digital

VAX/VMS Primer

Order No. AA-D030B-TE

VAX11

March 1980

This tutorial document introduces a new VAX/VMS user to the DIGITAL Command Language, file manipulation, program development, and elementary operating system concepts.

VAX/VMS Primer

Order No. AA-D030B-TE

SUPERSESSION/UPDATE INFORMATION:	This revised document supersedes the VAX/VMS Primer (Order No. AA-D030A-TE)
OPERATING SYSTEM AND VERSION:	VAX/VMS V02
SOFTWARE VERSION:	VAX/VMS V02

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, August
Revised, March

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

Contents

Figures

Tables

Preface

People use a VAX-11 computer system to do work, principally programming. To do their work, they use the VAX/VMS operating system, a software component of a VAX-11 computer system which allows many users to share the resources of the computer and its hardware devices (such as terminals, disks, magnetic tapes, and printers).

The objective of this primer is to introduce you, a new user, to the VAX/VMS operating system. Thus, the primer presents:

- Some of the fundamental operations you must be familiar with to use the operating system, such as:

  Using a terminal
  Using an editor
  Developing programs
  Correcting mistakes

- Examples of using the operating system's command language, DCL (DIGITAL Command Language). DCL is the primary method you use to tell the operating system what work you want it to do. In the primer is information on:

  Entering commands
  Interpreting messages
  Tailoring commands for specific uses

- Frequently used terms and concepts you encounter when using the operating system, such as:

  Terminal session
  Batch job
  File specification
  Logical names

- The places to go for more information about the operating system, its command language, and its other features and services. For example:

  Table 1 lists general information categories for VAX-11 documentation.

  Summary sections at the end of each chapter point to sources of specific information about material discussed in that chapter.

Depending on your prior experience or knowledge, you may want to read this primer carefully or just skim it for specific details.

Table 1: Sources of Information in VAX-11 Documentation

For information on:	Look in these manuals:
VAX/VMS Features and Documentation	VAX-11 Information Directory and Index VAX/VMS Summary Description and Glossary
The Command Language	VAX/VMS Command Language User's Guide VAX/VMS Guide to Using Command Procedures
VAX-11 Programming Languages:	
VAX-11 MACRO	VAX-11 MACRO User's Guide VAX-11 MACRO Language Reference Manual
VAX-11 FORTRAN	VAX-11 FORTRAN User's Guide VAX-11 FORTRAN Language Reference Manual
VAX-11 BASIC	VAX-11 BASIC User's Guide VAX-11 BASIC Language Reference Manual
VAX-11 BLISS-32	VAX-11 BLISS-32 User's Guide BLISS Language Guide
VAX-11 COBOL-74	VAX-11 COBOL-74 User's Guide VAX-11 COBOL-74 Language Reference Manual
VAX-11 PASCAL	VAX-11 PASCAL User's Guide VAX-11 PASCAL Language Reference Manual
Program Development, Testing, and Control	VAX-11 Text Editing Reference Manual VAX-11 EDT Editor Reference Manual 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 User's Guide RMS-11 User's Guide
VAX/VMS Installation, Management, and Operations	VAX/VMS Release Notes VAX-11 Software Installation Guide VAX/VMS System Manager's Guide VAX/VMS Operator's Guide VAX/VMS UETP User's Guide

(Continued on next page)

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

For information on:	Look in these manuals:
Programming Utilities and Development Tools	VAX-11 Utilities Reference Manual VAX-11 PATCH Utility Reference Manual VAX-11 Sort User's Guide VAX/VMS System Services Reference Manual VAX/VMS I/O User's Guide VAX-11 Run-Time Library Reference Manual VAX-11 Guide to Creating Modular Library Procedures VAX/VMS System Dump Analyzer Reference Manual
Compatibility Mode Programming	VAX-11/RSX-11M User's Guide VAX-11/RSX-11M Programmer's Reference Manual RMS-11 User's Guide RMS-11 MACRO-11 Reference Manual

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-1978 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

Interactive users communicate with the VAX/VMS operating system through a **terminal**; a terminal looks and sometimes behaves like a typewriter, except that it is connected to the computer. You tell the system what you want it to do by typing commands at the terminal. The system responds by performing your request. If the system cannot properly interpret what you type, it displays an error message on your terminal screen.

Batch users communicate with the system by entering all the commands they want to execute on cards and placing the cards in the system card reader. The system produces a printed listing describing the outcome of the 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: hardcopy terminals and video display terminals.

Hardcopy terminals print on continuous forms of paper. Figure 1–1 shows an example of a hardcopy terminal, the LA120.

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

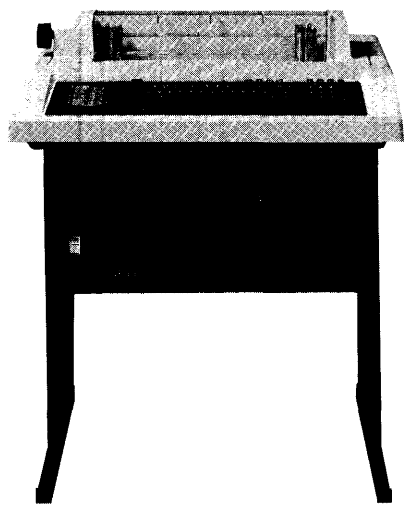


Figure 1–1: The LA120 Terminal

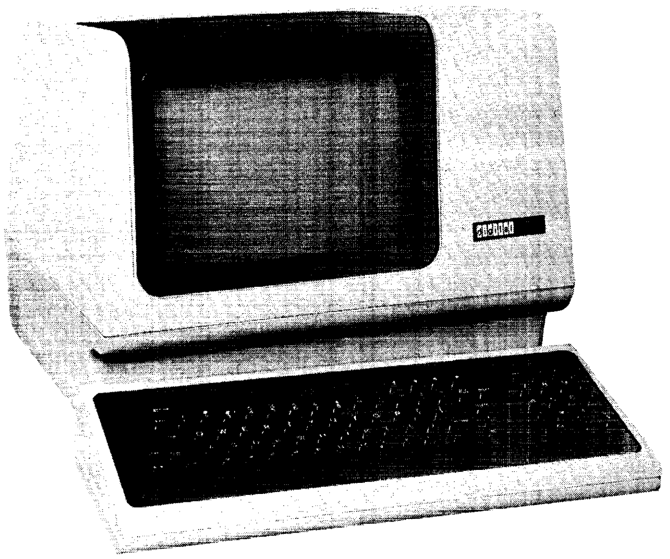


Figure 1–2: The VT100 Terminal

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 a typewriter, in terms of the positions of the alphabetic and numeric keys. 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.

The symbols in the diagram are the symbols used throughout this text as a shorthand notation to refer to pressing these keys. For example, the symbol RET in text means that you press the key labeled RETURN, CTRL/Y means that you hold down the CTRL key while you press the Y key.

1.3 Logging In

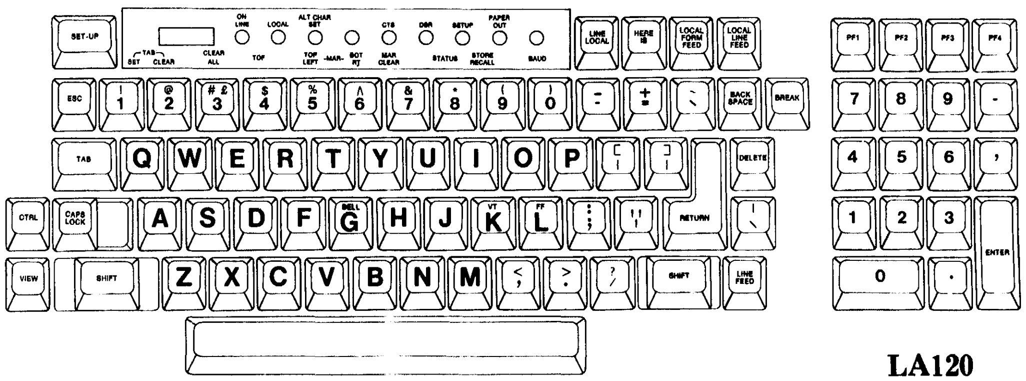
To access the system from a terminal and identify yourself to the system, you must **log in**. When you log in successfully, you establish a **terminal session**.

However, before you can log in to the system, 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**.

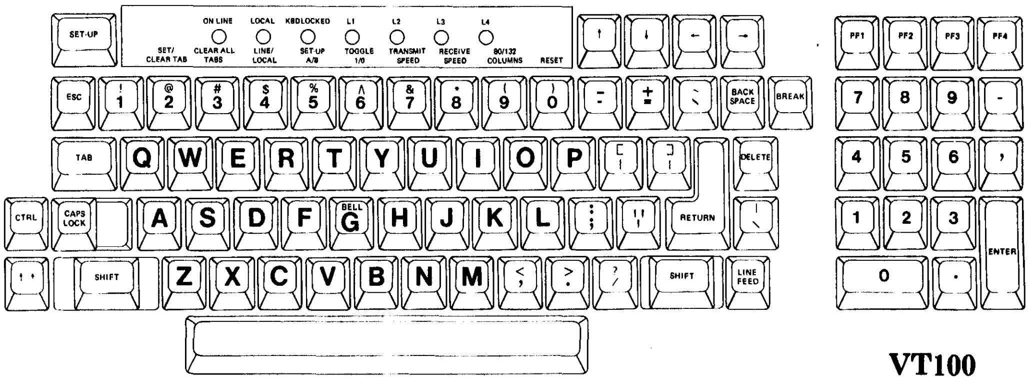
Your user name is a unique name that identifies you to the system and distinguishes you from other users. In many cases, a user name is the same as a person's real first or last name.

