VAX/VMS
Primer

Order No. AA-D030C-TE

digital
software

VAX/VMS
Primer

Order No. AA-D030C-TE

May 1982

This tutorial document introduces the DIGITAL Command Language, the EDT editor, file manipulation, program development, and basic operating system concepts.

REVISION/UPDATE INFORMATION: This revised document supersedes the VAX/VMS Primer (Order No. AA-D030B-TE).

SOFTWARE VERSION: VAX/VMS Version 3.0

digital equipment corporation · maynard, massachusetts

First Printing, August 1978
Revised, March 1980
Revised, May 1982

HOW TO ORDER ADDITIONAL DOCUMENTATION

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

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

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

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

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

Contents

Tables

Preface

A primer is a book for beginners. The *VAX/VMS Primer* introduces new users to the VAX/VMS operating system. Depending upon your prior experience, you may want to read this book carefully or just skim it for specific details.

Between you and the computer is VAX/VMS, the operating system. VAX/VMS allows users to share the resources of the computer and its hardware devices, such as disks, tapes, terminals, and printers.

Communication with the operating system consists of commands that you give it and messages that it gives in response. The set of commands that an operating system recognizes constitutes its command language.

DIGITAL Command Language, or DCL, is the name of the command language you use to communicate with VAX/VMS. One of the main objectives of this primer is to introduce you to the DIGITAL Command Language.

Eventually, you will need to know more information about VAX/VMS, its command language, and its services than this primer presents. Sections entitled "For More Information" conclude each of the following chapters; these sections direct you to further sources of specific information about material discussed in the preceding chapter. In addition, the *VAX–11 Information Directory and Index* provides a guide to the entire VAX/VMS software document set. See Table 1 for a list of VAX–11 documentation organized by subject.

Table 1: Sources of Information in VAX-11 Documentation

For information on:	Look in these manuals:
VAX/VMS Documentation	VAX-11 Information Directory and Index
VAX/VMS Installation, Management, and Operations:	VAX/VMS Release Notes VAX-11/780 Software Installation Guide VAX-11/750 Software Installation Guide VAX-11/730 Software Installation Guide VAX-11/782 User's Guide VAX/VMS System Management and Operations Guide VAX/VMS UETP User's Guide
DIGITAL Command Language	VAX/VMS Command Language User's Guide VAX/VMS Guide to Using Command Procedures
Program Development, Testing, and Control	VAX-11 SOS Text Editing Reference Manual EDT Editor Manual VAX/VMS DIGITAL Standard Runoff User's Guide VAX/VMS Magnetic Tape User's Guide VAX-11 Linker Reference Manual VAX-11 Symbolic Debugger Reference Manual VAX/VMS Real-Time User's Guide VAX/VMS Guide to Writing a Device Driver VAX/VMS System Messages and Recovery Procedures Manual
File and Record Management	Introduction to VAX-11 Record Management Services VAX-11 Record Management Services Reference Manual VAX-11 Record Management Services Utilities Reference Manual VAX-11 Record Management Services Tuning Guide
Programming Utilities and Development Tools	VAX-11 Utilities Reference Manual VAX-11 PATCH Utility Reference Manual VAX-11 SORT/MERGE User's Guide VAX/VMS System Services Reference Manual VAX/VMS I/O User's Guide VAX-11 Run-Time Library User's Guide VAX-11 Run-Time Library Reference Manual VAX/VMS System Dump Analyzer Reference Manual VAX-11 Guide to Creating Modular Library Procedures
Compatibility Mode Programming	VAX-11/RSX-11M Programmer's Reference Manual VAX-11/RSX-11M User's Guide

Table 1: (Cont.) Sources of Information in VAX-11 Documentation

For information on:	Look in these manuals:
Networking	DECnet-VAX System Manager's Guide DECnet-VAX User's Guide
VAX-11 Programming Languages	VAX-11 BASIC Installation Guide and Release Notes VAX-11 BASIC Language Reference Manual VAX-11 BASIC User's Guide
	BLISS Language Guide BLISS Pocket Guide VAX-11 BLISS-32 User's Guide VAX-11 BLISS-16 User's Guide
	Installing VAX-11 C Programming in VAX-11 C
	VAX-11 COBOL Installation Guide/Release Notes VAX-11 COBOL Language Reference Manual VAX-11 COBOL Pocket Guide VAX-11 COBOL-74 Translator Utility VAX-11 COBOL User's Guide VAX-11 COBOL-74 Installation Guide/Release Notes VAX-11 COBOL-74 Language Reference Manual VAX-11 COBOL-74 User's Guide
	CORAL 66 Language Reference Manual
	VAX-11 FORTRAN Installation Guide/Release Notes VAX-11 FORTRAN Language Reference Manual VAX-11 FORTRAN User's Guide
	VAX-11 MACRO Language Reference Manual VAX-11 MACRO User's Guide
	VAX-11 PASCAL Installation Guide/Release Notes VAX-11 PASCAL Language Reference Manual VAX-11 PASCAL Primer VAX-11 PASCAL User's Guide
	Introduction to VAX-11 PL/I Programming in VAX-11 PL/I VAX-11 PL/I Encyclopedia Reference VAX-11 PL/I Guide to Program Debugging VAX-11 PL/I Installation and System Management Guide VAX-11 PL/I Language Summary VAX-11 PL/I User's Guide

Graphic Conventions Used in this Primer

ESC RET
DEL TAB

These symbols indicate that you press the ESCAPE, RETURN, DELETE, or TAB key on the terminal.

CTRL/Y CTRL/Z
CTRL/R CTRL/S
CTRL/U CTRL/Q
CTRL/C CTRL/O

These symbols indicate that you hold down the CTRL key while you press a terminal key, for example, Y. These symbols represent control key sequences. In some examples, control key sequences are shown as a circumflex (ˆ) and a letter, for example ˆY, because that is how the system displays them.

```
$ show time
  05-JUN-1982 11:55:22
```

In examples of commands you enter and system responses, all the lines you type are shown in red letters. Everything the system prints or displays is shown in black letters.

```
$ type myfile.dat
      .
      .
      .
```

A vertical ellipsis in an example means that not all the data the system would display in response to the particular command is shown; or that not all the data a user would enter is shown.

All **commands** are

A word or phrase in bold indicates a term defined in the Glossary at the end of this primer.

Chapter 1
Accessing the System and Typing Commands

To communicate with the VAX/VMS **operating system** you use a **terminal** that is connected to the computer. You tell the operating system what to do by typing a **command** on the terminal's keyboard. The system responds by executing your command. If the system cannot interpret what you type, it displays an error message at your terminal; when the command has been successfully executed, you type another.

When you communicate with the system in this manner, you are an **interactive** user. A **batch** user, in contrast, communicates with the system by submitting all commands at one time in a batch **job**. This primer emphasizes how to use VAX/VMS interactively. Chapter 6 shows how any of the commands described can be submitted in a batch job.

1.1 Terminals

The terminals you can use to communicate with the VAX/VMS operating system fall into two general categories: hard-copy terminals and video display terminals.

Hard-copy terminals print on continuous forms of paper. Figure 1–1 shows one type of hard-copy terminal, the LA120.

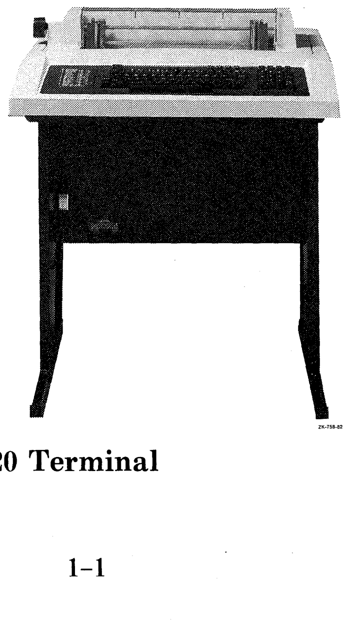


Figure 1–1: The LA120 Terminal

Video display terminals display your typed input and system responses on a screen similar to that of a television. Figure 1–2 shows an example of a video display terminal, the VT100.

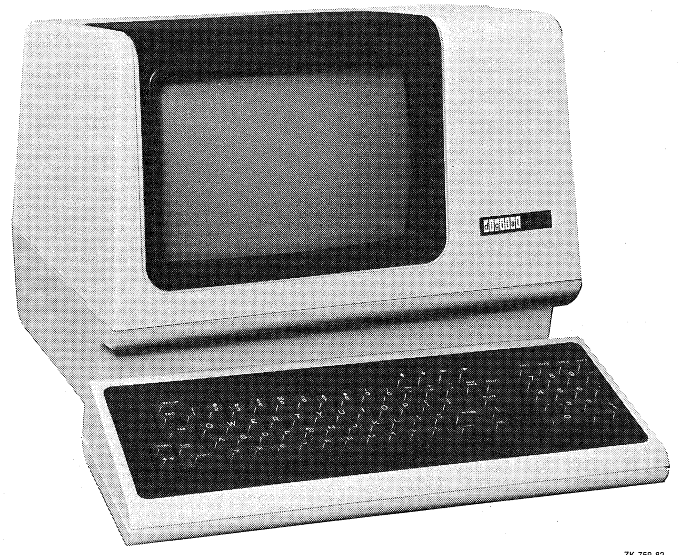


Figure 1–2: The VT100 Terminal

1.2 Keyboards

Figure 1–3 shows the keyboard layouts of the LA120 and VT100 terminals. All terminal keyboards have the same basic configuration as typewriter keyboards. However, a terminal has additional keys that provide special signals to the operating system; to use the terminal effectively, you should become familiar with these keys.

In Figure 1–3, arrows point to the most commonly used of the special keys. The symbols in the figure are used throughout this text as a shorthand notation to refer to pressing these special keys. For example, the symbol RET in text means that you press the key labeled RETURN. CTRL/x means that you hold down the key labeled CTRL while you press another key. To issue a CTRL/Y, for instance, hold the CTRL key down and press the Y key.

1.3 Logging In

To begin a session at the terminal, you must **log in**. Logging in consists of getting the system's attention and identifying yourself as an authorized user.

Before you can log in to the system, however, you must have an account. Accounts are set up by the system manager, or whoever is responsible at your installation for authorizing the use of the system. This person must provide you with a **user name** and a **password**.

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

Your password, however, is for your protection. If you maintain its secrecy, other users cannot use system resources under your user name or gain access to files you wish to keep private.

When you log in, you must enter both your user name and your password before VAX/VMS allows you to begin typing commands.

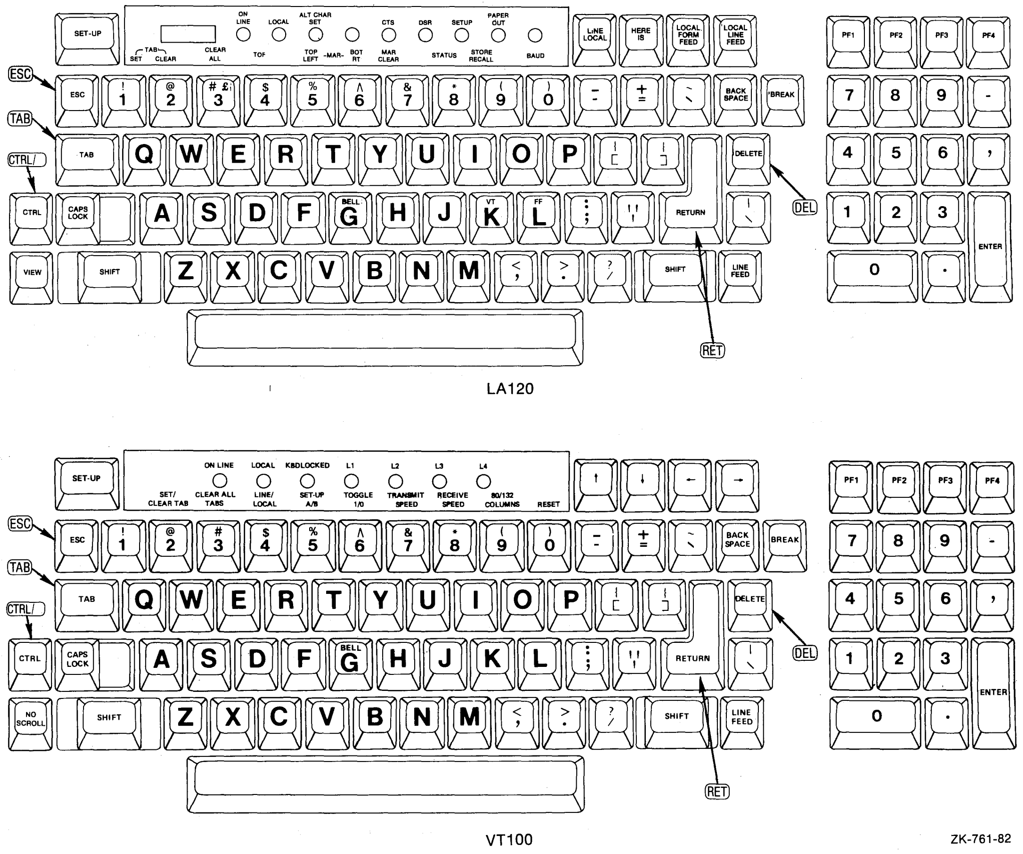


Figure 1-3: The LA120 and VT100 Keyboard Layouts

1.3.1 Getting the Terminal Ready

Before you use the terminal, be sure that:

- The terminal is plugged in and the power is turned on.
- If the terminal has a LOCAL/REMOTE switch, the switch is set to REMOTE. (If you are using a dial-up connection, check installation instructions for special procedures.)

The terminal should then be ready to accept your login. If you have any problems with the login procedure described in the next section, get help from the system operator or system manager. The terminal may not be properly connected to the computer, or the baud rate (the speed at which the terminal transmits or receives characters) may not be correctly set.

1.3.2 Gaining Access to the System

Press RET or CTRL/Y to signal the system that you want to log in. The system responds by prompting you for your user name. Enter your user name and press RET. The system displays your user name as you type it. After you enter your user name and press RET, the system prompts you to enter your password. When you type the password, the system does not display it; this preserves the secrecy of your password.

The login sequence looks like this:

```
RET
Username:   MALCOLMRET
Password:   RET
          WELCOME TO VAX/VMS VERSION 3.0
$
```

The dollar sign is a symbol the system uses as a **prompt**. When VAX/VMS displays this character in the left margin, it indicates that the login was successful and that you can begin entering commands to the system.

Note that if you type your user name or your password incorrectly, the system displays an error message. When an error message appears, you must repeat the login procedure.

1.4 Entering Commands

All commands to the system are words, generally verbs, that describe the functions they perform. You can type them in upper or lower case. For example:

```
$ show timeRET
```

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

```
  17-JUL-1982 11:55:40
$
```

The commands are part of the DIGITAL Command Language (DCL), which has its own vocabulary of **keywords** and rules of grammar. The vocabulary consists of commands, **parameters**, and **qualifiers**. The grammar consists of rules for using these keywords.

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

```
$ print myfile.lisRET
```

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

Command qualifiers restrict or modify the function the command is to perform. For example:

```
$ print/copies=2 myfile.lis(RET)
```

In this command, /COPIES=2 is a qualifier that indicates how many copies of the file MYFILE.LIS you want printed. A slash character (/) precedes the qualifier.

The entire command string, including the command and any parameters or qualifiers it may have, is called the **command line**. The keywords (commands, qualifiers, and parameters) that make up the DIGITAL Command Language have predefined meanings; therefore, you must use them exactly as defined, in some cases supplying a value to complete them. For example, the /COPIES qualifier for the PRINT command requires a value: you supply the number of copies you want printed.

