commands or to enter a series of keystrokes.

EVE does not allow you to define the RETURN key (CTRL/M), the space bar, or any printing characters (such as letters, digits, and punctuation marks) on the main keyboard. In addition, DIGITAL recommends that you do not define the following keys and control key sequences:

DELETE <X]
F6 (VT200- and VT300-series)
Help (PF2 on VT100-series)
CTRL/C
CTRL/I (Tab key)
CTRL/R
CTRL/S
CTRL/T
CTRL/Q
CTRL/U
CTRL/X
CTRL/Y

You can define all other keys, including control keys. You can redefine the DO key, as long as you assign the DO command to another key.

8.1.9.1 Defining Keys to Execute an EVE Command

The DEFINE KEY command assigns an EVE command to a single key or control key sequence. You can, in effect, create your own editing keys to enter EVE commands that you use frequently. If you press a key to which you have assigned an EVE command, EVE provides the help text for that EVE command. Key definitions are discarded when you terminate an EVE editing session unless you use the SAVE EXTENDED EVE command (see Section 8.1.9.4) to save key definitions from one editing session to the next.

To define a key, do the following:

1. Press the DO key, enter the command DEFINE KEY, and press RETURN.
2. Type the EVE command that you want to assign to the key and press RETURN.
3. Press the key to be associated with the EVE command.

The message *Key defined* appears if you have successfully defined a key.

You can also assign EVE commands to keys by creating an initialization file. The initialization file contains EVE commands that EVE executes when you invoke the editor. Each command line in the initialization file should contain a DEFINE KEY command. The command syntax is as follows:

```
DEFINE KEY [=key-name] command
```

The first parameter is the key to be defined, and the second parameter is the command to assign to the key. For example, the following command line assigns the MOVE BY WORD command to keypad key 1:

```
Command: DEFINE KEY=KP1 MOVE BY WORD
```

The following command assigns the FILL command to CTRL/F:

```
Command: DEFINE KEY=CTRL/F FILL
```

You can use three different separators when specifying key names: an underscore, a hyphen, or a slash. For example, the CTRL/F key can appear as CTRL_F, CTRL-F, or CTRL/F.

To remove a key definition, use the UNDEFINE KEY command.

Section 8.1.9.3 contains more examples of defining keys to execute EVE commands.

8.1.9.2 Defining Keys to Enter a Learn Sequence

The LEARN command assigns a sequence of keystrokes called a *learn sequence* to a single key or control key. Learn sequences allow you to enter the same series of keystrokes in a buffer any number of times by pressing one key. If you press a key to which you have assigned a learn sequence, EVE provides a HELP message stating that you have defined the key; the HELP diagram labels as *sequence* any keys to which you have assigned a learn sequence, but does not describe what the key sequence is. All learn sequences are discarded when you terminate an EVE editing session unless you use the SAVE EXTENDED EVE command (see Section 8.1.9.4).

Define a learn sequence as follows:

1. Press the DO key, enter the LEARN command, and press RETURN.
2. Type the keystrokes to be remembered.
3. Press CTRL/R followed by the key to be associated with the learn sequence.

The message *Key sequence remembered* appears if you have successfully defined a key.

The following example shows how to define a learn sequence that will insert a string of text into your file when you press CTRL/F. Invoke EVE to edit the file RHYMES.DAT.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

Buffer RHYMES.DAT                                     | Insert | Forward
3 lines read from file WORKDISK:[USER]RHYMES.DAT
```

First, move to the end of the buffer. To begin the definition of the learn sequence, enter the LEARN command.

```
She rhymes with tree,
also with bee,
and this one makes three.

[End of file]

 Buffer RHYMES.DAT                                    | Insert | Forward
Command: LEARN
```

Insert the text *And what is a rhyme?* at the end of your file. This is the text that EVE is to remember.

```
She rhymes with tree,
also with bee,
and this one makes three.
And what is a rhyme?
[End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward

 Press keystrokes to be learned. Press CTRL/R to remember these keystrokes.
```

Press CTRL/R.

```
She rhymes with tree,
also with bee,
and this one makes three.
And what is a rhyme?
 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward

 Press the key that you want to use to see what was just learned:
```

Press CTRL/F, the key to which you are assigning the learn sequence.

```
She rhymes with tree,
also with bee,
and this one makes three.
And what is a rhyme?
 [End of file]

 Buffer RHYMES.DAT                                   | Insert | Forward

 Key sequence remembered
```

For the rest of the editing session, press CTRL/F to insert the text *And what is a rhyme?* at the current cursor position.

8.1.9.3 Defining a GOLD Key

You can assign two editing functions to one editing key if you create a *GOLD* key. One editing function is performed by pressing the editing key. The other function is performed by first pressing the GOLD key and then pressing the same editing key. To define a GOLD key, enter the SET GOLD KEY command and press the key you want to use as the GOLD key. Once defined, the message *GOLD key set.* appears in the Messages buffer.

Once you have defined a GOLD key, you can use the GOLD editing keys that EVE predefines (on VT200-series and VT300-series terminals). To see a diagram of these commands, enter HELP KEYPAD. The GOLD editing keys appear in reverse video.

You can also create your own key definitions using the GOLD key. The following example demonstrates how to define a GOLD key and assign two commands to a single key. The example defines the number 4 key on the numeric keypad as the GOLD key and then assigns the BOTTOM and TOP commands to the CTRL/G key. Thus, pressing CTRL/G alone enters the BOTTOM command, and pressing the GOLD key followed by CTRL/G enters the TOP command.

Invoke EVE to edit the file RHYMES.DAT. Define a GOLD key by pressing DO, typing SET GOLD KEY, and pressing RETURN.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]
 Buffer  RHYMES.DAT                                | Insert | Forward
Command:  SET GOLD KEY
```

Press the number 4 key on the numeric keypad.

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]
Buffer  RHYMES.DAT                                | Insert | Forward
Press the key that you want to use as the GOLD key:
GOLD key set
```

Press the DO key, type DEFINE KEY, and press RETURN.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]
 Buffer  RHYMES.DAT                                | Insert | Forward
Command:  DEFINE KEY
```

Type BOTTOM and press RETURN.

```
 And what is a rhyme?
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]
 Buffer  RHYMES.DAT                                | Insert | Forward
EVE command:  BOTTOM
```

Press CTRL/G.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]
 Buffer  RHYMES.DAT                                | Insert | Forward

 Key defined
```

Now define the GOLD CTRL/G key to enter the TOP command. Press the DO key, type DEFINE KEY, and press RETURN.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]
 Buffer  RHYMES.DAT                                | Insert | Forward
Command:  DEFINE KEY
```

Type TOP and press RETURN.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]

 Buffer  RHYMES.DAT                                | Insert | Forward
EVE Command:  TOP
```

Press the GOLD key (number 4 on the numeric keypad) and then press CTRL/G.

```
 She rhymes with tree,
 also with bee,
 and this one makes three.
 [End of file]

 Buffer  RHYMES.DAT                                | Insert | Forward
 Key defined
```

For the rest of your editing session, when you press CTRL/G, EVE executes the BOTTOM command; and when you press the GOLD key (number 4 on the numeric keypad) followed by CTRL/G, EVE executes the TOP command. If you press the GOLD key by mistake, press the Select key to cancel it.

You can define only one GOLD key at a time. To remove a GOLD key definition, enter the SET NOGOLD KEY command, then press the key you want to undefine. Alternatively, define another GOLD key, which removes the original GOLD key definition.

You can also define a GOLD key by inserting a command in an initialization file, using the following format:

SET GOLD KEY keyname

For example, the following command defines keypad key 4 as the GOLD key:

```
SET GOLD KEY KP4
```

To save key definitions from one editing session to the next, use the SAVE EXTENDED EVE command, described in Section 8.1.9.4.

8.1.9.4 Saving Key Definitions and Learn Sequences

The command definitions and learn sequences that you assign to keys extend the power of the EVE editor, making editing faster and more efficient. You can save your definitions cumulatively in a section file. EVE creates a section file automatically when you save definitions with the SAVE EXTENDED EVE command. Enter the SAVE EXTENDED EVE command before you terminate the editing session, using the following format:

SAVE EXTENDED EVE filename

The section file is saved in your current default directory unless you include a device and directory in the file specification. The file type defaults to TPU$SECTION. You can specify the same file specification each time you execute the SAVE EXTENDED EVE command. By doing this, you add any new key definitions and learn sequences to the same section file.

To use this extended version of EVE, you must include the /SECTION qualifier in the command line when you invoke EVE. For example, to invoke EVE to edit the file RHYMES.DAT using the section file WORKDISK:[USER]MYDEFS.TPU$SECTION, enter the following command:

```
$ EDIT/TPU/SECTION=WORKDISK:[USER]MYDEFS.TPU$SECTION RHYMES.DAT
```

Remember, you can define a command symbol to invoke this or any other lengthy command line. See the *Guide to VMS Text Processing* for further information on building section files.

8.1.9.5 Creating Initialization Files

Rather than defining keys or setting the characteristics of an editing session interactively, you can put EVE commands and key definitions in an initialization file. You can execute an initialization file when invoking EVE or during an editing session, by using the @ command; for example,

```
Command: @SETUP_INIT
```

An initialization file is an ASCII file containing standard EVE commands. When invoked, EVE executes an initialization file after the section file.

Each command in an initialization file begins on a separate line. You can add comments to the file to document it, as long as you precede the comments with an exclamation mark and place them on a line separate from a command. An initialization file has a file type of EVE. The following is an example of an initialization file:

```
     set tabs every 5
     set left margin 15
     set right margin 75
     overstrike mode
     define key=Ctrl/D erase word
     define key=Gold/W start of line
     define key=KP5 fill paragraph
     !
     !Binds the EDT forward function (KP4 on
     !EDT keypad) to GOLD F
     !
     define key=Gold/F EDT KP4
```

An initialization file can be specified with the /INITIALIZATION qualifier, defined as EVE$INIT in your LOGIN.COM file, or named EVE$INIT.EVE in your SYS$LOGIN directory. The following command invokes EVE with the initialization file named MY_INIT:

```
$ EDIT/TPU/INIT=WORK1:[ALEXIS]MY_INIT
```

By default, EVE uses the initialization file whose logical name is EVE$INIT. If you define this logical name in your LOGIN.COM file, EVE automatically uses your initialization file when you invoke the editor. For example, you could insert the following command in your LOGIN.COM file:

```
$ DEFINE EVE$INIT WORK1:[ALEXIS]MY_INIT.EVE
```

Since an initialization file is executed after a section file, the definitions in an initialization file override those in a section file. For this reason, place commands that define the editing environment in your initialization file. Commands that define the environment include the following:

- SET CURSOR BOUND or FREE
- SET FIND WHITESPACE or NOWHITESPACE
- SET KEYPAD
- SET GOLD KEY
- SET LEFT MARGIN
- SET RIGHT MARGIN
- SET SCROLL MARGINS
- SET TABS AT or EVERY
- SET TABS SPACES, MOVEMENT, or INSERT
- SET TABS VISIBLE or INVISIBLE
- SET WIDTH
- SET WILDCARD VMS or ULTRIX
- SET WRAP or NOWRAP
- The default mode of the buffer: CHANGE MODE, OVERSTRIKE MODE, or INSERT MODE
- The default direction of the buffer: CHANGE DIRECTION, FORWARD, or REVERSE

8.1.10 Using the TPU Command

EVE is an editor built on the VAX Text Processing Utility (VAXTPU), which is a programmable text processing utility. The TPU command allows you to enter any VAXTPU statement or series of statements that can be expressed on one command line.

To enter a VAXTPU statement, press DO, enter the command TPU followed by the VAXTPU statement you want to execute, and press RETURN. For example, to execute the TPU APPEND_LINE statement, which places the current line at the end of the previous line, enter the command TPU APPEND_LINE. For more information on the TPU command, type HELP TPU. See the *VAX Text Processing Utility Manual* for a complete list of VAXTPU statements and procedures.

8.1.11 Using DCL Within EVE

You can execute a DCL command from within EVE, or you can use a subprocess to switch between the DCL command level and an EVE editing session very quickly.

8.1.11.1 Executing a DCL Command

To execute a DCL command from within EVE, press the DO key, type the EVE command DCL and the DCL command you wish to execute, and press RETURN. The message *Creating DCL subprocess ...* appears in the Message buffer.

When the DCL command has executed, EVE creates another window, if necessary, and displays the DCL command and its output in the DCL buffer. (The cursor remains in the buffer it was in before you executed the DCL command.) You can move the cursor to the DCL buffer, select and remove text, and copy it to the editing buffer. Do not enter DCL commands that generate continuous output or run programs that do screen management of their own, such as the Phone Utility. The DCL command is most useful when you want to capture output in a buffer.

8.1.11.2 Creating a Subprocess

You can create a subprocess to switch between an EVE editing session and DCL command level without terminating your editing session. To create a subprocess, press the DO key, type the SPAWN command, and press RETURN. EVE suspends the current editing session and connects your terminal to a new VMS subprocess. The DCL prompt ($) appears on your screen.

While the most common reasons to spawn a subprocess are to invoke the Mail Utility and to run screen-oriented programs, your subprocess can invoke any VMS utility or execute any DCL command.

To return to your editing session, log out of the subprocess by typing the DCL command LOGOUT and pressing RETURN. EVE resumes the editing session, and the cursor appears in the location it occupied before you spawned the subprocess.

You can also supply a DCL command as a parameter to the SPAWN command to create a specific subprocess. For example, to execute the Mail Utility, press DO, type SPAWN MAIL, and press RETURN. The prompt for the Mail Utility appears on the screen (*MAIL>*). When you exit from the Mail Utility, you are automatically logged out of the subprocess and EVE resumes the editing session.

8.2 The EDT Editor

EDT is an interactive text editor. With EDT you can create a new file, insert text into it, and modify that text. You can also edit text in existing files.

EDT provides both line and keypad editing. In line editing, you type the editing command and the range of text you want the command to affect. In keypad editing, you move the cursor directly to the text you want to change and press keypad keys to enter the editing commands.

An efficient way to use EDT on a video terminal is to use the keypad as the primary editing mode in combination with various line-editing commands. You can also redefine certain keypad and control keys to perform editing functions not available in keypad editing. This chapter describes keypad editing as the main editing mode and includes supplementary line-editing commands and key definitions. (If you are using a hardcopy terminal, you can use only line-editing commands.)

See the *VAX EDT Reference Manual* for a list of the line-editing commands available with EDT.

8.2.1 Invoking and Terminating EDT

An editing session begins when you invoke EDT with the DCL command EDIT. In an editing session, you can create and edit a new file, or you can edit an existing file. The session ends when you enter the EXIT or QUIT command.

8.2.1.1 Invoking EDT

To invoke EDT, type the DCL command EDIT and specify as a parameter the file you want to edit. If the specified file already exists, EDT saves the existing versions and places a copy of the latest version in your buffer. (A buffer is the temporary storage area in which you edit text.) The existing versions of the file remain unchanged. For example, to edit an existing file named MEMO.TXT, enter the following command line:

```
$ EDIT MEMO.TXT

Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The first few lines of the latest version of the file appear on the screen. The cursor is positioned at the top of the screen, and EDT is ready to receive a keypad-editing command.