Your password is for your protection. If you maintain its secrecy, other users cannot gain access to the system under your user name.

When you access the system, you must enter both your user name and your password before you are allowed to begin typing **commands**.



LA120



VT100

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 Getting the Computer's Attention

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 followed by RET. After you enter your name, the system prompts you to enter your password. When you type the password, the system does not echo it; that is, the password is not displayed on the terminal.

The login sequence looks like the following:

```
(RET)
Username:  MALCOLM(RET)
Password:(RET)
       WELCOME TO VAX/VMS VERSION 2.00
$
```

The dollar sign is a symbol the system uses as a prompt. When this character appears on the far left of the terminal, 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 and that you must repeat the login procedure.

1.4 Entering Commands

All commands to the system are words that describe the functions they perform. For example:

```
$ show time(RET)
```

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

```
  17-JUL-1978 11:55:40
$
```

When you enter commands, you can type them using either uppercase or lowercase letters, or a combination of both.

Commands utilize command **parameters** to define what is to be acted upon by the command and command **qualifiers** to define how that action is to occur.

You use command parameters to define the object of a command verb or to specify a command function. For example:

```
$ print myfile.lis(RET)
```

In this command, MYFILE.LIS is a parameter for the PRINT command. The PRINT command requires an object; in this case, the name of the **file** you want printed.

You use command qualifiers to 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. You must always precede each qualifier in a command with a slash character (/).

All command qualifiers and those parameters which do not require a **file name** are constructed using **keywords**. Keywords have a predefined meaning for the commands in which they are used. You must use them as defined, in some cases supplying a value to complete them. For example, the /COPIES qualifier for the PRINT command needs a value: you supply the number of copies you want printed.

1.4.1 Defaults

A **default** is an action taken by a command when you do not specify a choice. Defaults are applied to command qualifiers as well as to the names of files you specify as command parameters. For example, the /COPIES qualifier shown in the PRINT command has a default value of 1 because printing one copy of a file is a common need. If you do not tell the print command how many copies of the file you want, it prints a single copy by default.

1.4.2 System Responses

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:

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

You can use this job identification number to determine the status of your job in the queue.

Not all commands display informational messages; in fact, successful completion of a command is most commonly indicated by a $ prompt for another command. Nonsuccessful completion is always indicated by an error message or messages.

1.4.3 Recovering from Errors

Some of the keys noted on the keyboard in Figure 1–3 provide line-editing functions. By using these keys to correct errors you make while typing lines, you will not pass a bad command to the system.

The line-editing function keys are:

DEL

Backspaces over one character typed on the current line, then deletes the character. Some video display terminals actually move the print 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

Deletes the current line and performs a carriage 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 carriage 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, but 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. This 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.lisRET

    .
    .
CTRL/Y
$ stopRET
$
```

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

1.4.4 Error Messages

If, despite the use of line-editing function keys, you enter a command incorrectly, the system responds with an error message. If you enter a command name incorrectly, the system displays a message and prompts for a command line as if no command had been entered:

```
$ capyRET
%DCL-W-IVVERB, unrecognized command
  \CAPY\
$
```

The three-part code preceding the descriptive part 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 a command 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 file. The second message indicates the reason, that is, the file cannot be found. The facility name in this message, RMS, is the VAX/VMS file system; error messages related to file handling are generally RMS messages.

1.4.5 Truncating and Abbreviating Commands

When you type commands or keywords, you do not always need to type the full command or keyword name. In fact, you never have to type more than four characters, and in many cases you can type only one or two characters. The rule to follow is: a four-character abbreviation will always work and you must type at least the minimum number of characters necessary to make the command or keyword unique.

For example, the SET and SHOW commands both begin with the letter "S." Therefore, when you type either of these commands, you must type at least two characters, SE or SH, to make the command unique.

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

1.5 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 a **file specification** parameter.

If a command requires two or more parameters, it prompts for each parameter. In response to each prompt, you can enter 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 specification is entered separately. You could, however, enter both file specifications following the first prompt, as shown below:

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

1.6 The HELP Command

When you are using the 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:

```
$ help RET
```

When you type this command, the system responds by displaying information about using the HELP command.

If you type:

```
$ help print RET
```

The information displayed includes a synopsis of what the PRINT command does, the valid qualifiers and their default values, and the parameters required by the command.

The HELP command also tells you what additional information is available, in the form of keywords that you can also specify. For example, the HELP PRINT command tells you that additional information is available about the PRINT command qualifiers. Then, you can type:

```
$ help print qualifiers RET
```

The HELP command displays a list of valid qualifiers for the PRINT command.

1.7 Logging Out

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

```
$ logout RET
```

The system responds:

```
MALCOLM  logged out at 17-JUL-1978 12:43:10.38
```

Note that shutting your terminal off, or setting the REMOTE/LOCAL switch to LOCAL while you are logged on, does not cause the system to **log** you **out** automatically. You must use the LOGOUT command to end a terminal session. If you shut a terminal off without logging out properly, another user can later turn the terminal on and use your account to access your files.

1.8 For More Information

The *VAX/VMS Command Language User's Guide* is the primary reference for information about the DIGITAL Command Language and the DCL **command interpreter**. 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
Files: Using the Editor

Before you can use the VAX/VMS operating system to create and run programs, you should have an understanding of how the operating system uses and identifies files. You should also know something about the editor, an interactive program you can use to create and update source programs or data files.

2.1 Files

A file is a collection of logically related data located on a medium, such as a disk, tape, or card deck. Many system commands require input files or produce output files. To access files that already exist, or to give names to files that you create with system commands, you must know how to identify files.

The system uniquely identifies a file by its file specification. Some of the characteristics of a file specification are described below.

A file is first identified by its location, that is, the physical device on which it is stored. When you log in to the system, the system assumes that all the files you create or use are on a specific disk, your **default disk**. This default is provided for you by the system manager who sets up your account.

Since a disk can contain files belonging to many different users, each disk has a set of files called directories. A **directory** is simply a catalog of the files on that disk that belong to a particular user. As with the default disk, the system also assumes that the files you refer to are cataloged in a default directory.

You can find out what your current default disk and directory are by issuing the SHOW DEFAULT command:

```
$ show default(RET)
  DBA2:[MALCOLM]
```

This response from the SHOW DEFAULT command indicates that the default disk device is DBA2 and the default directory is named MALCOLM.

2.2 File Names and File Types

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

```
filename.type
```

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

The file type can be from one to three alphanumeric characters; it must be preceded by a period. The file type describes more specifically the kind of data in the file. Again, you can choose any alphanumeric characters for the file type. However, the system recognizes several default file types used for special purposes.

Among these default file types are:

File Type	**Use**
FOR	VAX-11 FORTRAN language source statements
MAR	VAX-11 MACRO assembly source statements
DAT	Data file
LIS	Output listing from a compiler or the assembler
OBJ	Object module output from a compiler or the assembler
EXE	Executable program image

For example, if you create a file containing source statements for a FORTRAN program, you should use the file type FOR.

2.2.1 Version Numbers

Every file also has a **file version number** associated with it to distinguish among copies of the file made during various updating operations.

You specify a version number in a file specification by placing it after the file type, preceded by a semicolon (;) or a period (.). For example:

```
$ print myfile.lis;2(RET)
```

This PRINT command requests that the version numbered 2 of the file MYFILE.LIS be printed.

When you create a new file, the system always assigns it a version number of 1. When you make an update, the system increments the version by 1. When you refer to an existing file without specifying a version number, the system always locates the most recent version (that is, the highest number). You can override these default version numbers by specifying an explicit version number in a command, as shown in the example above.

2.3 Using the Editor

An **editor** is a program that lets you create and modify files. When you issue the EDIT command from the terminal, you enter the file name and file type of the file you want to create or modify, as shown below:

```
$ edit newfile.dat(RET)
```

This command invokes the editor and begins an editing session, during which you use a special set of commands that the editor understands and uses to process the file named NEWFILE.DAT.

When you use the editor, what you type is not recorded in a file until you instruct the editor to copy the input data from system memory and write it into a disk file. You can then access the file at a later time and make changes to it, such as adding or deleting text.

The default VAX/VMS editor is called SOS. When you issue the EDIT command, SOS identifies itself and you begin your editing session. The following sections describe how to use the editor to create and make changes to a file.

2.3.1 What an SOS File Looks Like

SOS "sees" text in terms of lines; a line is a string of characters, spaces, and tabs ending with a (RET). Every line in an SOS file has a number which is displayed with the text. The three numbers below start three separate SOS lines:

```
    .
00100 (RET)
00200 This is an SOS line(RET)
00300     111111111122222222223333333333444444444455555555556666666666
7777777777888888888899999999991111111111222222222233333333334444444444
5555555555666666666677777777778888888888999999999900000000001111111111
2222222222333333333344444444445555555555666666666677777777778888888888
9999999999000000000011111111112222222222333333333344444444445555555555
6666666666777777777788888888889999999999000000000011111111112222222222
333333333344444444445555555555********** THE MAXIMUM SOS LINE IS 500
CHARACTERS LONG ****(RET)
```

When you use SOS to create a new file, SOS assigns the line numbers as you enter lines into the file. If you use SOS to edit a file that does not have line numbers (for example, a file created with some other editor), SOS assigns the line numbers when it opens the file for editing.

2.3.2 Creating a New File

To invoke SOS, use the EDIT command, giving the file specification of the file you want to create. SOS will supply your default disk and directory name for the file if you do not specify the disk and directory when you give the file specification.

SOS responds to the EDIT command by displaying its current mode (either Input mode or Edit mode) and the full file specification of the file. If the file you specify is a new one, SOS places itself in Input mode and prompts you to begin entering input, as shown in the following example.