The rules of grammar for the DIGITAL Command Language (that is, the order of the words, the spacing, and the punctuation) are also strictly defined. The *VAX/VMS Command Language User's Guide* contains a dictionary of DCL commands and discusses the rules of grammar.

1.4.1 Command Prompting

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

```
$ print(RET)
$_File:    myfile.dat(RET)
```

In this example, no parameter was entered, so the system prompted for the name of the file to be printed.

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

```
$ copy(RET)
$_From:   file1.dat(RET)
$_To:     file2.dat(RET)
```

In this example, each file name is entered separately. You could, however, enter both file names after the first prompt, as shown below:

```
$ copy(RET)
$_From:   file1.dat  file2.dat(RET)
```

You could also enter the entire command line on one line.

```
$ copy  file1.dat  file2.dat(RET)
```

1.4.2 Abbreviating Commands

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

For example, the SET, SEARCH, and SHOW commands all begin with the letter "S." To make the SHOW command unique, you must type at least two characters, SH. To make the SET and SEARCH commands unique, you must type three characters, SET and SEA respectively.

The examples in this primer show full commands so that you can become familiar with the commands and what they do.

1.4.3 Recovering from Errors

Some of the keys noted on the keyboard in Figure 1-3 provide editing functions that you can use to correct mistakes you make while typing commands. These keys are:

DEL

Backspaces over one character typed on the current line, then deletes the character. Most video display terminals actually move the **cursor** (an underline or block that marks your position) backward and erase the character when you press DEL. Otherwise, the terminal prints a backslash character (\ ), then each deleted character, then another backslash before it prints the next character you enter. On some terminals, the key that performs the delete function is marked RUBOUT.

CTRL/U

Ignores the current line and performs a return so you can reenter the entire line. Use CTRL/U when a line contains a number of mistakes and it would be tedious to use DEL.

CTRL/R

Performs a return and displays the current line, leaving the print element or cursor at the end of the line so you can continue typing input. Use CTRL/R when you have deleted a lot of characters on a line, and cannot read the line easily because of the backslash characters. For example:

```
$ pron\no\int mu\u\yCTRL/R
$ print my
```

CTRL/C and CTRL/Y

Cancel an entire command, regardless of how many lines were used to enter it.

You can also use CTRL/Y or CTRL/C to interrupt the system while it is executing a command. Such an interruption is useful in cases when you have entered a command and you want to stop it. Press CTRL/Y (or CTRL/C) and then issue the STOP command, as shown below:

```
$ type myfile.lis RET
    .
    .
    .
CTRL/Y
$ stop RET
$
```

In this example, CTRL/Y interrupted the typing of a long file and the STOP command terminated the output.

CTRL/S and CTRL/Q

To suspend and resume the upward movement, or **scrolling**, of the terminal display, use CTRL/S and CTRL/Q. To temporarily stop the display from scrolling, press CTRL/S; to continue the scrolling, press CTRL/Q. (The NO SCROLL key on VT100 terminals performs the same function.)

1.5 System Responses

The system can respond to your command in several ways. It can execute the command, indicating successful completion with the dollar sign prompt. It can execute the command and inform you in a message of what it has done. It can, if execution is not successful, inform you of errors you have made. It can even act for you, supplying values that you have not supplied yourself.

1.5.1 Defaults

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

1.5.2 Information Messages

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

```
$ print myfile.lis RET
  Job 210 entered on queue SYS$PRINT
```

Not all commands display informational messages; in fact, successful completion of a command is most commonly indicated by a dollar sign prompt for another command. Unsuccessful completion is always indicated by one or more error messages.

1.5.3 Error Messages

If you enter a command incorrectly, the system displays an error message and prompts for a command line as if no command had been entered:

```
$ capy(RET)
%DCL-W-IVVERB, unrecognized command
  \CAPY\
$
```

The three-part code preceding the text of the message indicates that the message is from DCL, the command interpreter; that it is a warning (W) message; and that the mnemonic for this particular message is IVVERB.

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

```
$ print nofile.dat(RET)
%PRINT-W-OPENIN, error opening DBA1:[MALCOLM]NOFILE.DAT;
as input
-RMS-E-FNF, file not found
```

The first message is from the PRINT command: it tells you it cannot open the specified file. The second message indicates the reason for the first, that is, the file cannot be found. "RMS" refers to the VAX/VMS file-handling facility, Record Management Services; error messages related to file handling are generally VAX-11 RMS messages.

1.6 The HELP Command

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

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

```
$ help(RET)
```

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

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

```
Topic? print RET
```

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

If you want to know more about one of the PRINT command's qualifiers, respond to the prompt "PRINT subtopic?" with that qualifier. For example, to display information about the /COPIES qualifier of the PRINT command, type:

```
PRINT subtopic? /copies RET
```

If you know the subtopic on which you need help to begin with, you can simply type:

```
$ help print/copies RET
```

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

1.7 Logging Out

When you have finished using the computer, use the LOGOUT command to end the terminal session:

```
$ logout RET
```

The system responds:

```
MALCOLM   logged out at 17-JUL-1982 12:43:10.38
```

Note that neither shutting off your terminal, nor setting the REMOTE/-LOCAL switch to LOCAL, automatically causes you to **log out**. To ensure that you have logged out, you should use the LOGOUT command to end a terminal session. If you shut a terminal off without logging out properly, another user may be able to turn the terminal on later and use your account.

1.8 For More Information

The *VAX/VMS Command Language User's Guide* is the primary reference manual for information about the DIGITAL Command Language. The manual contains complete descriptions of DCL commands, defines the grammar of the DCL command language, and illustrates command usage with many examples.

Chapter 2
Using an Editor: EDT

Before you use the VAX/VMS operating system, you need to know how to identify and create files. This chapter briefly describes file identification and editing with the EDT **editor**. If you are familiar with editors, you may want to skip this chapter and go directly to Chapter 3, Commands to Manipulate Files.

2.1 Files

A file is the basic unit of storage for VAX/VMS. All user information is stored in files, usually on auxiliary storage media such as magnetic tapes or disks. In order to retrieve a file from storage or to create and store a new one, you must be able to identify the file for the system.

VAX/VMS uses a unique **file specification** to identify each file. A complete file specification includes the user's **network node**, the **device** on which the file is stored, the **directory** in which the file is cataloged, and the **name**, **type**, and **version** of the file. Only the file name, type, and version are discussed here. For a more complete explanation of file specifications, see Chapter 3.

2.1.1 File Names

By taking advantage of defaults, you can identify a file by specifying only its file name and file type in the format:

```
filename.type
```

The file name can have up to nine characters selected from the letters A through Z and the numbers 0 through 9. When you create files, you can give them any names that are meaningful to you.

2.1.2 File Types

The file type can be from zero to three characters and must be preceded by a period. Again, you can choose from the letters A through Z and the numbers 0 through 9 for the file type. However, the file type usually describes specifically the kind of data in the file, and the system recognizes several default file types used for special purposes. (See Sections 3.1.4 and 4.1 for tables of default file types.)

2.1.3 File Versions

In addition to a file name and file type, every file also has a version number to distinguish it from other copies of the file. The version number is the last item in the file specification and is connected to the file type by a semicolon (;) or a period (.). The following example shows the format for version two of the file MYFILE.DAT:

```
myfile.dat;2
```

The system supplies version numbers by default if you do not specify them yourself. When you create a file, the system always assigns it a version number of 1. When you edit the file, its version number is automatically increased by 1. If you refer to an existing file without specifying a version number, the system always locates the most recent version (that is, the file with the highest version number).

You can override these default version numbers by explicitly specifying a version number:

```
$ print myfile.dat;3
```

This PRINT command requests that the third version of the file MYFILE.DAT be printed.

2.2 Editors

To edit a file you use a program called an editor, which has its own set of commands for modifying files. VAX/VMS supports four editors. (There may be other editors in addition to these at your installation.) Two, SUM and SUMSLP, are batch editors. To use a batch editor, you submit a list of the changes you want made to the file and the editor makes the changes as a batch job.

VAX/VMS also supports the SOS and EDT editors. With both, you conduct an interactive editing session: you give an editing command to which the editor responds. This interaction continues until you give the final command to store or delete the edits you have made.

To invoke any of these editors type the EDIT command, specifying as a qualifier the name of the editor you want to use and, as a parameter, the name and type of the file you want to edit. For example, to invoke SOS to edit the file MYFILE.DAT type:

```
$ edit/sos myfile.dat
```

Because /EDT is the default qualifier for the EDIT command, you invoke the EDT editor when you type:

```
$ edit myfile.dat
```

2.3 Introduction to EDT

The DIGITAL Standard Editor, EDT, has a number of advanced features that allow you to tailor it to your own needs. Some of these features are mentioned in Section 2.6; however, most are beyond the scope of this primer. For other sources of information about EDT and other editors, see Section 2.7.

EDT provides two basic methods of editing: line and keypad editing. When you use EDT's **line editor**, you specify both the editing command and the line(s) of text you want the command to affect. The EDT line editor is available on both hard-copy and video display terminals.

The EDT keypad editor is designed to be used on video terminals that display the editing as it takes place. When you use EDT's keypad editor, you move the cursor directly to the text you want to change and press either one or two keys to perform most of the commands.

If you use a video terminal, you can use both keypad and line editing in EDT. If you use a hard-copy terminal, you should skip Section 2.4 and go to Section 2.5, Line Editing in EDT.

2.4 Keypad Editing in EDT

The **keypad** is the small group of keys to the right of the larger keyboard on your terminal. Each key in the keypad performs at least one editing function; most of the keys perform two.

As you can see in the figure below, the keypads of the VT100 and the VT52 differ slightly, so that the keys performing a particular command are not always the same on both terminals. Therefore, the sample editing sessions that follow generally refer to the command (WORD) and its function (moves the cursor forward or backward in word units), rather than to the key itself (key 1). To learn which functions the keys on your keypad perform, refer to the diagrams below or the *EDT Editor Reference Card*.

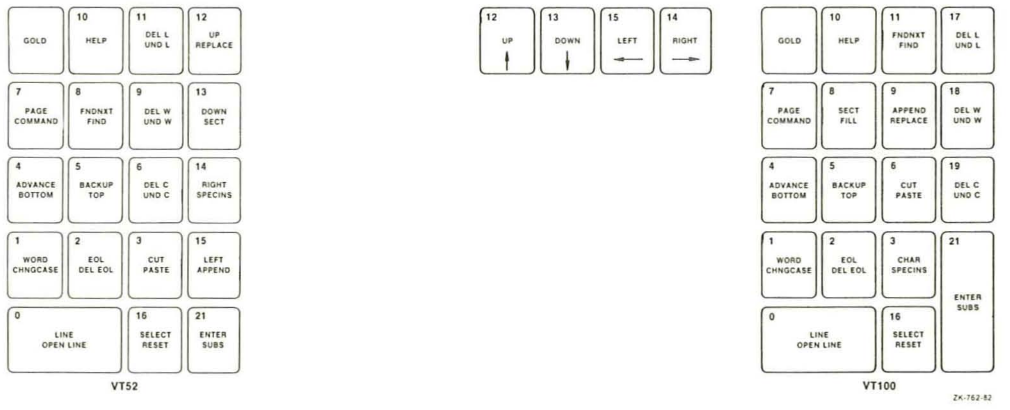
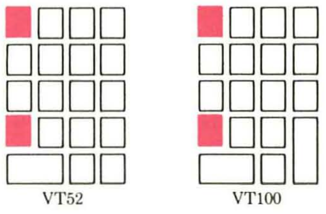


Figure 2-1: The VT52 and VT100 Keypads

2.4.1 The Gold Key

As Figure 2–1 illustrates, most of the keys perform two editing functions. To use the alternate, or lower, function of a key, press the GOLD key before you press the second key. For example, key 1 (on both the VT52 and the VT100) performs two functions: WORD and CHNGCASE (Change Case). To enter the WORD command, press key 1; to enter the CHNGCASE command, press the GOLD key first and then key 1.

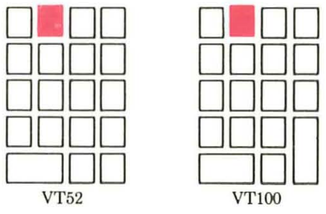
In the examples that follow, small diagrams of the VT52 and VT100 keypads highlight the keys that perform the command being described. The text to the left of the diagrams displays the effect of the command. For example, the CHNGCASE command is pictured as:



Note that the GOLD key must be pressed before the second key in order to invoke that key's alternate function.

2.4.2 The HELP Command

The EDT keypad editor provides a HELP key that displays a diagram of the keypad and the various key functions:



If you want information about a specific key, press the HELP key and then press the key in question. For example, if you want to know the function of the WORD command, press the HELP key to find which key performs the WORD function and then press key 1 for an explanation.

You can continue to get help on specific keys by pressing them. When you want to return to your editing, type a space. EDT resumes the display of your file, returning the cursor to its previous position.

2.4.3 Creating a File

To create a file with EDT, type the EDIT command, specifying the file's name as a parameter:

```
$ edit myfile.dat RET
Input file does not exist
[EOB]
*
```

The first line that appears is the message "Input file does not exist" telling you that you have no existing file named MYFILE.DAT (since you are creating that file). The second line, "[EOB]," denotes the end of the **buffer**, the temporary storage area in which you edit text before it is stored in a file. The third line, the asterisk, is the EDT line editor prompt.

To use EDT's keypad editor, type "C" (for the CHANGE command) after the asterisk prompt and press RETURN. The screen goes blank with only [EOB] in the upper left hand corner, indicating that there is nothing currently stored in the buffer. (In keypad editing, EDT always displays the [EOB] symbol on the line after the last character in the buffer.)

Using the main keyboard, type the following text to create a new file:

```
Hey, diddle diddle, RET
the cat played the fiddle, RET
the cow jumped over the moon. RET
This is the fourth line. RET
Use CTRL/Z to change from RET
keypad to line editing.
[EOB]
```

What you type is not recorded in a file until you instruct EDT to copy what you have entered from the temporary storage of a buffer to a file. You can then retrieve the file at a later time and make changes to it, such as adding and deleting text.

To save, or store, your edits, enter the EXIT command. First, invoke the line mode prompt by typing CTRL/Z. Then type EXIT after the asterisk and press RETURN:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.CTRL/Z
[EOB]

*EXIT RET
```

When you enter the EXIT command, the system saves the contents of the buffer in a file named MYFILE.DAT;1, issues the message below, and returns you to the DCL command level:

```
DB2:[MALCOLM]MYFILE.DAT;1 6 lines
$
```

If you do not want to save the edits you have just made, type QUIT after the asterisk prompt. EDT will delete the contents of the buffer without creating a file and return you to the DCL command level.

2.4.4 Editing a File

You invoke EDT to edit an existing file the same way you do to create one: type the EDIT command, specifying the file name as a parameter.