If you invoke EDT to create a file, the following message appears:

```
$ EDIT NEWFILE.TXT

[EOB]


Input file does not exist
```

Only the EDT message and the end-of-buffer symbol, [EOB], appear on the screen, and EDT is ready to receive keypad-editing commands. See Section 8.2.2.1 for a description of EDT line commands.

Note: **In the previous examples, you enter EDT in keypad (change) mode because a startup command file (SYS$LOGIN:EDTINI.EDT) containing the SET MODE CHANGE command has been executed. If this command is not executed in an EDT startup command file, you will enter EDT in line mode. See Section 8.2.7.1 for more information on EDT startup command files.**

8.2.1.2 Terminating EDT

To terminate an EDT session, press CTRL/Z. This puts you into line-editing mode. You can type EXIT or QUIT at the asterisk (*) prompt. QUIT terminates the editing session and does not save your edits. EXIT saves your edits in a new version of the file. (Note that the existing versions of a file remain unchanged regardless of how the editing session is terminated.)

To save your edited text, use the line-editing command EXIT to terminate EDT. When you enter the EXIT command, EDT creates an output file containing the edited version of the input file. By default, the output file has the same name and type as the input file, with the version number incremented by 1.

For example, if you enter the EXIT command after editing a file named MEMO.TXT;3, EDT creates a higher version named MEMO.TXT;4 as follows:

```
*EXIT
DISK1:[USER]MEMO.TXT;4  2 lines
$
```

To override the default output file name, enter the EXIT command with a new file specification as the parameter. For example, if you end the same editing session with EXIT MICE.TXT, EDT names the output file MICE.TXT;1, provided no other file named MICE.TXT exists.

```
*EXIT MICE.TXT
DISK1:[USER]MICE.TXT;1  2 lines
$
```

To terminate EDT without saving your edits, use the line-editing command QUIT. All edits you have made to the text are ignored, and no output file is created.

```
*QUIT
$
```

The QUIT command is a useful way to terminate EDT when you have opened a file by mistake. No new file version is created.

8.2.2 Entering EDT Commands

Enter most keypad-editing commands by pressing a keypad key. Enter line-editing commands by typing them after the line-editing prompt and pressing RETURN.

8.2.2.1 Entering EDT Line Commands

EDT prompts for line-editing commands with an asterisk. Line-editing commands usually operate on a range of one or more lines of text that you specify as a parameter for the command. For example, to display an entire file on your screen, enter the TYPE command and specify WHOLE as the parameter as follows:

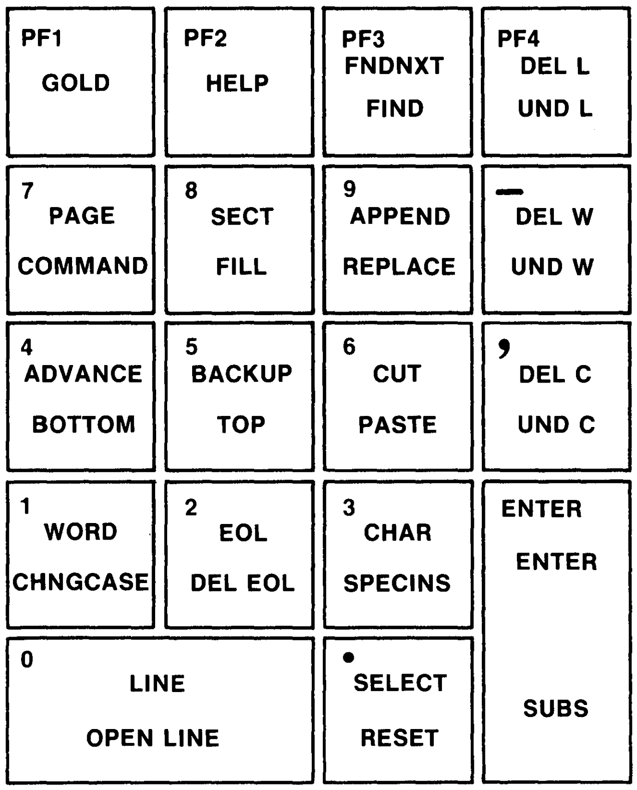
```
*TYPE WHOLE
```

You can abbreviate EDT line-editing commands. For clarity, the examples in this chapter show complete line-editing commands.

8.2.2.2 Entering Keypad Commands

In keypad editing, the screen displays editing changes as you make them. You type text from the main keyboard and enter keypad-editing commands from the numeric keypad. (To initiate keypad editing, you must first enter the line-editing command CHANGE or have SET MODE CHANGE in your EDT startup file. See Section 8.2.4.2 for information on the CHANGE command.)

The following figure shows the keypad keys and their functions.

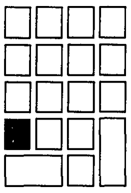


ZK-1688-84

Each key in the keypad performs at least one editing command; most perform two. Pressing a key invokes the regular, or upper, function. To invoke the alternate, or lower, function of a key, press the GOLD key (labeled PF1) first, followed by the desired key. In the examples that follow, a small diagram of the keypad highlights the key or keys that perform the command being described. The text associated with the keypad illustrates the effect of that editing command.

For example, key 1 performs both the WORD and the CHNGCASE functions. To invoke the WORD command, press WORD: the cursor moves to the beginning of the next word.

WORD

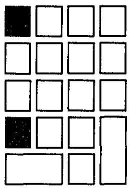


```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
```

To invoke the CHNGCASE command, press the GOLD key first and then CHNGCASE. The character at the cursor or the characters highlighted with the select key changes from lowercase to uppercase or from uppercase to lowercase.

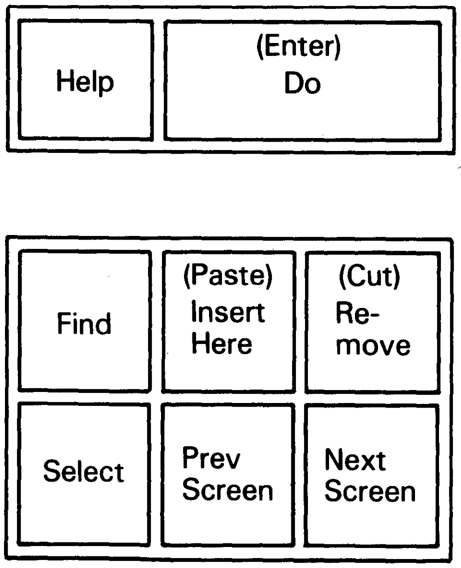
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
```

CHNGCASE



```
Once The weather turns cold, mice may find a crack in your foundation
and enter your house.  They're looking for food and shelter from the harsh
weather ahead.

[EOB]
```

The supplemental editing keys on the VT200 keypad perform the same functions as some of the EDT keypad keys. (See the *VAX EDT Reference Manual* for more information about these supplemental editing keys.)



ZK-1677-84

8.2.2.3 Canceling EDT Commands

Use CTRL/C to cancel the currently executing EDT command without affecting previous edits. For example, to stop the display of a long file, press CTRL/C.

```
*TYPE WHOLE
        .
        .
        .
CTRL/C
CANCEL
Aborted by CTRL/C
*
```

The display stops and the CTRL/C message appears.

8.2.3 Getting HELP in EDT

EDT provides a help facility for each of the EDT editing modes.

8.2.3.1 Getting HELP on Keypad-Editing Commands

To display a diagram of the keypad keys and their functions, enter change mode (assuming you are in line-editing mode) and then press the HELP key (labeled PF2). (On VT200-series terminals, you can also use the HELP key on the supplemental editing keypad.) To display information about a particular keypad command, first press the HELP key and then press the keypad key.

8.2.3.2 Getting HELP on Line-Editing Commands

To display a list of EDT topics on which information is available, type HELP and press RETURN. To display information about a particular command or topic, type HELP followed by the name of the topic and press RETURN. EDT responds with a display of information about the topic and a list of related topics about which information is available. To display information about the use of a particular command qualifier, type HELP plus the command and that qualifier and press RETURN. For example, to display information on the use of /QUERY with the COPY command, enter the following command line:

```
*HELP COPY /QUERY
```

8.2.3.3 Getting HELP on Nokeypad-Editing Commands

Nokeypad commands are used to construct key definitions. To display a list of the nokeypad-editing commands on which information is available, enter the following HELP command in line mode:

```
*HELP CHANGE SUBCOMMANDS
```

To display information about nokeypad entities (units of text upon which nokeypad-editing commands operate), enter the following HELP command in line mode:

```
*HELP CHANGE ENTITIES
```

8.2.4 Changing Editing Modes

You can easily switch back and forth between line and keypad editing; you can also enter line-editing commands from keypad mode. Before using keypad commands, be sure that your terminal type is set properly. (Use SHOW TERMINAL to display the setting and SET TERMINAL/INQUIRE to set the terminal type.)

8.2.4.1 Changing from Keypad to Line Editing

To change from keypad editing to line editing, press CTRL/Z. The asterisk prompt appears at the bottom of your screen, indicating EDT is ready to accept line-editing commands.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
CTRL/Z



*
```

8.2.4.2 Changing from Line to Keypad Editing

To change from line editing to keypad editing, enter the CHANGE command:

```
*CHANGE
```

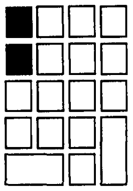
The first 22 lines of the file are displayed on your screen. If the file has fewer than 22 lines, the [EOB] symbol appears below the last line of the file.

8.2.4.3 Entering Line-Editing Commands from Keypad Mode

The keypad COMMAND function allows you to enter line-editing commands without leaving keypad mode. First, enter COMMAND (by pressing GOLD and then COMMAND) to invoke the *Command:* prompt, then type the line-editing command and press ENTER. (If you press RETURN by mistake, ^M appears; delete the ^M by pressing the DELETE key on the main keyboard, and press ENTER.) The following example enters the line-editing command SET QUIET, which suppresses the sound made when EDT issues an error message:

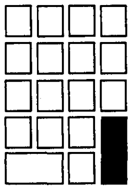
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
```

COMMAND



```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
Command:  SET QUIET
```

ENTER



8.2.5 Recovering from Interruptions

You can recover from interruptions to your editing session in the following ways:

- Deleting extraneous characters—Pressing CTRL/W removes extraneous characters (such as a broadcast message or a message indicating that you have received electronic mail) from the screen and restores the previous display. Use CTRL/W to ensure that the cursor is in the correct position.
- Resuming an interrupted editing session—The DCL command CONTINUE resumes an editing session that was interrupted by pressing CTRL/Y, so long as only built-in DCL commands were entered after pressing CTRL/Y. (See Chapter 1 for a list of built-in commands.) For example, you could press CTRL/Y, enter the command SHOW TIME, and return to your editing session with the CONTINUE command.

(Press CTRL/W to refresh the screen display. The text of your editing session is once again displayed.)

- Recovering a lost session—By default, EDT keeps a journal file with the same file name as the input file and a file type of JOU. If the editing session ends without interruption, the journal file is deleted when you terminate the session. If the editing session is aborted (for example, during a system failure, in response to pressing CTRL/Y, or entering the QUIT/SAVE command), you can recover your edits (with the exception of those commands entered just prior to the interruption). Enter the same command line you used to begin the editing session, adding the /RECOVER qualifier. For example:

```
$ EDIT/RECOVER MEMO.TXT
```

EDT will reproduce the editing session, reading the commands from the journal file and executing them on the screen.

8.2.6 EDT Keypad Editing

While line editing allows you to manipulate large portions of text easily, keypad editing provides easy manipulation of small units of text. Several EDT keypad commands enable you to find, insert, delete, substitute, and move text in a file. The cursor can be moved through a file in a variety of ways, and the position of the cursor in a file determines how text will be affected by EDT commands.

8.2.6.1 Manipulating the Cursor

You can manipulate the cursor with commands that move it unit by unit through the text or with commands that move it directly to a particular location. Several commands that move the cursor are controlled by the ADVANCE and BACKUP commands, which set the cursor's direction forward and backward. Unless otherwise stated, this chapter assumes the default direction of the cursor to be ADVANCE.

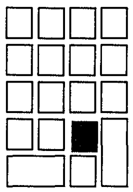
You can move the cursor by character, word, and line units.

Use one of the following keys to move the cursor by character:

- RIGHT ARROW — Moves the cursor one character to the right.
- LEFT ARROW — Moves the cursor one character to the left.
- CHAR — Moves the cursor one character in the current direction.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

CHAR

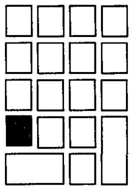


```
    Once the weather turns cold, mice may find a crack in your
    foundation and enter your house.  They're looking for food and
    shelter from the harsh weather ahead.
    [EOB]
```

The WORD command moves the cursor to the beginning of the next or previous word.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

WORD



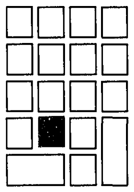
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The following keys move the cursor by line:

- UP ARROW—Moves the cursor up one line.
- DOWN ARROW—Moves the cursor down one line.
- EOL—Moves the cursor to the end of the current or previous line.

```
    Once the weather turns cold, mice may find a crack in your
    foundation and enter your house.  They're looking for food and
    shelter from the harsh weather ahead.
    [EOB]
```

EOL



```
    Once the weather turns cold, mice may find a crack in your
    foundation and enter your house.  They're looking for food and
    shelter from the harsh weather ahead.
    [EOB]
```

- F12 (the BACKSPACE key on VT100-series terminals)—Moves the cursor to the beginning of the previous line.

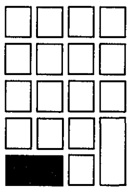
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.  F12 (BACKSPACE)
```

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
```

- LINE—Moves the cursor to the beginning of the next line or previous line.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
```

LINE

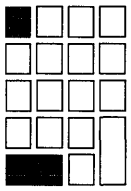


```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The OPEN LINE command terminates a line without moving the cursor. (The RETURN key also terminates a line, but moves the cursor to the next line.) The OPEN LINE command is useful when you want to insert a blank line or a new line of text. When the cursor is placed at the beginning of a line and the OPEN LINE command is entered, the text on that line is moved down so that the cursor is at the beginning of a blank line as follows:

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

OPEN LINE



```
Once the weather turns cold, mice may find a crack in your

foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

To move the cursor by large units, use the SECT and PAGE commands. The SECT and PAGE commands allow you to scan several lines of text at a time. The direction in which EDT moves depends upon whether ADVANCE or BACKUP is set.