```
$ edit newfile.dat(RET)
Input:DBA2:[MALCOLM]NEWFILE.DAT;1
00100
```

The line number prompt (00100) appears when you are in **Input mode**. In Input mode, each line you enter is placed in the file.

Terminate each line by pressing (RET). After each line, SOS prompts with a new line number for the next line. By default, SOS begins numbering at 100 and uses an increment of 100 for each line.

If you make mistakes while you are entering input lines, you can use the line-editing function keys on your terminal to correct mistakes, or (CTRL/U) to discard a line.

When you have finished entering text, press (ESC) to signal SOS that you do not want to enter more input. (On some terminals, the key that performs the escape key function is labeled ALTmode, SELect, or PREfix.)

You can press (ESC) either at the end of the last line of input, or following the prompt for the next line. When you press (ESC), SOS echoes it on your terminal as a dollar sign ($) character. SOS leaves Input mode and enters **Edit mode**. In Edit mode, SOS Interprets each line that you enter as a command for it to perform a particular operation. When SOS is in Edit mode, it displays an asterisk (*) in column 1.

Figure 2–1 shows a sample editing session to create a new file.

```
Terminal Display                                Comments

$ edit newfile.dat(RET)                         Invoke SOS
Input:DBA2:[CRAMER]NEWFILE.DAT;1                SOS new file message
00100   SOS is now in Input mode.(RET)          Prompt; first line entered
00200   It talks each lo ^U                     Discard line
00200   It takes every line you (RET)           Retype line 200
00300   enter and(RET)                          Enter line 300
00400   gives it a line number.(RET)            Enter line 400
00500   (RET)                                   A blank line
00600   Get out of Input mode(RET)              Enter line 600
00700   with an escape.$                        (ESC) is displayed as $
*                                               SOS Edit mode prompt
```

Figure 2-1: Creating a File with the SOS Editor

2.3.3 Writing a File onto Disk

As you are entering input for a file, you can request SOS to write the data into a disk file to save what you have entered. There are two SOS commands for this purpose: W (Save World) and E (End).

If you want to stay in SOS, and make changes to lines you have entered, use the W (Save World) command:

```
*w(RET)
```

SOS responds by writing the file to disk, telling you the file specification of the saved file, and returning control to you in SOS Edit mode.

```
[DBA2:[MALCOLM]NEWFILE.DAT;1]
*
```

If you want to terminate the session with SOS, save your edited file, and return to the DCL command level, use the E (End) command instead of the W command:

```
*e RET
```

SOS also responds to the End command by displaying the file specification before it exits:

```
[DBA2:[MALCOLM]NEWFILE.DAT;1]
$
```

The dollar sign prompt indicates that you have ended your editing session and can now enter system commands.

2.3.4 Editing an Existing File

When you invoke SOS to edit a file that already exists, SOS places itself in Edit mode when it opens the file. For example, if you issue the EDIT command to edit the file NEWFILE.DAT and this file already exists, SOS responds as shown below:

```
$ edit newfile.dat RET
Edit:DBA2:[MALCOLM]NEWFILE.DAT;1
*
```

The asterisk prompt indicates that SOS is ready to accept editing commands.

2.3.5 SOS Commands

SOS has a command language of its own. SOS commands are usually one-character abbreviations of a verb, for example E for End, I for Input, F for Find, and so on.

Some SOS commands also accept qualifiers that modify or restrict the action of a command.

Many SOS commands accept qualifiers that specify a line number or a range of line numbers. The line numbers tell SOS on which lines to perform the requested command. For example, the P (Print) command requests SOS to display a line or a range of lines in the file on the terminal. To display lines 300 and 400, type:

```
*p300:400 RET
00300   enter and
00400   gives it a line number.
```

For some commands, a qualifier is a character string that you want SOS to locate or change. You must terminate character string qualifiers by pressing ESC. For example, the Find command requests SOS to locate and display the next line that contains a particular character string. To locate the line that contains the string "escape", type:

```
*fescape ESC RET
00700   with an escape.
```

After you enter a command to SOS, you must always use RET to pass the command to SOS for execution. SOS responses depend on the operation performed. If you make an error typing a command — for example, a spelling error — SOS issues the message:

```
Illegal syntax of command
```

You must retype the command.

2.3.6 Specifying Line Numbers in SOS Commands

When you edit a file with SOS, SOS always keeps track of the **current line**. This is the number of the line that you are "looking at" at any particular time. When you use SOS commands that add or delete lines, the current line can move toward a smaller number or toward a larger number in the file.

Although you do not always need to know the line number of a line you want to look at or change, you may at times want to specify a line number in a command.

For commands that accept a line number or range of line numbers for a qualifier, you can use a variety of shorthand notations to represent line numbers. The shorthand consists of punctuation marks that have specific meanings to SOS when they appear in place of or with line numbers. Some of these punctuation marks and their meanings are summarized in Table 2-1.

Table 2-1: Punctuation Marks for SOS Line Number Arguments

Mark	Meaning	Example	Explanation of Example
:	Range	**100:500**	Lines 100 through 500
.	Current line	**.:500**	Current line through line 500
^	First line	**^:.**	First line through current line
*	Last line	**.:***	Current line through last line
;	Increment [1]	**300;10**	Increment new lines by 10 while doing this command

[1] The use of increments is discussed in Section 2.3.10, "Inserting Lines in a File."

2.3.7 Summary of Frequently Used SOS Commands

Table 2-2 lists some of the more frequently used SOS commands. This table does not represent a complete list of commands, nor even all of the possible things that each of the commands listed can do. Rather, it summarizes the commands and qualifiers described in this chapter.

2.3.8 Examples of SOS Editing Commands

The following sections show examples of using the SOS commands listed in Table 2-2. The examples also illustrate combinations of line number qualifiers, using positions and ranges for different commands.

Table 2-2: A Subset of SOS Commands

Command	Qualifiers[1]	Function
[RET]	None	Print the next line in the file
[ESC]	None	Print the previous line in the file
P	position or range	Display line(s) at the terminal
I	position	Insert new line(s) into the file
N	increment and range	Renumber the lines in the file
R	position or range	Replace one or more lines with new line(s)
D	position or range	Delete line(s) from the file
F	string [ESC]	Find and print the next line containing the specified string
S	string [ESC] string [ESC]	Substitute one string for another
E	None	End the editing session

[1] Key:

position	means you can specify a single line number
range	means you can specify a range of line numbers
increment	is a numeric value for line-number incrementing
string	is any character string

For the purposes of the examples, suppose you have created an SOS file as follows:

```
$ edit sostest.dat[RET]
Input:DBA2:[MALCOLM]SOSTEST.DAT;1
00100   This is the first line[RET]
00200   of a file to try out SOS.[RET]
00300   Chokeberries, Persimmon Seeds, Mangoes.[RET]
00400   Apples, Pears, Plums, Cherries.[RET]
00500   Here is the fifth line.[RET]
00600   The sixth line.[RET]
00700   And the seventh.[RET]
00800   My little persimmon seed.[RET]
00900   The singed shores, the bears who talk,[ESC]
*w [RET]
[DBA2:[MALCOLM]SOSTEST.DAT;1]
*
```

The lines in this sample file are used to illustrate the SOS commands in the remainder of this section. Note that the W (Save World) command saved the input text, and that SOS issues its Edit mode prompt for you to enter commands.

It is good practice to use the W command frequently during an editing session, to protect your input or modifications from being lost through some user error.

2.3.9 Scanning a File

There are a number of ways you can skip through a file, displaying lines on the terminal before making changes to them.

For convenience, SOS interprets two of the keys on your keyboard as commands. These are RET and ESC.

When SOS is in Edit mode and you press RET on a line that has no command on it, SOS moves forward one line in the file and displays the next line. Conversely, if you press ESC, SOS backs up one line and displays the previous line.

For example, after you issue the W command above, the current line is still positioned at the last line of text that was entered. The following example shows the effect of using RET and ESC.

```
*ESC
00800   My little persimmon seed.
*ESC
00700   And the seventh.
*RET
00800   My little persimmon seed.
*RET
00900   The singed shores, the bears who talk,
*RET
No such line exists
*
```

If SOS is already pointing at the last line of the file, and you press RET, SOS informs you that there are no more lines, as the example shows.

To display a line that is not immediately before or after the current line and to position the current line pointer at that line, use the P (Print) command. For example:

```
*p500RET
00500   Here is the fifth line.
```

When you specify a line number, you can omit the leading zeros.

To display the current line (in case you lose track of where you are in a file), use the period (.), as follows:

```
*p.RET
00500   Here is the fifth line.
```

The P command also prints a range of lines, for example:

```
*P500:700(RET)
00500   Here is the fifth line.
00600   The sixth line.
00700   And the seventh.
```

To position SOS to point to the first line in the file, and to display the line, use the circumflex (ˆ), as follows:

```
*P^(RET)
00100   This is the first line
```

On some terminals, this function of the circumflex character is performed by an up-arrow (↑).

Similarly, you can display the last line in the file by specifying:

```
*P*(RET)
00900   The singed shores, the bears who talk,
```

If you use the P command without specifying a line number or a range of line numbers, SOS displays 16 lines, beginning with the current line.

2.3.10 Inserting Lines in a File

The I (Input) command tells SOS to enter Input mode, so you can enter new lines of text into the file. You must specify where you want the new lines to go. The following example shows how you would request SOS to enter Input mode to insert a new line of text following the current line:

```
*P500(RET)
00500   Here is the fifth line.
*i.(RET)
00550      add a new line(ESC)
*
```

The SOS input prompt indicates the new line will be numbered 550. After you enter the line, SOS returns to Edit mode when you press either (RET) or (ESC) to terminate the line. The current line pointer is positioned at the last line of text entered.

To insert a new line of text following a line that is not the current line, specify the line number, as follows:

```
*i600(RET)
00650   Terminate the line with return or escape(ESC)
*
```

The asterisk prompt indicates that SOS has returned to Edit mode.

2.3.10.1 Line Number Increments for Inserting Lines — SOS has two ways to handle multiple-line insertions. One way is for insertions after the last line in a file, another for insertions between existing lines.

When you begin inserting lines following the last line in the file, SOS does not return to Edit mode, but expects that you want to enter more than one line. For example, the following sequence shows how to position the current line at the end of the file and begin entering new lines into the file.