```
$ edit myfile.dat
     1            Hey, diddle diddle,
*
```

The first line of the latest version of the file, as well as the line editor prompt, will appear on the screen. (Note that if you type the wrong file name in the EDIT command line, you can terminate EDT without altering that file by using the QUIT command.)

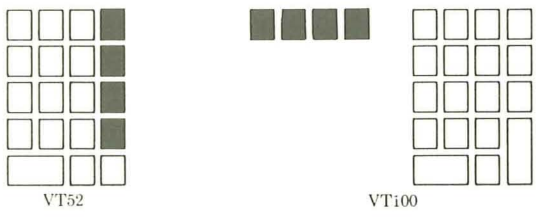
Type "C" for the CHANGE command to switch from line to keypad editing: EDT displays the first 22 lines of your file on your screen. (The EDT keypad editor does not display line numbers.)

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```

You can now begin editing your file.

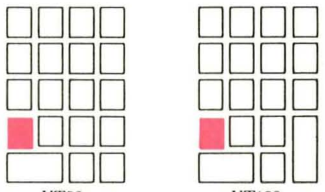
2.4.5 Manipulating the Cursor

The right and left arrow keys move the cursor one character in the direction of the arrow on the key. (When the left arrow is at the beginning of a line, however, it moves vertically, and when the right arrow is at the end of a line of text or the beginning of a blank line, it also moves vertically.) The location of the arrow keys on your terminal depends on whether you have a VT52 or a VT100:



You can also move the cursor by larger units of text, such as words, lines, sections, and pages. For example, give the WORD command to move the cursor to the beginning of the next word:

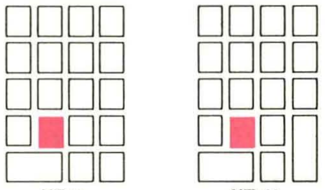
```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



(If your cursor does not move as described, it may be set to move backward rather than forward. See Section 2.4.7 to change the cursor's direction.)

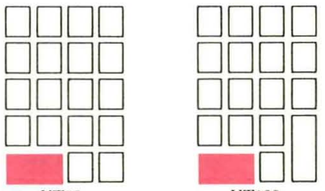
There are several ways to move through a file by line. To move the cursor to the end of the line, give the EOL (End Of Line) command:

```
Hey, diddle diddle,_
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



(The BACKSPACE key on the main keyboard moves the cursor in the opposite direction — to the left margin.) To move up or down a line you can use the up and down arrows, or you can use the LINE command, which moves the cursor to the beginning of the next line:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```

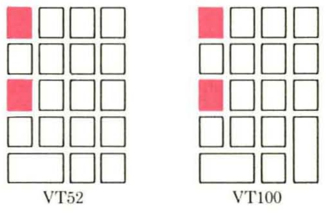


2.4.6 Scanning a File

In addition to moving the cursor by character, word, and line units, you can scan several lines of text at a time with the SECT (Section) and PAGE commands. SECT moves the cursor across a 16-line section of text; PAGE moves the cursor across a page[1] of text.

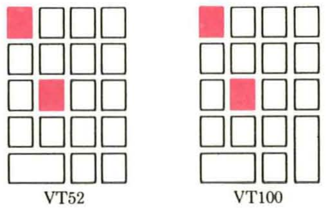
Another way to move the cursor quickly through a large portion of text is to move it directly to the beginning or end of the file. The BOTTOM command moves the cursor to the line following the last character:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



The TOP command moves it to the first character:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```

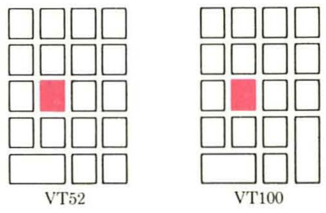


2.4.7 Changing the Cursor's Direction

Many commands (including the CHAR, WORD, LINE, EOL, SECT, and PAGE commands) move the cursor forward or backward unit by unit, depending upon whether you last set the direction of the cursor with the ADVANCE or BACKUP command. Each of these commands controls the cursor's direction until you set the cursor in the opposite direction with the other command.

To illustrate: give the BACKUP command to set the cursor in the backward direction. (The cursor does not move when you set its direction.)

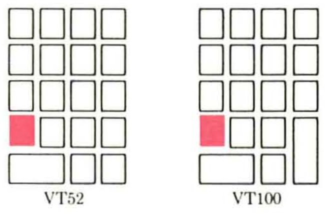
```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



1. By default a PAGE is defined as the text between form feed characters (ASCII characters that determine the start of each line printer page).

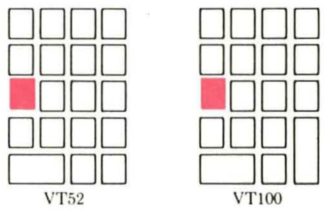
Now move the cursor one word (backward) with the WORD command:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]

Backup past top of buffer
```



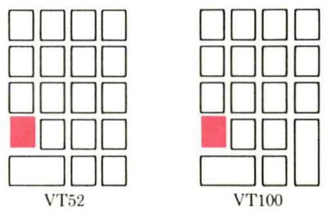
You should hear a bell (or buzzer) and see a message indicating that the command requests EDT to backup past the top of the buffer. Now reset the cursor's direction with the ADVANCE command. (Again, the cursor does not move.)

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



Move the cursor forward one word with the WORD command:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



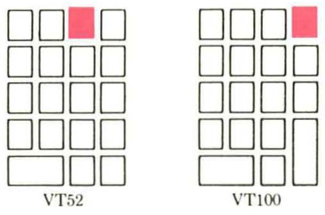
This time the cursor moves forward to the beginning of the next word. The cursor will remain set in this direction until you give the BACKUP command; you can change the cursor's direction whenever it is convenient.

2.4.8 Deleting and Restoring Text

The delete commands work in units similar to those that manipulate the cursor: there are commands to delete by character, by word, and by line. The deleted text is stored in a buffer so that you can restore it with an undelete command.

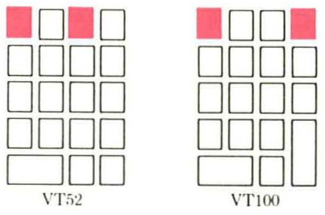
The delete and undelete commands work in the same way: you can delete by character (DEL C), word (DEL W), or line (DEL L); and you can restore that character (UND C), word (UND W), or line (UND L). For example, enter the DEL L (Delete Line) command:

```
Hey, the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



The deleted line disappears and is replaced on the screen by the line following it. Now give the UND L (Undelete Line) command to restore the line:

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```



The line reappears. The UND C (Undelete Character) and UND W (Undelete Word) commands work in exactly the same way, allowing you to restore the last character or word you deleted.

Note that the undelete commands restore only the corresponding units of text that were most recently deleted. If you have deleted two lines of text with the DEL L (Delete Line) command, for example, the UND L (Undelete Line) command will restore only one line — the line most recently deleted.

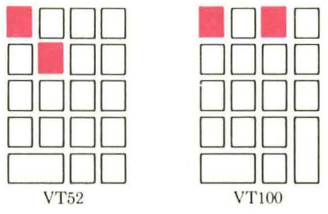
2.4.9 Locating Text

In addition to scanning text, you can move the cursor to a specific location in the file with the FIND and FNDNXT commands. The FIND command searches for a particular character string between the current position of the cursor and the beginning or end of the buffer, depending on whether the ADVANCE or BACKUP command is in control. The FIND command is especially useful with long files that would be tedious to scan with other commands.

For example, if you want to edit a particular string of text, specify that string and EDT will search for it. If EDT finds the string, the cursor will move to the beginning of the string. If EDT cannot find the string, the bell will ring and the message "String was not found" will appear on the screen.

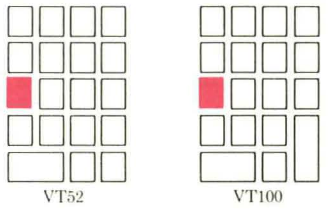
Assume you want to locate the word MOON. You could instruct EDT to search for the string by first giving the FIND command to invoke the Search for: prompt.

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]

Search for:_
```



Type MOON in response to the prompt and then press ADVANCE (since you want EDT to search in the forward direction).

```
Hey, diddle diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]

Search for: MOON
```



When EDT finds the string, it positions the cursor at the first character in the string. (In a long file the message "Working" may flash on the screen while EDT searches for the string.)

To find the next occurrence of the same string, give the FNDNXT (Find Next) command. If there is no other occurrence of the string (as in this instance), EDT will issue the message "String was not found."

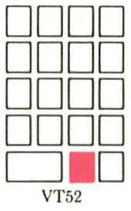
Note that the directional setting of the cursor determines the outcome of the search. You can use either ADVANCE or BACKUP to enter the search string, depending on the direction in which you want the cursor to search. (You can also use the ENTER command, which applies the current direction to the search.)

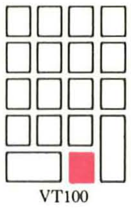
2.4.10 Moving Text

Moving text from one place to another in a file is called "cutting and pasting" in EDT: you cut out the text you want to move and paste it in the place where you want it. The first step in this operation is to select the range of text you want to move.

For example, to move the first line of text to the end of the file, move the cursor to the beginning of line one and press SELECT. This marks the beginning of the select range.

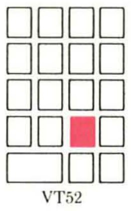
```
Hey, diddle, diddle,
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```

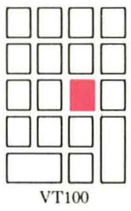




To mark the end of the select range, move the cursor to the end of line one. All text between the select point and the cursor will be affected by the CUT command. (A VT100 highlights the select range with **reverse video.**) Now press CUT:

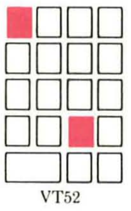
```
_
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
[EOB]
```

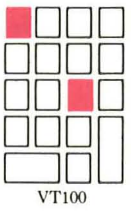




The select range (in this case, line one) disappears from the screen. EDT holds the text you delete with the CUT command in the PASTE buffer. To restore it, move the cursor to the location where you want to place the text and enter the PASTE command. For this example, move the cursor to the end of the file. Now enter the PASTE command:

```
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
Hey, diddle diddle,_
[EOB]
```





You can continue to use the PASTE command wherever you wish, since the text remains in the paste buffer until it is replaced by another CUT operation. The CUT and PASTE commands are especially useful in moving large pieces of text from one place to another in a file.

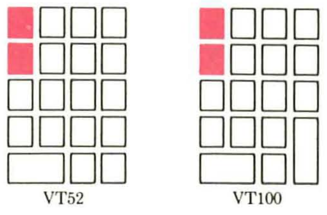
Select ranges are also used with other commands, such as the APPEND, FILL, and SUBSTITUTE commands. If you make a mistake during the process of marking the select range (or any keypad command requiring a sequence of keys), use the RESET command to cancel the first part of the command and start over.

2.4.11 Entering Line Commands in Keypad Mode

Occasionally, you may want to use line editing commands without actually changing from keypad to line mode. You can do so with the COMMAND function.

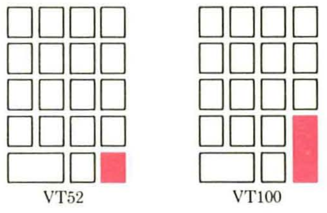
For example, to give the SET QUIET line command (which suppresses the ringing of the bell or buzzer when an EDT error occurs), first invoke the Command: prompt. Then type the command after the prompt.

```
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
Hey, diddle diddle,
[EOB]

Command: SET QUIET_
```



Use ENTER to submit the command. (If you press RETURN by mistake, ^M will appear. Delete the ^M and press ENTER.)

```
the cat played the fiddle,
the cow jumped over the moon.
This is the fourth line.
Use CTRL/Z to change from
keypad to line editing.
Hey, diddle diddle,_
[EOB]
```



You can also enter the EXIT and QUIT commands with the COMMAND function.

2.4.12 Changing From Keypad to Line Editing

If at some point during keypad editing you want to change to line editing, type CTRL/Z. The line-editing prompt, an asterisk, will appear at the bottom of the screen, indicating that EDT is ready to receive line-editing commands.

2.4.13 Subset of EDT Keypad-Editing Commands

Table 2-1 presents some of the most frequently used keypad commands and the main function of each. For a complete list of EDT keypad commands, see the *EDT Editor Manual*.

Table 2-1: Subset of EDT Keypad-Editing Commands

Command	Function
ADVANCE	Sets cursor movement in forward direction
BACKUP	Sets cursor movement in backward direction
BOTTOM	Moves the cursor to the bottom of the buffer
CHNGCASE	Changes the case of specified characters
CUT	Deletes specified text from the main buffer and stores it in the paste buffer
DEL C	Deletes the character at the cursor
DEL EOL	Deletes text from the cursor to the end of the current line
DEL L	Deletes text from the cursor through the end of the current line, moving the following line up to the cursor
DEL W	Deletes text up to the first character of the next word
EOL	Moves the cursor to the end of the current line
FIND	Locates specified text
FNDNXT	Locates the next occurrence of specified text
HELP	Invokes EDT's keypad help facility
LINE	Moves the cursor to the beginning of the next line
PAGE	Moves the cursor across one page of text
PASTE	Inserts the contents of the paste buffer at the cursor's position
RESET	Cancels GOLD, SELECT, or any key sequence
SECT	Moves the cursor across 16 lines of text
SUBS	Deletes specified text and inserts the contents of paste buffer
TOP	Moves the cursor to the top of the current buffer
UND C	Restores the character deleted by the last DEL C
UND L	Restores the line deleted by the last DEL L
UND W	Restores the word deleted by the last DEL W
WORD	Moves the cursor one word
CTRL/W	Restores the video display
CTRL/Z	Changes from keypad editing to line editing

2.5 Line Editing in EDT

In line editing, you do not move the cursor directly to the character or word you want to edit. Instead, you type the command and specify the line or range of lines you want the command to affect. (Although most of the line-editing commands can be abbreviated, this chapter presents the full commands so that you will recognize their functions.)

2.5.1 The HELP Command

The EDT line editor has a HELP command, which works much like the DCL command HELP. If you need help with EDT line commands, type HELP after the asterisk prompt, and EDT will display all the line-editing commands available. To get help with a particular command, type HELP and the name of the command (for example, HELP TYPE); EDT will display information about that command.

2.5.2 Creating a File

To create a file with EDT's line editor, give the EDIT command, specifying the file's name as a parameter:

```
$ edit yourfile.dat
Input file does not exist
[EOB]
*
```

The first line is the message "Input file does not exist," indicating that you have no file in your default directory with the name YOURFILE.DAT (since you are creating that file). The second line, "[EOB]," denotes the end of the buffer, the temporary storage area in which you edit text before storing it in a file. The third line, the asterisk, is the prompt for EDT's line editor.

To insert text in your file, type INSERT after the prompt and press RETURN: the cursor moves forward several spaces. EDT inserts what you type after this command as text until you signal completion with CTRL/Z.

Type the following lines from the main keyboard, using the DELETE key and CTRL/U, which deletes text between the cursor and the left margin if you make a mistake. (You cannot return to a preceding line to make an addition or correction once you have pressed RETURN, unless you give additional commands.)

```
*INSERT RET
              Hey, diddle diddle, RET
              the cat played the fiddle, RET
              the cow jumped over the moon. CTRL/Z
[EOB]
*
```

Terminate the insertion of text by typing CTRL/Z. EDT displays the [EOB] sign and asterisk prompt, indicating it is ready to receive another line-editing command.

When you use EDT, what you type is not recorded in a file until you instruct the editor to copy your text from the temporary storage of a buffer to a file, which you can later retrieve for editing. When you enter the EXIT command, EDT saves the contents of the buffer in a file named YOURFILE.DAT;1, issues the message below, and returns you to the DCL command level (signified by the dollar sign prompt). Type EXIT after the asterisk prompt:

```
*EXIT RET
DB2:[MALCOLM]YOURFILE.DAT;1 3 lines
$
```

If you do not want to save your edits, type QUIT after the prompt. EDT will delete the contents of the buffer without creating a file and return you to the DCL command level.

2.5.3 Editing a File

You enter and exit EDT to edit an existing file the same way you do to create one. To invoke EDT, give the EDIT command and specify the file name as a parameter. The first line of the latest version of the file, as well as the line editor prompt will appear.

```
$ edit yourfile.dat
        1          Hey, diddle diddle,
*
```

Note that if you type the wrong file name in the EDIT command line, you can terminate EDT without altering that file by using the QUIT command.