- SECT—Moves the cursor across a 16-line section of text in EDT's current direction. If there are fewer than 16 lines, SECT moves the cursor across the existing lines.

  (On the VT200-series terminals, the supplemental editing keypad key Next Screen moves the cursor 16 lines forward, regardless of EDT's current direction. The supplemental editing keypad key Prev Screen moves the cursor 16 lines backward, regardless of EDT's current direction.)

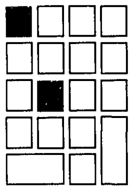
- PAGE—Moves the cursor to the next or previous page boundary (form feed) or to the end or top of the buffer if there is no boundary. To insert form feeds in your text, use CTRL/L.

The TOP and BOTTOM commands allow you to move directly to the beginning or end of a buffer. (See Section 8.2.6.8 for more information about buffers.)

- TOP—Moves the cursor to the beginning, or top, of the buffer.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

TOP

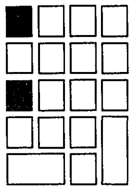


```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

- BOTTOM—Moves the cursor to the end, or bottom, of the buffer.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

BOTTOM



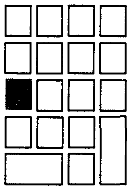
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The ADVANCE and BACKUP commands control the cursor's direction for the following EDT keypad-editing commands: CHAR, CHNGCASE, EOL, FIND, FNDNXT, LINE, PAGE, SECT, SUBS, and WORD. Each of the directional commands remains in effect until you set the cursor in the opposite direction with the other command.

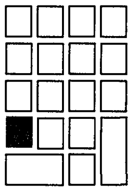
- ADVANCE—Sets the cursor's direction forward so that subsequent commands move the cursor in the forward direction. For example, if you enter the WORD command after using ADVANCE, the cursor moves forward one word.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

ADVANCE



WORD

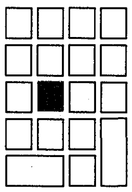


```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

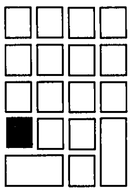
- BACKUP—Sets the cursor's direction in the backward direction so that subsequent commands move the cursor toward the top of the buffer. For example, if you enter the WORD command after using BACKUP, the cursor moves backward one word.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

BACKUP



WORD



```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The cursor remains set in the backward direction until you press ADVANCE. For example, if you enter a second WORD command in the preceding example you receive a message indicating that the command requests EDT to back up past the top of the buffer.

The ADVANCE and BACKUP commands are particularly important in string searches; see Section 8.2.6.4 for more information on searches.

8.2.6.2 Inserting Text

To insert text in EDT keypad editing, position the cursor where you want the text to be inserted and begin typing; the cursor remains one position to the right of the last character inserted. Inserting text in the middle of a line moves both the cursor and the remainder of the line one position to the right for each character inserted. When the line exceeds 80 characters, the text you type will either wrap to the following line or disappear off your screen, depending on the status of the SET SCREEN, SET [NO]TRUNCATE, and SET [NO]WRAP commands. (See Section 8.2.7.2 for information about screen formatting commands.)

8.2.6.3 Deleting and Restoring Text

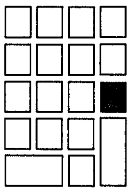
The delete commands work like the cursor movement commands. In EDT keypad editing, you can delete by character using the Delete key (⌫) (DELETE on VT100-series terminals) and DEL C; by word using F13 (LINEFEED on VT100-series terminals) and DEL W; and by line using DEL L, DEL EOL, and CTRL/U.

The deleted text is stored in a buffer so that you can also restore the character (UND C), word (UND W), or line (UND L) most recently deleted wherever and as many times as you need. Note that the undelete commands restore only the corresponding units of text that were most recently deleted. For example, if you have deleted two lines of text with the DEL L (delete line) command, the UND L (undelete line) command will restore only one line, the line most recently deleted.

The ⌫ key on the main keyboard (the DELETE key on VT100-series terminals) deletes the character immediately to the left of the cursor. The EDT keypad-editing command DEL C deletes the character directly at the cursor. The UND C command restores the last character deleted with either the ⌫ (DELETE) key or the DEL C command. For example:

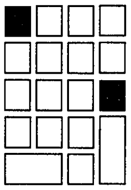
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

DEL C



```
nce the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

UND C

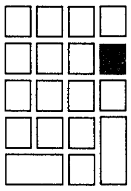


```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The F13 key on the main keyboard (the LINEFEED key on VT100-series terminals) deletes to the beginning of the current or preceding word. The DEL W command deletes to the end of the current word. Blank spaces are considered part of the word they follow, while all other word delimiters are considered to be separate words. The UND W command restores the last word deleted with either the F13 (LINEFEED) key or the DEL W command. For example:

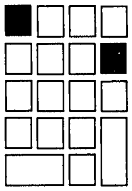
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

DEL W



```
Once the weather turns cold, may find a crack in your
foundation and enter your house.  They're looking for food
and shelter from the harsh weather ahead.
[EOB]
```

UND W



```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

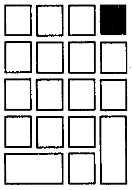
The following commands delete a line (or part of a line) of text:

- DEL L—Deletes from the cursor to the end of the line, including the line terminator. If the cursor is at the beginning of the line, the entire line is deleted, and the cursor is positioned at the beginning of the next line.
- DEL EOL—Deletes from the cursor to the end of the line (excluding the line terminator), leaving the cursor at the end of the truncated line.
- CTRL/U—Deletes from the cursor to the next previous beginning of line, leaving the cursor at the beginning of the previous line. (If CTRL/U is used when the cursor is at the beginning of the line, the previous line is deleted.)

The UND L command restores the last line (or part of a line) that was deleted with the DEL L, DEL EOL, or CTRL/U command. For example:

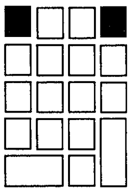
```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

DEL L



```
Once the weather foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

UND L



```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.
[EOB]
```

The EDT line-editing command DELETE is useful for deleting large sections of text. Generally, you use line numbers to specify a range for a line-editing command. For example, to delete lines 306 through 860, enter the following:

```
*DELETE 306 THRU 860
555 lines deleted
    861     Rodents have had a profound effect on human civilization.
*
```

Note that the EDT line-editing command SET NUMBERS (the default) must be in effect for line numbers to be displayed in EDT line editing.

You can also use certain keywords (such as WHOLE, REST, BEFORE) as range specifiers. For example, if you are in the middle of a long buffer and want to delete from the cursor to the end of the buffer, enter the following:

```
*DELETE REST
43 lines deleted
[EOB]
*
```

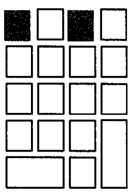
(You can also specify range by using the EDT keypad-editing command SELECT. See Section 8.2.6.7 for information on SELECT.)

8.2.6.4 Locating Text

You can move the cursor to a character string you specify with the FIND and FNDNXT EDT keypad-editing commands. The FIND command searches for the specified character string between the current position of the cursor and the beginning or end of the buffer (depending on whether the ADVANCE or the BACKUP command is in control). EDT does not distinguish between uppercase and lowercase letters unless you use the SET SEARCH EXACT line-editing command. When EDT finds the string, it positions the cursor at the first character in the string (unless the SET SEARCH END command is in effect, and the cursor is positioned at the last character in the string). In a long file the message "Working" may flash on the screen while EDT searches for the string.

For example, to delete a comma after the word "house" in the following text, you can use the FIND command to move the cursor to the string "house." First, enter the EDT keypad command FIND by pressing the GOLD key and then the FIND key (on the VT200-series terminal you can also use the FIND key located on the supplemental editing keypad). Next, type the string you want to locate (the search string) after the Search for: prompt.

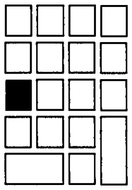
FIND



```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.  Once inside, they may gnaw
through electrical wires and raid your food. Because mice reproduce
so quickly, what started as one or two mice can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
Search for:   house
```

To search in the forward direction, use the ADVANCE command to enter the search string.

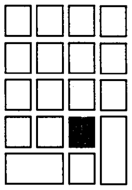
ADVANCE



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  Because elephants reproduce so
quickly, what started as one or two elephants can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
```

Use the CHAR command to move the cursor to the comma after the word "house". Then use the DEL C command to delete the comma.

CHAR



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  Because elephants reproduce so
quickly, what started as one or two elephants can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
```

To find the next occurrence of the string located with the FIND command, use the FNDNXT (find next) command. If there is no other occurrence of the string (as in the example above), EDT issues the message "String was not found."

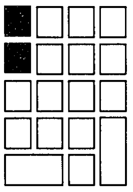
Note: **Note that the directional setting of the cursor determines the direction of the search. After you press FIND, you can press either ADVANCE or BACKUP (depending on the direction in which you want to search) to enter the search string. You can also use the ENTER command, which applies the current direction to the search.**

8.2.6.5 Substituting Text

To substitute one character string for another, you can use the SUBS keypad-editing command or the SUBSTITUTE line-editing command. The EDT line-editing command can make global substitutions; that is, it can replace every occurrence of one character string in the specified range with another string using only one EDT line-editing command. In contrast, you must use the keypad SUBS command (press the GOLD key followed by the SUBS key) for each substitution you make. (If you do not specify a range, the line-editing command SUBSTITUTE replaces only the first occurrence of the search string in the current line with the substitute string.)

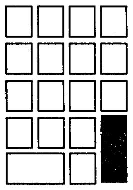
For example, to substitute the string "mice" for "elephants" throughout a buffer, enter the line-editing command SUBSTITUTE, the old string, and the new string, separating all three with the same delimiter. You can use any nonalphanumeric character (except the percent sign and underscore) as a delimiter for the SUBSTITUTE command, as long as the delimiting character is not part of either string. To apply the command to the entire buffer in a global substitution, specify WHOLE as the parameter. When the operation has been completed, EDT displays each occurrence of the substitution and the total number of substitutions. The following example substitutes the string "mice" for each occurrence of the string "elephants" in the following text:

COMMAND



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  Because elephants reproduce so
quickly, what started as one or two elephants can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
Command:  SUBSTITUTE\mice\elephants\WHOLE
```

ENTER



```
1   Once the weather turns cold, elephants may find a crack
4   in your electrical wires and raid your food.  Because elephants reproduce
5   so quickly, what started as one or two elephants can quickly become an
3 substitutions
Press return to continue
```

```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  Because elephants reproduce so
quickly, what started as one or two elephants can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
```

Note that a global substitution replaces all occurrences of the string, regardless of case or surrounding characters. If you want EDT to search for exact comparisons of case, use the SET SEARCH EXACT command. If the search string occurs in the middle of a longer string, the substitution will still be made. For instance, a global substitution of "IN" for "AT" would change all words containing the string "AT". ("LATER" would become "LINER", "THAT" would become "THIN", "SAT" would become "SIN", and so on.)

To get EDT to prompt you before each substitution, use the /QUERY qualifier with the SUBSTITUTE command.

```
Command:  SUBSTITUTE\AT\IN\WHOLE/QUERY
```

EDT prompts you with a ? to verify each substitution. You can respond with one of the following:

Y	Yes, do the substitution.
N	No, do not do the substitution.
Q	Quit, terminate the command.
A	All, do the rest of the substitutions without query.

8.2.6.6 Moving Text

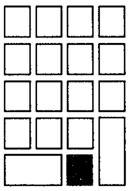
Both EDT keypad and line commands can move text; however, only line-editing commands transfer text between buffers and files.

8.2.6.7 Moving Text Within the File

The EDT keypad-editing command CUT deletes a selected range of text and the PASTE command inserts it at the cursor's current position. (On the VT200-series terminals, the supplemental editing keys Remove and Insert Here perform the same functions as the EDT keypad commands CUT and PASTE.) For instance, to move the first sentence in the second paragraph of the example to the end of that paragraph, move the cursor to the beginning of the sentence and press SELECT. (On the VT200-series terminals, the supplemental editing key SELECT performs the same function as the EDT keypad command SELECT.) This marks the beginning of the selected range. (You can cancel the SELECT command with the RESET command.)

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.  Once inside, they may gnaw
through electrical wires and raid your food. Because mice reproduce
so quickly, what started as one or two mice can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
```

SELECT

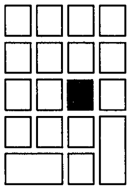


To mark the end of the selected range, move the cursor to the end of the sentence. The terminal highlights a selected range in reverse video. (The selected range includes the text up to the character preceding the cursor.)

```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  Because elephants reproduce so
quickly, what started as one or two elephants can quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent these rodents from ever getting in.
[EOB]
```

Press CUT to delete the selected text.

CUT

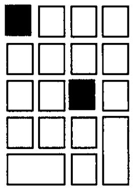


```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in.
[EOB]
```

Deleted text remains in the PASTE buffer until you perform another CUT operation. To restore the text, move the cursor to the appropriate position and enter the PASTE command. (The text will be inserted directly in front of the cursor.)

```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in.
[EOB]
```

PASTE

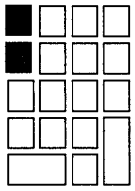


```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.
[EOB]
```

Because the selected text is held in the PASTE buffer until you perform another CUT operation (or give the line-editing command CLEAR PASTE), you can paste the text contained in the PASTE buffer as many times as you want. You can also enter the PASTE buffer to edit the text it contains. (See Section 8.2.6.8 for information on using multiple buffers.)

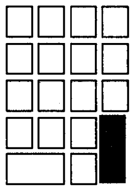
After moving the text, you may want to use the FILL command to reorganize selected text so that the maximum number of whole words are fitted within the current line width. The default line width is 80 characters, but you can use the SET WRAP command to use another line length for filling text. For example, you can set the line length to 71 characters with the EDT line-editing command SET WRAP and then fill a selected range of text.