```
*i*(RET)
01000  Enter new lines of text as long(RET)
01100  as you want; SOS keeps prompting for more lines(RET)
01200  until you use escape to return to Edit mode.(ESC)
*
```

Note that SOS numbers the new lines using the default increment for line numbers, 100, beginning after the last line in the file.

When you want to insert more than one line between two existing lines in the file, specify a line number increment when you issue the I command. For example, the following commands tells SOS to insert new lines following line 700, and to number each new line in increments of 5:

```
*i700;5(RET)
00705  A new line here, incremented by 5.(RET)
00710  Another new line.(RET)
00715  Last new line.(RET)
00720  (ESC)
*
```

The semicolon (;) in the above example represents a temporary increment: SOS numbers new lines by 5 as long as it is in Input mode as a result of this I command.

In this example, (ESC) is pressed after the prompt for the line 720. You could have pressed (ESC) rather than (RET) at the end of line 715. The result is the same; the last new line that is actually entered in the file is line 715.

2.3.10.2 Renumbering Lines — If you insert many new lines between existing lines, SOS may run out of room to write the new lines into the file. If so, it issues the message:

```
Insufficient line numbers for insertion
*
```

Before you can enter any more lines at this position in the file, you must use the N (reNumber) command to renumber the lines:

```
*n(RET)
*
```

This command renumbers all lines in the file. SOS uses an increment of 100, unless you have specified another increment during the current SOS editing session. If you have, SOS uses the last increment you specified. For example, if you had specified an increment of 5 (as in the insert example above), all lines in the file would be renumbered in increments of five. Note that when

SOS finishes renumbering the lines it issues the asterisk prompt. The SOS current line pointer is positioned at the end of the file.

The line numbers shown in the remaining examples in this chapter reflect the result of a reNumber command.

2.3.11 Deleting Lines

The D (Delete) command deletes one or more lines from a file. For example, to position the line pointer at line 400 and delete the line, enter the commands:

```
*p400(RET)
00400   Apples, Pears, Plums, Cherries.
*d.(RET)
1 Line(s) deleted (00400/1).
*
```

SOS responds by telling you the line number of the deleted line. When you use the D command to delete only the current line, you can omit the period (.); SOS deletes the current line by default. The D command does not change the position of the current line pointer. To move the line pointer to the next line in the file, you must press (RET).

The /1 in the SOS response shown above is a page number. With SOS you can use page numbers to divide large files into manageable units. In fact, SOS will create a new page for you if you attempt to include more than 65535 line numbers in one page. In small files, like the ones in this primer, the entire file is usually on a single page, page 1.

2.3.12 Replacing Lines

The R (Replace) command combines the Delete and Input commands. When you specify a line number or a range of line numbers, SOS first deletes the specified line(s), then goes into Input mode to accept a line of input. The following example shows how to replace a single line:

```
*p1400(RET)
01400   The singed shores, the bears who talk,
*r.(RET)
1 Line(s) deleted (001400/1).
01400   The singing sands, the beasts that talk,(ESC)
*
```

After you enter the new line, SOS returns to Edit mode when you press either (ESC) or (RET) unless you have replaced the last line in the file. Note that the line replaced here, entered as line 900 during the original input in the file, is now line 1400 as a result of the reNumber command.

You can also specify a range of lines, for example:

```
*r500:700(RET)
3 Line(s) deleted (00500/1:00700)
00500
```

This command requests SOS to delete lines 500 through 700 and to place itself in Input mode to accept replacement lines for lines 500, 600, and 700.

When you want to replace one or more lines with a larger number of lines of input, specify a line number increment with the R command. Increment arguments for the R command are the same as for the I (Input) command, described above. For example:

```
*r500:700;10(RET)
```

This command deletes lines 500 through 700 and causes SOS to enter Input mode; SOS numbers the input lines you type in increments of 10 until you press (ESC) to return to Edit mode.

2.3.13 Editing with the Find and Substitute Commands

The F (Find) and S (Substitute) commands give you an easy way to locate and change lines in a file when you do not know the line numbers. Use the F command to request SOS to locate a particular character string. The character string can be as long as 200 characters. You must terminate the string with (ESC) and then use (RET) to terminate the command line.

When SOS searches for a string, it begins the search at the line following the current line and searches toward the end of the file. For example:

```
*fpersimmon(ESC)(RET)
```

SOS searches for the string "persimmon" beginning with the line following the current line. When it locates the string, it displays the line, then issues the Edit mode prompt:

```
01300   My little persimmon seed.
*
```

If it does not find the string, it issues the message:

```
String not found, search failed
*
```

In this case, the current line remains at its original position. If you want SOS to search the lines before the current line in the file, position the current line at the beginning of the file and repeat the F command, as shown below:

```
*P^(RET)
00100   This is the first line
*f(RET)
00300   Chokeberries, Persimmon Seeds, Mangoes.
```

SOS always remembers the last Find command string, so you do not need to retype the character string to repeat the command. Note in the above example that the occurrence of the requested string (persimmon) begins with an uppercase P. The Find command does not distinguish between uppercase and lowercase letters unless you use the Exact option, as shown below:

```
*fpersimmon(ESC),e(RET)
01300   My little persimmon seed.
```

The S (Substitute) command substitutes one character string for another. The following lines show an example of an S command:

```
*p1300(RET)
01300   My little persimmon seed.
*spersimmon seed(ESC)chickadee(ESC)(RET)
01300   My little chickadee.
*
```

After locating and displaying line 1300, the S command requests SOS to change the string "persimmon seed" to "chickadee." The (ESC) key delimits each of the character string arguments. SOS displays the line with the change.

In the above example, the S command is issued when the line containing the specified string is the current line. If the string you want to change is not on the current line, SOS begins searching for the string (as it would for a Find command). When it locates the string, it performs the substitution and prints the line.

The Substitute command changes all occurrences of a string on a line. For example:

