Educational Services

digital™

VMS Utilities and Commands I
Student Workbook – Volume II

EY–3501E–SB–0002

Second Edition, December 1988

This document was prepared using VAX DOCUMENT, Version 1.0

CONTENTS

FIGURES

TABLES

MANAGING FILES

5.1 INTRODUCTION

File management on a VMS system involves moving files between devices, directories, and/or systems; protecting files from undesired manipulation; and maintaining and organizing collections of files in a directory.

The VMS system provides the following means to help manage files:

- Devices that store files.
- A file system that organizes, protects, and retrieves files stored on the system.
- Commands and utility programs that allow you to communicate with the devices and the system.

This module shows you how to organize and maintain a collection of files.

5.2 OBJECTIVES

To store and retrieve the many files used during daily operations, and to protect these files from unauthorized use, a user should be able to:

- Locate files in directories
- Locate directories in directory trees
- Locate directory trees on volumes
- Locate volumes on devices
- Display contents of files
- Add and remove files from a directory
- Locate files on tape volumes
- Specify devices that do not support files
- Protect files from access by unauthorized users

5.3 RESOURCES

- *Guide to VMS Files and Devices*
- *VMS DCL Dictionary*

5.4 NAMING A FILE

This section describes the necessary terminology for a user to effectively perform daily tasks on a system. The following sections give a more detailed explanation of devices, directories, file names, file types, and version numbers.

5.4.1 File Specifications

A *file* is a logically related collection of records. You name a file on a VMS system by giving it a *file specification*. The file specification is broken into five parts. Each part gives the system a different piece of information it needs to locate the file. The system distinguishes one part from another by the location of special characters called *delimiters* that you place within the file specification.

5.4.1.1 Use of Delimiters in a Local Disk File Specification

```
DEVICE:[DIRECTORY]FILENAME.TYPE;VERSION-NUMBER
      1     2             3    4
```

1	:	specifies the end of a device name
2	[and]	specifies the beginning and end of a directory name
3	.	specifies the beginning of a file type and end of a file name
4	;	specifies the beginning of a version number

Table 5–1 shows the parts of a file specification and their syntax.

Table 5–1 Syntax of a Local Disk File Specification

```
DBA0:[SMITH]MYFILE.DAT;7
```

Part	Reference	Rules of Naming	Example
Device	Storage device name	1 to 255 characters	DBA0:
Directory	Catalog of files	1 to 39 characters	[SMITH]
Name	Name of file	0 to 39 characters	MYFILE
Type	Kind of file	0 to 39 characters	DAT
Version	Unique number used to differentiate files with the same name and type	1 to 32767 (integer)	7

The following characters are allowed in directory names, file names, and file types:

- A through Z
- 0 through 9
- Underscore (_)
- Dollar sign ($)
- Hyphen (-)

The system interprets all alphabetic characters in file names and types as uppercase letters.

NOTE
DIGITAL systems use the dollar sign ($) in a number of system–wide variables. Therefore, to minimize confusion, it is recommended not to use the dollar sign in user–defined file specifications.

5.5 DEVICE SPECIFICATIONS

A *device specification* can consist of one of the following: a *logical device name*, a *physical device name*, or a *generic device name*. A logical device name is a synonym for a physical device name, usually established by the system manager. A physical device name refers to a specific physical device on the system. It has the following components:

- A *device code* indicating the type of device you want to use.

  (For reasons internal to VMS, the device code is not always the same as the first two characters of a device type. See the list of device types in Table 5–17 of Appendix A of this module.)

- A *controller character* indicating the controller to which the desired device is attached.

- A *unit number* indicating the relative location of the desired device among the devices on the particular controller.

Table 5–2 illustrates the components of a physical device name. Table 5–17 of Appendix A lists the device type and device code for the most commonly used physical devices.

Table 5–2 Naming a Device

Device Specification	Function	Value	Default Value
Device type code	Identifies device type	2–13 characters	None
Controller character	Names controller to which device is attached	One or more of the characters A–Z	A
Unit number	Names relative position on controller of desired unit	Decimal numbers from 0–65535	0
Device specification delimiter	Marks the end of the device specification	: (colon)	None

NOTE
Refer to Appendix A of this module for further information regarding devices.

5.5.1 Peripheral Devices

A typical VMS system may have a large number of *peripheral devices*. These devices are classified as either *mass storage devices* or *record–oriented devices*. Disk and tape drives are examples of mass storage devices, while terminals, printers, and card readers are examples of record–oriented devices. To use a peripheral device, you must describe its location to the operating system by giving a device specification. For example, LPA0: represents a particular line printer on the system.

5.5.2 Logical Names Used to Represent Device and File Specifications

The VMS system allows you and the system manager to define *logical names* to be used in place of part or all of a file specification. For example, in the file specification:

```
DBA0:[SMITH]MYFILE.DAT;7
```

you can equate the device name DBA0: to the logical device name DISK_USER, and then write the file specification as:

```
DISK_USER:[SMITH]MYFILE.DAT;7
```

By using the logical device name DISK_USER, the system manager achieves *device independence*. When desired, the system manager can move the file structure from the device named DBA0: to a free device, device DBA1: for example, and equate the logical device name DISK_USER to the device named DBA1:. Users can continue to specify files with the logical device name DISK_USER even after the device has changed.

This module uses logical names defined by the system manager only. User–defined logical names are discussed in the **Customizing the User Environment** module.

5.6 DIRECTORY STRUCTURE

All files stored on a disk are listed in a directory of some type. Each disk has one *master file directory (MFD)* that catalogs all *user file directories (UFDs)*. Your default directory is one of many UFDs. Each directory has a specific name that can be used in file specifications. The conventions for naming the MFD and UFDs are shown in Table 5–3.

5.6.1 The User File Directory (UFD)

Each UFD contains an alphabetical list of file names and pointers to files. It has a protection code that prevents other users from viewing or accessing its contents. A UFD is implemented by a disk file and has a file type of DIR.

Files are placed in your UFD in one of the following ways:

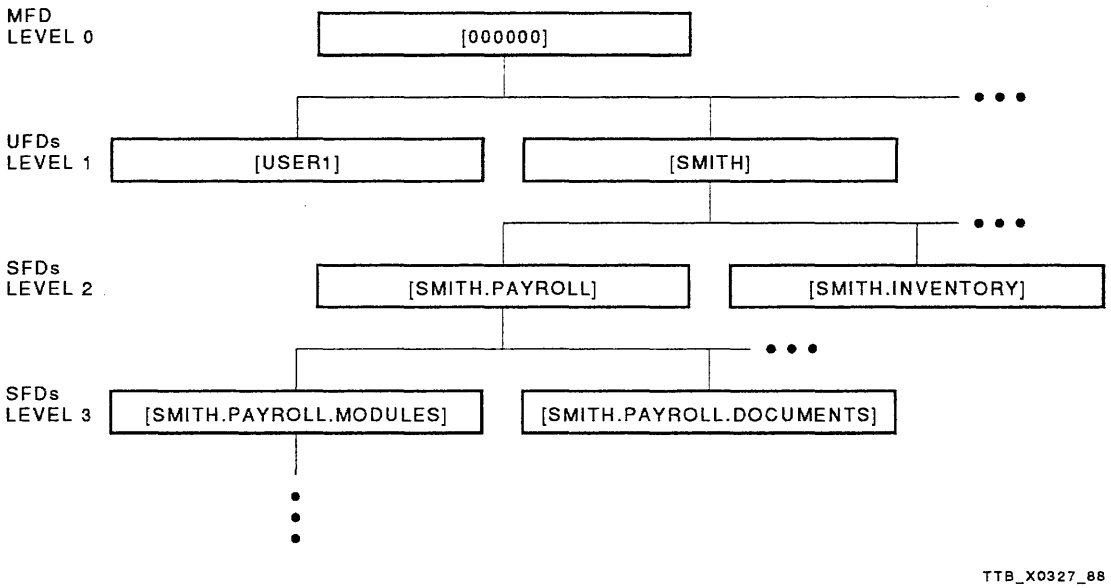
- You can create files in your UFD by using an editor such as EDT or EVE.
- Another user can place files in your directory.
- Files can be copied from another source.
- Some programs, when executed, can produce files as output.