COMMAND



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.
[EOB]
Command:  SET WRAP 71
```

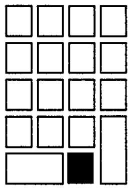
ENTER



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.
[EOB]
```

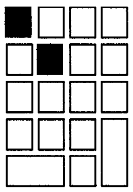
EDT will now wrap lines of inserted text and fill lines of selected text at a line width of 71 characters. Use the SELECT command to mark the text you want to affect and then enter the EDT keypad command FILL.

SELECT



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in.  Because elephants reproduce so quickly, what started as
one or two elephants can quickly become an invasion.
[EOB]
```

FILL



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.   If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.
[EOB]
```

There are several EDT line-editing commands that move text. For example, the MOVE and COPY commands each perform a function similar to those of the keypad CUT and PASTE operations. MOVE deletes text from one location and inserts it in another; COPY inserts a copy of the text where specified without deleting any text. The EDT line-editing commands INCLUDE and WRITE perform tasks not possible with EDT keypad-editing commands:

- INCLUDE—Copies a file into the buffer you are currently editing or the buffer you specify. Follow the VMS conventions for file specifications when specifying the file to be copied to the buffer. For example, the following command copies the file named MEM.DAT to the buffer named BUF1:

```
Command:  INCLUDE MEM.DAT =BUF1
```

- WRITE—Copies a specified range of text from a buffer (the current buffer by default) to a specified file. If you do not specify a range, the WRITE command copies the entire contents of the current buffer. For example, the following command copies the contents of the current buffer to the file ANIMALS.TXT:

```
Command:  WRITE ANIMALS.TXT
$DISK1:[USER]ANIMALS.TXT;1  11 lines
```

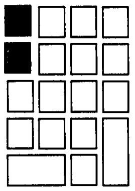
The message displays the new file specification and length.

8.2.6.8 Using Multiple Buffers

When you begin editing a file with EDT, you are working on a copy of the file in a buffer called MAIN. (EDT also uses a buffer called PASTE to store the text that you delete with the CUT and APPEND commands; you can edit this buffer just as you can edit other text buffers.) You can create other buffers to store pieces of text during your EDT editing session. You can enter and edit these buffers; you can copy text to and from them; and you can write their contents to specified files.

To create a buffer, press the COMMAND key. Type the line-editing command FIND followed by the equal sign and the name you are giving the buffer, then press the ENTER key. For example, the following command creates a buffer named BUF1:

COMMAND



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.
[EOB]
Command:  FIND=BUF1
```

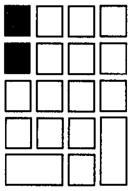
When you enter this command, the system responds by displaying only the [EOB] symbol, which indicates that the current buffer, BUF1, is empty. You can now insert and edit text just as you would in the MAIN buffer. To return to the MAIN buffer, follow the same procedure, typing FIND=MAIN rather than FIND=BUF1. To return to your previous position in the MAIN buffer, include a period after the buffer's name as follows:

```
Command:  FIND=MAIN.
```

The buffer named BUF1 remains intact until you exit from EDT, regardless of whether you enter the EXIT or QUIT command. That is, you can enter, edit, and exit from a buffer as necessary. However, when you exit from EDT, only the buffer MAIN is saved.

The SHOW BUFFER command displays the number of lines contained in each buffer and indicates (with an equal sign) the current buffer. The following example indicates that there are three buffers (including MAIN and PASTE, which always exist) and that MAIN is the current buffer:

COMMAND

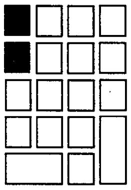


```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.


[EOB]
Command:  SHOW BUFFER
=MAIN  11   lines
 PASTE  3   lines
 BUF1   2   lines
Press return to continue
```

Pressing the RETURN key returns the cursor to its previous position in the buffer.

You can further manipulate the contents of a buffer by specifying the buffer's name in an EDT line-editing command. For example, if you are in the MAIN buffer and want to save the contents of BUF1 in a file named RODENT.TXT before exiting from EDT, enter the following command:

COMMAND



```
Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and shelter
from the harsh weather ahead.  Once inside, they may gnaw through
electrical wires and raid your food.  If you seal the cracks and holes
on the exterior of your house, you can prevent these rodents from ever
getting in. Because elephants reproduce so quickly, what started as one
or two elephants can quickly become an invasion.
```

```
[EOB]
Command:   WRITE RODENT.TXT =BUF1
$DISK1:[USER]RODENT.TXT;1  2 lines
```

EDT returns a message indicating that the file has been created, and the cursor is returned to its previous location in the buffer.

8.2.7 Controlling EDT Sessions

You can control some of the characteristics of an EDT editing session with the SET commands. You can redefine the functions of many keys by using the EDT line-editing command DEFINE KEY (or CTRL/K in keypad editing) plus one or more EDT nokeypad-editing commands. (EDT nokeypad-editing commands perform keypad operations; you can combine them in key definitions to extend your editing capacity. See the *VAX EDT Reference Manual* for a list of EDT nokeypad-editing commands.) You can also define a macro (a sequence of line-editing commands) and define keys in EDT. You can enter these control commands interactively, or you can include them in an EDT startup command file.

8.2.7.1 Startup Command Files

An EDT startup command file contains EDT line-editing commands that are executed when you invoke EDT before you receive control of the editor. A startup command file is useful for setup operations in EDT; it can include specifications for screen format, definitions of text entities, and definitions of keys and macros. Generally, EDT reads a systemwide startup command file at the beginning of your editing session. If no systemwide startup command file exists on your system, EDT looks for a file named EDTINI.EDT in your default directory and processes the commands in that file.

To create an EDTINI.EDT file, invoke an editor and specify EDTINI.EDT as the file specification as follows:

```
$ EDIT EDTINI.EDT
```

Now list the commands, one per line. Some typical commands you might want to put in a startup command file follow:

```
SET QUIET
SET WRAP 60
SET SEARCH BOUNDED
SET TAB 5
DEFINE KEY GOLD P AS "PAR."
SET MODE CHANGE
```

When you exit from the editor, EDTINI.EDT is saved in your default directory. Every time you invoke the editor, the commands in your EDTINI.EDT file are in effect.

To specify an EDT startup command file named something other than EDTINI.EDT, you must include the file specification in the EDIT command line n as follows:

```
$ EDIT/COMMAND=startup-file-spec  file-spec
```

For a list of EDT line and nokeypad editing commands, see the *VAX EDT Reference Manual.*

8.2.7.2 Controlling Screen Format with SET Commands

Several EDT commands control the format of a screen display. Some are listed below. See the *VAX EDT Reference Manual* for a comprehensive list of the SET commands.

- SET LINES n—Controls the number of lines that EDT displays on the screen. This number, which can be set from 1 to 22, defaults to 22. To set the screen to 15 lines, for example, type:

  ```
  Command: SET LINES 15
  ```

  Note that if you are editing at slow baud rates, setting the number of lines low will increase your editing speed.

- SET SCREEN width—Controls the maximum length of the line EDT displays; the default width is 80 characters. (When there are more characters than the SET SCREEN command specifies, EDT displays a diamond at the end of the line.)

  ```
  Command: SET SCREEN 132
  ```

  If you use the SET SCREEN command to make the screen wider than 80 on either a VT100- or VT200-series terminal, EDT changes the terminal's screen width to 132.

- SET [NO]TRUNCATE—Controls whether the characters that exceed the SET SCREEN width are displayed on the next line. The default is SET TRUNCATE, which ends the display of a line at the value of SET SCREEN.

  ```
  Command: SET [NO]TRUNCATE
  ```

- SET [NO]WRAP n—Specifies n character positions as the point at which text will be moved to the beginning of the next line. When you are inserting text in EDT keypad mode and the cursor position exceeds the value of n, EDT wraps the next full word to the next line. (However, when you insert text in the middle of a line, that line does not always wrap.) The default is NOWRAP. To wrap the text exceeding 75 characters, for example, type:

  ```
  Command: SET WRAP 75
  ```

SET commands have corresponding SHOW commands; see the *VAX EDT Reference Manual* for a list of SHOW commands.

8.2.7.3 Controlling Editing Functions with SET Commands

Several commands control EDT's responses during an editing session, as follows. (See the *VAX EDT Reference Manual* for a comprehensive list of the SET commands.)

- SET ENTITY—Defines boundaries for the WORD, SENTENCE, PARAGRAPH, and PAGE entities. (The SENTENCE and PARAGRAPH entities are not used by any default key definitions; consequently, they are useful only in the key definitions you create with the DEFINE KEY command.) For example, the default boundaries for the WORD entity are a line feed, tab, form feed, line terminator, and space. To make the period and comma the only delimiters of the word entity, enter the following SET ENTITY command:

  ```
  Command: SET ENTITY WORD '.,'
  ```

- SET MODE—Controls the EDT editing mode to be entered when the processing of the EDTINI.EDT file is completed (either line or change mode, which is keypad mode). For example, to enter change mode instead of line mode at the beginning of editing sessions, insert the following command at the end of your EDT startup command file:

  ```
  SET MODE CHANGE
  ```

- SET QUIET—Suppresses the sound made when EDT issues an error message in keypad mode. The default is NOQUIET.

8.2.7.4 Defining Keys

To redefine a key, assign one or more EDT nokeypad-editing commands (listed in the *VAX EDT Reference Manual*) to the key with the DEFINE KEY command:

```
Command:  DEFINE KEY key AS "command(s)"
```

You can redefine all keypad keys. You can define any GOLD keyboard key sequence except the keyboard digits, minus sign, and ⌫ key (the DELETE key on VT100-series terminals). See the *VAX EDT Reference Manual* for a diagram of the keypad key numbers you use to define keypad keys. Although you can define many keys as control keys, do not redefine the specialized control keys CTRL/C, CTRL/M, CTRL/O, CTRL/P, CTRL/Q, CTRL/R, CTRL/S, CTRL/T, CTRL/U, CTRL/Y, and CTRL/Z.

The following example of a key definition redefines the GOLD key and the S key combination to perform a global substitution, with prompts for the search string and the replacement string:

```
*DEFINE KEY GOLD S AS "EXT S/?*'REPLACE: '/?*'     WITH: '/WHOLE."
```

The string within quotation marks consists of an EDT line-editing command (EXT, which tells EDT that the rest of the line is an EDT line-editing command); an EDT line-editing command (SUBSTITUTE, abbreviated "S"); its qualifier (WHOLE); prompts (the question marks followed by the prompts REPLACE and WITH, and the asterisk (*) directly following the question mark prompt, allowing you to use either ENTER or RETURN to enter your response). Note that the period at the end of the definition (before the final quotation mark) is necessary to make the command execute immediately when the key is pressed. Subsequent use of GOLD S performs the substitution, prompting for old and new strings and displaying the substitutions.

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.  Once inside, they may gnaw
through electrical wires and raid your food.  Because mice reproduce
so quickly, what started as one or two mice could quickly become an
invasion.  If you seal the cracks and holes on the exterior of your
house, you can prevent the rodents from ever getting in.
[EOB]
GOLD + S
REPLACE: mice ENTER  WITH: elephants ENTER

Once the weather turns cold, elephants may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.  Once inside, they may gnaw
through electrical wires and raid your food.  Because elephants
reproduce so quickly, what started as one or two elephants could
quickly become an invasion.  If you seal the cracks and holes on the
exterior of your house, you can prevent the rodents from ever getting in.
```

Placing key definitions (such as the GOLD + S definition) in an EDT startup command file makes the redefined keys available during every editing session.

8.2.7.5 Defining EDT Macros

An EDT macro allows you to execute a sequence of EDT line-editing commands whenever you invoke the macro. To define a macro, use the EDT line-editing command DEFINE MACRO to define the name of a buffer as the macro name. Then create and enter a buffer with the same name as the macro. (See Section 8.2.6.8 for information about using multiple buffers.) Once in the buffer, type the EDT line-editing commands in the desired sequence, one command per line. For example, the following macro inserts a four-line heading.

```
INSERT;NAME:
INSERT;DEPT:
INSERT;DATE:
INSERT;SUBJ:
[EOB]
```

Then exit from the buffer. To invoke the macro, enter its name as an EDT line-editing command. The lines of the heading are inserted at the cursor position:

```
NAME:
DEPT:
DATE:
SUBJ:
```

To make a macro available during other editing sessions, you can place the DEFINE MACRO command and the macro command sequence in an EDT startup command file. When you include a macro definition in a startup command file, be sure the command sequence contains the commands for entering the macro buffer (FIND=buffer-name.) and returning to the MAIN buffer (FIND=MAIN.). Note that you must precede each command in the sequence with the INSERT command. For more information about key and macro definitions, see the *VAX EDT Reference Manual.*

9 Processing Files with DIGITAL Standard Runoff

DIGITAL Standard Runoff (DSR) is a text formatter that processes source files into formatted text and intermediate files, and creates tables of contents and indexes. You process the source and intermediate files with the DCL commands RUNOFF, RUNOFF/CONTENTS, and RUNOFF/INDEX. With DSR, you can produce several types of documents including a memo, letter, and full-length manuscript.

The DSR source file is a file you created with EDT, EVE, or another text editor. This file has a default file type of .RNO and contains text, DSR formatting commands, flags and control characters. (DSR flags are special characters that you insert in text to specify, for example, emphasis of text, case of characters, and spacing of characters.) When the source file is processed, the DSR commands cause the text to be formatted into sections, paragraphs, lists, and so on. You can direct DSR to provide title pages, footnotes, tables of contents, indexes, and appendixes for the source files it processes by inserting control characters and other special identifiers within your text.

See the *VAX DIGITAL Standard Runoff Reference Manual* for rules on entering DSR commands and a description of each command.

DSR commands start with the control flag, which by default is represented by a period. DSR commands can be typed in uppercase, lowercase, or a combination of uppercase and lowercase. You can abbreviate DSR command names, but the abbreviations must be exactly as listed in the *VAX DIGITAL Standard Runoff Reference Manual.*

9.1 Formatting Text

Using DSR, you can format text into paragraphs or lists, or maintain a literal display. You can adjust the margins and the spacing between lines or blocks of lines. DSR also provides commands for leaving blocks of space on a page and for writing notes and footnotes.

You can control such features as bolding and underlining with embedded flags. By default, DSR treats certain characters as flags rather than text. For example, by default, a character is underlined if it is preceded by an ampersand (&). (To place an actual ampersand in the text, you must enter an ampersand preceded by an underscore (_&) flag, or you must turn off the flag governing that character.) One approach is to turn off all flags at the start of your DSR file. The following commands turn off all flags that are on by default:

Flag Command	Flag Character
.NO FLAGS ACCEPT	Underscore (_) by default
.NO FLAGS COMMENT	Exclamation point (!) by default
.NO FLAGS LOWERCASE	Backslash (\) by default
.NO FLAGS SPACE	Number sign (#) by default
.NO FLAGS SUBINDEX	Right angle bracket (>) by default
.NO FLAGS UNDERLINE	Ampersand (&) by default
.NO FLAGS UPPERCASE	Circumflex (^) by default

When you need a particular flag, set the flag on (.FLAGS command), write the text that uses the flag, and set the flag off (.NO FLAGS command). (See Section 9.1.9.) You can change the special flag character when you specify the .FLAGS command.