```
*p300(RET)
00300   Chokeberries, Persimmon Seeds, Mangoes.
*s,(ESC) and(ESC)(RET)
00300   Chokeberries and Persimmon Seeds and Mangoes.
```

This Substitute command substitutes the string "and" for each comma (,) in the line.

2.3.14 For More Information

The *VAX-11 Text Editing Reference Manual* contains a complete description of how to use SOS. In that manual, you will find additional examples of the commands presented in this chapter, and information on many more commands and special editing techniques.

Another editor used with the VAX/VMS operating system is named EDT. For information on the features and use of EDT, refer to the *VAX-11 EDT Editor Reference Manual.*

Chapter 3
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 executable image
- Executing and debugging the program

These steps are common to all the languages available on the VAX/VMS operating system.

3.1 Creating The Program

When you write a program, you must create a file that contains the program source statements. If you are an interactive user, you use the editor to create the source program. If you are a batch user, you create the file on punched cards, and submit the cards in your batch job.

3.2 Compiling or Assembling the Program

You use a DCL command to invoke a language processor, either a **compiler** or an **assembler**. There is a command to invoke each processor.

Command	**Invokes**
BASIC	VAX-11 BASIC compiler
BLISS	VAX-11 BLISS-32 compiler
COBOL/C74	VAX-11 COBOL-74 compiler
FORTRAN	VAX-11 FORTRAN compiler
MACRO	VAX-11 MACRO assembler
PASCAL	VAX-11 PASCAL compiler

The language processors check your source program for syntax and programming errors, and then translate your input source file into a binary form that can be interpreted by the computer. The translated code — that is, the **object module** — is written into a file called an object module file.

3.3 Linking the Object Module

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

The LINK command invokes the 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 linker creates an **image**, which is a file containing your program in an executable format.

3.4 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 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 compilation, linking, and running steps to test the result. Figure 3–1 illustrates these steps in program development.

The remaining sections in this chapter show the commands to create, compile, and execute a program in these programming languages:

- VAX–11 FORTRAN
- VAX–11 MACRO

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.

Each section also contains a sample program. 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 is a list of additional documentation that you can consult for further information about the language, programming considerations, and other tools provided by VAX/VMS for program development.

If you are a high-level language programmer, you can scan the section on FORTRAN program development to learn the concepts of program development. For pointers to where you can find more information, see the *VAX–11 Information Directory and Index*.

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.

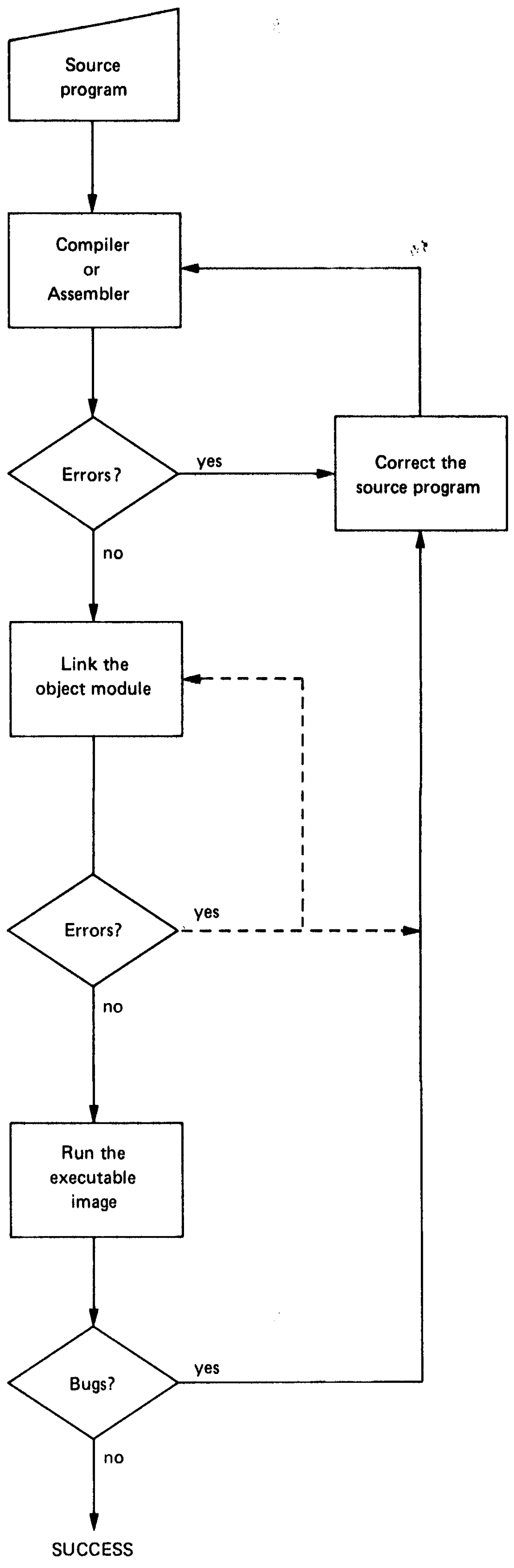


Figure 3-1: Steps in Program Development

3.5 A FORTRAN Program

The steps required to prepare a VAX-11 FORTRAN[1] program to run on VAX/VMS are illustrated in Figure 3-2. Figure 3-2 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.

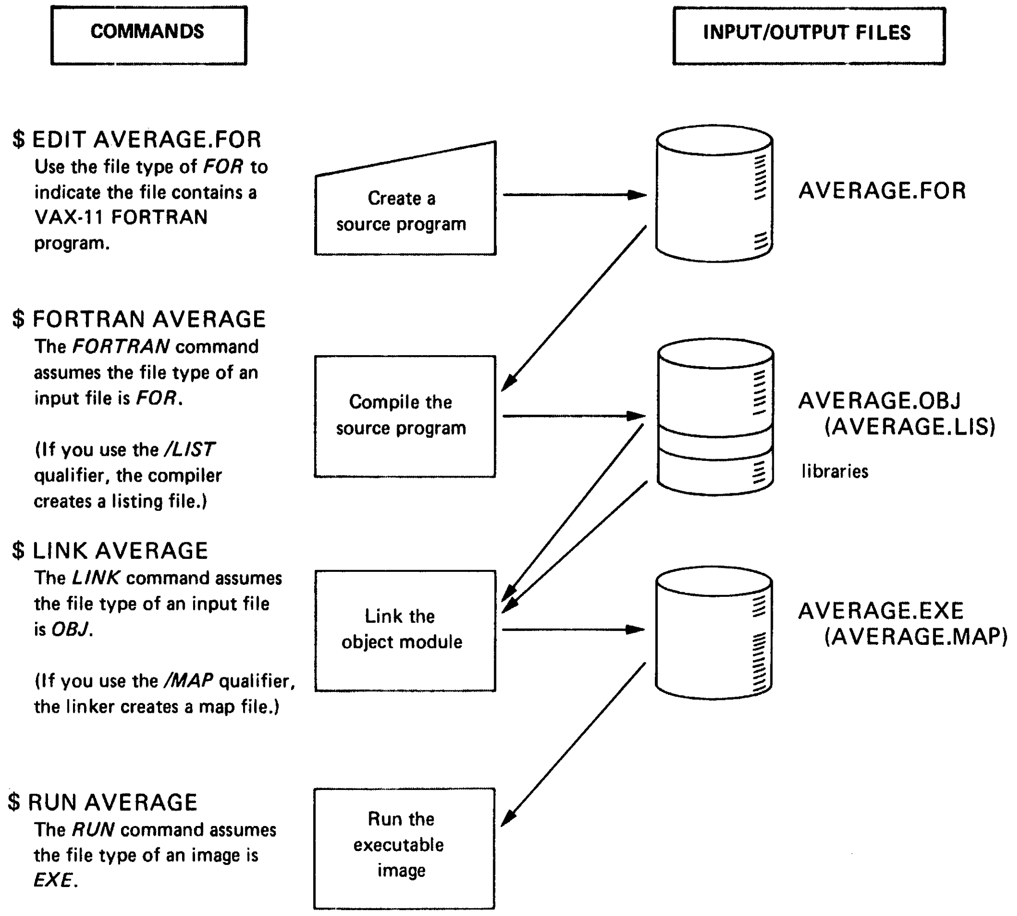

COMMANDS
INPUT/OUTPUT FILES
$ EDIT AVERAGE.FOR
Use the file type of *FOR* to indicate the file contains a VAX-11 FORTRAN program.
Create a source program
AVERAGE.FOR
$ FORTRAN AVERAGE
The *FORTRAN* command assumes the file type of an input file is *FOR*.
(If you use the */LIST* qualifier, the compiler 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

Figure 3-2: Commands for FORTRAN Program Development

3.5.1 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:DBA2:[MALCOLM]AVERAGE.FOR;1
00100
```

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

The line number prompt indicates that SOS is ready to accept input lines.

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. The terminal has internal tab settings at every eight character positions.

```
                PROGRAM AVERAGE

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

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

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

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

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

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

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

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

                STOP
                END
```

The program AVERAGE reads and writes lines to the current input and output devices; it prompts for 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.

3.5.2 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:

```
$ edit average.for(ESC)
Edit:DBA2[MALCOLM]AVERAGE.FOR;1
*
```

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

```
*fi10(ESC)(RET)
1500   20    FORMAT I10
*r.*(RET)
1500   20    FORMAT (I10)(ESC)
*
```

The Find command locates the FORMAT statement that is in error and the Replace command replaces the line. If you had more than one error in your source file, correct these errors, too.

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

```
*e(RET)
[DBA2:[MALCOLM]AVERAGE.FOR;2]
$
```

Notice that SOS 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.

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.

3.5.3 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.

3.5.4 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)
Edit:DBA2:[MALCOLM]AVERAGE.FOR;2
*fTOTAL/ (ESC)(RET)
2300 40      AVERAG = TOTAL/N
*r.(RET)
2300 40      AVERAG = TOTAL/(N-1)(ESC)
*e(RET)
[DBA2:[MALCOLM]AVERAGE.FOR;3]
$
```

The Find command locates the string TOTAL/ and the Replace command replaces the line. The End command writes a new version of the file onto disk.

Now, repeat the compiling, linking, and running:

```
$ 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.

3.5.5 Debugging a FORTRAN 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*.

3.6 A MACRO Program

The steps required to prepare a VAX-11 MACRO[1] program to run under VAX/VMS are illustrated in Figure 3-3. Figure 3-3 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.