2.5.4 Specifying Ranges

EDT initially assigns line numbers in increments of one to each line in the buffer. You can use these numbers to specify the line or range of lines you want the command to act upon. EDT also recognizes certain words in **range specifications**; you can combine these words with line numbers in a range specification. For example, if you want to delete the first two lines in the file, you could type: "DELETE 1 THRU 2" or "DELETE BEGIN:2."

Table 2–2 lists a small subset of range specifications possible in line editing to help you begin using the line editor. Once you have begun, you can use the line editor's HELP facility and the *EDT Editor Manual* for complete instructions on specifying ranges.

Table 2–2: Subset of EDT Line-Editing Range Specifications

Specification	Description
.	The current line
n THRU n	Line number n through line number n
n : n	Line number n through line number n
BEGIN	The first line of the buffer
END	The empty line after the last line of the buffer
BEFORE	All the lines before the current line
REST	The lines including and after the current line
WHOLE	All the lines in the buffer

In addition to using range specifications, you can press RET to move forward one line:

```
           1   Hey, diddle diddle,
*RET
           2   the cat played the fiddle,
*RET
           3   the cow jumped over the moon.
```

2.5.5 Inserting and Deleting Text

The INSERT and DELETE commands take similar parameters. When you specify a range, EDT inserts the text before the first line of the specified range. (If you do not specify a range, EDT inserts the text immediately before the current line.) For example, to add text to the end of the buffer, you could specify line four or END to indicate the end of the buffer:

```
*INSERT END RET
                 This is the fourth line.RET
                 Type CHANGE to switchRET
                 from line to keypad editing.CTRL/Z
[EOB]
*
```

Terminate the insertion of text with CTRL/Z. EDT assigns line numbers to the new text and issues an asterisk prompt, indicating that it is ready to receive another editing command.

The DELETE command works in much the same way, deleting those lines of text specified in the range. If you do not specify a range, the current line is deleted by default. (There is no line-editing command equivalent to the UNDELETE keypad commands.)

2.5.6 Locating Text

To display the entire contents of the buffer, use the TYPE command and specify WHOLE as the parameter:

```
*TYPE WHOLERET
          1      Hey, diddle diddle,
          2      the cat played the fiddle,
          3      the cow jumped over the moon.
          4      This is the fourth line.
          5      Type CHANGE to switch
          6      from line to keypad editing.
[EOB]
*
```

If you want to locate a text string in a long file without displaying the entire buffer contents, you can use quotation marks. For example, to find the first occurrence of the word FOURTH, type:

```
*"fourth"RET
    4        This is the fourth line.
*
```

EDT moves the cursor to the line containing the first occurrence of the string. If you want to display all occurrences of the string, use the TYPE ALL command, followed by the string in quotation marks.

2.5.7 Substituting Text

To replace one character string in the current line with another, use the SUBSTITUTE command. To replace every occurrence of one string with another throughout the file, use WHOLE as the parameter.

For example, to replace all occurrences of the string THE with the string A, type SUBSTITUTE, the old string, and the new string, separating all three with **delimiters**, such as slashes. (You must use the same delimiter throughout the command line.) Then specify WHOLE as the parameter:

```
*SUBSTITIUTE/the/a/WHOLE(RET)
 2           a cat played a fiddle
 3           a cow jumped over a moon.
 4           This is a fourth line.

5 substitutions
*
```

EDT displays the total number of substitutions made and issues the asterisk prompt.

2.5.8 Copying and Moving Text

The MOVE command in line editing is similar to the CUT and PASTE commands in keypad editing: the MOVE command deletes text in one location and inserts it in another. The COPY command, in contrast, duplicates a range of text in another location, without altering the original text.

For example, with the COPY command you can duplicate the entire file: first type the command, then the range of lines to be copied, and finally the range of lines to which you want to copy the text.

```
*COPY 1:6 TO END
6 lines copied
*
```

Lines 1 through 6 are duplicated in lines 7 through 12. (You can display the change with TYPE WHOLE.)

When you have finished editing your file and are ready to store your changes, enter the EXIT command (as described in Section 2.5.2).

2.5.9 Changing from Line to Keypad Editing

If at some point you want to change from line editing to keypad editing, give the CHANGE command by typing "C" after the asterisk prompt. The screen will display without line numbers the first 22 lines of the file, and you can begin keypad editing immediately.

2.5.10 Subset of EDT Line-Editing Commands

Table 2–3 presents some of the more frequently used EDT line-editing commands and the main function of each. For a complete list of commands, see the *EDT Editor Manual*.

Table 2–3: Subset of EDT Line-Editing Commands

Command	Function
CHANGE	Changes from line editing to keypad editing
COPY	Duplicates text in specified location
DELETE	Deletes specified range of lines
EXIT	Ends an editing session by storing the buffer contents in a file
FIND	Locates specified line
HELP	Invokes EDT's line editing help facility
INCLUDE	Copies specified file into text buffer
INSERT	Inserts text in the buffer
MOVE	Deletes text in one location and inserts it in another
QUIT	Ends an editing session by deleting the contents of the buffer
RESEQUENCE	Assigns new line numbers in increments of one
SUBSTITUTE	Replaces one character string with another
TYPE	Displays a specified range of lines
WRITE	Copies a specified range from the buffer to the specified file

2.6 Special Features of EDT

EDT offers several special features, a few of which are mentioned here. For more information about these and other features of EDT, see the *EDT Editor Manual*.

2.6.1 Multiple Buffers

When you edit a file with EDT, you are working on a copy of the file in a buffer called MAIN. There are other buffers in addition to this main buffer. One buffer, called PASTE, is maintained by EDT for its own use, but the others are available for you to use as separate workspaces.

These additional buffers are useful in working with multiple pieces of text. You can move part or all of a buffer into another buffer or several other buffers. You can also copy a file into one of these buffers. To create a buffer for additional workspace, you first name it, after which you can enter and exit that buffer at any time.

2.6.2 Journal Files

When you edit a file with EDT, EDT keeps a journal file of all your edits. This file has the same file name as the file you are working on and the file type of

JOU. Should a system failure or inadvertent CTRL/Y end your editing session, the journal file has a record of all your edits, with the possible exception of those made just prior to the interruption.

To recover your lost edits, you use the same EDIT command and the exact qualifiers you used to begin the original editing session — plus the /RECOVER qualifier. For instance, if you began a session with the command EDIT YOURFILE.DAT, you would type EDIT/RECOVER YOURFILE.DAT to recover your edits using the journal file. EDT will then reenact the editing session, reading the commands from the journal file and executing them on the screen.

If no interruption occurs, EDT deletes the journal file when you exit from the editing session.

2.6.3 SET and SHOW Commands

The SET commands let you control certain characteristics of EDT's operation by setting certain parameters. Among the characteristics that you can set are: the length of line displayed on the screen (SET SCREEN width), the default delimiters for textual units, such as words and sentences (SET ENTITY), and the display of line numbers during line editing (SET[NO]NUMBERS). The SHOW commands display most of the characteristics you can set with the SET command.

2.6.4 Start-up Command Files

You can create an EDT start-up command file to specify the default characteristics for editing sessions. EDT looks for a start-up command file when you begin an editing session; if you have one, it reads the commands and applies the specifications to the current editing session. For example, if you place the command SET MODE CHANGE in your start-up command file, you will begin editing in keypad mode whenever you enter EDT.

2.6.5 Defining Keys

You can redefine the functions of the keys in the keypad and several of the CONTROL and keyboard keys with the line command DEFINE KEY or the keypad command CTRL/K. You can redefine keys for the current editing session, or you can put the new key definitions in your start-up command file.

2.6.6 Defining Macros

You can group several line-editing commands together and use them as one unit, or macro. To do so, give the DEFINE MACRO command and enter the commands in the proper sequence. Later, EDT will execute that series of commands whenever you type the name of the buffer containing the macro. You can define a macro for the current editing session only, or you can put it in your start-up command file.

2.7 For More Information

There are several sources of information about EDT. Both the line and keypad HELP facilities provide a quick reference for EDT commands. EDTCAI, the computer-aided course on EDT, is especially good for users unfamiliar with text editors. The *EDT Editor Reference Card* provides a summary of EDT's keypad editing features, and the *EDT Editor Manual* contains an extensive description of how to use EDT, including a complete list of EDT commands, their parameters, and qualifiers.

For information about SOS, see the *VAX-11 SOS Text Editing Reference Manual*; for information about batch editors SUM and SUMSLP, see the *VAX-11 Utilities Reference Manual.*

Chapter 3
Commands to Manipulate Files

The previous chapter described how to create and edit files using the EDT editor. This chapter explains how to use DCL commands to manipulate files: how to identify, create, delete, and purge files; how to create and list directories; and how to copy and rename files.

3.1 Identifying Files

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

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

The punctuation marks (colons, brackets, period, semicolon) are required syntax that separate the various components of the file specification.

3.1.1 Nodes

When computer systems are linked together to form a network, each system in the network is called a node, and is identified within the network by a unique node name. Your system may or may not be part of a larger network. To find out, type SHOW NETWORK. If your system is part of a network, you will see a list of node names displayed on your screen. (The node labeled "LOCAL" is your own system's node name.) If your system is not a part of a network, a message indicating that no network is available will appear.

If your system is a network node, you may be able to gain access to a file located at another node on the network by adding a **node specification** to the first part of the file specification. (This specification will allow you access to the file only if the user owning the file has permitted other users access to it.) If you do not specify a node, the system assumes by default that the file belongs to your own, or local, node. See the DECnet–VAX documentation for an explanation of gaining access to files across a network.

3.1.2 Devices

The second part of a file specification, the device name, identifies the physical device on which a file is stored. A device name has three parts:

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

Some examples of device names are:

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

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

3.1.3 Directories and Subdirectories

Since a disk can contain files belonging to many different users, each user of a given disk has a directory that catalogs all the files belonging to him on that device.

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

```
$ show default(RET)

DBA2:[MALCOLM]
```

This response from the SHOW DEFAULT command indicates that the user's default device is DBA2 and the default directory is [MALCOLM].

You can gain access to files in other directories (including directories that catalog files belonging to other users) by specifying the directory name in a file specification. For example, to display on your terminal the contents of a file named CONTENTS.DAT belonging to a user whose directory is [JONES], issue the TYPE command as shown below:

```
$ type [jones]contents.dat(RET)
```

Note that the file specification does not include a device name. For this command to execute successfully, the directory [JONES] must be on your default disk device. This is because the system always applies a default when you omit a device name. If user JONES's directory is on the disk DBB2 you would issue the TYPE command as:

```
$ type dbb2:[jones]contents.dat RET
```

In both of these examples, it is assumed that the user Jones has given other users access to files in the directory. You can explicitly allow or restrict access to your own files, either generally or on a file-by-file basis, with the SET PROTECTION command. See the *VAX/VMS Command Language User's Guide* for information about directory and file protection and for a description of the SET PROTECTION command.

Files can also be cataloged in subdirectories. A **subdirectory** is a file (cataloged in a higher directory) that contains additional files. A subdirectory name is formed by **concatenating** its name to the name of the directory that lists it. For example:

```
$ type [jones.datafiles]memo.sum RET
```

This TYPE command requests a display of the file MEMO.SUM that is cataloged in the subdirectory [JONES.DATAFILES]. The subdirectory file name is DATAFILES.DIR, and is cataloged in the directory [JONES].

Subdirectories are described in more detail in Section 3.8.

3.1.4 File Names, Types, and Versions

By taking advantage of your default node, disk, and directory, you can identify a file uniquely by specifying only its file name and file type in the format:

```
filename.type
```

The file name can have from one to nine characters chosen from the letters A through Z and the numbers 0 through 9. When you create files, you can give them any names that are meaningful to you.

The file type can be from zero to three characters, and must be preceded by a period; again, you can choose any of the letters A through Z or the numbers 0 through 9 for the file type. However, the file type usually describes more specifically the kind of data in the file, and the system recognizes several default file types used for special purposes. For example, each high-level language has a default file type for source programs. (See Section 4.1 for a table of these file types.)

Among the other default file types are:

File Type	**Use**
DAT	Data file
EDT	Start-up command file for EDT editor
EXE	Executable program image file
JOU	Journal file used by the EDT editor
LIS	Output listing file
MAI	Mail message file
OBJ	Object module file output from a compiler or assembler

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

You rarely need to specify the version number with a file specification. The system assumes default values for version numbers, as it does with devices, directories, and file types. Version number defaults are determined as follows:

1. For an **input file,** the system uses the highest existing version number of the file.
2. For an **output file,** the system adds 1 to the highest existing version number.

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

3.1.5 Wild Card Characters

A **wild card character** is a symbol that you can use with many DCL commands to apply the command to several files at once, rather than specifying each file individually. Two wild card characters, the asterisk (*) and the percent sign (%) can be used in specifications of a directory, file name, and file type. The asterisk can also be used to specify version numbers.

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

```
$ print testfile.dat;*
```

If there were no wild card character in the above example, the PRINT command by default would apply only to the most recent version of the file TESTFILE.DAT. The following command prints all versions of all files in the current directory with the file type of DAT:

```
$ print *.dat;*
```

To print all versions of all files in the directory with the file name of TEST, type:

```
$ print test.*;*
```

The percent sign allows you to specify all files containing any single character in the position that the percent sign occupies in the file specification. For example, to print the latest version of several files with a file type of TXT and a file name that begins with CHAP but ends in a series of different numbers, as in CHAP1.TXT, CHAP2.TXT, and CHAP3.TXT, type:

```
$ print chap%.txt
```

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

3.2 Creating Files

Chapter 2 explains how to create a file by using the EDIT command to invoke an editor. You can also use the CREATE command to make a new file. Specify the file name as a parameter. You can insert text immediately, terminating the insertion with CTRL/Z.

```
$ create myfile.dat
This is the only line. CTRL/Z
```

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

3.3 Deleting Files

Quite often, as you develop and revise programs you end up with many versions. Since these files take up space on your disk, you may want to delete versions of files that you no longer need.

The DELETE command deletes specific files. When you use the DELETE command, you must specify a file name, file type, and version number (having to specify a version number provides some protection against accidental deletion). However, any of these file components can be specified as a wild card character. You can also enter more than one file specification on a command line separating the file specifications with commas. Some examples of the DELETE command are:

Command	**Result**
`$ delete average.obj;1`	Deletes the file named AVERAGE.OBJ;1
`$ delete *.lis;*`	Deletes all files with file types of LIS (thus, this command deletes all versions of all program listings)
`$ delete a.dat;1,a.dat;2`	Deletes the first two versions of the same data file

3.4 Purging Files

You may want to clean up your directory by getting rid of all early versions of particular files. If you have many versions of a file, naming them all in the DELETE command would be tedious.

The PURGE command allows you to delete all but the most recent version of a file; therefore, no version number is required by the PURGE command. For example:

```
$ purge average.for RET
```

This command deletes all files named AVERAGE.FOR except the file with the highest version number.

The /KEEP qualifier of the PURGE command allows you to specify that you want to keep more than one version of a file. For example:

```
$ purge/keep=2 test.dat RET
```

This command deletes all but the two most recent versions of the file TEST.DAT.

3.5 Displaying Files at Your Terminal

The TYPE command displays a file at your terminal. For example:

```
$ type test.dat RET

This is the first line of
a file created with
the EDT editor.
```

While a file is being displayed, you can interrupt the output by using any of the following CTRL key combinations:

CTRL/S suspends the terminal display of the file and the processing of the command. To resume display, press CTRL/Q. The interrupted command displays lines beginning at the point at which processing was interrupted.