Following are several examples that illustrate how to use various DSR commands to produce formatted output:

```
.BREAK
.BLANK
.BLANK 2
.RIGHT MARGIN 34
.CENTER;Twelve Days of Dieting
```

In the previous example, the .BREAK command ends the current line. The .BLANK command without a parameter inserts a blank line in the text. The .BLANK command with the parameter 2 inserts two blank lines in the text. The .RIGHT MARGIN command sets the right margin at position 34. The .CENTER command centers the text following the semicolon.

On any line containing a DSR command, the first item on the line must be a DSR command and the control flag must occupy position 1. Depending on the particular commands, a line containing a command may contain additional commands and text. In the following example, the DSR command occupies its own line. The end of the line terminates the command.

```
.BLANK 2
```

In the following example, the command and text are placed on one line. The semicolon acts as the command terminator.

```
.CENTER;Twelve Days of Dieting
```

In the following example, two commands are placed on one line. The beginning of the second command terminates the first command.

```
.BLANK 2.CENTER;Twelve Days of Dieting
```

The following example demonstrates the use of the DSR commands .BLANK and .CENTER to format text:

```
.RIGHT MARGIN 34
.CENTER;Twelve Days of Dieting
.CENTER;Watching Your Weight Increase
.BLANK 2
On the twelfth day of dieting, Millitsa gave to me,
.BREAK
Twelve hot fudge sundaes,
.BREAK
Eleven Hostess Twinkies,
.BREAK
Ten cherry cheese cakes,
.BLANK
Nine ladyfingers,
.BREAK
Eight date nut muffins,
.BREAK
Seven oatmeal cookies,
.BREAK
Six bags of Fritos,
.BREAK
Five coffee rings,
.BLANK
Four sticky buns,
.BREAK
Three Clark bars,
.BREAK
Two marbled cakes,
.BREAK
And a pizza with pepperoni.
```

The preceding lines of text coded in DSR produce the following output:

```
           Twelve Days of Dieting
        Watching Your Weight Increase


On the twelfth day of dieting, Millitsa gave to me,
Twelve hot fudge sundaes,
Eleven Hostess Twinkies,
Ten cherry cheese cakes,

Nine ladyfingers,
Eight date nut muffins,
Seven oatmeal cookies,
Six bags of Fritos,
Five coffee rings,

Four sticky buns,
Three Clark bars,
Two marbled cakes,
And a pizza with pepperoni.
```

Do not edit the output file from a DSR operation. Instead, modify the DSR file and reprocess it. If you do make minor modifications to the output file, the file does not have the carriage return record attribute (which causes each record in the file to produce a new line automatically when the file is displayed or printed). The carriage return and line feed control characters are embedded in the file.

9.1.1 Filling and Justifying Text

By default, DSR fills and justifies text. Filling adds words to each output line until the addition of another word would exceed the right margin. Justification inserts additional space between words on a line so that the last word reaches the right margin. The following example demonstrates how text is filled and justified:

```
.RIGHT MARGIN 45
For it so falls out,
That what we have
we prize not
to the worth
while we enjoy it;
but being lacked,
lacked and lost,
Why,
then we rack the value.
```

The preceding lines of text coded in DSR produce the following output:

```
For it so falls out,  That  what we have
we prize not to the worth while we enjoy
it; but being lacked, lacked  and  lost,
Why, then we rack the value.
```

If you do not want filling or justification, you must explicitly specify the .NO FILL command or the .NO JUSTIFY command before you write the text. Turn filling or justifying back on by specifying the .FILL or .JUSTIFY command. The following example turns off justification but retains filling, which produces a ragged right margin:

```
.RIGHT MARGIN 45
.NO JUSTIFY
For it so falls out,
That what we have
we prize not
to the worth
while we enjoy it;
but being lacked,
lacked and lost,
Why,
then we rack the value.
.JUSTIFY
```

The preceding lines of text coded in DSR produce the following output:

```
For it so falls out, That what we have
we prize not to the worth while we enjoy
it; but being lacked, lacked and lost,
Why, then we rack the value.
```

The next example turns off both filling and justifying, which formats the output lines in the same way as the input lines:

```
.NO FILL.NO JUSTIFY
For it so falls out,
That what we have
we prize not
to the worth
while we enjoy it;
but being lacked,
lacked and lost,
Why,
then we rack the value.
.FILL.JUSTIFY
```

The preceding lines of text coded in DSR produce the following output:

```
For it so falls out,
That what we have
we prize not
to the worth
while we enjoy it;
but being lacked,
lacked and lost,
Why,
then we rack the value.
```

9.1.2 Adjusting Margins and Centering Text

By default, margins are set at 0 and 70. Change the margin settings with the .LEFT MARGIN and .RIGHT MARGIN commands. The following example changes the right margin to position 60:

```
.RIGHT MARGIN 60
    .
    .
    .
```

To indent one or more lines of text on the left, increase the left margin setting. After you write the indented text, decrease the left margin setting by the same amount. Indent on the right by decreasing the right margin setting, writing the text, and increasing the right margin setting. The following example indents text by 10 spaces on either margin:

```
    .
    .
    .
.LEFT MARGIN +10.RIGHT MARGIN -10
indented text
.LEFT MARGIN -10.RIGHT MARGIN +10
    .
    .
    .
```

You can indent a single line of text from the left margin with the .INDENT command. The following example indents a line of text eight spaces:

```
.INDENT 8
I wandered lonely as a cloud
```

Indent a single line of text from the right margin with the .RIGHT command. You can also use the .RIGHT command to position a single line of text against the right margin as follows:

```
.RIGHT 0
I wandered lonely as a cloud
```

To center text between two margins, use the .CENTER command. Place the text to be centered on the line following the command as follows:

```
.CENTER
I wandered lonely as a cloud
```

You can also end the .CENTER command with a semicolon, and place the text to be centered on the same line as follows:

```
.CENTER;I wandered lonely as a cloud
```

9.1.3 Formatting Paragraphs

To separate text into paragraphs, place the .PARAGRAPH command between paragraphs. By default, the .PARAGRAPH command indents the first line of a paragraph five spaces, inserts one blank line before starting a new paragraph, and tests to ensure that room remains on the page for at least four lines of text. You can change the parameter values when you type your first .PARAGRAPH command or by placing a .SET PARAGRAPH command at the top of the file. The following example does not indent the first line of each paragraph and ensures that room remains on the page for at least three lines of text:

```
.SET PARAGRAPH 0,1,1
paragraph of text
.PARAGRAPH
paragraph of text
.PARAGRAPH
   .
   .
   .
```

You can also separate text by inserting .BLANK or .SKIP commands. The .BLANK command inserts the exact number of lines specified by the parameter (which defaults to 1). The .BLANK command does not provide for indentation or for testing the room left on the page; perform these actions with the .INDENT and .TEST PAGE commands. The following example separates two blocks of text with one blank line after first testing to ensure that at least three lines remain on the page:

```
paragraph of text
.TEST PAGE 3
.BLANK
paragraph of text
```

The .SKIP command takes into account the spacing you have in effect. When the default is in effect (spacing is 1), .BLANK and .SKIP are equivalent. However, if multiple spacing is in effect, the .SKIP command multiplies the skip value by the spacing value. You can specify something other than single spacing with the .SPACING command. (The .SPACING command also affects the test page value in the .PARAGRAPH and .SET PARAGRAPH commands.) The following example demonstrates double spacing, with two extra lines between blocks of text:

```
.SPACING 2
block of text
.SKIP
block of text
.SKIP
    .
    .
    .
```

9.1.4 Formatting Literal Text

To have text appear in the output file exactly as it appears in the DSR source file, enclose it with .LITERAL and .END LITERAL commands as follows:

```
.RIGHT MARGIN 34
.BLANK 2
.LITERAL
           Twelve Days of Dieting
        Watching Your Weight Increase


On the twelfth day of dieting, Millitsa gave to me,
Twelve hot fudge sundaes,
Eleven Hostess Twinkies,
Ten cherry cheese cakes,

Nine ladyfingers,
Eight date nut muffins,
Seven oatmeal cookies,
Six bags of Fritos,
Five coffee rings,

Four sticky buns,
Three Clark bars,
Two marbled cakes,
And a pizza with pepperoni.
.END LITERAL
```

The preceding lines coded in DSR produce the following output:

```
           Twelve Days of Dieting
        Watching Your Weight Increase


On the twelfth day of dieting, Millitsa gave to me,
Twelve hot fudge sundaes,
Eleven Hostess Twinkies,
Ten cherry cheese cakes,

Nine ladyfingers,
Eight date nut muffins,
Seven oatmeal cookies,
Six bags of Fritos,
Five coffee rings,

Four sticky buns,
Three Clark bars,
Two marbled cakes,
And a pizza with pepperoni.
```

Literal text is not filled and not justified; lines are the same as in the input file. Except for the .END LITERAL command, commands and flags are not recognized in literal text. In addition, commands and flags placed before the literal text do not affect the literal text except that you can set the left margin, set tab stops, set spacing, start bolding, and start underlining. You cannot reset any of these items until the literal text ends (for example, you would have to bold the entire literal block). The following example indents the literal text (the .MARGIN commands must be outside the literal block):

```
the text filled and justified
.LEFT MARGIN +5
.LITERAL
the literal text
.END LITERAL
.LEFT MARGIN -5
```

If you want the positional effect of literal text but also want to use commands and flags, you can turn off filling and justifying (.NO FILL and .NO JUSTIFY) and turn it back on after the literal text. If the literal text contains blank lines, specify .KEEP when you turn off the filling and justification as follows:

```
the text filled and justified
.NO FILL
.NO JUSTIFY
.KEEP
the literal text
.FILL
.JUSTIFY
.NO KEEP
```

For a small number of lines, you might place .BREAK or .BLANK commands (specify .BLANK 0 for single spacing) between the literal lines.

9.1.5 Formatting Lists

You can format text as lists. Although by default DSR creates a numbered list, you can create a bulleted or lettered list or use any other character or symbol to precede the list elements.

9.1.5.1 Numbered Lists

The following three commands format a numbered list:

- .LIST—Starts the list by leaving one blank line and indenting. The text of the list is indented nine spaces if the left margin is currently 0. Otherwise, the text of the list is indented four spaces.
- .LIST ELEMENT—Starts an element (an item) within the list. You can have any number of elements.
- .END LIST—Ends the list and writes a blank line.

Each element is preceded by a number starting with 1. Elements are separated by blank lines as follows:

```
.LIST
.LIST ELEMENT;grosbeak
.LIST ELEMENT;goldfinch
.LIST ELEMENT;redpoll
.LIST ELEMENT;sparrow
.END LIST
```

The preceding lines of text coded in DSR produce the following output:

```
    1.  grosbeak

    2.  goldfinch

    3.  redpoll

    4.  sparrow
```

Text for each list element can be placed after the semicolon terminating the command (as shown in the example) or can be placed on a line or lines following the command. A single list element continues until the next .LIST ELEMENT command or an .END LIST command occurs. The list element can contain commands and flags.

To change the spacing between list elements, specify the number of spaces as parameter 1 to the .LIST command. For example, .LIST 0 places no spaces between list elements.

9.1.5.2 Bulleted Lists

If you do not want your list to be numbered, as it is by default, substitute another character for the numbers by specifying that character as the second parameter of the .LIST command. Enclose the character in quotation marks or apostrophes; the character does not have to be preceded by a comma if the first parameter is not specified. In the following example, the lowercase "o" gives the effect of an unfilled bullet:

```
.LIST "o"
.LIST ELEMENT;ferret
.LIST ELEMENT;mink
.LIST ELEMENT;rabbit
.LIST ELEMENT;sable
.LIST ELEMENT;raccoon
.END LIST
```

The preceding lines of text coded in DSR produce the following output:

```
    o  ferret

    o  mink

    o  rabbit

    o  sable

    o  raccoon
```

9.1.5.3 Nested Lists

You can nest one list within another as long as the nested list is entirely within one element of the outer list as follows:

```
.LIST 0
.LIST ELEMENT;German
.LIST ELEMENT;Russian
.LIST ELEMENT;Swedish
.LIST ELEMENT;Yugoslavian
.LIST 0,"o"
.LIST ELEMENT;Serbian
.LIST ELEMENT;Croatian
.LIST ELEMENT;Macedonian
.END LIST
.LIST ELEMENT;Turkish
.LIST ELEMENT;Scottish
.LIST ELEMENT;Irish
.END LIST
```

The preceding lines of text coded in DSR produce the following output:

```
    1.  German
    2.  Russian
    3.  Swedish
    4.  Yugoslavian
          o  Serbian
          o  Croatian
          o  Macedonian
    5.  Turkish
    6.  Scottish
    7.  Irish
```

9.1.5.4 Lists Beginning with Letters and Roman Numerals

By default, DSR numbers lists with decimal numbers. You can, however, produce a list whose elements are preceded by uppercase or lowercase letters or Roman numerals by specifying the .DISPLAY ELEMENTS command. You must place the .DISPLAY ELEMENTS command between the .LIST command and the first .LIST ELEMENT command. The following example produces a numbered list using lowercase Roman numerals:

```
.LIST 0
.DISPLAY ELEMENTS RL
.LIST ELEMENT;tan
.LIST ELEMENT;beige
.LIST ELEMENT;rust
.LIST ELEMENT;brown
.END LIST
```

The preceding lines of text coded in DSR produce the following output:

```
      i.  tan
     ii.  beige
    iii.  rust
     iv.  brown
```

9.1.6 Leaving Space on a Page

The text of a file usually starts on the fourth or fifth line from the top of a page, depending on the layout of the document. To leave extra space at the top of a page, specify an amount in a .FIGURE command (the .BLANK command does not work at the top of a page). To leave extra space in the middle of a page, use either the .FIGURE or .BLANK command. If you want a certain amount of space all on one page, use the .FIGURE or .FIGURE DEFERRED command. These commands insert the required space on the next page if it does not fit on the current page. While the .FIGURE command leaves the rest of the current page blank, the .FIGURE DEFERRED command fills the rest of the current page using the text and commands that follow the .FIGURE DEFERRED command.