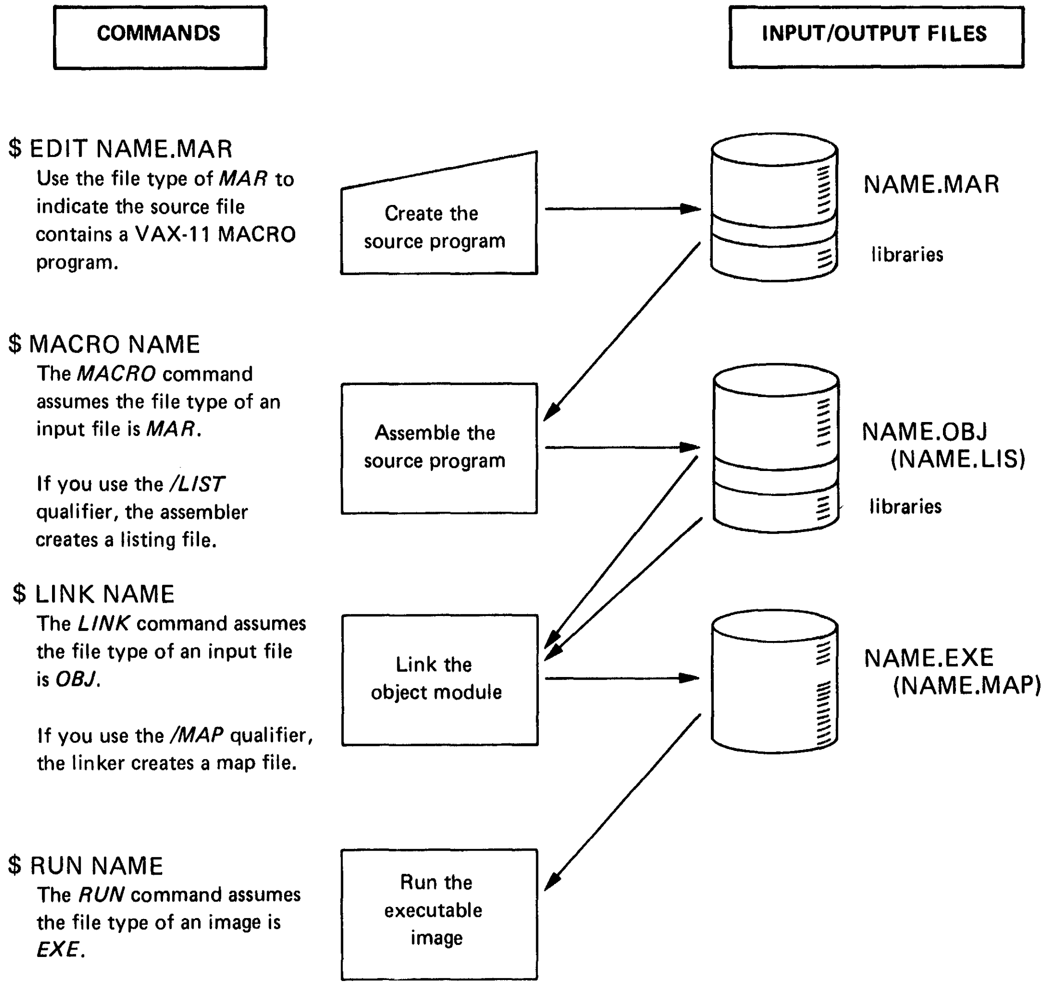

COMMANDS
INPUT/OUTPUT FILES
$ EDIT NAME.MAR
Use the file type of MAR to indicate the source file contains a VAX-11 MACRO program.
Create the source program
NAME.MAR
libraries
$ MACRO NAME
The MACRO command assumes the file type of an input file is MAR.
If you use the /LIST qualifier, the assembler creates 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

Figure 3-3: Commands for MACRO Program Development

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

3.6.1 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:DBA2:[MALCOLM]NAME.MAR;1
00100
```

The line number prompt indicates that SOS is 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. The terminal has internal tab settings at every eight character positions.

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.

```
 100            .TITLE NAME
 200            .IDENT /01/
 300            .PSECT  RWDATA,WRT,NOEXE
 400
 500    ; DEFINE CONTROL BLOCKS FOR TERMINAL INPUT AND OUTPUT
 600
 700    TRMFAB: $FAB      FNM=SYS$INPUT,RAT=CR,FAC=<GET,PUT> ;FAB FOR TERMINAL
 800
 900    TRMRAB: $RAB      FAB=TRMFAB,UBF=BUFFER,USZ=BUFSIZ, -
1000                      ROP=PMT, PBF=PMSG1, PSZ=P1SIZ
1100
1200    BUFFER: .BLKB     132                         ; INPUT READ BUFFER
1300    BUFSIZ= .-BUFFER                              ; BUFFER LENGTH
1400
1500    PMSG1:  .ASCII    /ENTER YOUR NAME:           ; PROMPT MESSAGE
1600    P1SIZ=  .-PMSG1                               ; MESSAGE SIZE
1700
1800    OUTMSG: .ASCII    /HELLO, YOUR NAME IS/       ; OUTPUT MESSAGE
1900    OUTBUF: .BLKB     30                          ; MOVE NAME HERE
2000    OUTLEN: .LONG     OUTBUF-OUTMSG
2100    MSGSIZ: .BLKL     1                           ; ADD LENGTHS HERE
2200
2300            .PSECT    NAME,EXE,NOWRT
2400            .ENTRY    BEGIN,0                     ; ENTRY MASK
2500
2600            $OPEN     FAB=TRMFAB                  ; OPEN TERMINAL FILE
2700            BLBC      R0,ERROR                    ; EXIT IF ERROR
2800            $CONNECT  RAB=TRMRAB                  ; ESTABLISH RAB
2900            BLBC      R0,ERROR                    ; EXIT IF ERROR
3000
3100            $GET      RAB=TRMRAB                  ; ISSUE PROMPT
3200            BLBC      R0,ERROR                    ; EXIT IF ERROR
3300
3400    ; MOVE NAME ENTERED INTO OUTPUT MESSAGE, AND FIX UP LENGTH
3500
3600            MOVC3     TRMRAB+RAB$W_RSZ,BUFFER,OUTBUF
3700            MOVZWL    TRMRAB+RAB$W_RSZ,MSGSIZ
3800            ADDL      MSGSIZ,OUTLEN
3900
4000    ; AFTER CONSTRUCTING OUTPUT MESSAGE, OUTPUT IT
4100
4200            MOVAL     OUTMSG,TRMRAB+RAB$L_RBF     ; UPDATE RAB: ADDRESS
4300            MOVW      MSGSIZ,TRMRAB+RAB$W_RSZ     ; UPDATE RAB: SIZE
4400            $PUT      RAB=TRMRAB
4500            BLBC      R0,ERROR                    ; EXIT IF ERROR
4600
4700    ; ALL DONE, CLOSE THE FILE
4800
4900            $CLOSE    FAB=TRMFAB
5000
5100    ERROR:
5200
5300            RET
5400            .END      BEGIN
```

3.6.2 The MACRO Command

When you enter the MACRO command from the terminal, 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 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:
1500(1)
```

This message indicates that the ASCII string argument coded on line 1500 is incorrect; you must terminate the string with a slash (/) character.

To correct the error, edit the source file:

```
$ edit name.mar(RET)
Edit:DBA2:[MALCOLM]NAME.MAR;1
```

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

```
*p1500(RET)
01500 PMSG1:  .ASCII /ENTER YOUR NAME:    ;PROMPT MESSAGE
*r.(RET)
01500 PMSG1:  .ASCII /ENTER YOUR NAME:/   ;PROMPT MESSAGE(ESC)
*
```

The Print command prints the line (1500) that was in error; the Replace command replaces the line. If you had more than one error in your source file, correct these errors, too.

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

```
* e RET
[DBA2:[MALCOLM]NAME.MAR;2]
$
```

Notice that SOS 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.

3.6.3 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 RMS routines used.

3.6.4 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 uses the string you entered to create an output string and outputs it. A typical run of this program might appear as follows:

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

As you can see, the program is writing only the first 11 characters of the output message. If you examine the listing, you can see that on line 4300 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)
Edit:DBA2:[MALCOLM]NAME.MAR;2
*p4300(RET)
04300     MOVW    MSGSIZ,TRMRAB+RAB$W_RSZ ;UPDATE RAB:SIZE
*r.(RET)
04300     MOVW    OUTLEN,TRMRAB+RAB$W_RSZ ;UPDATE RAB:SIZE(ESC)
*e(RET)
[DBA2:[MALCOLM]NAME.MAR;3]
$
```

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.

3.6.5 Debugging a MACRO 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.*

3.7 For More Information

The two programs and the command examples presented in this section have shown only the simplest cases, using defaults for getting a program to run. As you learn to use the MACRO assembler or FORTRAN compiler on VAX/VMS you will want to create more complex programs. The VAX/VMS operating system provides many programming capabilities beyond those of the VAX-11 MACRO assembler and instruction set, or the VAX-11 FORTRAN language.

Some of the manuals you will find useful are described below.

- The *VAX-11 FORTRAN Language Reference Manual* describes the features and syntax of the FORTRAN language.
- The *VAX-11 FORTRAN User's Guide* provides details on how to use FORTRAN on the VAX/VMS operating system.
- The *VAX-11 MACRO Language Reference Manual* describes the features and syntax of the MACRO langauge.
- The *VAX-11 MACRO User's Guide* provides details on how to use the VAX-11 MACRO assembler.
- The *VAX/VMS Command Language User's Guide* contains reference information for all the commands that have been used in the examples in this section.
- 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 *Introduction to VAX-11 Record Management Services* describes the file formats used in VAX/VMS, and the *VAX-11 Record Management Services Reference Manual* and the *VAX-11 Record Management Services User's Guide* describe the macros that can be used to create, read, and update files.

As you write new programs, you may want to use the VAX-11 Common Run-Time Procedure Library, which contains procedures you can call from your programs. For example, there are library procedures that:

- Perform common mathematical functions, such as computing the sine of an angle
- Manipulate character string data for input and output routines
- Convert data from one type to another, for example, changing a numeric ASCII string to a binary value or a floating-point number to scientific notation

A procedure in the Run-Time Library is automatically located by the linker when you link a program that calls it. Moreover, these procedures can be called and used by any language supported by the VAX/VMS operating system. For details on how to use these procedures, see the *VAX-11 Run-Time Library Reference Manual.*

Additional programming capabilities are available through the VAX/VMS system services. System services are operating system procedures you can use to synchronize program events using timers, or to establish communications among separate programs. For details on how to use these system services, and information on other kinds of services, see the *VAX/VMS System Services Reference Manual.*

Chapter 4
Files: Commands to Manipulate Files

The two previous chapters showed how to create files using the editor and illustrated the files that are required to develop executable programs.

This chapter describes in more detail how you can use the command language to manipulate files, including files that reside on disks other than your default disk, or on devices other than disks. This chapter contains examples of:

- Identifying files
- Listing files in a directory
- Creating and using subdirectories
- Displaying files at your terminal
- Deleting files
- Copying files
- Printing files
- Renaming files

4.1 Identifying Files

A complete file specification contains all the information the system needs to know 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.

4.1.1 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**. If your system is a network node, you can gain access to a file located at another node by including the node name in the file specification.

4.1.2 Devices

A **device name** identifies the physical device on which a file is stored. A device name has three fields:

1. The device type. Each hardware device has a unique name that identifies it. For example, an RP06 disk is DB and a TE16 magnetic tape is MT.
2. A controller designator. The controller designator identifies the hardware controller to which the device is attached.
3. The unit number. The unit number 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 assumes that the file is on your default disk device.

4.1.3 Directories

If you type a file specification and omit a directory name, the system assumes the file is in your default directory. However, you can access 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 permitted other users to access files in the directory. The system protects all users' files, by default. 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.

Files can also be cataloged in subdirectories. A **subdirectory** is a file (cataloged in a directory) that lists 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 and how to create them are described in more detail in Section 4.3.

4.1.4 Changing Defaults

The SET DEFAULT command allows you to change your default disk and/or default directory during a terminal session. 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 with DCL commands or with your own programs. (You must, of course, have access to a directory on the disk.) The changes you make with the SET DEFAULT command remain in effect until you issue another SET DEFAULT command, or until you log out.

4.1.5 Version Numbers

A version number is a decimal number (ranging from 1 to 32767) that the system assigns to a file when the file is created or updated. When you initially create a file — for example, with the editor — or when you first compile or assemble a source program to create an object module file, the system assigns the file a version number of 1. Subsequently, when you update a file or create additional versions of it, the system automatically increments the version number.

Thus, if you edit a source file a few times and compile the program, you have several different versions of the files. For example:

Command	**Output File**
`edit test.for`	test.for;1
`fortran test`	test.obj;1
`edit test.for`	test.for;2
`fortran test`	test.obj;2

You rarely need to specify a 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 (.).

4.2 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 DBA1:[CRAMER] 1

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

Total of 6 files. 3
```