5.6.2 Directory Names in the Hierarchy

Table 5–3 Directory Names

Directory Type	Example	Naming Convention
Master File Directory (MFD)	[000000]	Each disk contains one MFD, named [000000]. Given by the system.
User File Directory (UFD)	[SMITH]	The owner's user name is the Level 1 directory name in most instances.
Subdirectory (SFD)	[SMITH.PAYROLL]	You choose the names for the subdirectories you create.

Figure 5–1 illustrates Master File Directories and User File Directories in the hierarchy.



The names given in the rectangles are directory names.

Figure 5–1 Naming Directories

5.7 DEFAULTS FOR FILE SPECIFICATIONS

Each part of a file specification is called a *field.* The system supplies defaults for each field. To use the default value for a field, omit the value for the field. To override the default value for a field, supply a value for the field. Table 5–4 shows the parts of the file specification and the defaults for each part.

Table 5–4 File Specification Defaults

Part of File Specification	Default
Device	Device established at login by the system manager
Directory	Directory established at login by the system manager
Name	None
Type	Depends on the DCL command
Version	The highest version number

For example, the following file specification specifies the highest version of the file named MYFILE.DAT, catalogued in the directory named [SMITH] of the volume mounted on the disk drive named DBA0:.

```
DBA0:[SMITH]MYFILE.DAT;7
```

Note that the node name is not utilized when you access a file on your local node, and the version number of 7 signifies that the highest version of MYFILE.DAT is MYFILE.DAT;7.

5.7.1 Using Temporary Default Fields Within a Parameter

DCL establishes temporary file specification default fields within a *command parameter* that has more than one file specification. Each file specification within the parameter is used to establish temporary field defaults for subsequent file specifications. By taking these temporary defaults, the VMS system minimizes the typing you have to do. Temporary defaults are established for the following file specification fields: device name, directory name, file name, and file type. (Recall that the node name is not utilized when you access a file on your local node and that the version number always defaults to the highest version number.)

Because of the temporary defaults established by DCL, the results of the following two commands will be the same:

```
$ PRINT DBA0:[SMITH]FILE1.LIS,DBA0:[SMITH]FILE1.DAT,DBA0:[SMITH]FILE2.DAT
$ PRINT DBA0:[SMITH]FILE1,.DAT,FILE2
```

Note that the first **PRINT** command consists of file specifications that do not utilize the temporary defaults established by DCL. The second **PRINT** command utilizes the following DCL temporary defaults:

1. The file specification DBA0:[SMITH]FILE1 takes advantage of the default file type LIS, which is associated with the **PRINT** command. (Note that you do not place the period (.) after the file name FILE1. If you use the period delimiter, the system assumes the file has a null file type.)

2. The file specification .DAT utilizes the following temporary defaults:

   Device name – DBA0:
   Directory name – [SMITH]
   File name – FILE1

   (Note that the file type DAT is used to establish a temporary default.)

3. The file specification FILE2 utilizes the following temporary defaults:

   Device name – DBA0:
   Directory name – [SMITH]
   File type – DAT

Table 5–5 shows examples of operations you can use to manipulate files in your default directory.

Table 5–5 Manipulating Files in Your Default Directory

Operation	Comments and Examples
Copying a file	The COPY command creates a new file from an old file. `$ COPY DISK:[SMITH]OLD.TXT DISK:[SMITH]NEW.TXT` *(or)* `$ COPY OLD.TXT NEW.TXT`
Changing existing file name to new file name	The RENAME command changes the file name, file type, or version number of an existing file. `$ RENAME DISK:[SMITH]OLD.TXT DISK:[SMITH]NEW.TXT` *(or)* `$ RENAME OLD.TXT NEW.TXT`
Removing a file	The DELETE command removes a file. A specific version number must be used to remove a file. `$ DELETE DISK:[SMITH]MYFILE.TXT;2` *(or)* `$ DELETE MYFILE.TXT;2`
Removing files on an interactive basis	The /CONFIRM qualifier initiates a system prompt to confirm whether or not the file should be deleted. A "Yes" response deletes the file; a "No" response does not delete the file. `$ DELETE/CONFIRM DISK:[SMITH]MYFILE.TXT;*` *(or)* `$ DELETE/CONFIRM MYFILE.TXT;*` System prompts: `DISK:[SMITH]MYFILE.TXT;3 delete? [N]:` `DISK:[SMITH]MYFILE.TXT;2 delete? [N]:` `DISK:[SMITH]MYFILE.TXT;1 delete? [N]:`

Table 5–5 (Cont.) Manipulating Files in Your Default Directory

Operation	Comments and Examples
Removing all versions of all files except the latest version	The DCL command PURGE removes all but the highest numbered version of all files. `$ PURGE` The PURGE command used with a file specification removes all but the highest numbered version of that file. `$ PURGE MYFILE.TXT`
Appending one or more files to the end of another file	The APPEND command adds the contents of one or more files to the end of the specified output file. In this example, two files are being appended to the output file. `$ APPEND MYFILE.TXT,OLDFILE.TXT NEWFILE.TXT`
Searching files for all occurrences of the specified search string(s)	The SEARCH command searches one or more files for the specified string(s) and lists all lines containing the specified string(s). In this example, the search string "March 13" must be enclosed in quotation marks because it contains a space character. `$ SEARCH MYFILE.TXT "March 13"`
Comparing contents of two files and displaying differences	The DIFFERENCES command compares the contents of two files and creates a listing of the records that do not match. `$ DIFFERENCES MYFILE.TXT YOURFILE.TXT`
Controlling the listing output from differences	The /OUTPUT qualifier tells the system to send the listing of differences to a file. The default output is usually the terminal. `$ DIFFERENCES/OUTPUT=DIFF.TXT MYFILE.TXT YOURFILE.TXT`

5.8 FINDING FILES AND DETERMINING THEIR CHARACTERISTICS

As you create files and add them to your directory hierarchy, your collection grows rapidly in size and complexity. You find yourself increasing the use of files listed in other directories and stored on other devices. To work in such an environment, you should be able to locate particular files among the thousands stored on the disks and tapes of your system.

The operating system includes a powerful utility that helps you locate and display the characteristics of files. To invoke this utility, enter the **DIRECTORY** command, followed by one or more command qualifiers and a file specification. You use the **DIRECTORY** command to find files on the peripheral storage devices of your system. It can also display the contents of directories or the characteristics of files.

This section shows how to use the **DIRECTORY** command to find a file and determine its:

- Owner UIC
- Protection code
- Size
- Date of creation or modification

Table 5–6 shows the commands used to find and determine the characteristics of files. Examples 5–1 and 5–2 demonstrate the use of the commands discussed in Tables 5–5 and 5–6.