CTRL/C or CTRL/Y interrupts command processing. The system then prompts you to enter another command.

3.6 Printing Files

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

A printed file is preceded by a **header page** describing the file so you can identify your own listing. For example, if you issue the following command, the header page will show your user name and the file name, type, and version number of the file.

```
$ print db2:[jones]average.lis RET
  Job 210 entered on queue SYS$PRINT
```

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

The PRINT command also has qualifiers that allow you to control the number of copies of the file to print, the type of forms to print the file on, and so on. More information on these qualifiers can be found in the *VAX/VMS Command Language User's Guide.*

3.7 Listing Files in a Directory

The DIRECTORY command lists the names of files in a particular directory. If you type the DIRECTORY command with no parameters or qualifiers, the command displays the files listed in your default directory on the terminal. For example:

```
$ directory RET

DIRECTORY DBA2:[MALCOLM]❶

AVERAGE.EXE;2   AVERAGE.EXE;1   AVERAGE.FOR;2   AVERAGE.FOR;1
AVERAGE.OBJ;2   AVERAGE.OBJ;1❷

Total of 6 files.❸
```

The following notes are keyed to this sample output of the DIRECTORY command:

❶ The disk and directory name.

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

❸ The total number of files in the directory.

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

```
$ directory average.for(RET)

DIRECTORY DBA1:[CRAMER]

AVERAGE.FOR;2    AVERAGE.FOR;1

Total of 2 files.
```

3.8 Creating Subdirectories

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

The CREATE/DIRECTORY command creates a subdirectory. For example:

```
$ create/directory [malcolm.testfiles](RET)
```

This command creates the subdirectory file TESTFILES.DIR in the directory [MALCOLM]. You can specify the subdirectory name, [MALCOLM.TESTFILES], in commands or programs.

3.9 Changing Your Default Directory

To establish another directory or subdirectory as your default directory, use the SET DEFAULT command. For instance, you could create a new file in the subdirectory [MALCOLM.TESTFILES] by changing your default directory and then creating the file with the EDIT command:

```
$ set default [malcolm.testfiles](RET)
$ edit newfile.txt(RET)
Input file does not exist
[EOB]
*
```

The new file will be cataloged in the subdirectory [MALCOLM.TESTFILES]. (You could also do this by specifying the subdirectory as part of the file specification when you use the EDIT command.)

You can also use the SET DEFAULT command to change your default disk. For example:

```
$ set default dbb2:(RET)
```

After you issue this command, the system uses the disk DBB2 as the default disk for all files that you access or create.

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

3.10 Copying Files

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

When you issue the COPY command, you specify first the name(s) of the input file(s) you want to copy, then the name of the output file. For example, the following COPY command copies the contents of the file PAYROLL.TST to a file named PAYROLL.OLD.

```
$ copy payroll.tst payroll.old RET
```

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

You could copy a file from the directory [MALCOLM] to the subdirectory [MALCOLM.TESTFILES], and give it the new name, OLDFILE.DAT.

```
$ copy newfile.dat [malcolm.testfiles]oldfile.dat RET
```

When you copy files from devices other than your default disk, you must specify the device name in the COPY command. For example, the following COPY command copies a file from your default directory onto an RK06 disk.

```
$ copy payroll.tst dma1: RET
```

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

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

- Mounting the volume, with the MOUNT command.
- Ensuring that the volume has a directory for cataloging the file. If no directory exists, use the CREATE command to create one.

Note that the VAX/VMS operating system protects access to volumes that individuals maintain for private purposes, as well as access to system volumes. For details on the commands and procedures necessary to prepare and use disks and tapes, see the *VAX/VMS Command Language User's Guide*.

3.11 Renaming Files

The RENAME command changes the identification of one or more files. For example, the following command changes the name of the most recent version of the file PAYROLL.DAT to TEST.OLD.

```
$ rename payroll.dat test.old(RET)
```

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

```
$ rename [malcolm]test.old [malcolm.testfiles]
```

You can use wild card characters if you want to change a number of files that have either a common file name or file type. For example:

```
$ rename payroll.*;* [malcolm.testfiles]*.*;*(RET)
```

This RENAME command changes the directory name for all versions of all files that have file names of PAYROLL. The files are now cataloged in the subdirectory [MALCOLM.TESTFILES].

3.12 For More Information

The *VAX/VMS Command Language User's Guide* describes in more detail the commands presented here. Part II of that manual lists all the commands in alphabetical order and includes descriptions of parameters and qualifiers, as well as giving additional examples of each command.

Remember, too, that while you are using the terminal, you can use the HELP command to receive on-the-spot assistance if you cannot remember a parameter or qualifier. Or, you can let the system prompt you for command parameters, if you cannot remember the order in which you have to enter them.

See the DECnet–VAX documentation for further explanation of networks and node specifications.

Chapter 4
Program Development

Four steps are required to develop a program:

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

4.1 Creating the Program

In order to run your program, you must first create a file of the program source statements. The default file type corresponds to the language in which the program is written. For instance, if your program is written in VAX–11 BASIC, its file type default is BAS. The following are default file types for source program files written in VAX–11 languages:

File Type	**Contents**
BAS	Input source file for the VAX–11 BASIC compiler
B32	Input source file for the VAX–11 BLISS–32 compiler
C	Input source file for the VAX–11 C compiler
COB	Input source file for the VAX–11 COBOL compiler
COB	Input source file for the VAX–11 COBOL–74 compiler
COR	Input source file for the VAX–11 CORAL–66 compiler
FOR	Input source file for the VAX–11 FORTRAN compiler
MAR	Input source file for the VAX–11 MACRO assembler
PAS	Input source file for the VAX–11 PASCAL compiler
PLI	Input source file for the VAX–11 PL/I compiler

4.2 Compiling or Assembling the Program

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

Both **compilers** and **assemblers** are programs that translate source programs into binary **machine code** that can be interpreted by the computer. An **assembly language** is usually designed for a specific computer, and it generally assembles line for line into machine code. Most high-level languages, on the other hand, are designed to be universal, and usually compile one line of source code into several lines of machine code. If your source program is written in assembly language (in this case, VAX–11 MACRO), you invoke the VAX–11 MACRO assembler to translate it. If it is written in a high-level language (such as BASIC, C, COBOL, FORTRAN, PASCAL, or PL/I), you invoke the appropriate VAX–11 language compilers to compile the program.

There is a DCL command to invoke each language processor:

Command	**Invokes**
BASIC	VAX–11 BASIC compiler
BLISS	VAX–11 BLISS-32 compiler
CC	VAX–11 C compiler
COBOL	VAX–11 COBOL compiler
COBOL/C74	VAX–11 COBOL–74 compiler
CORAL	VAX–11 CORAL–66 compiler
FORTRAN	VAX–11 FORTRAN compiler
MACRO	VAX–11 MACRO assembler
PASCAL	VAX–11 PASCAL compiler
PLI	VAX–11 PL/I compiler

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

```
$ basic myfile
```

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

4.3 Linking the Object Module

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

The LINK command invokes the VAX-11 Linker. The linker searches system libraries to resolve references to routines or symbols that are not defined within the object modules it is linking. You can request the linker to include more than one object module as input, or specify your own libraries of object modules for it to search. The format of the LINK command is:

```
$ link myfile
```

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

The linker creates an image, which is a file containing your program in an executable format.

4.4 Executing the Program

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

```
$ run myfile
```

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

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

The remaining sections of this chapter illustrate the steps of program development with two sample programs: a MACRO example for assembly language users and a FORTRAN example for high-level language users. These sections describe the input and output files used in each step and the naming conventions for the files. They also present optional command qualifiers you can use to create additional output files, including program listings. If you have access to a terminal, you can create the programs and issue the commands that are described.

At the end of the chapter, Section 4.7 lists additional documentation with further information about the tools VAX/VMS provides, for program development. For information about a particular VAX-11 language, see the document set for that language.

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

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

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

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

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

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

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

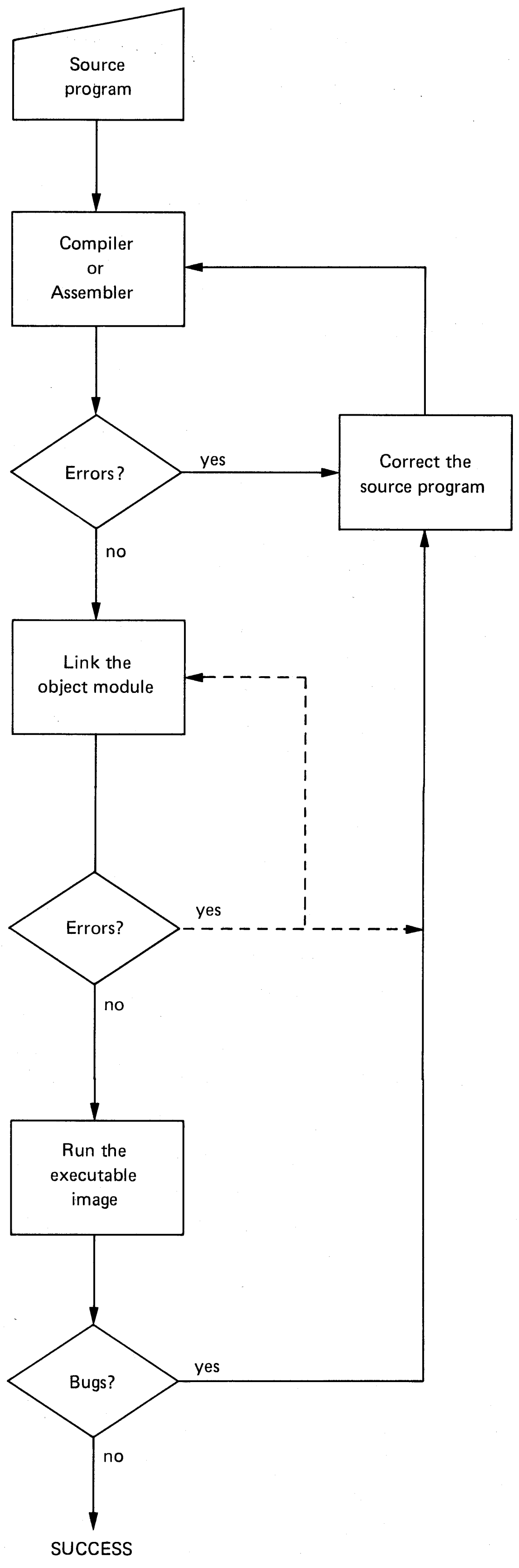


Figure 4–1: Steps in Program Development

4.5 A FORTRAN Program

The steps required to prepare a VAX-11 FORTRAN[1] program to run on VAX/VMS are illustrated in Figure 4–2. Figure 4–2 also notes the default file types used by the FORTRAN, LINK, and RUN commands. For any of these commands, you can specify an explicit file type to override the defaults when you name an input or output file.

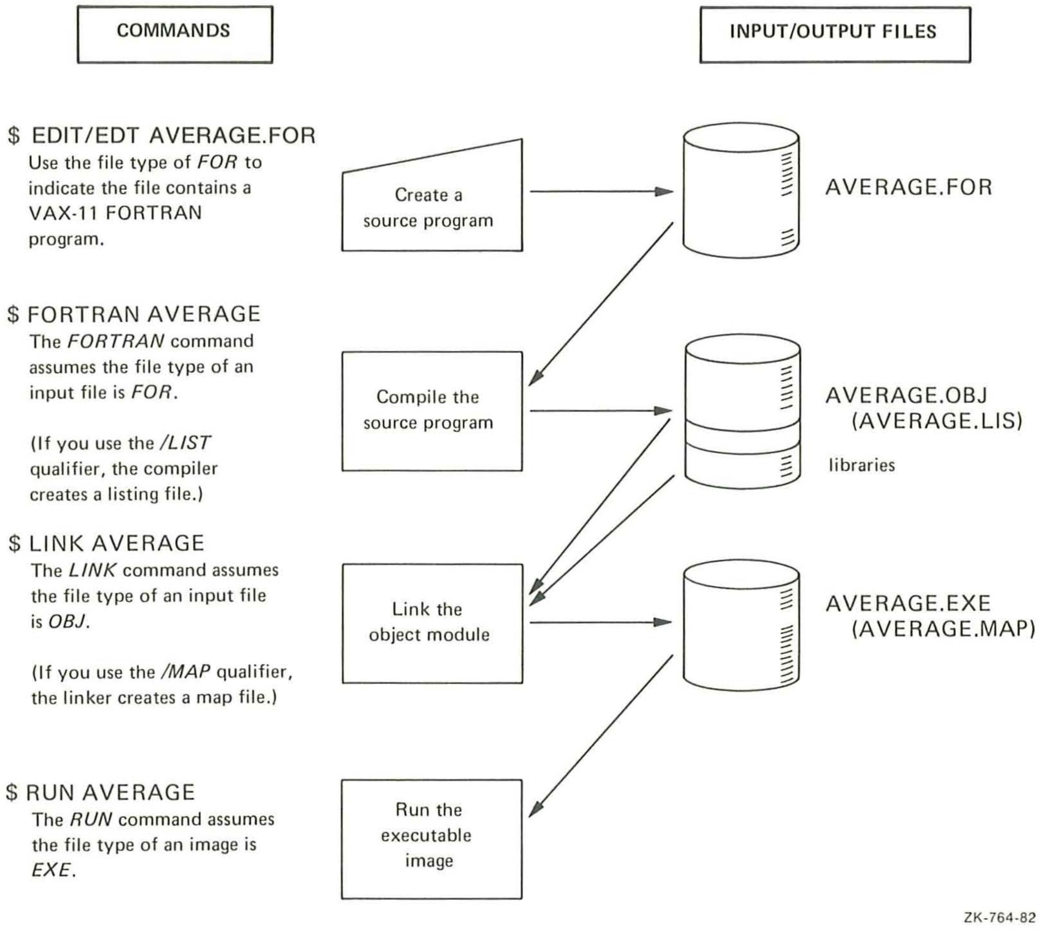

COMMANDS
INPUT/OUTPUT FILES
$ EDIT/EDT AVERAGE.FOR
Use the file type of *FOR* to indicate the file contains a VAX-11 FORTRAN program.
Create a source program
AVERAGE.FOR
$ FORTRAN AVERAGE
The *FORTRAN* command assumes the file type of an input file is *FOR*.
(If you use the */LIST* qualifier, the compiler creates a listing file.)
Compile the source program
AVERAGE.OBJ
(AVERAGE.LIS)
libraries
$ LINK AVERAGE
The *LINK* command assumes the file type of an input file is *OBJ*.
(If you use the */MAP* qualifier, the linker creates a map file.)
Link the object module
AVERAGE.EXE
(AVERAGE.MAP)
$ RUN AVERAGE
The *RUN* command assumes the file type of an image is *EXE*.
Run the executable image
ZK-764-82

Figure 4–2: Commands for FORTRAN Program Development

4.5.1 Creating the Source Program

Use the editor (described in Chapter 2) to create a source program interactively. For example, to create the FORTRAN program called AVERAGE, issue the EDIT command as follows:

```
$ edit average.for RET
Input file does not exist
[EOB]
*
```

1. The VAX-11 FORTRAN compiler is referred to simply as FORTRAN throughout this manual.

The asterisk prompt indicates that EDT is ready to accept a line-editing command.