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

1. The disk and directory name.
2. The file names, file types, and version numbers of each file in the directory.
3. The total number of files in the directory.

4.2.1 Listing Information about Specific Files

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

```
$ directory average.for;*(RET)

DIRECTORY DBA1:[CRAMER]

AVERAGE.FOR;2    AVERAGE.FOR;1

Total of 2 files.
```

The asterisk in the above command line is a **wild card character**. Whenever it appears in a file specification, it indicates that all files that satisfy the explicit parts of the file specification are to be located. As another example:

```
$ directory *.obj(RET)
```

This command results in a display of all files in the default directory that have file types of OBJ. For example:

```
DIRECTORY DBA1:[CRAMER]

AVERAGE.OBJ;2    AVERAGE.OBJ;1

Total of 2 files.
```

4.3 Creating and Using Subdirectories

Normally, the system manager provides each system user with only one directory to catalog and maintain files. If you are a frequent user of the system and work on several applications, you will find it convenient to organize a directory into several directory files. You can create subdirectories in any directory in which you can create files. Each directory can list files that have a common use.

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. For example, you could copy files from a default directory to the [MALCOLM.TESTFILES] subdirectory using the copy command:

```
$ copy newfile.* [malcolm.testfiles](RET)
```

In fact, there are no files in a subdirectory until you either create them there or copy them from some other directory.

To establish [MALCOLM.TESTFILES] as your default directory, use the SET DEFAULT command:

```
$ set default [malcolm.testfiles](RET)
$ edit newfile.tst(RET)
Input:DBA1:[MALCOLM.TESTFILES]NEWFILE.TST;1
```

The SET DEFAULT command requests the system to use the subdirectory as the default directory to search for input files or to create output files. The EDIT command invokes the SOS editor. SOS displays the subdirectory name in the file specification of the file NEWFILE.TST.

To return to your established default directory, reissue the SET DEFAULT command:

```
$ set default [malcolm]RET
```

You can change the default directory as many times during a terminal session as you want. Use the SHOW DEFAULT command to determine, at any time, the current directory.

4.4 Displaying Files at Your Terminal

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

```
$ type sostest.datRET
This is the first line
of a file to try out SOS.
Chokeberries, Persimmon Seeds, Mangoes.
.
.
.
```

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

CTRL/S suspends the output 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 and the system prompts you to enter another command.

4.5 Deleting Files

Quite often, as you develop and update programs you end up with many versions of source files, object modules, and program images. 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. 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 two early versions of the same data file

4.5.1 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 on the DELETE command would be tedious.

The PURGE command allows you to delete all but the most recent version(s) of a file. For example:

```
$ purge average.for(RET)
```

The 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.

4.6 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 other directories, 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.

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 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.*

4.7 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 printer file is preceded by a header page describing the file so you can identify your 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 job. By referring to this number, you can determine whether the file was printed by using the SHOW QUEUE command or request that it not be printed by using the DELETE/ENTRY command.

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. These characteristics can be changed for a file that has not yet been printed by using the SET QUEUE/ENTRY command.

4.8 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 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.

4.9 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.

Chapter 5
Logical Names: Files for Program Input/Output

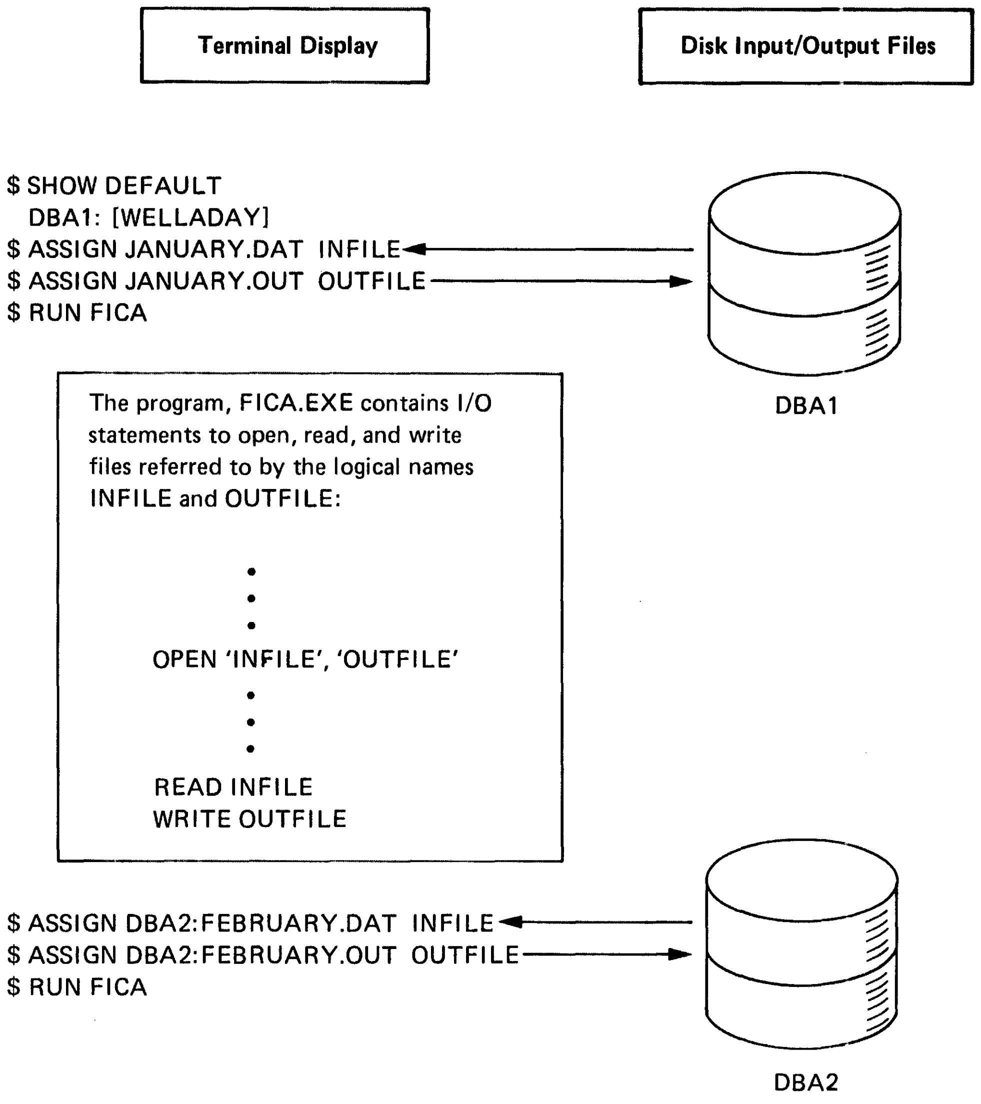
When you design programs to read and write data, you can code the programs to read or write different files each time you run them. This is called device and file independence.

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.



```
$ SHOW DEFAULT
  DBA1: [WELLADAY]
$ ASSIGN JANUARY.DAT  INFILE
$ ASSIGN JANUARY.OUT  OUTFILE
$ RUN FICA
```

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

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

```
$ ASSIGN DBA2:FEBRUARY.DAT  INFILE
$ ASSIGN DBA2:FEBRUARY.OUT  OUTFILE
$ RUN FICA
```

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 command lines 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 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.

5.3.4 For More Information

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.

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.

5.4.1 For More Information

For details on how to use RMS macros, see the *VAX-11 Record Management Services User's Guide* and *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

The command lines used as examples in this book will acquaint you with some frequently used sequences of commands and demonstrate basic uses of the command language.

As you continue to use the command language, however, 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 specialize the command language for your individual needs. For example, you can:

- Use assignment statements to 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 Assignment Statements

Assignment statements equate character strings, arithmetic values, or expressions to symbolic names. Assignment statements and the symbols they create have many specialized uses, particularly in command procedures.

Symbolic names for character strings are defined as shown in the following example:

```
$ time := show time(RET)
```

TIME is a symbolic name, and the string SHOW TIME is the value of the symbol.