Table 5–6 Commands Used to Find and Determine the Characteristics of Files

Operation	Comments and Examples
Listing all files in your directory	The DIRECTORY command lists all files and information about them in your directory. `$ DIRECTORY`
Checking for a unique file in your directory	The file specification must be included to obtain information concerning a particular file in your directory. `$ DIRECTORY MYFILE.TXT`
Obtaining all information about a particular file in your directory	The /FULL qualifier overrides the default directory display, which is BRIEF. Omit the file specification to obtain full information about all files in your directory. `$ DIRECTORY/FULL MYFILE.TXT` *(or)* `$ DIRECTORY/FULL`
Determining the size of files in your directory	The /SIZE qualifier lists the size of files in 512–byte blocks used. The /SIZE=ALL qualifier lists the size of files both in blocks used and blocks allocated by the system. `$ DIRECTORY/SIZE MYFILE.TXT` *(or)* `$ DIRECTORY/SIZE=ALL MYFILE.TXT`

Table 5–6 (Cont.) Commands Used to Find and Determine the Characteristics of Files

Operation	Comments and Examples
Finding files created or modified before or after a specified time	The /BEFORE= qualifier selects those files that are dated before the specified time. The /SINCE= qualifier selects those files that are dated after the specified time. Both the /BEFORE and /SINCE qualifiers can also use the keywords YESTERDAY, TODAY, and TOMORROW. The /CREATED qualifier is the default. It selects files based on their creation date. The /MODIFIED qualifier selects files based on the dates they were modified. `$ DIRECTORY/BEFORE=09:00/CREATED MYFILE.TXT` *(or)* `$ DIRECTORY/SINCE=YESTERDAY/MODIFIED OLD.RNO`
Determining the owner and protection of a file	The /OWNER qualifier determines if the owner's UIC will be displayed. The /PROTECTION qualifier determines if the protection of the file is displayed. `$ DIRECTORY/OWNER/PROTECTION MYFILE.TXT`

Example 5–1 illustrates some file manipulation commands.

```
1  $ DIRECTORY

   Directory DISK:[SMITH]

   CLASS.LIST;4          CLOCK.EXE;1          COLOR.COM;4          DEG.EXE;1
   JOE_EVE.TPU$SECTION;1                      MYFILE.TXT;3         MYFILE.TXT;2
   MYFILE.TXT;1          NOTE.COM             REMIND.EXE;1         REMLOG.EXE;1
   TRNG.PLAN;6           VT100.CLR;1

   Total of 13 files.

2  $ CREATE FILE1.TXT
   From the time when man first began making numerical calculations,
   he has been inventing devices to aid him in the handling of
   numbers. These devices have been particularly useful where the
   calculations have been repetitive. With the advent of the modern
   computer, more and more control has been given to the machine
   to reduce the repetition and thus the possibility of human error.
   CTRL/Z

   $ CREATE FILE2.TXT
   From the time when man first began making numerical calculations,
   he has been inventing devices to aid him in the handling of
   numbers. These devices have been particularly helpful when the
   calculations have been repetitive. With the advent of the modern
   computer, more and more control has been given to the machine
   to reduce the repetition and therefore the possibility of human error.
   CTRL/Z

3  $ APPEND/LOG MYFILE.TXT TRNG.PLAN
   %APPEND-S-APPENDED, DISK:[SMITH]MYFILE.TXT;1 appended to
   DISK:[SMITH]TRNG.PLAN;6 (6 records)

4  $ PURGE/LOG MYFILE.TXT
   %PURGE-I-FILPURG, DISK:[SMITH]MYFILE.TXT;2 deleted (4 blocks)
   %PURGE-I-FILPURG, DISK:[SMITH]MYFILE.TXT;1 deleted (4 blocks)

5  $ DELETE/LOG MYFILE.TXT;3
   %DELETE-I-FILDEL, DISK:[SMITH]MYFILE.TXT;3 deleted (3 blocks)
```

Example 5–1 Using VMS Commands to Maintain Your Default Directory

Notes on Example 5–1:

In Example 5–1, assume you are the user who issues the commands. Your default directory is [SMITH].

1 `$ DIRECTORY`

To list the files in your default directory use the **DIRECTORY** command. The first line of output displayed at your terminal, directory DISK:[SMITH], reports the name and location of your default directory. The remaining lines list the names of the files it catalogs.

2
```
$ CREATE FILE1.TXT
$ CREATE FILE2.TXT
```

You can create additional files in your directory at any time. This module only uses the **CREATE** command for cataloging text files in your directory file. You can use the DCL line–editing commands discussed in the **Getting Started** module to format each line you enter with the **CREATE** command. Pressing CTRL/Z ends file input.

Several text editor utilities are available on the VMS system. Text editors are discussed in the **Creating and Editing Text Files** module.

3 `$ APPEND/LOG MYFILE.TXT TRNG.PLAN`

To concatenate MYFILE.TXT with TRNG.PLAN, enter the **APPEND** command, followed by the name of the file you want to append (MYFILE.TXT), followed by the name of the file you want to append it to (TRNG.PLAN). This operation does not create any new files. The contents of TRNG.PLAN are modified to include the file MYFILE.TXT. The contents of MYFILE.TXT are unaffected. To instruct the system to display a message when it completes the operation, add the **/LOG** qualifier to the **APPEND** command.

4 `$ PURGE/LOG MYFILE.TXT`

Since your default directory contains multiple versions of MYFILE.TXT, you can delete all but the most recent version. To do so, enter the **PURGE** command. To display the name of each file the system deletes, append the **/LOG** qualifier.

5 `$ DELETE/LOG MYFILE.TXT;3`

You decide that you no longer need the contents of MYFILE.TXT. To delete the remaining MYFILE.TXT from your directory, enter the **DELETE** command. By specifying the version number 3, you delete only that version of the file. Any other versions of the file would remain in your directory.

Example 5–2 illustrates how to use VMS commands to compare two files and find their differences. In this example, you compare the files FILE1.TXT and FILE2.TXT that you created in Example 5–1.

```
1  $ TYPE FILE1.TXT
   From the time when man first began making numerical calculations,
   he has been inventing devices to aid him in the handling of
   numbers. These devices have been particularly useful where the
   calculations have been repetitive. With the advent of the modern
   computer, more and more control has been given to the machine
   to reduce the repetition and thus the possibility of human error.

   $ TYPE FILE2.TXT
   From the time when man first began making numerical calculations,
   he has been inventing devices to aid him in the handling of
   numbers. These devices have been particularly helpful when the
   calculations have been repetitive. With the advancement of the modern
   computer, greater control has been given to the machine
   to reduce the repetition and therefore the possibility of human error.

2  $ DIFFERENCES FILE1.TXT FILE2.TXT/CHANGE_BAR=:
   ************
   File DISK:[SMITH]FILE2.TXT;1
       1   From the time when man first began making numerical calculations,
       2   he has been inventing devices to aid him in the handling of
       3 : numbers. These devices have been particularly helpful when the
       4 : calculations have been repetitive. With the advancement of the modern
       5 : computer, greater control has been given to the machine
       6 : to reduce the repetition and therefore the possibility of human error.
   ************

   Number of difference sections found: 1
   Number of difference records found: 4

   DIFFERENCES /IGNORE=()-
       DISK:[SMITH]FILE1.TXT;1-
       DISK:[SMITH]FILE2.TXT;1/CHANGE_BAR=(":")
```

Example 5–2 Comparing Files

Notes on Example 5–2:

1
```
$ TYPE FILE1.TXT
$ TYPE FILE2.TXT
```

Before comparing FILE1.TXT with FILE2.TXT, you decide to display the contents at your terminal. To do so, enter the **TYPE** command.

2
```
$ DIFFERENCES FILE1.TXT FILE2.TXT/CHANGE_BAR=:
```

You want to compare FILE1.TXT with FILE2.TXT to determine how they differ. You have already displayed the contents of both files. You would like to display FILE2.TXT with the lines that differ marked in an appropriate way (with a colon, for example). To do this, enter the **DIFFERENCES** command followed by the specifications of the files you want to compare. To generate a marked listing of FILE2.TXT, specify that file last. Terminate the specification of the file with the **/CHANGE_BAR** qualifier. Following a heading, the system displays the contents of the second file at your terminal. A colon precedes each line that differs from the first file.

5.8.1 Using Wildcards in File Specifications

At times it is useful to specify a group of files that have parts of a file specification in common. For example, you may want to delete all the files of a given file type in your default directory. *Wildcards* allow this. Table 5–7 describes wildcard symbols that you can use in the name, type, and version fields of a file specification.