The following example writes 40 blank lines to the current page if they will fit; otherwise, the 40 blank lines are placed at the top of the next page, and the current page is filled from the input lines following the .FIGURE DEFERRED line.

```
text filled and justified
.FIGURE DEFERRED 40
text filled and justified
```

You can also create space on a page by entering the .LITERAL command, pressing RETURN once for each empty line, and entering the .END LITERAL command. This technique allows you to see the amount of space you are creating. However, the space will be split if you cross page boundaries.

9.1.7 Formatting Notes

The .NOTE command narrows the left and right margins, inserts a blank line, writes a centered title, inserts another blank line, and writes the text that follows the .NOTE command. The .END NOTE command writes a blank line and restores the margin settings. The title defaults to the word NOTE.

The following example writes a note with the title CAUTION:

```
.NOTE CAUTION
Do not operate the machine outdoors in wet weather.
Do not operate the machine in a wet area indoors or outdoors.
Such actions may lead to a severe electrical shock.
.END NOTE
```

The preceding lines of text coded in DSR produce the following output:

```
                    CAUTION

     Do not operate the machine outdoors  in  wet  weather.
     Do  not  operate  the machine in a wet area indoors or
     outdoors.  Such  actions may  lead  to  a  severe
     electrical shock.
```

9.1.8 Formatting Footnotes

The .FOOTNOTE command places the text following the command at the bottom of the page if enough room exists. If the entire footnote does not fit, the whole footnote is placed at the bottom of the next page. No automatic formatting occurs; you must determine how to format the footnote with DSR commands. In addition, no automatic footnote symbols are provided. The .END FOOTNOTE command ends the footnote and automatically restores any case, fill, justify, and margin settings you might have changed within the footnote.

The following example demonstrates the use of a footnote:

```
Press the START button firmly.
Release the START button as soon as the engine starts.(1)
.FOOTNOTE.BLANK.LEFT MARGIN +4.INDENT -4
(1)
If the engine does not crank, ensure that the battery cables
are firmly connected to the battery. Sometimes the cables are
disconnected for shipping.
.END FOOTNOTE
Push the choke in until you hear it click.
Pull the throttle about halfway down but do not let the
engine stall.
After about two minutes, push the choke all the way in
and pull the throttle all the way down.
.PARAGRAPH 0
Before engaging the drive train, ensure that you are in a
comfortable position to operate the machine.
Two seat controls are provided for your comfort.
```

The preceding lines of text coded in DSR produce the following output:

```
Press the START button firmly.  Release the START button as
soon as the engine starts.(1) Push the choke in until you
hear it click.  Pull the throttle about halfway down but do
not let the engine stall.  After about two minutes, push the
choke all the way in and pull the throttle all the way down.

Before engaging the drive train, ensure that you are in a
comfortable position to operate the machine.  Two seat
controls are provided for your comfort.

(1) If the engine does not crank, ensure that the battery
    cables are firmly connected to the battery. Sometimes
    the cables are disconnected for shipping.
```

9.1.9 Bolding and Underlining Text

The following steps describe how to bold a single character:

1 Turn the bold flag on. (The bold flag is off by default.)

2 In the text, precede the character to be bolded by the bold flag (an asterisk by default).

3 Turn the bold flag off to enable the use of the flag as a normal character.

The following example bolds the numbers 3 and 7:

```
.FLAGS BOLD
Follow route *3 to route *7.
.NOFLAGS BOLD
```

The following steps describe how to underline a single character:

1 Turn the underline flag on. (The underline flag is on by default.)
2 In the text, precede the character to be underlined by the underline flag (an ampersand by default).
3 Turn the underline flag off to enable the use of the flag as a normal character.

The following example underlines the letters A and B:

```
.FLAGS UNDERLINE
&A is for Amy and &B is for Basil.
.NOFLAGS UNDERLINE
```

The following steps describe how to bold or underline a block of text:

1 Turn on the uppercase and lowercase flags (they are on by default) in addition to the bold or underline flag.
2 Start the block of text with the uppercase flag (a circumflex by default) followed by the bold or underline flag.
3 End the block of text with the lowercase flag (a backslash by default) followed by the bold or underline flag.

The following example bolds a line of text:

```
.FLAGS BOLD
.FLAGS UPPERCASE
.FLAGS LOWERCASE
^*KEEP OFF THE GRASS, PLEASE\*
.NOFLAGS BOLD
.NOFLAGS UPPERCASE
.NOFLAGS LOWERCASE
```

9.2 Laying Out a Document

By default, DSR produces a document of consecutively numbered pages (that is, if you do not specify .LAYOUT, .CHAPTER, or .APPENDIX commands). On each page, the text area is the fourth line through the bottom line. Page numbers appear in the upper right corner as "Page 2," "Page 3," and so on, starting with page 2. Running heads (chapter names or other designated text) appear in the upper left corner.

You can adjust the position of page numbers and running heads with the .LAYOUT command. Layout codes 1, 2, and 3 center the page number at the bottom of the page and adjust the running heads as follows:

- Layout code 1 centers running heads at the top of the page.
- Layout code 2 moves the running heads to one upper corner or the other depending on whether the page number is odd or even.
- Layout code 3 puts the running heads in the upper left corner and puts the date in the upper right corner.

Specify the .LAYOUT command at the beginning of your file. The following command adjusts the layout to code 2:

```
.LAYOUT 2,3
```

9.2.1 Chapters and Appendixes

To divide a document into chapters, start each chapter with the .CHAPTER command. The title of the chapter must follow the command name on the same line. The lines following the .CHAPTER command are part of that chapter until you enter another .CHAPTER command or an .APPENDIX command.

By default, chapters are numbered consecutively within the document, beginning with Chapter 1. You can force the numbering of a chapter (for example, in order to place each chapter in a separate file) by preceding the .CHAPTER command with a .NUMBER CHAPTER command. The following example begins Chapter 2:

```
.NUMBER CHAPTER 2
 .CHAPTER starting procedures
```

The preceding lines of text coded in DSR produce the following output:

```
<12 blank lines>
                             CHAPTER 2

                        STARTING PROCEDURES
```

DSR starts a chapter on a new page with 12 blank lines at the top of the page. The number of the chapter and the chapter title are centered on the page in uppercase characters. You can adjust the appearance of the chapter number with the .DISPLAY CHAPTER command.

The .APPENDIX, .NUMBER APPENDIX, and .DISPLAY APPENDIX commands work similarly to the chapter commands. However, by default, appendixes are lettered sequentially starting with Appendix A. The following example starts Appendix C:

```
.NUMBER APPENDIX C
 .APPENDIX connecting the battery
```

9.2.2 Sections

The .HEADER LEVEL command divides a document into sections and subsections identified by a decimal numbering scheme to a maximum depth of 6. The topmost section is header level 1. Each level is numbered sequentially starting with 1 unless a .NUMBER LEVEL command precedes the .HEADER LEVEL command. The following example shows a document with three sections:

```
text
.HEADER LEVEL 1 normal starting
text
.HEADER LEVEL 1 cold weather starting
text
.HEADER LEVEL 1 troubleshooting
text
```

The preceding lines of text coded in DSR produce the following output:

```
text
1  NORMAL STARTING
text
2  COLD WEATHER STARTING
text
3  TROUBLESHOOTING
text
```

Subsections are numbered within their respective higher-level sections as follows:

```
text
.HEADER LEVEL 1 normal starting
text
.HEADER LEVEL 2 cold engine
text
.HEADER LEVEL 2 warm engine
text
.HEADER LEVEL 1 cold weather starting
text
.HEADER LEVEL 2 above zero
text
.HEADER LEVEL 2 below zero
text
.HEADER LEVEL 1 troubleshooting
text
```

The preceding lines of text coded in DSR produce the following output:

```
text
1  NORMAL STARTING
text
1.1  Cold Engine
text
1.2  Warm Engine
text
2  COLD WEATHER STARTING
text
2.1  Above Zero
text
2.2  Below Zero
text
3  TROUBLESHOOTING
text
```

If the sections are within chapters or appendixes, the section number is prefixed by the chapter or appendix identifier and a decimal point as follows:

```
.NUMBER CHAPTER 2
.CHAPTER starting procedures
text
.HEADER LEVEL 1 normal starting
text
.HEADER LEVEL 1 cold weather starting
text
.HEADER LEVEL 1 troubleshooting
text
```

The preceding lines of text coded in DSR produce the following output:

```
the chapter heading and text
2.1  NORMAL STARTING
text
2.2  COLD WEATHER STARTING
text
2.3  TROUBLESHOOTING
text
```

You can force the numbering of a section with the .NUMBER LEVEL command. The following example forces the start of section 1.2:

```
.NUMBER LEVEL 1,2
.HEADER LEVEL 2 Warm Engine
```

You can change the appearance of section headers with the .STYLE HEADERS command. The default .STYLE HEADERS settings cause level 1 headers to be written in all uppercase and level 2 headers to be written with the initial letter of every word in uppercase. The following command changes the settings so that the headers below level 1 are written exactly as you type them:

```
.STYLE HEADERS 3,1,0,7,7,2,1,9,2
```

9.2.3 Running Heads

By default, chapter and appendix titles appear as the first line of running heads. If the document does not contain chapters or appendixes, no running heads appear. Running heads are always placed at the top of the page; their exact position can be changed with the .LAYOUT command.

To use level 1 header titles as the second line of running heads, enter the following commands at the start of your DSR file:

```
.SUBTITLE
.AUTOSUBTITLE
```

To use titles other than chapter and section titles as running heads, use the .TITLE and .SUBTITLE commands. The .TITLE command affects the first line of the running head; the .SUBTITLE command affects the second line of the running head.

The title specified by a .TITLE command remains in effect until you specify another .TITLE or .CHAPTER command. If you want to use a specified title in place of a chapter name, enter the .TITLE command immediately after the .CHAPTER command. The title specified by a .SUBTITLE command remains in effect until you specify another .SUBTITLE command or until a .HEADER LEVEL command occurs (if automatic subtitles are in effect).

9.2.4 Pagination

If the document is not divided into chapters, pagination is sequential throughout the document. If the document is chapter oriented, pagination is sequential throughout the document only if .LAYOUT 3 is in effect. Otherwise, pagination is sequential within each chapter and appendix; the page number starts with the chapter or appendix identifier and a hyphen.

You can suspend the numbering of pages with the .NO NUMBER command (unless .LAYOUT 3 is in effect). Create a document that is not paged (no running heads, no page numbers) by entering the command .NO PAGING.

9.3 Processing DSR Files

Enter the RUNOFF command to process a DSR file. Specify the name of the DSR file as the parameter; the file type defaults to RNO. The following example processes a file named EXAMPLE.RNO in your default directory.

```
$ RUNOFF EXAMPLE
```

If you do not specify the /OUTPUT qualifier, the RUNOFF command produces an output file with the same name as the input file and a file type of MEM. The preceding example produces an output file named EXAMPLE.MEM. The following example produces an output file named EXAMPLE.MEM from a DSR file named TEMPLATE.RNO:

```
$ RUNOFF/OUTPUT=EXAMPLE TEMPLATE
```

See the *VAX DIGITAL Standard Runoff Reference Manual* for a complete description of the RUNOFF command and its qualifiers.

9.3.1 Producing a Table of Contents

The table of contents you produce can display chapter titles and numbers, header levels, and appendix titles and letters. To produce a table of contents, do the following:

1. Enter the command RUNOFF/INTERMEDIATE, specifying the RNO file as the parameter. The name of the intermediate file produced is the same as the name of the DSR file with a file type of BRN, unless you specify a different name. You will also get the usual output (MEM) file; you can specify /NOOUTPUT if you do not want the MEM file.
2. Enter the command RUNOFF/CONTENTS, specifying the intermediate file as the parameter. This command produces an unformatted table of contents file with the same name as the input file but with a file type of RNT.
3. Enter the RUNOFF command, specifying the RNT file as the parameter. You must specify the file type. This command produces a formatted table of contents file with a file type of MEC.

The following example processes a file named OPER.RNO. It produces an output file named OPER.MEM and a table of contents named OPER.MEC:

```
$ RUNOFF/INTERMEDIATE OPER
$ RUNOFF/CONTENTS OPER
$ RUNOFF OPER.RNT
```

To produce a table of contents from more than one file, you must concatenate the intermediate files when you enter the RUNOFF/CONTENTS command. (You cannot use wildcard characters.) The following example produces output files and a single table of contents from three DSR files:

```
$ RUNOFF/INTERMEDIATE OPER1
$ RUNOFF/INTERMEDIATE OPER2
$ RUNOFF/INTERMEDIATE OPER3
$ RUNOFF/CONTENTS/OUTPUT=OPER OPER1+OPER2+OPER3
$ RUNOFF OPER.RNT
```

The table of contents is based on the .CHAPTER, .APPENDIX, and .HEADER LEVEL commands in your DSR file. You can control the formatting to some extent with the qualifiers to the RUNOFF/CONTENTS command. You can write additional information to the table of contents with the command .SEND TOC.

See the *VAX DIGITAL Standard Runoff Reference Manual* for more information about producing a table of contents with DSR.

9.3.2 Producing an Index

To create an index, you enter .INDEX and .ENTRY commands throughout your DSR file. (You can also use the index flag to index a word of text.)

The .INDEX command names an item to be placed in the index. Position the .INDEX command as close as possible to the text being indexed. The item appears in the index followed by the number of the page on which it was written to the formatted text (MEM) file. The following example makes index entries for each section:

```
text
.HEADER LEVEL 1 Normal Starting
.INDEX Normal starting
text
.HEADER LEVEL 1 Cold Weather Starting
.INDEX Cold weather starting
text
.HEADER LEVEL 1 Troubleshooting
.INDEX Troubleshooting
text
```

The index entries would appear as follows:

```
Cold weather starting, 2-2
Normal starting, 2-1
Troubleshooting, 2-3
```

Use the subindex flag (by default a right angle bracket) to indicate subentries in the index. A subentry is listed under the higher-level item in the index and has its own page number. The subindex flag must be turned on. The following example produces one index entry for "Starting" under which are listed the two subentries "normal" and "cold weather":

```
text
.HEADER LEVEL 1 Normal Starting
.FLAGS SUBINDEX
.INDEX Normal starting
.INDEX Starting>normal
.NOFLAGS SUBINDEX
text
.HEADER LEVEL 1 Cold Weather Starting
.FLAGS SUBINDEX
.INDEX Cold weather starting
.INDEX Starting>cold weather
.NOFLAGS SUBINDEX
text
.HEADER LEVEL 1 Troubleshooting
.INDEX Troubleshooting
text
```

The index entry for "Starting" would appear as follows:

```
Starting
  cold weather, 2-2
  normal, 2-1
```

You can make an entry without a page number in the index with the .ENTRY command. Usually these index entries are used for cross references. The following example produces an index entry for "Weather," under which the subentry "see cold weather" appears without a page number:

```
.FLAGS SUBINDEX
.ENTRY Weather>see cold weather
```

The index entry would appear as follows:

```
Weather
  see cold weather
```

After you enter the index commands to your file, you are ready to run the indexing program. To produce an index, do the following:

1 Enter the command RUNOFF/INTERMEDIATE, specifying your RNO file as the parameter. The name of the intermediate file produced is the same as the name of the DSR file but with a file type of BRN, unless you specify a different name. You will also get the usual output (MEM) file; you can specify /NOOUTPUT if you do not want the MEM file.