The program AVERAGE is shown below. When you type the input statements, you can use the TAB key to align the statement and comments columns. Tabs are set at every eight character positions. The EDT line editor assigns line numbers to help you locate text; the line numbers are not part of the file, however. (To display the line numbers, give the TYPE WHOLE command after the asterisk prompt.)

```
1               PROGRAM AVERAGE
2
3       C       COMPUTES THE AVERAGE OF NUMBERS ENTERED AT TERMINAL
4       C       TO TERMINATE THE PROGRAM, ENTER 9999
5
6               TOTAL = 0                       ! INITIALIZE ACCUMULATOR
7               N = 0                           ! INITIALIZE COUNTER
8
9       5       N = N + 1
10              WRITE (6,10)                    ! PROMPT TO ENTER NUMBER
11
12      10      FORMAT (' ENTER NUMBER, END WITH 9999')
13              READ (5,20) K                   ! READ NUMBER FROM TERMINAL
14
15      20      FORMAT I10
16              IF (K .EQ. 9999) GOTO 40        ! 9999 MEANS NO MORE INPUT
17              TOTAL = TOTAL + K               ! COMPUTE TOTAL WITH NUMBER
18              GOTO 5
19
20      C       NOW, COMPUTE AVERAGE BY DIVIDING TOTAL BY THE NUMBER, OF
21      C       TIMES THROUGH THE LOOP
22
23      40      AVERAG = TOTAL/N
24              WRITE (6,50) AVERAG             ! DISPLAY THE RESULT
25
26      50      FORMAT ('     AVERAGE IS ',F10.2)
27
28              STOP
29              END
```

The program AVERAGE reads and writes lines to the current input and output devices; it prompts the user to enter numbers and then computes the average of the numbers entered. This program purposely has a syntax error and a bug, so you can get an idea of how to use VAX/VMS to correct programming errors.

4.5.2 The FORTRAN Command

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

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

To compile the source program AVERAGE, issue the command:

```
$ fortran average RET
```

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

If the compilation is successful — that is, if the compiler did not detect any errors — the system displays a prompt for the next command:

```
$
```

If there are any errors, the FORTRAN compiler displays information on the terminal. If you entered the source program AVERAGE exactly as it appeared above, then you received the message:

```
%FORT-F-ERROR 33, Missing operator or delimiter symbol
    [FORMAT I] in module AVERAGE at line 8
%FORT-F-ENDNOOBJ, DB2:[MALCOLM]AVERAGE.FOR;1,
            completed with 1 diagnostic-
            object deleted
```

This message indicates that the FORMAT statement was incorrectly coded; you must put parentheses around the format specification.

To correct the error, edit the source file. First, invoke EDT:

```
$ edit average.for RET
    1   Program AVERAGE
*
```

Now, use editor commands to correct the error, as shown below:

```
*replace 15
1 line deleted
        20       FORMAT (I10) CTRL/Z
*
```

The REPLACE command deletes the line specified and inserts the line or lines you type. If you had more than one error in your source file, correct these errors, too.

When you are satisfied with the changes, use the EXIT command to write the updated file onto disk:

```
*exit RET
DBA2:[MALCOLM]AVERAGE.FOR;2   29 lines
$
```

Notice that EDT has created a new version of the file AVERAGE.FOR.

Now you can recompile the program:

```
$ fortran average RET
```

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

When you compile a source program, use the /LIST qualifier on the FORTRAN command to request the compiler to create a program listing. For example:

```
$ fortran/list average(RET)
```

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

```
$ print average(RET)
```

The PRINT command uses the default file type of LIS.

4.5.3 Linking the Object Module

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

```
$ link average(RET)
```

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

4.5.4 Running the Program

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

```
$ run average(RET)
```

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

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

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

To correct the error, edit the source file again:

```
$ edit average.for(RET)
     1        Program Average
*Substitute\TOTAL/N\Total/(N-1)\ 23(RET)
    23      40
AVERAG = TOTAL/(N-1)
1 Substitution
*Exit(RET)
DBA2:[MALCOLM]AVERAGE.FOR;3 29 lines
$
```

The SUBSTITUTE command deletes the first string, TOTAL/N, and inserts the second, TOTAL/(N-1), in the line specified. The EXIT command writes a new version of the file onto disk.

Now, repeat the compile, link, and run steps:

```
$ fortran average(RET)
$ link average(RET)
$ run average(RET)
ENTER NUMBER, END WITH 9999
33(RET)
ENTER NUMBER, END WITH 9999
66(RET)
ENTER NUMBER; END WITH 9999
99(RET)
ENTER NUMBER, END WITH 9999
9999(RET)
AVERAGE IS   66.00
FORTRAN STOP
$
```

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

4.5.5 Debugging the Program

The VAX/VMS operating system has a **debugger**, a program that permits you to debug your programs interactively. When you want to use the debugger, you have to compile the source program with the /DEBUG and /NOOPTIMIZE qualifiers, as follows:

```
$ fortran/debug/nooptimize average(RET)
```

These qualifiers make the later use of the debugger program possible with this FORTRAN program. When the compilation completes, use the /DEBUG qualifier when you link the object module:

```
$ link/debug average(RET)
```

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

For information on how to use the debugger, see the *VAX-11 FORTRAN User's Guide*.

4.6 A MACRO Program

The steps required to prepare a VAX-11 MACRO[1] program to run under VAX/VMS are illustrated in Figure 4-3. Figure 4-3 also notes the default file types used by the MACRO, LINK, and RUN commands. For any of these commands, you can specify an explicit file type to override the default when you name an input or output file.

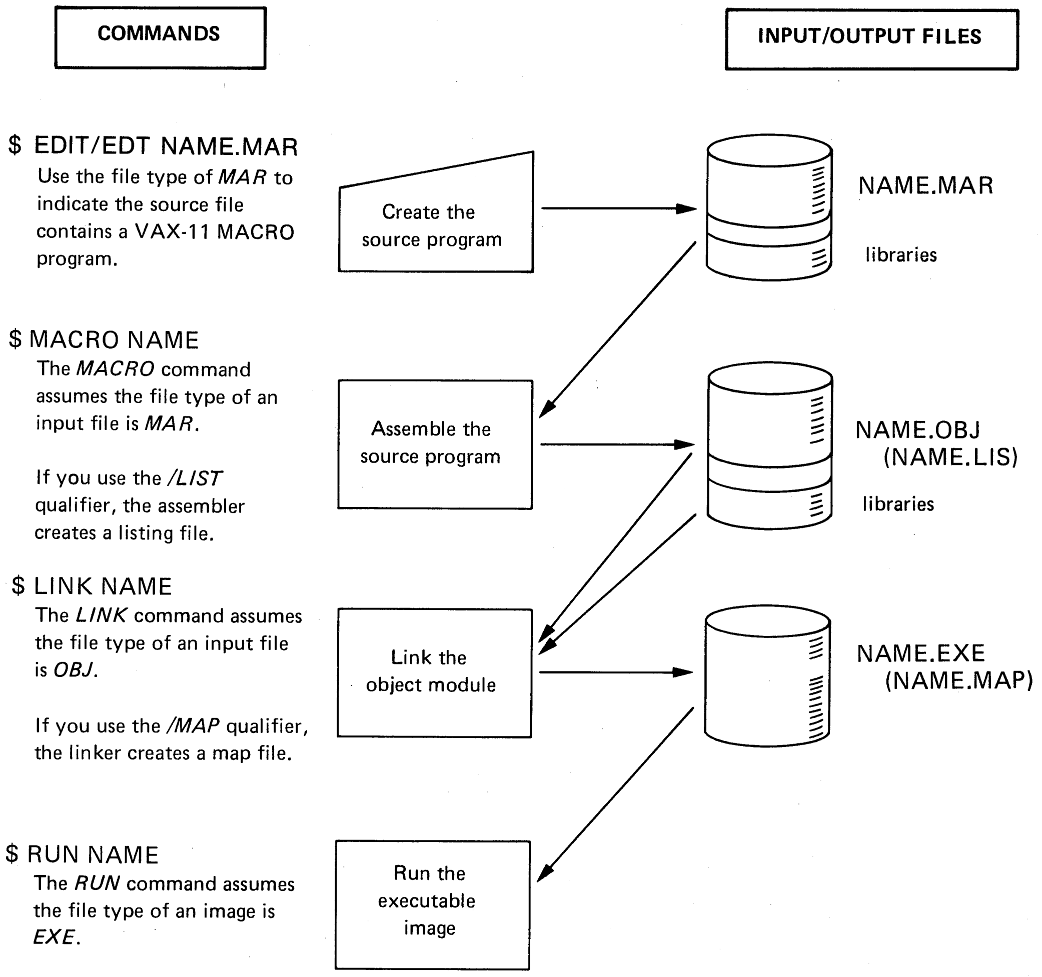

COMMANDS
INPUT/OUTPUT FILES
$ EDIT/EDT NAME.MAR
Use the file type of *MAR* to indicate the source file contains a VAX-11 MACRO program.
Create the source program
NAME.MAR
libraries
$ MACRO NAME
The *MACRO* command assumes the file type of an input file is *MAR*.
If you use the */LIST* qualifier, the assembler creates a listing file.
Assemble the source program
NAME.OBJ
(NAME.LIS)
libraries
$ LINK NAME
The *LINK* command assumes the file type of an input file is *OBJ*.
If you use the */MAP* qualifier, the linker creates a map file.
Link the object module
NAME.EXE
(NAME.MAP)
$ RUN NAME
The *RUN* command assumes the file type of an image is *EXE*.
Run the executable image
ZK-765-82

Figure 4-3: Commands for MACRO Program Development

1. The VAX-11 MACRO assembler is referred to simply as MACRO throughout this manual.

4.6.1 Creating the Source Program

Use the editor (described in Chapter 2) to create a source program interactively. For example, to create the MACRO program called NAME, issue the EDIT command as follows:

```
$edit name.mar RET
Input file does not exist
[EOB]
*Insert RET
```

EDT is now ready to accept input lines.

The program NAME is shown below. When you type the input statements, you can use the TAB key to align the operand and comments columns. Tabs are set at every eight character positions. The EDT line editor assigns line numbers to help you locate text; the line numbers are not part of the file, however. (To see the line numbers, give the TYPE WHOLE command after the asterisk prompt.)

The program uses VAX–11 RMS to read and write lines to the current terminal; it issues a prompting message asking for the user's name and redisplays whatever is entered in response. This program purposely has a syntax error and a bug, so you get an idea of how to use VAX/VMS to correct programming errors.

```
1               .TITLE NAME
2               .IDENT /01/
3               .PSECT  RWDATA,WRT,NOEXE
4
5       ; DEFINE CONTROL BLOCKS FOR TERMINAL INPUT AND OUTPUT
6
7       TRMFAB: $FAB    FNM=SYS$INPUT,RAT=CR,FAC=<GET,PUT> ;FAB FOR TERMINAL
8
9       TRMRAB: $RAB    FAB=TRMFAB,UBF=BUFFER,USZ=BUFSIZ, -
10                      ROP=PMT, PBF=PMSG1, PSZ=P1SIZ
11
12      BUFFER: .BLKB   132                     ; INPUT READ BUFFER
13      BUFSIZ= .-BUFFER                        ; BUFFER LENGTH
14
15      PMSG1:  .ASCII  /ENTER YOUR NAME:       ; PROMPT MESSAGE
16      P1SIZ=  .-PMSG1                         ; MESSAGE SIZE
17
18      OUTMSG: .ASCII  /HELLO, YOUR NAME IS/   ; OUTPUT MESSAGE
19      OUTBUF: .BLKB   30                      ; MOVE NAME HERE
20      OUTLEN: .LONG   OUTBUF-OUTMSG
21      MSGSIZ: .BLKL   1                       ; ADD LENGTHS HERE
22
23              .PSECT  NAME,EXE,NOWRT
24              .ENTRY  BEGIN,0                 ; ENTRY MASK
25
26              $OPEN   FAB=TRMFAB              ; OPEN TERMINAL FILE
27              BLBC    R0,ERROR                ; EXIT IF ERROR
28              $CONNECT RAB=TRMRAB             ; ESTABLISH RAB
29              BLBC    R0,ERROR                ; EXIT IF ERROR
30
31              $GET    RAB=TRMRAB              ; ISSUE PROMPT
32              BLBC    R0,ERROR                ; EXIT IF ERROR
33
34      ; MOVE NAME ENTERED INTO OUTPUT MESSAGE, AND FIX UP LENGTH
35
36              MOVC3   TRMRAB+RAB$W_RSZ,BUFFER,OUTBUF
37              MOVZWL  TRMRAB+RAB$W_RSZ,MSGSIZ
38              ADDL    MSGSIZ,OUTLEN
39
```

```
40    ; AFTER CONSTRUCTING OUTPUT MESSAGE, OUTPUT IT
41
42            MOVAL     OUTMSG,TRMRAB+RAB$L_RBF ; UPDATE RAB: ADDRESS
43            MOVW      MSGSIZ,TRMRAB+RAB$W_RSZ ; UPDATE RAB: SIZE
44            $PUT      RAB=TRMRAB
45            BLBC      R0,ERROR                ; EXIT IF ERROR
46
47    ; ALL DONE, CLOSE THE FILE
48
49            $CLOSE    FAB=TRMFAB
50
51    ERROR:
52
53            RET
54            .END      BEGIN
```

4.6.2 The MACRO Command

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

1. Produces an object module that has the same file name as the source file and a file type of OBJ
2. Uses MACRO assembler defaults when it creates output files (qualifiers on the command line can override these defaults)
3. Searches the system macro library for definitions for system macros, such as the VAX-11 RMS macros $FAB and $RAB used in the sample program NAME.MAR

To assemble the source program NAME, issue the command:

```
$ macro/list name RET
```

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

If the assembly is successful — that is, if the assembler did not detect any errors — the system displays a prompt for the next command:

```
$
```

If errors occur, a message is displayed at the terminal. If you entered the source program NAME exactly as it appeared above, then you received an error message:

```
%MACRO-E-UNTERMARG, Unterminated argument
There were 1 error, 0 warnings, and 0 information messages on
lines:
15(1)
```

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

To correct the error, edit the source file. First, invoke EDT:

```
$ edit name.mar RET
     1       .TITLE NAME
*
```

Now, use editor commands to locate the line and correct the error, as shown below:

```
*REPLACE 15
1 line deleted
     PMSG1:.ASCII/ENTER YOUR NAME:/;PROMPT MESSAGE CTRL Z
*
```

The REPLACE command deletes the line (15) that was in error, and inserts the line or lines you type in its place. If you had more than one error in your source file, correct these errors, too.

When you are satisfied with the changes, use the exit command to write the updated file onto disk:

```
*exit RET
DBA2:[MALCOLM]NAME.MAR;2 54 lines
$
```

Notice that EDT has created a new version of the file NAME.MAR.

Now, you can reassemble the program:

```
$ macro/list name RET
```

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

4.6.3 Linking the Object Module

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

```
$ link name RET
```

This LINK command creates a file named NAME.EXE, which is an executable program image. The linker automatically includes in the executable image any library procedures required by the VAX–11 RMS routines used.