Wildcards are also used:

- To abbreviate a file specification
- To specifically match one character in file names or file types
- In conjunction with each other or separately

Table 5–7 Wildcards Used to Specify File Names, Types, and Versions

Symbol	Meaning
* Asterisk	Match 0–39 characters in a file name, file type, or version number
% Percent	Match exactly one character in a file name or file type

Example 5–3 illustrates the directory named [SMITH], which contains eight files. Table 5–8 selectively refers to these files, using wildcard characters.

```
Directory WORK2:[SMITH]

PAY.FOR;2           PAY.FOR;1           PAY1.FOR;1          PAY2.FOR;14
PAYOFF.FOR;3        PROBLEMS.TXT;4      REPORT.MEM;9        REPORT.RNO;6

Total of 8 files.
```

Example 5–3 A Sample Directory File

Table 5–8 Using Wildcards to Specify Files

Directory Specification	Description	Corresponding Files
`$ DIRECTORY PAY.FOR;*`	All versions of PAY.FOR	PAY.FOR;2 PAY.FOR;1
`$ DIRECTORY *.*;1`	All files with a version number of 1	PAY.FOR;1 PAY1.FOR;1
`$ DIRECTORY *.*;*`	All files, types, and versions[1]	All files in the directory
`$ DIRECTORY PAY%.FOR;*`	All versions of files with file types of FOR and file names beginning with PAY, followed by exactly one character	PAY1.FOR;1 PAY2.FOR;14
`$ DIRECTORY PAY*.*;*`	All files whose first three letters are PAY, including all file types and all versions	PAY1.FOR;1 PAY.FOR;2 PAY.FOR;1 PAYOFF.FOR;3 PAY2.FOR;14

[1]Issuing the DIRECTORY command with no qualifiers or wildcards lists all files, types, and versions by default.

5.9 ORGANIZING YOUR DIRECTORY STRUCTURE

In addition to your master file directory and user file directories, you can also have *subdirectories*. Subdirectories are used to better organize the directory structure, to protect files from accidental modification or loss, and to decrease the time for the system to find files.

Each UFD can have a maximum of seven levels of subdirectories below it. Each directory level has a level identifier of 1–39 characters that makes up part of the directory name.

Files are usually grouped by:

- Function (all command files)
- Application (all files for a given project)
- Type (all FORTRAN files)

Subdirectories can catalog other subdirectories as well as files. Table 5–9 shows the three levels of directories and conventions for naming them.

5.9.1 Directory Names in the Hierarchy

Table 5–9 Directory Names

Directory Type	Example	Naming Convention
Master File Directory (MFD)	[000000]	Each disk contains one MFD, named [000000], which is given by the system.
User File Directory (UFD)	[SMITH]	The owner's user name is usually the Level 1 directory name.
Subdirectory (SFD)	[SMITH.DOC]	The subdirectory name includes the directory name where it is created. Each subdirectory name is separated by a period.

The ability to create and maintain a subdirectory structure starting with your UFD is a powerful tool in organizing your files. To create and maintain a subdirectory hierarchy, you should be able to perform the following operations:

- Create subdirectories
- Move around in the directory hierarchy
- Display a hierarchy and its contents at your terminal
- Determine the name of your current default directory or subdirectory
- Move files from one directory to another
- Assign a protection code to a subdirectory
- Remove a subdirectory from the hierarchy

The following pages show you how to do this. You can also refer to Table 5–5 for more examples of how to manipulate files in your default directory.

5.9.2 Creating a Subdirectory

To create a subdirectory, use the **CREATE/DIRECTORY** command in the format:

$ CREATE/DIRECTORY [directory.subdirectory]

- The subdirectory name must be enclosed in brackets
- The subdirectory name includes the directory name where it is created
- Separate subdirectory names with a period
- The directory or subdirectory itself is a file
  - — The directory or subdirectory has a file type of DIR
  - — The version number of file type DIR is 1

Example:

```
$ CREATE/DIRECTORY/LOG [SMITH.DOC]
(System response:)
%CREATE-I-CREATED DISK:[SMITH.DOC] created
```

- The directory [SMITH] is a Level 1 directory
- The subdirectory [.DOC] is the next level below the directory [SMITH]
- The **/LOG** qualifier displays, at your terminal, the fact that the subdirectory was created

To create another subdirectory beneath the [.DOC] subdirectory, use the following command format:

$ CREATE/DIRECTORY [directory.subdirectory.subdirectory]

Example:

```
$ CREATE/DIRECTORY/LOG [SMITH.DOC.FORTRAN]
%CREATE-I-CREATED DISK:[SMITH.DOC.FORTRAN] created
```

- The subdirectory [.FORTRAN] is now listed under the subdirectory [SMITH.DOC]

Figure 5–2 illustrates four levels of directories in a hierarchy.

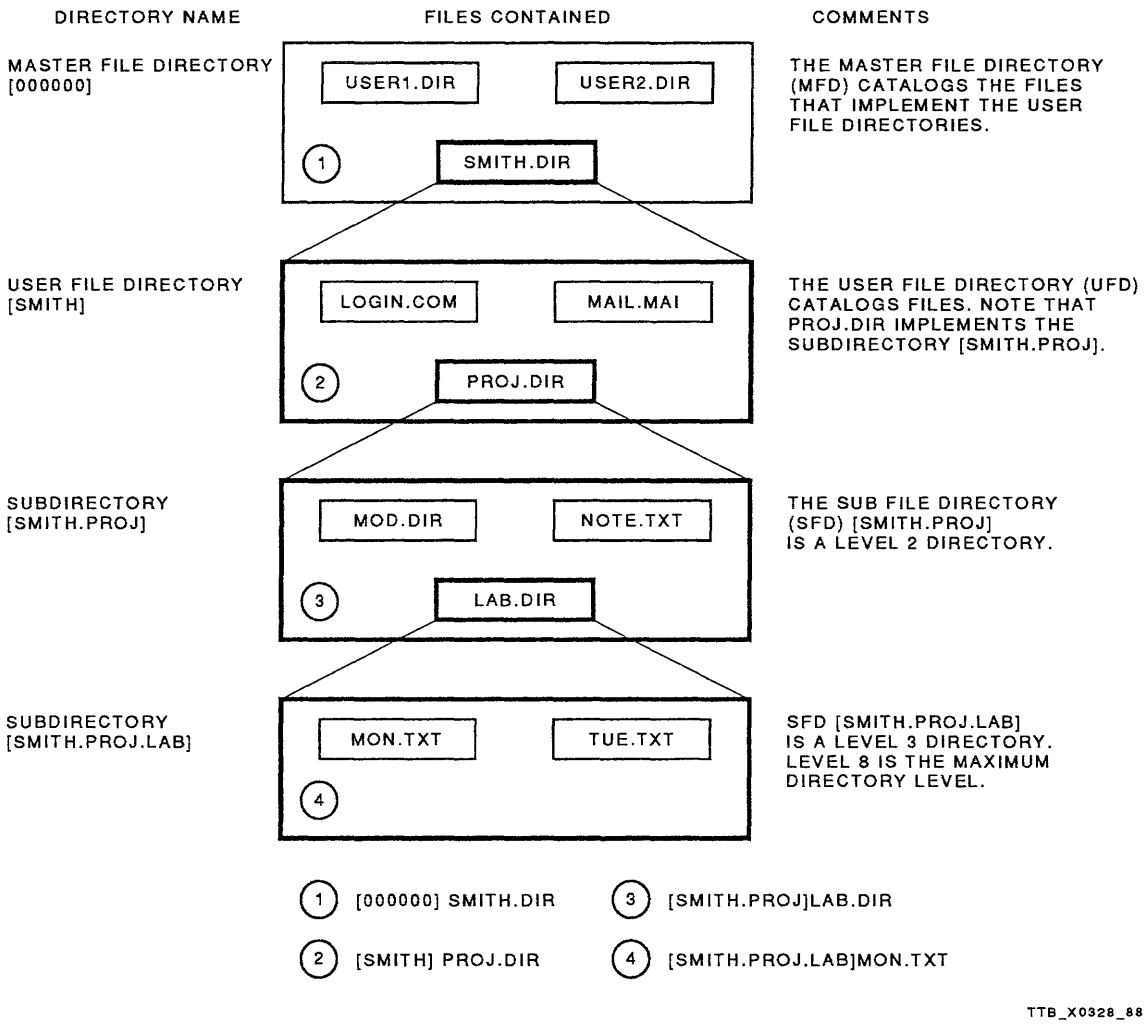


Figure 5–2 File Specification in the Directory Hierarchy

5.10 MOVING WITHIN A DIRECTORY HIERARCHY

There are three characters used to move within a directory hierarchy. They are:

- Hyphen (–)
- Period (.)
- Ellipsis (...)