2 Enter the command RUNOFF/INDEX, specifying the intermediate file as the parameter. This command produces an unformatted index file with the same name as the input file but with a file type of RNX.

3 Enter the RUNOFF command, specifying the RNX file as the parameter. You must specify the file type. This command produces a formatted index file with a file type of MEX.

The following example processes a file named OPER.RNO. It produces an output file named OPER.MEM, a table of contents named OPER.MEC, and an index named OPER.MEX.

```
$ RUNOFF/INTERMEDIATE OPER
$ RUNOFF/CONTENTS OPER
$ RUNOFF/INDEX OPER
$ RUNOFF OPER.RNT
$ RUNOFF OPER.RNX
```

To produce an index from more than one file, you must concatenate the intermediate files when you enter the RUNOFF/INDEX command. (You cannot use wildcard characters.) The following example produces formatted text files, a single table of contents, and a single index from three DSR files:

```
$ RUNOFF/INTERMEDIATE OPER1
$ RUNOFF/INTERMEDIATE OPER2
$ RUNOFF/INTERMEDIATE OPER3
$ RUNOFF/CONTENTS/OUTPUT=OPER OPER1+OPER2+OPER3
$ RUNOFF/INDEX/OUTPUT=OPER OPER1+OPER2+OPER3
$ RUNOFF OPER.RNT
$ RUNOFF OPER.RNX
```

9.3.3 Printing Output Files

The following are guidelines for printing nonlaser-output files produced by DSR:

- Copying files—You can copy a file (with the COPY command) to a printer, but you may occasionally lose a form feed. You lose a form feed when the size of an output page equals 66 or the value that SYS$LP_LINES had at the time the file was created if SYS$LP_LINES was defined as a logical name. (Note that the current value of SYS$LP_LINES has no effect on the output file after it is created.)
- Generating automatic form feeds—In general, use the default for the DCL command PRINT (PRINT/FEED) to generate form feeds automatically when printing nonlaser files. However, you may occasionally generate a blank page. More precisely, you generate a blank page when an output page equals 66 or the value that SYS$LP_LINES had at the time the file was created if SYS$LP_LINES was defined as a logical name. (Note that the current value of SYS$LP_LINES has no effect on the output file.) If you specify PRINT/NOFEED, you eliminate the chance of a blank page, but you take the chance of losing pages under the same circumstances as a COPY operation.

The following example shows a laser printer setup in a command procedure. *Dsr$ln01* is the name you assign the laser printer form. *Lpb0* is the name of the printer.

```
$ ! Define form for dsr output on ln01 laser printer
$
$ DEFINE/FORM dsr$ln01 /MARGIN=(BOTTOM=0) -
                       /NOWRAP -
                       /NOTRUNCATE -
                       /STOCK=DEFAULT -
                       /DESCRIPTION="dsr ln01 form definition"
$
$ ! Set up ln01 laser printer for dsr output
$
$ SET PRINTER lpb0     /NOTRUNCATE -
                       /NOWRAP -
                       /TAB -
                       /PRINTALL -
                       /FF -
                       /NOCR
```

The print command should specify /NOFEED, the name of the form, and the name of the queue. You should equate this command to a global symbol in your login file or in the system login file. The following example makes LNPRINT a global symbol that prints LN01 files:

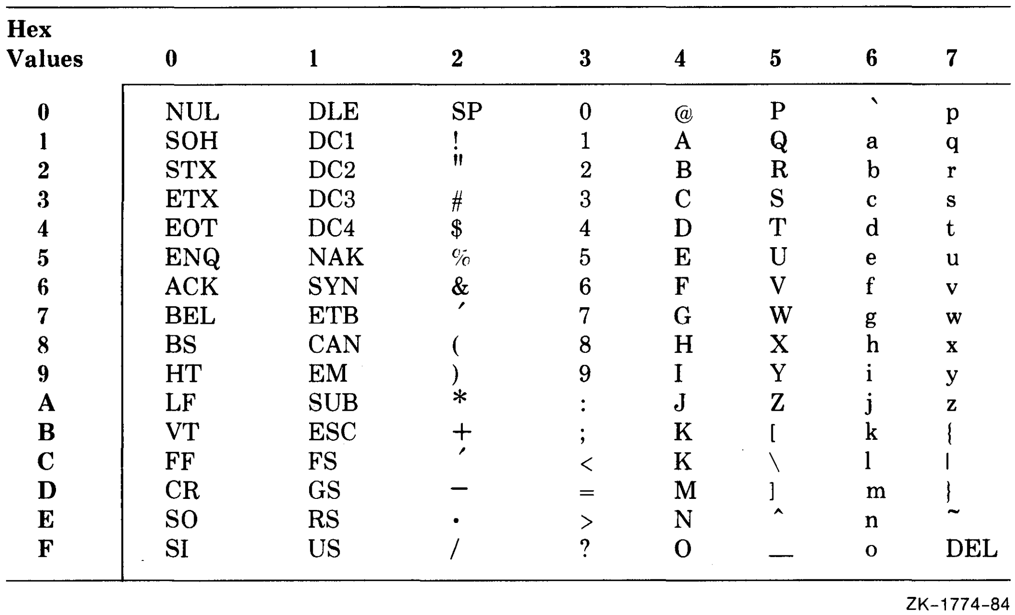
```
$ ! Set up lnprint as print command for ln01 laser printer
$
$ LNP*RINT == "PRINT/NOFEED/FORM=dsr$ln01/QUEUE=ln01_queue"
```

A Character Sets

The following tables present the ASCII character set and the DEC Multinational Character Set.

A.1 ASCII Character Set

The ASCII character set consists of the characters shown in the following table. The characters with names are not printable. (The ASCII character set comprises the first 127 characters of the DEC Multinational Character Set; descriptions of the nonprintable characters are located in the table in Section A.3.) You can calculate the numeric value of a character by constructing a 2-digit hexadecimal number in which the column position of the character represents the 16s position of the hexadecimal number and the row position of the character represents the units position of the number. For example, an uppercase *A* has the numeric value 41 hexadecimal. String comparisons are made using these values.



Hex Values	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	'	<	K	\	l	|
D	CR	GS	−	=	M	]	m	}
E	SO	RS	•	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

ZK-1774-84

A.2 ASCII and DEC Multinational Character Set Tables

Table A–1 represents the ASCII character set (characters with decimal values 0 through 127). The first half of each of the numbered columns identifies the character as you would enter it on a VT300-, VT200-, or VT100-series terminal or as you would see it on a printer (except for the nonprintable characters). The remaining half of each column identifies the character by the binary value of the byte; the value is stated in three radixes—octal, decimal, and hexadecimal. For example, the uppercase letter A has, under

A.2 ASCII and DEC Multinational Character Set Tables

ASCII conventions, a storage value of hexadecimal 41 (a bit configuration of 01000001), equivalent to 101 in octal notation and 65 in decimal notation.

Table A–1 Graphical Representation of the ASCII Character Set

ROW	COLUMN	0		1		2		3		4		5		6		7	
	BITS b8 b7 b6 b5 / b4 b3 b2 b1	0 0 0 0		0 0 0 1		0 0 1 0		0 0 1 1		0 1 0 0		0 1 0 1		0 1 1 0		0 1 1 1	
0	0 0 0 0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	`	140 96 60	p	160 112 70
1	0 0 0 1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	a	141 97 61	q	161 113 71
2	0 0 1 0	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	b	142 98 62	r	162 114 72
3	0 0 1 1	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	c	143 99 63	s	163 115 73
4	0 1 0 0	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	d	144 100 64	t	164 116 74
5	0 1 0 1	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	e	145 101 65	u	165 117 75
6	0 1 1 0	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	f	146 102 66	v	166 118 76
7	0 1 1 1	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	g	147 103 67	w	167 119 77
8	1 0 0 0	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	h	150 104 68	x	170 120 78
9	1 0 0 1	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	i	151 105 69	y	171 121 79
10	1 0 1 0	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	j	152 106 6A	z	172 122 7A
11	1 0 1 1	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	k	153 107 6B	{	173 123 7B
12	1 1 0 0	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	l	154 108 6C	|	174 124 7C
13	1 1 0 1	CR	15 13 D	GS	35 29 1D	-	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	m	155 109 6D	}	175 125 7D
14	1 1 1 0	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	n	156 110 6E	~	176 126 7E
15	1 1 1 1	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	_	137 95 5F	o	157 111 6F	DEL	177 127 7F

KEY

CHARACTER	ESC	33	OCTAL
		27	DECIMAL
		1B	HEX

ZK-1752-84

A.2 ASCII and DEC Multinational Character Set Tables

The ASCII character set comprises the first half of the DEC Multinational Character Set. Table A–2 represents the second half of the DEC Multinational Character Set (characters with decimal values 128 through 255). The first half of each of the numbered columns identifies the character as you would see it on a VT300- or VT200-series terminal or printer (these characters cannot be output on a VT100-series terminal). Section A.3 describes how to enter symbols from the DEC Multinational Character Set.

Table A–2 Graphical Representation of the DEC Multinational Extension to the ASCII Character Set

8		9		10		11		12		13		14		15		COLUMN	
1 0 0 0		1 0 0 1		1 0 1 0		1 0 1 1		1 1 0 0		1 1 0 1		1 1 1 0		1 1 1 1		b8 b7 BITS b6 b5 b4 b3 b2 b1	ROW
	200 128 80	DCS	220 144 90	▨	240 160 A0	°	260 176 B0	À	300 192 C0		320 208 D0	à	340 224 E0		360 240 F0	0 0 0 0	0
	201 129 81	PU1	221 145 91	¡	241 161 A1	±	261 177 B1	Á	301 193 C1	Ñ	321 209 D1	á	341 225 E1	ñ	361 241 F1	0 0 0 1	1
	202 130 82	PU2	222 146 92	¢	242 162 A2	²	262 178 B2	Â	302 194 C2	Ò	322 210 D2	â	342 226 E2	ò	362 242 F2	0 0 1 0	2
	203 131 83	STS	223 147 93	£	243 163 A3	³	263 179 B3	Ã	303 195 C3	Ó	323 211 D3	ã	343 227 E3	ó	363 243 F3	0 0 1 1	3
IND	204 132 84	CCH	224 148 94		244 164 A4		264 180 B4	Ä	304 196 C4	Ô	324 212 D4	ä	344 228 E4	ô	364 244 F4	0 1 0 0	4
NEL	205 133 85	MW	225 149 95	¥	245 165 A5	µ	265 181 B5	Å	305 197 C5	Õ	325 213 D5	å	345 229 E5	õ	365 245 F5	0 1 0 1	5
SSA	206 134 86	SPA	226 150 96		246 166 A6	¶	266 182 B6	Æ	306 198 C6	Ö	326 214 D6	æ	346 230 E6	ö	366 246 F6	0 1 1 0	6
ESA	207 135 87	EPA	227 151 97	§	247 167 A7	·	267 183 B7	Ç	307 199 C7	Œ	327 215 D7	ç	347 231 E7	œ	367 247 F7	0 1 1 1	7
HTS	210 136 88		230 152 98	¤	250 168 A8		270 184 B8	È	310 200 C8	Ø	330 216 D8	è	350 232 E8	ø	370 248 F8	1 0 0 0	8
HTJ	211 137 89		231 153 99	©	251 169 A9	¹	271 185 B9	É	311 201 C9	Ù	331 217 D9	é	351 233 E9	ù	371 249 F9	1 0 0 1	9
VTS	212 138 8A		232 154 9A	ª	252 170 AA	º	272 186 BA	Ê	312 202 CA	Ú	332 218 DA	ê	352 234 EA	ú	372 250 FA	1 0 1 0	10
PLD	213 139 8B	CSI	233 155 9B	«	253 171 AB	»	273 187 BB	Ë	313 203 CB	Û	333 219 DB	ë	353 235 EB	û	373 251 FB	1 0 1 1	11
PLU	214 140 8C	ST	234 156 9C		254 172 AC	¼	274 188 BC	Ì	314 204 CC	Ü	334 220 DC	ì	354 236 EC	ü	374 252 FC	1 1 0 0	12
RI	215 141 8D	OSC	235 157 9D		255 173 AD	½	275 189 BD	Í	315 205 CD	Ÿ	335 221 DD	í	355 237 ED	ÿ	375 253 FD	1 1 0 1	13
SS2	216 142 8E	PM	236 158 9E		256 174 AE		276 190 BE	Î	316 206 CE		336 222 DE	î	356 238 EE		376 254 FE	1 1 1 0	14
SS3	217 143 8F	APC	237 159 9F		257 175 AF	¿	277 191 BF	Ï	317 207 CF	ß	337 223 DF	ï	357 239 EF	▨	377 255 FF	1 1 1 1	15

KEY

CHARACTER	ESC	33	OCTAL
		27	DECIMAL
		1B	HEX

ZK-1753-84

A.3 DEC Multinational Character Set

The DEC Multinational Character Set is an 8-bit character set with 256 characters; the first 128 characters in the set correspond to the ASCII character set. Each character has a value in the range 0 through 255 decimal.

In Table A–3, the graphic symbols shown in parentheses represent ASCII control characters. These are produced on most terminals by pressing the key indicated while holding down the CONTROL key. On VT300- and VT200-series terminals, graphic symbols with decimal values greater than 127 can be entered using the compose sequences. Press the Compose Character key followed by the EDT symbol; the graphic symbol is then displayed on your terminal. On VT300- and VT200-series terminals, you can enter symbols for characters 128 through 255 either in EDT or at DCL level.

On VT100-series terminals, graphic symbols with decimal values greater than 127 can only be entered from screen mode in EDT. Use the EDT keypad command SPECINS or the nokeypad command ASC to enter these characters in your text; EDT then displays the EDT symbol that corresponds to the character rather than displaying the graphic symbol itself.