4.6.4 Running the Program

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

```
$ run name(RET)
```

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

```
ENTER YOUR NAME:YORICK(RET)
HELLO,
$
```

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

To correct the error, edit the source file again:

```
$ edit name.mar(RET)
        1         .Title Name
*Replace 43
1 line deleted
              MOVW OUTLEN, TRMRAB+RAB$W_RSZ; UPDATE
RAB:SIZE(CTRL/Z)
   44         $PUT               RAB=TRMRAB
*EXIT
DBA2:[MALCOLM]NAME.MAR;3 54 lines
$
```

Now, repeat the assembling, linking, and running:

```
$ macro name(RET)
$ link name(RET)
$ run name(RET)
ENTER YOUR NAME:YORICK(RET)
HELLO, YOUR NAME IS YORICK
$
```

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

4.6.5 Debugging the Program

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

```
$ macro/enable=debug name(RET)
```

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

```
$ link/debug name(RET)
```

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

For information on how to use the debugger, see the *VAX-11 Symbolic Debugger Reference Manual.*

4.7 For More Information

The two program examples presented in this chapter show only the simplest cases, using defaults for getting a program to run. VAX/VMS provides many capabilities beyond those presented in these examples. Some of the VAX/VMS manuals you may find useful are described below.

- The *VAX/VMS Command Language User's Guide* contains reference information for all the commands that have been used in the examples in this chapter.
- The *VAX-11 Linker Reference Manual* describes how to use the linker and describes the options available to you when you link a program.
- The *VAX-11 MACRO Language Reference Manual* describes the features and syntax of the VAX-11 MACRO language.
- The *VAX-11 MACRO User's Guide* provides details on how to use the VAX-11 MACRO assembler.
- The *VAX-11 Symbolic Debugger Reference Manual* describes the features of the VAX-11 Symbolic Debugger.
- The *VAX/VMS System Services Reference Manual* presents additional programming capabilities through the VAX/VMS system services.
- The *Introduction to VAX-11 Record Management Services* describes the file formats used in VAX/VMS, and the *VAX-11 Record Management Services Reference Manual* describes the macros that can be used to create, read, and update files. The *VAX-11 Record Management Services Utilities Reference Manual* describes the VAX-11 RMS utilities that can be used to design, create, load, and analyze files. The *VAX-11 Record Management Services Tuning Guide* describes the concepts behind designing efficient files and provides tutorial explanations of the utilities.
- The VAX-11 Common Run-Time Procedure Library contains procedures that do such tasks as: perform common mathematical functions, manipulate character string data for input and output routines, and convert data

from one type to another (e.g., changing a numeric ASCII string to a binary value or a floating-point number to scientific notation). The *VAX-11 Run-Time Library Reference Manual* contains descriptions of all general purpose procedures in the Run-Time Library.

- The *VAX-11 Guide to Creating Modular Library Procedures* describes methods for designing and coding procedures for insertion in an object module library or a shareable image.

Each of the VAX-11 languages is documented in a separate set of manuals that generally includes a language reference manual and a user's guide for that particular language. For further information about a high-level language, see the document set for that language.

Separate documentation exists for the following VAX-11 languages:

VAX-11 BASIC
VAX-11 BLISS
VAX-11 C
VAX-11 COBOL
VAX-11 CORAL
VAX-11 FORTRAN
VAX-11 PASCAL
VAX-11 PL/I

See the *VAX-11 Information Directory and Index* for a list of the individual manuals in the document sets for the VAX-11 languages.

Chapter 5
Logical Names: Files for Program Input/Output

When you design programs to read and write data, you can code the programs to read or write different files each time you run them. This is called device and file independence.

In the VAX/VMS operating system, device independence is accomplished through the use of **logical names**. When you code a program, you refer to an input or output file according to the syntax requirements of the language you are using. After the program is compiled and linked, but before you run it, you can make the connection between the logical names you used in the program and the actual files or devices you want to use when you run the program.

The ASSIGN command makes this connection: it establishes the correspondence between a logical name (that is, the name you use in the program) and an **equivalence name** (that is, the actual file or device to use).

Figure 5-1 shows how logical names are used. The program FICA contains OPEN, READ, and WRITE statements in a general form; the program reads from a file referred to by the logical name INFILE, and writes to a file referred to by the logical name OUTFILE.

For different runs of the program, the ASSIGN command establishes different equivalence names for INFILE and OUTFILE. In the first example, the program reads the file JANUARY.DAT from the device DBA1 and writes to the file JANUARY.OUT on the same disk device. In the second example, it reads the file FEBRUARY.DAT from the device DBA2 and writes the file FEBRUARY.OUT to that device.

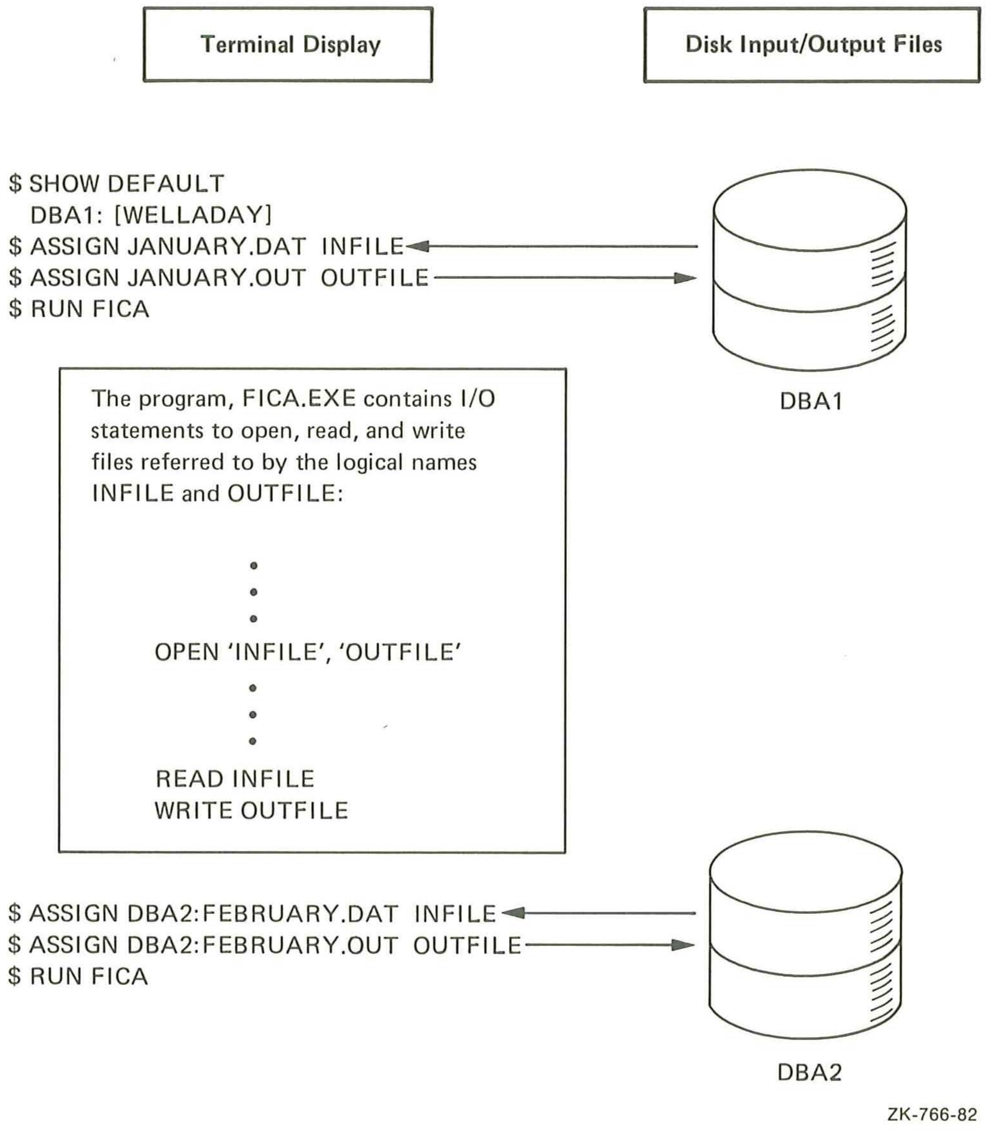

Terminal Display
Disk Input/Output Files
$ SHOW DEFAULT
DBA1: [WELLADAY]
$ ASSIGN JANUARY.DAT INFILE
$ ASSIGN JANUARY.OUT OUTFILE
$ RUN FICA
DBA1
The program, FICA.EXE contains I/O statements to open, read, and write files referred to by the logical names INFILE and OUTFILE:
OPEN 'INFILE', 'OUTFILE'
READ INFILE
WRITE OUTFILE
$ ASSIGN DBA2:FEBRUARY.DAT INFILE
$ ASSIGN DBA2:FEBRUARY.OUT OUTFILE
$ RUN FICA
DBA2
ZK-766-82

Figure 5-1: Using Logical Names

5.1 Logical Names in Commands

The use of logical names is not restricted to application programs. Commands that read or write files, such as COPY and TYPE, also accept logical names for a file specification. For example:

```
$ assign [chuck]personnel.rec myfile(RET)
$ type myfile(RET)
```

The ASSIGN command equates the logical name MYFILE to the file PERSONNEL.REC listed in the directory CHUCK. The TYPE command requests the system to display this file on the terminal.

A logical name can also define only the first portion of a file specification. For example:

```
$ assign dba2:[malcolm.testfiles] test(RET)
$ run test:memo(RET)
$ print test:memo.lis(RET)
```

The ASSIGN command equates the logical name TEST to the disk device and directory DBA2:[MALCOLM.TESTFILES]. Subsequently, the RUN command executes the program image MEMO.EXE cataloged in this subdirectory and the PRINT command prints another file. The system always examines file specifications to see if the portion of the file specification that precedes the colon (:) is a logical name; if it is (as in this example), the system substitutes the equivalence name.

5.2 System Default Logical Names

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

Logical Name	Use
SYS$INPUT	The default input stream from which the system reads commands and your programs read data Default interactive assignment: your terminal Default batch assignment: the command procedure or batch stream
SYS$OUTPUT	The default output stream to which the system writes responses to commands and your programs write data Default interactive assignment: your terminal Default batch assignment: batch job log file
SYS$ERROR	The default device to which the system writes all error and informational messages Default interactive assignment: your terminal Default batch assignment: batch job log file
SYS$DISK	Your default disk device Default assignment: set in User Authorization File

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

You can also assign a logical name to another logical name. For example, to test the program FICA shown in Figure 5-1, you could assign the logical name OUTFILE to the logical name SYS$OUTPUT, as follows:

```
$ assign sys$output outfile(RET)
```

Then, when FICA writes to the logical device OUTFILE, the output is directed to your terminal.

The remaining sections of this chapter contain additional language-specific examples of how logical names are used for program input and output.

5.3 FORTRAN Input/Output

The VAX/VMS system supplies several default logical names for use with FORTRAN programs. These logical names provide default devices for the input/output statements indicated.

Logical Name	Use
`FOR$READ`	Default input device read by READ statements that do not specify a logical unit number Default assignment: `SYS$INPUT`
`FOR$PRINT`	Default output device written by PRINT statements Default assignment: `SYS$OUTPUT`
`FOR$ACCEPT`	Default input device read by ACCEPT statements Default assignment: `SYS$INPUT`
`FOR$TYPE`	Default output device written by TYPE statements Default assignment: `SYS$OUTPUT`

You do not need to take any special action to direct input or output when you use these statements: the system translates the logical name SYS$INPUT or SYS$OUTPUT and locates the current equivalence for that logical name.

However, when you want to have a program read or write data from or to a device or file other than a default, or when you specify a logical unit number on an input/output statement, you can take special action: you can assign an equivalence name for the logical name.

5.3.1 Changing Default Logical Names

The ASSIGN command changes the equivalence for a logical name. For example, suppose you have a FORTRAN program named STAT that uses both TYPE and PRINT statements, as follows:

```
TYPE 30,A,B,C
   .
   .
   .
PRINT 100,D,E,F
```

To execute this program so that output from the PRINT statement goes to a disk file rather than to the terminal, enter an ASSIGN command before running the program:

```
$ assign statdata.out for$print(RET)
$ run stat(RET)
```

When STAT finishes execution, all output lines written by the PRINT statement are contained in the file STATDATA.OUT. The system uses your default disk device and directory to catalog the file.

5.3.2 Logical Names for Unit Numbers

The concept of logical names and default logical name assignments applies to specifying input/output files by logical unit numbers. Each logical unit number has an associated default logical name, and each logical name has a default equivalence name. These logical names and equivalence names are as follows:

Unit	Logical Name	Default Equivalence
1	FOR001	FOR001.DAT
2	FOR002	FOR002.DAT
3	FOR003	FOR003.DAT
4	FOR004	FOR004.DAT
5	FOR005	SYS$INPUT
6	FOR006	SYS$OUTPUT
.	.	.
.	.	.
.	.	.
n	FOR0nn	FOR0nn.DAT

For example, a program named BALANCE may contain the lines:

```
   READ (23,90)A,B,C

90 FORMAT (3F10.4)
```

In the above example, 23 is a logical unit number. Before executing this program, you can assign an equivalence name to the logical name FOR023 so that the READ statements read from a specific file, as follows:

```
$ assign libra.tst for023(RET)
$ run balance(RET)
```

If you do not assign FOR023, the system uses the default equivalence name FOR023.DAT. In either case, the system uses your current default disk and directory.

5.3.3 Logical Names in OPEN Statements

If you code a program that uses an OPEN statement to define an input or output file, you can specify the NAME parameter to give the file specification for the file. In this case, the system does not use the default equivalence name to locate a file for input or output, but uses the name specified.

A typical OPEN statement may look like the following:

```
OPEN (UNIT=19,NAME='WEATHER.STS')
```

When the program uses a READ statement to read the logical unit 19, it reads the file WEATHER.STS. Note that the system supplies your current disk and directory defaults to locate the file.

You can also specify a logical name with the NAME parameter. For example:

```
OPEN (UNIT=20,NAME='OUTFILE')
```

Before you execute the program containing this OPEN statement, you can assign an equivalence name to the logical name OUTFILE, as follows:

```
$ assign dma1:[scratch]test3.out outfile RET
```

Now, when an input/output statement in the program refers to logical unit 20, the system uses the equivalence name established for OUTFILE. Thus, the following statement results in a read from the file DMA1:[SCRATCH]TEST3.OUT:

```
READ (20,90)A,B,C
```

If there is no equivalence name for the logical name OUTFILE when you run this program, the system assumes that OUTFILE is a file specification. It uses your current disk and directory defaults and the default file type of DAT to complete the file specification for the output file.

You can find additional details on how to specify input and output files for FORTRAN programs in the *VAX-11 FORTRAN User's Guide*.

5.4 MACRO Input/Output

VAX-11 Record Management Services (RMS) provide macros for device- and file-independent input/output operations.

VAX-11 RMS uses control blocks to obtain information about the file or device you want to access (the File Access Block, or FAB) and the way you want to access records in the file (the Record Access Block, or RAB).

The $FAB macro constructs a FAB. When you code the $FAB macro, specify the file name (FNM) parameter to give the file specification of the file or device to which input/output is directed. For example:

```
OUTFAB: $FAB FNM=<WEATHER.STS>
OUTRAB: $RAB FAB=OUTFAB
```

The $RAB macro constructs a control block for record processing information.

The $OPEN and $CONNECT macros open the file for processing, and establish the connection between the FAB and the RAB. For example:

```
$OPEN FAB=OUTFAB
$CONNECT RAB=OUTRAB
```

When the program uses a $PUT macro to write to the output file defined by this FAB and RAB, it writes to the file WEATHER.STS. Note that the system supplies your current default disk and directory name to identify the file.