Subsequently, you can enter the word TIME as if it were a system command, as follows:

```
$ time RET
  18-JAN-1978 11:55:40
```

The system substitutes the string SHOW TIME for the symbol TIME and executes the command SHOW TIME. The substitution is automatic because the word TIME is the first word in the command line.

You can also define symbols for use as synonyms for system commands. 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.dat RET
```

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

6.1.1 Symbol Concatenation

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

```
$ pquals := /copies=2/forms=4/noburst RET
$ print report.dat'pquals' RET
```

The assignment statement equates the symbol name PQUALS to a list of qualifiers for the PRINT command. When the PRINT command is issued, the symbol PQUALS is enclosed in apostrophes. The system performs the substitution and executes the command:

```
PRINT REPORT.DAT/COPIES=2/FORMS=4/NOBURST
```

6.2 Command Procedures

Command procedures are files that contain lines and, in some cases, data to be used as command or program input.

The default file type for a command procedure is COM. For example, a procedure named AVERAGE.COM might look like the following:

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

Every line in this command procedure begins with a dollar sign ($). This is required syntax for all command lines in a procedure that the system must process.

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 each command line in the file and executes it.

Note that you can execute this command procedure for the AVERAGE.FOR program created in Chapter 3, if you substitute the device name of your terminal for the TTB3: symbol in the ASSIGN command.

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 symbols, and letting the system substitute the symbol's value while it executes the command procedure.

The examples in the next two sections show two ways to write a generalized procedure to compile, link, and run any FORTRAN program.

6.2.1.1 Global Symbols — Suppose that the command procedure DOFOR.COM contains the lines:

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

Before you execute this command procedure, you must define a value — in this case, a file name — for the symbol PROGRAM. Use an 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**. Assignment statements with one equal sign are **local symbols**. Local symbols are recognized only at the command level at which they are defined (thus, a symbol defined at the interactive level is recognized only at the interactive level, and not within a command procedure). Global symbols are recognized and substituted in any command procedure you execute. Thus, when you enter the following command line, the system substitutes the value AVERAGE for the symbol PROGRAM on each command line:

```
$ @dofor RET
```

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

6.2.1.2 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 name when you execute the command procedure, 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 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 (file type of LIS).

Since you want to keep your disk file directory uncluttered, you want to purge these listings on a regular basis. 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)
```

Then, the PURGE command line is automatically executed before you are logged out of the system.

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

```
$ log :== @log(RET)
```

Then, when you type the command line

```
$ log(RET)
```

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

6.2.2.1 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. If this is the case, 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, it automatically executes it.

For example, a LOGIN.COM file might contain:

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

Note that all the assignment statements define global symbols. Otherwise, the symbols would only be recognized within the file LOGIN.COM, and would be useless to you.

Since command procedures can be executed from within other command procedures, you may want to place the assignment statements you use for system 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:

```
$ TIME :== SHOW TIME
$ LIST :== DIRECTORY
$ LOG :== @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

Command procedures can be submitted to the operating system as batch jobs. Batch jobs submitted by all users are placed in a job queue, and processed from the queue.

If you want to execute a command procedure that requires a lot of processing time, you can submit the procedure to the batch job queue by using the SUBMIT command. For example:

```
$ submit payroll(RET)
  Job 312 entered on queue SYS$BATCH
```

In this command, PAYROLL is the file name of a command procedure. The SUBMIT command assumes a file type of COM. The response from the SUBMIT command indicates that the job was successfully queued for processing and gives the job identification the system assigned to the job.

If you do not have access to a terminal, and if your installation has a card reader available for reading batch jobs, you can submit the command procedure on punched cards. The card deck must be preceded by $JOB and $PASSWORD cards that identify you to the system.

6.3.1 A Sample Batch Job

A batch job (submitted either from the terminal or through a card reader) can contain both commands and data. Figure 6-1 illustrates a sample batch job, as it would be submitted in a card reader. The same job submitted interactively for execution would not contain the $JOB or $PASSWORD cards.

6.3.2 Batch Job Output

When the system processes a batch job, it equates the logical name SYS$INPUT with the command input stream, whether a command procedure file or a deck of cards. The logical names SYS$OUTPUT and SYS$ERROR are equated to a batch job log file. The system creates this file in your default directory as it processes a batch job that you queue. When the job completes, the system queues the file for printing and deletes it after printing.

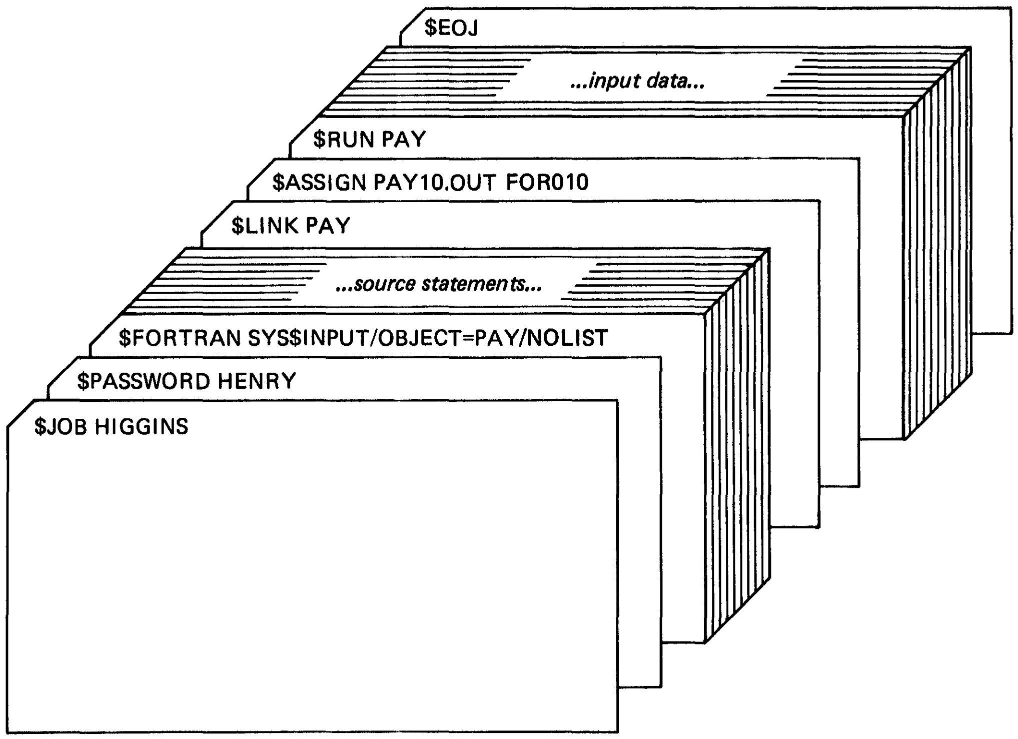


Figure 6-1: A Sample Batch Job for the Card Reader

Notes for Figure 6-1:

1. The JOB command specifies the user name of the user. HIGGINS, who is submitting the batch job and the PASSWORD command gives the password. These cards are the means the batch user uses to log in from a card reader.
2. The FORTRAN command specifies the input file using the logical name SYS$INPUT. All the cards following the card with the FORTRAN command, until the next card that begins with a dollar sign, are the source statements. The /OBJECT qualifier specifies the file name for the FORTRAN command to use when it creates the object module.
3. The LINK command links the object module just created. The ASSIGN command equates the file name PAY10.OUT to the logical name FOR010 so that when the program is run, the output file written to the logical unit 10 is written to the file PAY10.OUT in the user Higgins's current default directory.
4. The cards following the card with the RUN command contain the input for the program PAY. When the program reads from the logical device SYS$INPUT, the cards are read in and processed.
5. The EOJ command signals the end of 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
```

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.

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.

command

An instruction or request for the system (or a system program, such as the editor) 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 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.

current line

Pointer used by the editor to keep track of the position within a file being edited.

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

Assumed value supplied by the system when a command qualifier does not specifically override the normal command function; fields in a file specification that the system fills in when the specification is not complete.

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.

device name

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

directory

File listing files cataloged on a particular device for a user.

Edit mode

Mode of operation using the system editor in which every line that is typed at the terminal is interpreted as a command for the editor.

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 tranlates the logical name to its predefined equivalence name.

file

Collection of data, generally used to refer to data stored on a magnetic medium, such as a disk.

file name

A one- to nine-character name component of a file specification.

file specification

Unique indentification 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

A one- to three-character type component of a file specification. A file type generally describes the nature of a file, or how it is used, for example, FOR indicates a FORTRAN source program.

file version number

Numeric component of a file specification. As files are updated and changed, the file's version number is updated with each successive copy.

global symbol

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

image

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

Input mode

Mode of operation using the system editor in which every line that is typed at the terminal is accepted as input text for a file.

interactive

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

keyword

A command name, qualifier, or option. When users enter keywords, the keywords must be typed either verbatim or using a valid truncation.

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.

logout

The procedure that terminates an interactive user's communication with the operating system. The LOGOUT command executes the procedure and ends a terminal session.

node name

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.

parameter

Object of a command. A parameter can be a file specification or a keyword option. Or, a symbol value passed to a command procedure.

password

Protective keyword associated with a user name. A user logging into the system must supply the correct password before the system will allow access.

qualifier

Keyword that modifies the operation of a command. A qualifier is always preceded by a slash character (/).

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.

terminal session

Your activities on the system between the time you log in at a terminal and the time you log out.

user name

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

wild card character

A symbol used with many DCL commands in their file specifications. A wild card character, such as an asterisk (*), indicates that any item will satisfy the specification component.

VAX/VMS Primer
AA-D030B-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
- ☐ High-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 TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

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