A.3 DEC Multinational Character Set

Table A–3 Abbreviations and Descriptions of the DEC Multinational Character Set

Graphic	EDT Symbol	Decimal Value	Abbrev.	Description
(@)	^@	0	NUL	null character
(A)	^A	1	SOH	start of heading
(B)	^B	2	STX	start of text
(C)	^C	3	ETX	end of text
(D)	^D	4	EOT	end of transmission
(E)	^E	5	ENQ	enquiry
(F)	^F	6	ACK	acknowledge
(G)	^G	7	BEL	bell
(H)	^H	8	BS	backspace
(I)		9	HT	horizontal tabulation
(J)	<LF>	10	LF	line feed
(K)	<VT>	11	VT	vertical tabulation
(L)	<FF>	12	FF	form feed
(M)	<CR>	13	CR	carriage return
(N)	^N	14	SO	shift out
(O)	^O	15	SI	shift in
(P)	^P	16	DLE	data link escape
(Q)	^Q	17	DC1	device control 1
(R)	^R	18	DC2	device control 2
(S)	^S	19	DC3	device control 3
(T)	^T	20	DC4	device control 4
(U)	^U	21	NAK	negative acknowledge
(V)	^V	22	SYN	synchronous idle
(W)	^W	23	ETB	end of transmission block
(X)	^X	24	CAN	cancel
(Y)	^Y	25	EM	end of medium
(Z)	^Z	26	SUB	substitute
([)	<ESC>	27	ESC	escape
(\)	^\	28	FS	file separator
(])	^]	29	GS	group separator
(^)	^^	30	RS	record separator
(_)	^_	31	US	unit separator
		32	SP	space
!	!	33	!	exclamation point
"	"	34	"	quotation marks (double quote)
#	#	35	#	number sign
$	$	36	$	dollar sign
%	%	37	%	percent sign
&	&	38	&	ampersand
'	'	39	'	apostrophe (single quote)
(	(	40	(	opening parenthesis
)	)	41	)	closing parenthesis
*	*	42	*	asterisk
+	+	43	+	plus
,	,	44	,	comma
-	-	45	-	hyphen or minus
.	.	46	.	period or decimal point
/	/	47	/	slash

ZK-1737/1-84

Table A–3 (Cont.) Abbreviations and Descriptions of the DEC Multinational Character Set

Graphic	EDT Symbol	Decimal Value	Abbrev.	Description
0	0	48	0	zero
1	1	49	1	one
2	2	50	2	two
3	3	51	3	three
4	4	52	4	four
5	5	53	5	five
6	6	54	6	six
7	7	55	7	seven
8	8	56	8	eight
9	9	57	9	nine
:	:	58	:	colon
;	;	59	;	semicolon
<	<	60	<	less than
=	=	61	=	equals
>	>	62	>	greater than
?	?	63	?	question mark
@	@	64	@	commercial at
A	A	65	A	uppercase A
B	B	66	B	uppercase B
C	C	67	C	uppercase C
D	D	68	D	uppercase D
E	E	69	E	uppercase E
F	F	70	F	uppercase F
G	G	71	G	uppercase G
H	H	72	H	uppercase H
I	I	73	I	uppercase I
J	J	74	J	uppercase J
K	K	75	K	uppercase K
L	L	76	L	uppercase L
M	M	77	M	uppercase M
N	N	78	N	uppercase N
O	O	79	O	uppercase O
P	P	80	P	uppercase P
Q	Q	81	Q	uppercase Q
R	R	82	R	uppercase R
S	S	83	S	uppercase S
T	T	84	T	uppercase T
U	U	85	U	uppercase U
V	V	86	V	uppercase V
W	W	87	W	uppercase W
X	X	88	X	uppercase X
Y	Y	89	Y	uppercase Y
Z	Z	90	Z	uppercase Z
[	[	91	[	opening bracket
\	\	92	\	back slash
]	]	93	]	closing bracket
^	^	94	^	circumflex
_	_	95	_	underline (underscore)

ZK-1737/2-84

Table A–3 (Cont.) Abbreviations and Descriptions of the DEC Multinational Character Set

Graphic	EDT Symbol	Decimal Value	Abbrev.	Description
`	`	96	`	grave accent
a	a	97	a	lowercase a
b	b	98	b	lowercase b
c	c	99	c	lowercase c
d	d	100	d	lowercase d
e	e	101	e	lowercase e
f	f	102	f	lowercase f
g	g	103	g	lowercase g
h	h	104	h	lowercase h
i	i	105	i	lowercase i
j	j	106	j	lowercase j
k	k	107	k	lowercase k
l	l	108	l	lowercase l
m	m	109	m	lowercase m
n	n	110	n	lowercase n
o	o	111	o	lowercase o
p	p	112	p	lowercase p
q	q	113	q	lowercase q
r	r	114	r	lowercase r
s	s	115	s	lowercase s
t	t	116	t	lowercase t
u	u	117	u	lowercase u
v	v	118	v	lowercase v
w	w	119	w	lowercase w
x	x	120	x	lowercase x
y	y	121	y	lowercase y
z	z	122	z	lowercase z
{	{	123	{	opening brace
|	|	124	|	vertical line
}	}	125	}	closing brace
~	~	126	~	tilde
DEL	<DEL>	127	DEL	delete, rubout
	<X80>	128	---	|reserved|
	<X81>	129	---	|reserved|
	<X82>	130	---	|reserved|
	<X83>	131	---	|reserved|
	<IND>	132	IND	index
	<NEL>	133	NEL	next line
	<SSA>	134	SSA	start of selected area
	<ESA>	135	ESA	end of selected area
	<HTS>	136	HTS	horizontal tab set
	<HTJ>	137	HTJ	horizontal tab set with justification
	<VTS>	138	VTS	vertical tab set
	<PLD>	139	PLD	partial line down
	<PLU>	140	PLU	partial line up
	<RI>	141	RI	reverse index
	<SS2>	142	SS2	single shift 2
	<SS3>	143	SS3	single shift 3

ZK-1737/3-84

Table A–3 (Cont.) Abbreviations and Descriptions of the DEC Multinational Character Set

Graphic	EDT Symbol	Decimal Value	Abbrev.	Description
	<DCS>	144	DCS	device control string
	<PU1>	145	PU1	private use 1
	<PU2>	146	PU2	private use 2
	<STS>	147	STS	set transmit state
	<CCH>	148	CCH	cancel character
	<MW>	149	MW	message waiting
	<SPA>	150	SPA	start of protected area
	<EPA>	151	EPA	end of protected area
	<X98>	152	---	|reserved|
	<X99>	153	---	|reserved|
	<X9A>	154	---	|reserved|
	<CSI>	155	CSI	control sequence introducer
	<ST>	156	ST	string terminator
	<OSC>	157	OSC	operating system command
	<PM>	158	PM	privacy message
	<APC>	159	APC	application program command
	<XA0>	160	---	|reserved|
¡	<!!>	161	¡	inverted exclamation mark
¢	<C/>	162	¢	cent sign
£	<L->	163	£	pound sign
	<XA4>	164	---	|reserved|
¥	<Y->	165	¥	yen sign
	<XA6>	166	---	|reserved|
§	<S0>	167	§	section sign
¤	<X0>	168	¤	general currency sign
©	<C0>	169	©	copyright sign
ª	<a_>	170	ª	feminine ordinal indicator
«	<<<>	171	«	angle quotation mark left
	<XAC>	172	---	|reserved|
	<XAD>	173	---	|reserved|
	<XAE>	174	---	|reserved|
	<XAF>	175	---	|reserved|
°	<0^>	176	°	degree sign
±	<+->	177	±	plus/minus sign
^{2}	<2^>	178	^{2}	superscript 2
^{3}	<3^>	179	^{3}	superscript 3
	<XB4>	180	---	|reserved|
µ	</U>	181	µ	micro sign
¶	<P!>	182	¶	paragraph sign, pilcrow
·	<.^>	183	·	middle dot
	<XB8>	184	---	|reserved|
^{1}	<1^>	185	^{1}	superscript 1
º	<o_>	186	º	masculine ordinal indicator
»	<>>>	187	»	angle quotation mark right
¼	<14>	188	¼	fraction one quarter
½	<12>	189	½	fraction one half
	<XBE>	190	---	|reserved|
¿	<??>	191	¿	inverted question mark

ZK-1737/4-84

A.3 DEC Multinational Character Set

Table A–3 (Cont.) Abbreviations and Descriptions of the DEC Multinational Character Set

Graphic	EDT Symbol	Decimal Value	Abbrev.	Description
À	<A`>	192	À	uppercase A with grave accent
Á	<A´>	193	Á	uppercase A with acute accent
Â	<A^>	194	Â	uppercase A with circumflex
Ã	<A~>	195	Ã	uppercase A with tilde
Ä	<A">	196	Ä	uppercase A with umlaut,(diaeresis)
Å	<A*>	197	Å	uppercase A with ring
Æ	<AE>	198	Æ	uppercase AE diphthong
Ç	<C,>	199	Ç	uppercase C with cedilla
È	<E`>	200	È	uppercase E with grave accent
É	<E´>	201	É	uppercase E with acute accent
Ê	<E^>	202	Ê	uppercase E with circumflex
Ë	<E">	203	Ë	uppercase E with umlaut, (diaeresis)
Ì	<I`>	204	Ì	uppercase I with grave accent
Í	<I´>	205	Í	uppercase I with acute accent
Î	<I^>	206	Î	uppercase I with circumflex
Ï	<I">	207	Ï	uppercase I with umlaut, (diaeresis)
	<XD0>	208	---	[reserved]
Ñ	<N~>	209	Ñ	uppercase N with tilde
Ò	<O`>	210	Ò	uppercase O with grave accent
Ó	<O´>	211	Ó	uppercase O with acute accent
Ô	<O^>	212	Ô	uppercase O with circumflex
Õ	<O~>	213	Õ	uppercase O with tilde
Ö	<O">	214	Ö	uppercase O with umlaut, (diaeresis)
Œ	<OE>	215	Œ	uppercase OE ligature
Ø	<O/>	216	Ø	uppercase O with slash
Ù	<U`>	217	Ù	uppercase U with grave accent
Ú	<U´>	218	Ú	uppercase U with acute accent
Û	<U^>	219	Û	uppercase U with circumflex
Ü	<U">	220	Ü	uppercase U with umlaut, (diaeresis)
Ÿ	<Y">	221	Ÿ	uppercase Y with umlaut, (diaeresis)
	<XDE>	222	---	[reserved]
ß	<ss>	223	ß	German lowercase sharp s
à	<a`>	224	à	lowercase a with grave accent
á	<a´>	225	á	lowercase a with acute accent
â	<a^>	226	â	lowercase a with circumflex
ã	<a~>	227	ã	lowercase a with tilde
ä	<a">	228	ä	lowercase a with umlaut, (diaeresis)
å	<a*>	229	å	lowercase a with ring
æ	<ae>	230	æ	lowercase ae diphthong
ç	<c,>	231	ç	lowercase c with cedilla
è	<e`>	232	è	lowercase e with grave accent
é	<e´>	233	é	lowercase e with acute accent
ê	<e^>	234	ê	lowercase e with circumflex
ë	<e">	235	ë	lowercase e with umlaut, (diaeresis)
ì	<i`>	236	ì	lowercase i with grave accent
í	<i´>	237	í	lowercase i with acute accent
î	<i^>	238	î	lowercase i with circumflex
ï	<i">	239	ï	lowercase i with umlaut, (diaeresis)

ZK-1737/5-84

Table A–3 (Cont.) Abbreviations and Descriptions of the DEC Multinational Character Set

Graphic	EDT Symbol	Decimal Value	Abbrev.	Description
	<XF0>	240	---	[reserved]
ñ	<n˜>	241	ñ	lowercase n with tilde
ò	<o`>	242	ò	lowercase o with grave accent
ó	<o´>	243	ó	lowercase o with acute accent
ô	<o^>	244	ô	lowercase o with circumflex
õ	<o˜>	245	õ	lowercase o with tilde
ö	<o">	246	ö	lowercase o with umlaut, (diaeresis)
œ	<oe>	247	œ	lowercase oe ligature
ø	<o/>	248	ø	lowercase o with slash
ù	<u`>	249	ù	lowercase u with grave accent
ú	<u´>	250	ú	lowercase u with acute accent
û	<u^>	251	û	lowercase u with circumflex
ü	<u">	252	ü	lowercase u with umlaut, (diaeresis)
ÿ	<y">	253	ÿ	lowercase y with umlaut, (diaeresis)
	<XFE>	254	---	[reserved]
	<XFF>	255	---	[reserved]

ZK-1737/6-84

B Expressions

The following table lists data operations and comparisons in order of precedence, beginning with the highest:

Operator	Precedence	Description
+	1	Indicates a positive number
–	1	Indicates a negative number
*	2	Multiplies two numbers
/	2	Divides two numbers
+	3	(1) Adds two numbers (2) Concatenates two character strings
–	3	(1) Subtracts two numbers (2) Subtracts two character strings
.EQS.	4	Tests if two character strings are equal
.GES.	4	Tests if first character string is greater than or equal
.GTS.	4	Tests if first character string is greater than
.LES.	4	Tests if first character string is less than or equal
.LTS.	4	Tests if first character string is less than
.NES.	4	Tests if two character strings are not equal
.EQ.	4	Tests if two numbers are equal
.GE.	4	Tests if first number is greater than or equal to
.GT.	4	Tests if first number is greater than
.LE.	4	Tests if first number is less than or equal to
.LT.	4	Tests if first number is less than
.NE.	4	Tests if two numbers are not equal
.NOT.	5	Logically negates a number
.AND.	6	Combines two numbers with a logical AND
.OR.	7	Combines two numbers with a logical OR

The following tables demonstrate the results of logical operations on a bit-by-bit basis and a number-by-number basis. In logical operations, a character string beginning with an uppercase or lowercase T or Y is treated as the number 1; a character string beginning with any other character is treated as the number 0. In logical operations, odd numbers are considered true and even numbers and zero are considered false.

Given Bit A	Bit B	Results .NOT A	A .AND. B	A .OR. B
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Given Number A	Number B	Results .NOT A	A .AND. B	A .OR. B
odd	odd	even	odd	odd
odd	even	even	even	odd
even	odd	odd	even	odd
even	even	odd	even	even

Index

Index

Reader's Comments

Guide to Using VMS
AA–LA05A–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

Reader's Comments

Guide to Using VMS
AA–LA05A–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