You can also specify a logical name with the FNM parameter in the $FAB macro. For example:

```
OUTFAB: $FAB FNM=<OUTFILE>
OUTRAB: $RAB FAB=OUTFAB
```

Before you execute the program to write this output file, you can assign an equivalence name to the logical name OUTFILE, as follows:

```
$ assign dma1:[scratch]test3.out outfile(RET)
```

Now, when a $PUT macro refers to the RAB established for OUTFILE, the system uses the equivalence name. For example, the following line in a program results in a write to the file DMA1:[SCRATCH]TEST3.OUT:

```
$PUT RAB=OUTRAB
```

If there is no equivalence name for the logical name OUTFILE when you run this program, the system assumes that OUTFILE is a file specification. It uses your current disk and directory defaults to complete the file specification, but does not supply a default file type. The output file would be named OUTFILE.

5.5 For More Information

For more information about logical names see the *VAX/VMS Command Language User's Guide* and the *VAX/VMS System Services Reference Manual.*

For details about using VAX-11 RMS macros, see the *VAX-11 Record Management Services Reference Manual.* Additional information on using logical names in MACRO programs is contained in the *VAX-11 MACRO User's Guide.*

Chapter 6
Tailoring the Command Language

As you continue to use the command language, you will discover that it is a powerful and flexible programming and applications development tool. You can simplify the command language to save yourself time during interactive terminal sessions and to establish your own default commands and command qualifiers. You can create **command procedure** files to execute batch jobs, either interactively or from a card reader. You can construct command procedures to perform development and applications programming tasks.

This chapter provides some elementary information on techniques you can use to tailor the command language to your individual needs. For example, you can:

- Establish synonyms to use in place of command names and entire command strings, as well as to establish default qualifiers for commands.
- Create command procedures to perform a specialized set of commands.
- Submit command procedures for processing as batch jobs.
- Use command procedures to perform programming functions, using the command language as a high-level programming language.

6.1 Symbols

You can equate symbols to character strings or arithmetic values by defining them in **assignment statements**. In addition to their use in command procedures (see Section 6.2), symbols are useful as synonyms for long, frequently used command strings. For example, you can equate the symbol ST to the command SHOW TIME:

```
$ st = "show time"RET
```

and subsequently use the symbol ST in place of SHOW TIME:

```
$ stRET
    9-JUL-1982  10:45:19
```

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

```
$ list = "directory/full"RET
```

Then, if you issue the following command line, the system substitutes the name DIRECTORY/FULL for the symbol LIST:

```
$ list myfile.datRET
```

The system executes the command string DIRECTORY/FULL MYFILE.DAT.

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

```
$ pquals = "/copies=2/forms=4/noburst"RET
```

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

```
$ print report.dat'pquals'RET
```

The system recognizes the apostrophes and substitutes the appropriate value (in this case a string of qualifiers) for the symbol PQUALS: PRINT REPORT.DAT/COPIES=2/FORMS=4/NOBURST

(Information about the effect of these qualifiers on the PRINT command can be found in the *VAX/VMS Command Language User's Guide*.)

6.2 Command Procedures

A command procedure is a file containing a sequence of commands to be executed by the operating system. You submit a command procedure with one command: the Execute Procedure (@) character for interactive processing or the SUBMIT command for batch processing.

For instance, you could create a command procedure to compile, link, and run the program AVERAGE mentioned in Chapter 4. First, create a file containing the commands below. (The default file type for a command procedure file is COM.)

```
$ FORTRAN AVERAGE
$ LINK AVERAGE
$ ASSIGN/USER_MODE TTB3: SYS$INPUT
$ RUN AVERAGE
```

The dollar signs before the commands of a command procedure are required syntax, indicating that the subsequent line is a command to be executed by the operating system. (They should be placed in column one of each command line.)

To execute this command procedure, use the Execute Procedure command (@) as shown below:

```
$ @average RET
```

When this command is executed, the system searches for the file AVERAGE.COM. When it locates the file, the system reads and executes, in turn, each command line in the file.

Note that to execute this command procedure for the AVERAGE.FOR program created in Chapter 4, you must substitute the device name of your own terminal for TTB3: in the ASSIGN command (line 3) of the procedure.

6.2.1 Using Symbols in Command Procedures

The sample command procedure shown in the preceding section is not very flexible: it can be used to compile, link, and execute only the FORTRAN program named AVERAGE. Command procedures can be made more general by using undefined symbols in the procedure and defining the symbols when the procedure is executed.

The following examples show two ways to write a generalized procedure to compile, link, and run any FORTRAN program.

- Using global symbols in command procedures

  If you use the symbol PROGRAM rather than the file name AVERAGE in the command procedure DOFOR.COM below, you can later assign different file names to the symbol PROGRAM, making the command procedure independent of a particular source program file.

```
$ FORTRAN 'PROGRAM'
$ LINK 'PROGRAM'
$ RUN 'PROGRAM'
```

  Before you execute this command procedure you must define the symbol PROGRAM. Use the assignment statement as shown below:

```
$ program == "average" RET
```

  In this assignment statement, the two equal signs are required to make the symbol PROGRAM a **global symbol**. Global symbols are recognized and substituted in any command procedure you execute. (**Local symbols**, on the other hand, are restricted by the command level at which they were assigned; thus, a local symbol assigned in one command procedure cannot be used outside that command procedure. Local symbols are assigned with one equal sign.)

Now when you enter the following command line, the system substitutes the value AVERAGE for the symbol PROGRAM in each line of the command procedure:

```
$ @dofor RET
```

If you subsequently redefine the value of PROGRAM to a different file name and execute DOFOR.COM again, a different source program will be compiled, linked, and run.

- Passing parameters to command procedures

  An alternate way to code the procedure DOFOR.COM is to take advantage of special symbols that the system defines automatically when you execute a command procedure. These symbols, called parameters, are named P1, P2, P3, and so on up to P8, and are defined on the @ command line.

  For example, assume that DOFOR.COM has the lines:

```
$ FORTRAN 'P1'
$ LINK 'P1'
$ RUN 'P1'
```

  To define a value — in this example, the file name — for the symbol P1, enter the file name when you give the @ command to execute the command procedure DOFOR.COM, as follows:

```
$ @dofor average RET
```

  The system automatically equates the name AVERAGE to the symbol P1, the first (and, in this example, the only) parameter passed to the command procedure. P2 through P8 are equated to null strings. When the command procedure executes, the value AVERAGE is substituted for the symbol P1.

6.2.2 Redefining System Commands

You can use command procedures and symbol assignment statements together to redefine and expand system commands.

For example, suppose that during your terminal sessions you frequently compile and recompile programs, creating many listing files (with a file type of LIS). To keep your directory uncluttered, you may want to purge these listings regularly. To do this housekeeping, you could create a command procedure named LOG.COM that contains the lines:

```
$ PURGE *.LIS
$ LOGOUT
```

You can use this command procedure in place of the LOGOUT command when you want to end your terminal session, as follows:

```
$ @log RET
```

The PURGE command line is automatically executed before you log out.

Moreover, you could define a symbol named LO that is equated to the following command string:

```
$ lo == "@log" RET
```

Then, when you type the command line

```
$ lo RET
```

the system substitutes the symbol LO with the @LOG command string, and executes your command procedure.

6.2.3 A LOGIN.COM File

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

The command procedure file must be named LOGIN.COM, and it must be in your default disk directory. When you log in to the system, the system automatically searches for a file with this file name. If the system locates the LOGIN.COM file, the system automatically executes the commands within that file.

For example, a LOGIN.COM file might contain:

```
$ ST == "SHOW TIME"
$ LIST == "DIRECTORY"
$ LO == "@LOG"
$ ASSIGN [MALCOLM.TESTFILES] TEST
$ TEST == "SET DEFAULT [MALCOLM.TESTFILES]"
```

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

Command procedures can be executed from within other command procedures. You may want to place the global assignment statements you use for command synonyms in a separate file, and execute this procedure in the LOGIN.COM file. For example, suppose the file SYNONYM.COM contains the lines:

```
$ ST == "SHOW TIME"
$ LIST == "DIRECTORY"
$ LO == "@LOG"
```

Your LOGIN.COM file would contain the line:

```
$ @SYNONYM
```

When this command is executed, the definitions in the synonym file are established.

6.3 Batch Job Processing

If you use the Execute Procedure command (@) interactively, you cannot enter other commands to do other work while the procedure is executing. If you want to execute a command procedure that requires a great deal of processing time, you can submit the command procedure as a batch job. When you submit it, the batch job is queued by the operating system; your terminal is then free for you to continue working interactively.

Use the SUBMIT command to request the operating system to place the command procedure in the batch job queue. The SUBMIT command assumes your current disk and directory defaults, as well as the default file type of COM. For example:

```
$ SUBMIT DOFOR(RET)
Job 312 entered on queue SYS$BATCH
```

In this command, DOFOR is the file name of a command procedure. The system responds to the SUBMIT command with a message indicating that the job was successfully queued to the SYS$BATCH queue and has the job identification number of 312. As soon as the batch job is queued, you can continue interactive use of the terminal; the system will process the batch job.

6.4 Programming Command Procedures

The examples of assignment statements and command procedures in this chapter show only a few things you can do with command procedures. There is a special set of commands that you can use in command procedures to perform functions similar to those available in high-level programming languages. Some brief examples of these commands are shown below to illustrate the versatility of VAX/VMS command procedures. You can:

- Assign arithmetic values to symbol names, and use these symbols in assignment statements with arithmetic expressions. For example:

```
$ COUNT == 1
    .
    .
    .
$ COUNT == COUNT + 1
```

- Transfer control to a command line in a procedure that is not the next line in the file. For example:

```
$ LOOP:
    .
    .
    .
$ GOTO LOOP
```

- Conditionally execute a command based on a comparison of values, strings, or symbols. For example:

```
$ IF COUNT.LT.10 THEN GOTO LOOP
```

- Interactively define a value for a symbol by displaying a prompting message on the terminal. For example:

```
$ INQUIRE NUMBER
$ IF NUMBER.EQ.1 THEN GOTO NEXT
```

- Establish a default course of action should an error occur during processing of any command or program. For example:

```
$ ON ERROR THEN EXIT
```

6.5 For More Information

For additional examples of developing command procedures, see the *VAX/VMS Guide to Using Command Procedures.*

Glossary

assembler

Language processor that translates a source program containing assembly language directives and machine instructions into an object module.

assembly language

Machine oriented programming language. VAX–11 MACRO is the assembly language for the VAX–11 computer.

assignment statement

Definition of a symbol name to use in place of a character string or numeric value. Symbols can define synonyms for system commands or can be used for variables in command procedures.

batch

Mode of processing in which all commands to be executed by the operating system and, optionally, data to be used as input to the commands are placed in a file or punched onto cards and submitted to the system for execution.

buffer

A temporary storage area.

command

An instruction or request for the system to perform a particular action. An entire command can consist of the command name, parameters, and qualifiers.

command interpreter

The operating system component responsible for reading and translating interactive and batch commands. The default command interpreter for the VAX/VMS operating system interprets the DIGITAL Command Language (DCL).

command line

The entire command string, including the command and any parameters or qualifiers it may have.

command procedure

File containing a predefined sequence of commands to be executed by the operating system. The command procedure can be submitted for execution at the terminal or as a batch job.

compiler

Language processor that translates a source program containing high-level language statements (for example, FORTRAN) into an object module.

concatenate

To link together in a series.

cursor

A line or block indicator used in a video display terminal to indicate position.

debugger

Interactive program that allows you to display and modify program variables during execution and to step through a program to locate and detect programming errors.

default

Value supplied by the system when a user does not specify a required command parameter or qualifier.

default disk

The disk from which the system reads and to which the system writes, by default, all files that you create. The default is used whenever a file specification in a command does not explicitly name a device.

delimiter

Character that marks the beginning or end of a string.

device name

Identification of a physical device (for example, DBA2) or a logical name (for example, SYS$OUTPUT) that is equated to a physical device name.

directory

File cataloging a user's files on a particular device for a user.

editor

Program that creates or modifies files. In VAX/VMS, the default system editor is interactive.

equivalence name

Character string equated to a logical name, such that when a command or program refers to a file or device by its logical name, the system translates the logical name to its predefined equivalence name.

file

Collection of data treated as a unit; generally used to refer to data stored on magnetic tapes or disks.

file name

The name component of a file specification, consisting of from one to nine characters.

file specification

Unique identification of a file. A file specification describes the physical location of the file, as well as file name and file type identifiers that describe the file and its contents.

file type

The type component of a file specification, consisting of from one to nine characters. A file type generally describes the nature of a file, or how it is used. For example, FOR indicates a FORTRAN source program.

global symbol

A symbol defined with an assignment statement that is recognized in any command procedure that is executed.

header page

Printed page at the beginning of a listing that identifies the printed file.

image

Output from the linker, created from processing one or more object modules. An image is the executable version of a program.

input file

File containing data to be transferred into the computer.

interactive

Mode of communication with the operating system in which a user enters a command, and the system executes it and responds.

job

(1) The accounting unit equivalent to a process; jobs are classified as batch or interactive. (2) A print job.

keypad

The small set of keys next to the main keyboard on a terminal.

keyword

A command name, qualifier, or option. Keywords must be typed verbatim or truncated according to the rules of DCL.

line editor

Program that allows you to make additions and deletions to a file on a line by line basis.

linker

Program that creates an executable program, called an image, from one or more object modules produced by a language compiler or assembler. Programs must be linked before they can be executed.

local symbol

A symbol defined with an assignment statement that is recognized only within the command procedure in which it is defined.

logical name

Character string used to refer to files or devices by other than their specific names. A command or program can refer to a file by a logical name; the logical name can be equated to an equivalence name at any time; when the command or program refers to the logical name, the system translates the logical name to its defined equivalence name.

log in

To perform a sequence of actions at a terminal that establishes a user's communication with the operating system and sets up default characteristics for the user's terminal session.

log out

To terminate interactive communication with the operating system. The LOGOUT command executes the procedure and ends a terminal session.

machine code

A sequence of binary machine instructions in a form executable by the computer.

network

A collection of interconnected computer systems.

node specification

The component of a file specification which identifies the location of a computer system in a network of computer systems.

object module

Output from a language compiler or assembler that can be linked with other object modules to produce an executable image.

operating system

The system software that controls the operations of the computer.

output file

File to which the computer transfers data.

parameter

Object of a command. A parameter can be a file specification, a symbol value passed to a command procedure, or a word defined by the DIGITAL Command Language.

password

Protective word associated with a user name. A user logging in to the system must supply the correct password before the system will permit access.

prompt

Word(s) used by the system as cues to assist a user's response.

qualifier

Command modifier that describes the operation of a command. A qualifier is always preceded by a slash character (/).

queue

A line of items waiting to be processed.

range specification

Used with EDT line editor to define the line(s) to be affected by the editing command.

reverse video

A feature of the VT100 terminal that reverses the default video contrast. If black figures upon a white background is the default, reverse video displays white upon black. Used with some EDT keypad commands to highlight a range of text.

scrolling

A feature of a video terminal that allows the display of more than one screenful of text by vertical movement.

source program

A program written in a language other than machine code that must be compiled or assembled to be used.

subdirectory

Directory file cataloged in a higher-level directory that lists additional files belonging to the owner of the directory.

terminal

Hardware communication device, with a typewriter-like keyboard that receives and transmits information between users and the system.

user name

Name by which the system identifies a particular user. To gain access to the system, a user specifies a user name followed by a password.

version number

Numeric component of a file specification. When a file is edited, its version number is increased by one.

wild card character

A symbol used with many DCL commands in place of all or part of a file specification to refer to several files rather than specifying them individually.

Index

VAX/VMS Primer
AA-D030C-TE

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________

Name ____________ Date ____________

Organization ____________

Street ____________

City ____________ State ____________ Zip Code or Country ____________

— — Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03061

— — Do Not Tear - Fold Here —

Cut Along Dotted Line

Printed in U.S.A.