The hyphen and period characters are normally used in conjunction with the **SET DEFAULT** command to move from your current directory to another directory or subdirectory.

The ellipsis character can be used with the **DIRECTORY** command to list files in a directory and all subdirectories beneath it (see Example 5–4). Table 5–10 describes these three characters.

Table 5–10 Characters Used to Specify Directories

Symbol	Meaning
– (hyphen)	Move one level up in directory hierarchy
. (period)	Move one level down in directory hierarchy (MUST be followed by a subdirectory name)
... (ellipsis)	Use current directory and all directories below it

5.10.1 Using the SET DEFAULT Command

The DCL command **SET DEFAULT** changes the default device and/or the directory name for your current process.

- A physical device name MUST be terminated with a colon (:)
- A directory or subdirectory name MUST be enclosed in square brackets

The syntax for the **SET DEFAULT** command is:

$ SET DEFAULT device-name:[directory-name]
or
$ SET DEFAULT [directory-name.subdirectory-name]

Examples of its use:

```
$ SET DEFAULT [SMITH.DOC]
$ SET DEFAULT DISK2:[SMITH.FOR]
```

In the first example, the device name remains the same, and in the second example, the device name and directory name change.

5.10.2 Using the SHOW DEFAULT Command

The DCL command **SHOW DEFAULT** displays your current default device and directory names.

Examples:

```
$ SHOW DEFAULT
  DISK:[SMITH]
$ SET DEFAULT [SMITH.DOC]
$ SHOW DEFAULT
  DISK:[SMITH.DOC]
$ SET DEFAULT [-]
$ SHOW DEFAULT
  DISK:[SMITH]
```

5.10.3 Using the COPY and RENAME Commands

At times you will need to move files from one directory to another. For example, you may want to catalogue the files in your default directory by job or type. You could copy all the files with a BAS extension into a subdirectory called BASIC (assuming the BASIC subdirectory already exists). You would do this by using the **COPY** command with the directory and file specifications you want to copy from and to. If your default directory name is SMITH, the command would look like this:

```
$ COPY [SMITH]*.BAS [SMITH.BASIC]*.*;*
```

Table 5–11 gives additional examples of moving files from one directory to another.

Table 5–11 Commands to Move Files Within a Directory Hierarchy

Operation	Comments
Move files from one directory or subdirectory to another	Moves the most recent version of all files with the file type FOR from the directory [SMITH] to the subdirectory [SMITH.FORTRAN] `$ RENAME [SMITH]*.FOR [SMITH.FORTRAN]*.FOR`
Copy files from one directory or subdirectory to another	Copies all versions of files with the file type PAS from the [SMITH.UTLCOM] subdirectory to the [SMITH.UTLCOM.PASCAL] subdirectory `$ COPY [SMITH.UTLCOM]*.PAS;* -` `  [SMITH.UTLCOM.PASCAL]*.*;*`

Example 5–4 shows you how to create two levels of subdirectories under the UFD [SMITH] and how to move files into one of the subdirectories. See the notes to the example for a detailed explanation.

```
1  $ SHOW DEFAULT
     DISK:[SMITH]
2  $ CREATE/DIRECTORY/LOG [SMITH.COM]
   %CREATE-I-CREATED, DISK:[SMITH.COM] created
   $ CREATE/DIRECTORY/LOG [SMITH.UTLCOM]
   %CREATE-I-CREATED, DISK:[SMITH.UTLCOM] created
   $ CREATE/DIRECTORY/LOG [.UTLCOM.FIL]
   %CREATE-I-CREATED, DISK:[SMITH.UTLCOM.FIL] created
   $ CREATE/DIRECTORY/LOG [.UTLCOM.EDT]
   %CREATE-I-CREATED, DISK:[SMITH.UTLCOM.EDT] created
3  $ DIRECTORY [...]
4  Directory DISK:[SMITH]

   COM.DIR;1           FORCALL.MAR;1       MMUL.FOR;1          POLA.QUO;1
   PRINT.FOR;1         RANDOM.FOR;1        STRPROG.TXT;1       UTLCOM.DIR;1

   Total of 8 files.

   Directory DISK:[SMITH.UTLCOM]

   EDT.DIR;1           FIL.DIR;1

   Total of 2 files.

   Grand total of 2 directories, 10 files.
5  $ RENAME [SMITH]*.MAR,*.TXT [.UTLCOM.FIL]*.*
   $ SET DEFAULT [.UTLCOM.FIL]
   $ DIRECTORY

   Directory DISK:[SMITH.UTLCOM.FIL]

   FORCALL.MAR;1           STRPROG.TXT;1

   Total of 2 files.
```

Example 5–4 Using VMS Commands to Create and Maintain a Directory Hierarchy

Notes on Example 5–4:

1 `$ SHOW DEFAULT`

Before creating a subdirectory hierarchy, you need to know the name of your current default device and directory. To display this information, enter the **SHOW DEFAULT** command. Your default device is DISK. Your default directory is named SMITH.

```
2   $ CREATE/DIRECTORY/LOG [SMITH.COM]
        .
        .
    %CREATE-I-CREATED, DISK:[SMITH.UTLCOM.EDT] created
```

To create subdirectories, issue the **CREATE/DIRECTORY** command at your terminal, followed by a directory name.

By including the **/LOG** qualifier in each **CREATE/DIRECTORY** command string, you instruct the system to display a message at your terminal each time it successfully creates a subdirectory file.

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

Once you create a hierarchy of subdirectories, display your work by entering a **DIRECTORY** command string. Use an ellipsis (...) in the directory specification to force a search of all subdirectories associated with your UFD. No UFD name is required in the directory specification, since your default directory name is equated to its value.

```
4   Directory DISK:[SMITH]
```

The display produced by the **DIRECTORY** command reveals the skeleton of the hierarchy you have created. At the *top* of the structure is your UFD, SMITH, which now lists two directory files, COM.DIR and UTLCOM.DIR, in addition to other files.

The target subdirectories named COM and UTLCOM form the next level of your hierarchy. COM contains no files. UTLCOM, however, lists two directory files, EDT.DIR and FIL.DIR. These two subdirectories form the third level of your hierarchy.

```
5   $ RENAME [SMITH]*.MAR,*.TXT, [.UTLCOM.FIL]*.*
    $ SET DEFAULT [.UTLCOM.FIL]
    $ DIRECTORY
```

To change which directory catalogues a file, use the **RENAME** command. In this example, you choose to recatalogue all the most recent versions of files in the directory named [SMITH] with file types MAR and TXT to the directory named [.UTLCOM.FIL], retaining the same file name, file type, and version number.

Notice that the command that alters which directory catalogues a file is the **RENAME** command and not the **COPY** command. Use the **RENAME** command when all the files reside on the same disk structure. To move files from one disk structure to another, use the **COPY** command.

To see files that were renamed to the subdirectory FIL, set your default to [.UTLCOM.FIL] and issue a **DIRECTORY** command.

5.11 PROTECTING FILES IN YOUR DIRECTORY HIERARCHY

The VMS system allows you to access other users' files. To create a file in your directory with the contents of a file that exists in another user's directory, include the other user's directory name in the file specification when you enter the **COPY** command. For example:

```
$ COPY [MATTHEWS]WANTED.FIL NEW.FIL
```

The system places a copy of the text found in the file WANTED.FIL of the directory named [MATTHEWS] into your default directory, giving it the name NEW.FIL.

Since it is possible for users to manipulate the files of others, it is necessary to protect files from unwanted access. To access a disk file, a user must pass the following three levels of protection:

- *Volume protection*—Controls who can access a particular disk volume
- *Directory protection*—Controls who can access a particular directory
- *File protection*—Controls who can access a particular file

To access a tape volume, a user only needs to pass the volume level of protection, which controls who can manipulate files on tapes.

5.11.1 How the System Determines Access

When you attempt to access a file, your UIC is compared to the owner UIC of the file. Depending on the relationship of the UICs, you will fall into one or more of the following categories.

- SYSTEM—All users who have system privilege (SYSPRV) or low group numbers, usually from 1 through 10 (octal).
- OWNER—The user with the same UIC as the owner UIC of the file.
- GROUP—All users, including the owner, who have the same group number in their UICs as the file owner.
- WORLD—All users, including those in the first three categories.

The *protection code* describes the categories of users who have access to a file, and the type of access they have. For example, the protection code:

```
SYSTEM:RWED, OWNER:RWED, GROUP:RE, WORLD:RE
```

specifies that users in the SYSTEM and OWNER categories have READ, WRITE, EXECUTE, and DELETE access. Users in the GROUP and WORLD categories have only READ and EXECUTE access.

Figures 5–3 and 5–4 illustrate protection codes. Also, see Tables 5–12 and 5–13 for an explanation of access rights and user categories.

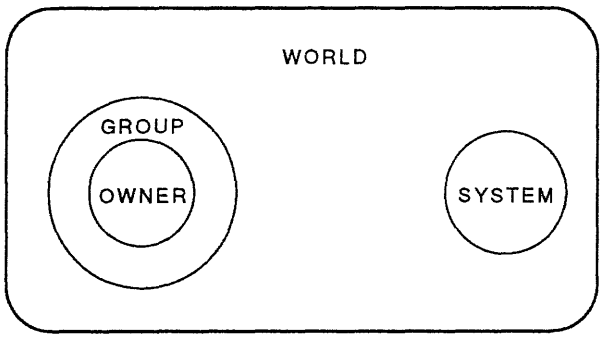


Figure 5–3 Interaction of Access Categories

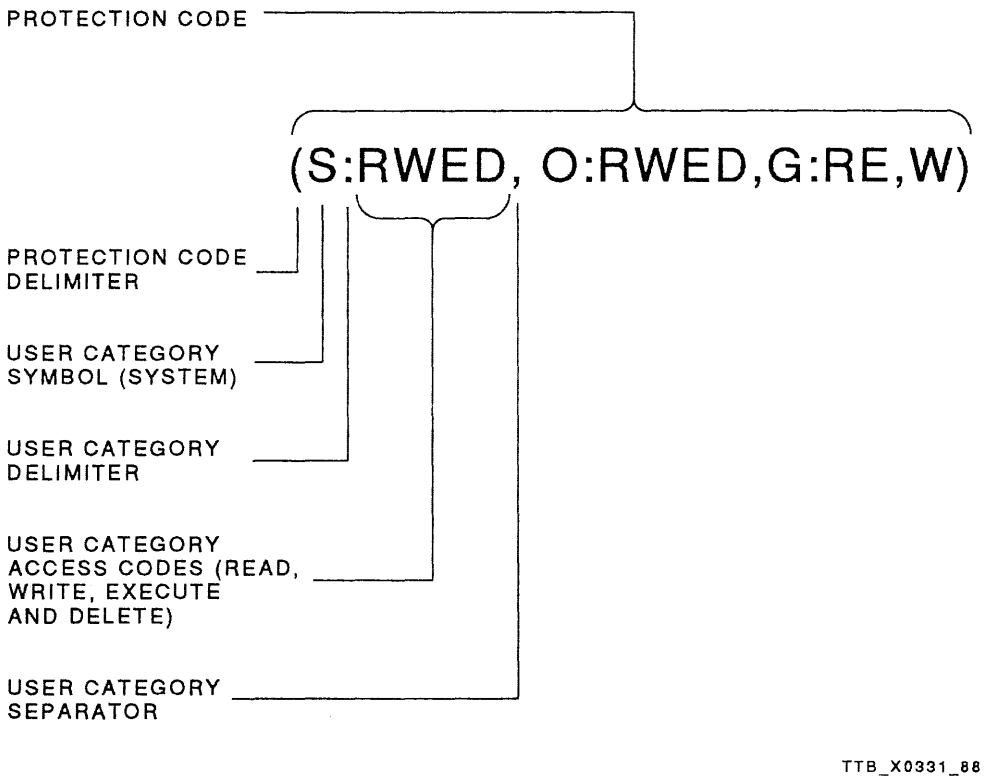


Figure 5–4 Elements of a Protection Code: Determines Which Users Have Access to a File

Table 5–12 Summary of Effects of Access Rights to Files

	(R)ead	(W)rite	(E)xecute	(D)elete
Disk Directory	Can read list of files in directory	Can modify list (Add files) Read access also needed	Can access explicitly named files	Can delete the directory
Disk File	Can read contents of file(s)	Can modify contents of file(s)	Can execute executable files	Can delete file(s)
Tape File	Can read list of files on tape	Can add files on the volume	Does not apply	Does not apply

Table 5–13 Determining a User's Category by Comparing User's UIC to File Owner's UIC

Category	Group Number	Member Number
SYSTEM	0–10 (Octal)	Does not matter
OWNER	Matches group number of file UIC	Matches member number of file UIC
GROUP	Matches group number of file UIC	Does not matter
WORLD	Does not matter	Does not matter

5.12 PROTECTION MECHANISMS

The VMS system uses the following two protection mechanisms:

- Access control lists (an optional protection)
- UIC–based protection

In granting access to a device or file, the VMS system checks the associated *access control list (ACL)* followed by the *UIC–based* protection. If the ACL allows access, then the UIC–based protection is not checked. If the ACL denies access, the system checks only the system and owner fields of the UIC–based protection code to determine whether you have access. If there is no ACL or you are not mentioned in the ACL, then the UIC–based protection is checked to determine access rights. Therefore, ACLs allow you to add limitations to the protection already provided by the UIC–based protection.

5.12.1 UIC–Based Protection

The VMS system assigns each file a UIC and a protection code when it is created. The system uses the UIC and protection code to determine who can:

- Read the file
- Modify the file
- Execute the file
- Delete the file

The format of your UIC is a pair of numbers in brackets, as follows:

[group,member]

The group number can range from 0 to 37777 (octal), while there can be from 0 to 177777 (octal) members in each group.

Examples:

Numeric: [100,30]

Alphanumeric: [GROUP11,SMITH] or [SMITH]

UICs specify the owners of objects, such as files, for which the VMS system provides protection. A user attempting to access a file falls into one of four access categories: System, Owner, Group, or World. The system places the user in a particular category by comparing the user's UIC to the UIC attribute of the file. The system manager assigns UICs.

Suppose a file has the following protection code:

S:R,O:R,G:R,W:RWED

You might expect that the owner would have only read access to the file. However, a user may be granted access to a file through more than one category. Any user in the Owner Category is also in the Group and World Categories. Therefore, the owner has read, write, execute, and delete access. The usual default protection codes supplied by the system are:

File	S:RWED,O:RWED,G:RE,W
Directory	S:RWE,O:RWE,G:RE,W
Volume	S:RWED,O:RWED,G:RWED,W:RWED

To alter the protection of files, you must have a system UIC, SYSPRV privilege, or own the file.

5.12.2 Access Control Lists

Access control lists are an optional layer of protection normally specified by the system manager. They are used to obtain more control than UIC–based protection. ACLs are usually used to provide access for specific users but not all users on the system.

Identifiers are the means of specifying the users in an ACL. There are three types of identifiers:

1. *UIC identifiers* that depend on the user identification codes (UICs) that uniquely identify each user on the system.
2. *General identifiers* that are defined by the system manager to identify groups of users (for example, STUDENT or PERSONNEL).
3. *System–defined identifiers* that describe certain types of users (BATCH, NETWORK, LOCAL, REMOTE). These identifiers are automatically defined by the system at installation time.

Users can have one or more identifiers. Files specify the access rights for holders of various identifiers. Since there may be many identifiers needed to represent different access needs for each user, you may need to create a list of entries, each of which defines groups of access rights. This list is the access control list (ACL). Each entry in this list is called an *access control list entry (ACE).*

5.12.3 Creating or Modifying an Access Control List

The following commands are used to obtain ACL information:

- **SHOW ACL file–name**
- **DIRECTORY/ACL file–name**
- **DIRECTORY/FULL file–name**
- **DIRECTORY/SECURITY file–name**

5.12.3.1 Access Control List Entries

Each ACL consists of one or more ACEs. There is no limit to the number of ACEs that an ACL can have, or to the number of characters in an ACE. You can add ACEs to your ACL by using the DCL command **EDIT/ACL file–name**, which invokes the ACL editor.

The format for an ACE is:

(TYPE,[OPTIONS],[ACCESS])

The first field, TYPE, determines the type of access protection. There are three types of ACEs:

1. *Identifier*—Controls the type of access allowed to a particular user or group of users

   The first field in the identifier ACE consists of the keyword IDENTIFIER followed by one or more identifiers. An identifier can be:

   - The user identification code (UIC)
   - A general identifier established by the system manager
   - A system–defined identifier

2. *Default Protection*—Defines the default protection for a directory
3. *Security Alarm*—Provides a security alarm when an object (such as a file or directory) is accessed in a particular way

The second field, OPTIONS, indicates options (if any) that apply to the ACE.

The third field, ACCESS, indicates the type of access to be granted to the file, such as READ, WRITE, EXECUTE, or DELETE.

The exact format of an ACE depends on its type, but all ACEs are enclosed in parentheses. Example 5–5 shows you how to modify the ACL for the file MYFILE.TXT.

```
$ DIRECTORY/FULL MYFILE.TXT

Directory DISK:[SMITH]

MYFILE.TXT;1                  File ID:  (25168,6,0)
Size:            1/3          Owner:    [GROUP11,SMITH]
Created:  17-DEC-1986 14:18   Revised:  17-DEC-1986 14:24 (3)
Expires:   <None specified>   Backup:    <No backup recorded>
File organization:  Sequential
File attributes:    Allocation: 3, Extend: 0, Global buffer count: 0,
                    No version limit
Record format:      Variable length, maximum 47 bytes
Record attributes:  Carriage return carriage control
Journaling enabled: None
File protection:    System:RWED, Owner:RWED, Group:RE, World:
Access Cntrl List:  None

Total of 1 file, 1/3 blocks.

$ EDIT/ACL MYFILE.TXT

(IDENTIFIER=VMS,ACCESS=READ+WRITE+EXECUTE+DELETE)
CTRL/Z

$ DIRECTORY/FULL MYFILE.TXT

Directory DISK:[SMITH]

MYFILE.TXT;1                  File ID:  (25168,6,0)
Size:            1/3          Owner:    [GROUP11,SMITH]
Created:  17-DEC-1986 14:18   Revised:  17-DEC-1986 14:45 (4)
Expires:   <None specified>   Backup:    <No backup recorded>
File organization:  Sequential
File attributes:    Allocation: 3, Extend: 0, Global buffer count: 0,
                    No version limit
Record format:      Variable length, maximum 47 bytes
Record attributes:  Carriage return carriage control
Journaling enabled: None
File protection:    System:RWED, Owner:RWED, Group:RE, World:
Access Cntrl List:  (IDENTIFIER=VMS,ACCESS=READ+WRITE+EXECUTE+DELETE)

Total of 1 file, 1/3 blocks.
```

Example 5–5 Modifying an Access Control List

5.13 DETERMINING AND ALTERING FILE PROTECTION

Each file on a disk has its own protection code. You can determine the current default protection by issuing the **SHOW PROTECTION** command:

```
$ SHOW PROTECTION
  SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD=NO ACCESS
```

This response is your default protection. To determine the current protection on a specific file or files, use the **/PROTECTION** qualifier with the **DIRECTORY** command. For example:

```
$ DIRECTORY/PROTECTION DIRECTORY.LIS

Directory DISK:[SMITH.DOC]

DIRECTORY.LIS;1      [100,200]      (RWED,RWED,RE,)
```

You can set the protection when you create a file or change the protection on an existing file using the **SET PROTECTION** command. For example, you can specify the protection for a file you create using the **COPY** command as follows:

```
$ COPY DISK1:[SMITH]DIRECTORY.LIS
DIR.LIS/PROTECTION=(SYSTEM:RW,OWNER:RWED,GROUP:RW,WORLD)
```

This command copies a file from the device DISK1 to your default disk directory. The protection code defines the protection for the new file DIR.LIS as:

- Users with system UICs can read and write to the file
- You (owner) have all types of access
- Other users in your group can read and write to the file
- All other users (world) have no access

If you do not define a protection code for a file when you create it, the system applies a default protection. If a version of the file already exists, protection is taken from the previous version. For a new file, the protection is determined in one of two ways:

1. If the directory where the file is to be placed has an associated access control list that specifies the DEFAULT_PROTECTION entry, the specified protection is used.

2. If the directory does not have an associated ACL, the default process protection is used. The default process protection is established explicitly with the **SET PROTECTION/DEFAULT** command, or by default when you log in. See Table 5–14 and Example 5–6 for more information on setting protections.

   NOTE
   To protect a file completely, you must protect both the file itself and the directory in which the file is listed. If you have files that must be protected against unauthorized access, be sure to specify the proper protection for both the directories and the files themselves.

Table 5–14 Commands Used to Determine and Alter File Protection

Operation	Comments and Examples
Displaying the default protection assigned to new files	The default protection applies to all newly created files in the current directory `$ SHOW PROTECTION`
Obtaining the protection code of a given file	Displays the current protection of a particular file `$ DIRECTORY/PROTECTION MYFILE.TXT`
Changing the default protection assigned to new files	The default protection, once changed, affects all future files created in this particular directory. Files created before changing the default protection will retain the previous protection. `$ SET PROTECTION=(S:RWED,O:RWED,G:RWE,W:RWE)/DEFAULT`
Changing the protection code of an existing file	The protection code can be changed to allow more or less access to a particular file. `$ SET PROTECTION=(S:RWED,O:RWE,G:RW,W:) MYFILE.TXT`

NOTE
If you omit a protection category when you issue the SET PROTECTION command, the protection for that category remains unchanged.

```
$ SET DEFAULT [SMITH.DOC]

$ SHOW PROTECTION
  SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD=NO ACCESS

$ DIRECTORY/OUTPUT=DIRECTORY.LIS
$ DIRECTORY/OWNER/PROTECTION

Directory DISK:[SMITH.DOC]

DIRECTORY.LIS;1       [GROUP11,SMITH]      (RWED,RWED,RE,)
EDT.DIR;1             [GROUP11,SMITH]      (RWE,RWE,RWE,RE)

Total of 1 file.

$ SET PROTECTION=(S:R,G:R)/DEFAULT

$ SHOW PROTECTION
  SYSTEM=R, OWNER=RWED, GROUP=R, WORLD=NO ACCESS

$ DIRECTORY/OUTPUT=DIRECTORY.LIS
$ DIRECTORY/OWNER/PROTECTION

Directory DISK:[SMITH.DOC]

DIRECTORY.LIS;2       [GROUP11,SMITH]      (R,RWED,R,)
DIRECTORY.LIS;1       [GROUP11,SMITH]      (RWED,RWED,RE,)
EDT.DIR;1             [GROUP11,SMITH]      (RWE,RWE,RWE,RE)

Total of 2 files.
```

Example 5–6 Changing Your Default Protection Code

5.14 DELETING A SUBDIRECTORY

Before you can delete a subdirectory, you must delete all the files catalogued in that subdirectory. When all the files have been deleted, set your default to the directory or subdirectory that contains the subdirectory name to be deleted. The directory protection on the subdirectory to be deleted must allow the owner DELETE access. If it does not, you must change the directory protection to reflect this. Example 5–7 shows how to delete a subdirectory from a directory, and Example 5–8 shows how to delete two levels of subdirectories.

```
$ SET DEFAULT [SMITH.DOC]
$ DIRECTORY

Directory DISK:[SMITH.DOC]

CLASS.LIST;4          CLOCK.EXE;1          COLOR.COM;4          DEG.EXE;1
JOE_EVE.TPU$SECTION;1                      MYFILE.TXT;1         NOTE.COM;4
REMIND.EXE;1          REMLOG.EXE;1         TRNG.PLAN;6          VT100.CLR;1

Total of 11 files.

$ DELETE *.*;*
$ DIRECTORY
  %DIRECT-W-NOFILES, no files found

$ SET DEFAULT [SMITH]
$ DELETE DOC.DIR;1
   %DELETE-W-FILNOTDEL, error deleting DISK:[SMITH]DOC.DIR;1
   -RMS-E-PRV, insufficient privilege or file protection violation

$ SET PROTECTION=(O:RWED) DOC.DIR

$ DELETE DOC.DIR;1

$ DIRECTORY DOC.DIR
  %DIRECT-W-FILES, no files found
```

Example 5–7 Deleting a Subdirectory from a Directory Hierarchy

```
1  $ SET DEFAULT [SMITH]
2  $ DIRECTORY [SMITH...]
   Directory DISK:[SMITH]
   DOC.DIR;1          MYFILE.TXT;1      MYTEXT.TXT;1      TXT.TXT;1
   Total of 4 files.

   Directory DISK:[SMITH.DOC]
   FORTRAN.DIR;1      MYFILE.TXT;1      MYTEXT.TXT;1      TXT.TXT;1
   YOUR.FILE;1
   Total of 5 files.

   Directory DISK:[SMITH.DOC.FORTRAN]
   MYFILE.TXT;1       MYTEXT.TXT;1      TXT.TXT;1         YOUR.FILE;1
   Total of 4 files.

   Grand total of 3 directories, 13 files.
3  $ SET PROTECTION=O:RWED [SMITH...]*.*;*
4  $ DELETE [SMITH...]*.*;*
   %DELETE-W-FILNOTDEL, error deleting DISK:[SMITH]DOC.DIR;1
   -RMS-E-MKD, ACP could not mark file for deletion
   -SYSTEM-F-DIRNOTEMPTY, directory file is not empty
   %DELETE-W-FILNOTDEL, error deleting DISK:[SMITH]FORTRAN.DIR;1
   -RMS-E-MKD, ACP could not mark file for deletion
   -SYSTEM-F-DIRNOTEMPTY, directory file is not empty
5  $ DIRECTORY [SMITH...]
   Directory DISK:[SMITH]
   DOC.DIR;1
   Total of 1 file.

   Directory DISK:[SMITH.DOC]
   FORTRAN.DIR;1
   Total of 1 file.

   Grand total of 2 directories, 2 files.
6  $ DELETE [SMITH...]*.*;*
   %DELETE-W-FILNOTDEL, error deleting DISK:[SMITH]DOC.DIR;1
   -RMS-E-MKD, ACP could not mark file for deletion
   -SYSTEM-F-DIRNOTEMPTY, directory file is not empty
7  $ DIRECTORY [SMITH...]
   Directory DISK:[SMITH]
   DOC.DIR;1
   Total of 1 file.
8  $ DELETE [SMITH...]*.*;*
   $ DIRECTORY [SMITH...]
   %DIRECT-W-NOFILES, no files found
```

Example 5–8 Removing Subdirectories from a Directory Hierarchy

Notes on Example 5–8:

1 The **SET DEFAULT** command moves the user to a Level 1 directory.

2 The **DIRECTORY** command is issued to obtain a listing of all files in the directory hierarchy.

3 Protection is set to enable the owner to delete all files and all subdirectories.

4 The **DELETE** command is issued to delete all files and all subdirectories. The subdirectories (DOC.DIR and FORTRAN.DIR) are not deleted during the first issuance of the **DELETE** command. A fatal error message is generated: "directory file is not empty."

5 The **DIRECTORY** command is issued, which establishes that all files have been deleted except the subdirectories DOC.DIR and FORTRAN.DIR.

6 The **DELETE** command is re–issued. This deletes the subdirectory [.FORTRAN] but the subdirectory [.DOC] is not deleted. A fatal error message is generated: "directory file is not empty."

7 The **DIRECTORY** command is issued, which establishes that the subdirectory DOC.DIR has not yet been deleted.

8 The **DELETE** command is issued again and now the subdirectory [.DOC] is deleted.

NOTE
Issuing the DIRECTORY command is an optional step. In this example, it is showing the remaining files during each step in the process of deleting subdirectories.

5.15 SPECIFYING DEVICES

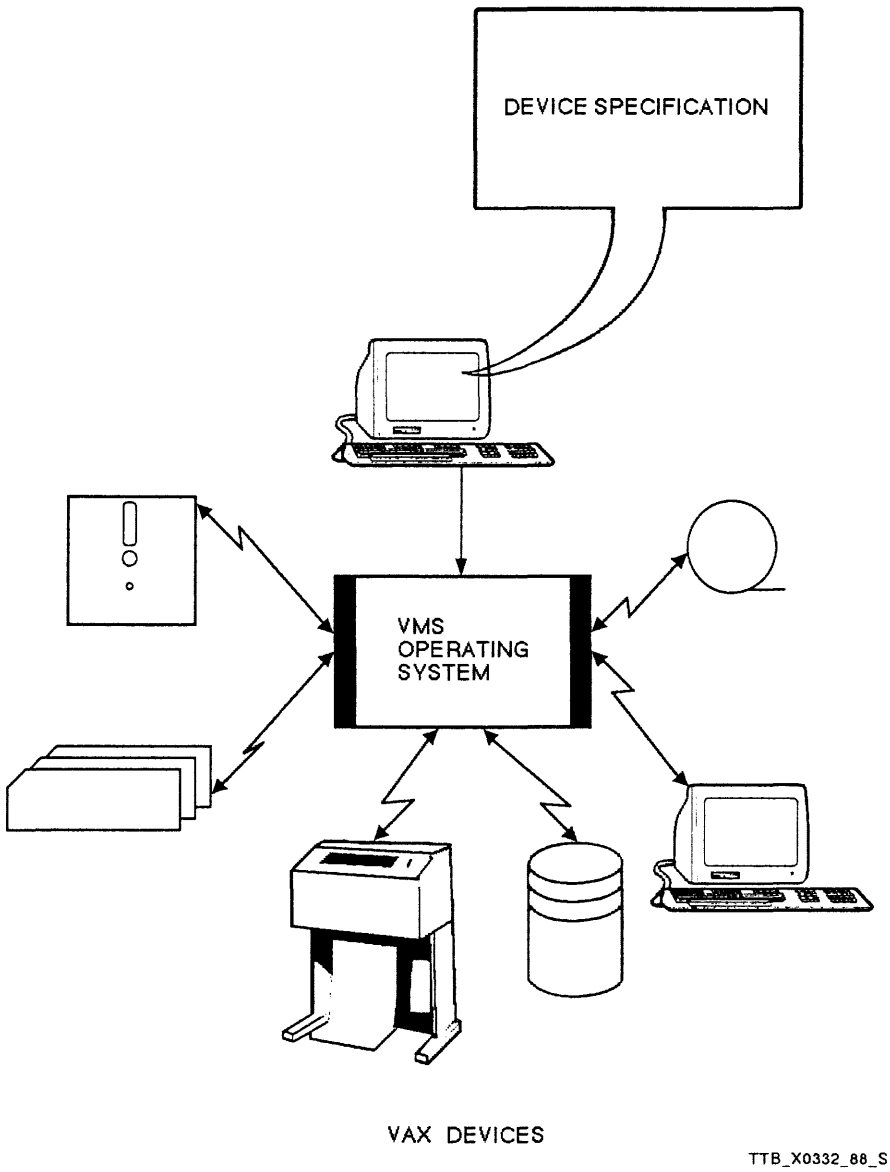


Figure 5–5 Device Specifications are Used to Identify the Desired Device for a Given Operation

As was previously discussed, you can assign logical names to specific devices on the system. Using these logical names, you can access files from other devices. If you specify a file using a logical device name, you can access the file regardless of which physical device holds the disk or tape containing your file. Your system manager will ensure that the logical device names are always equated to the correct physical devices. The device name should precede the directory name and be terminated with a colon (:).

For example:

```
$ DIRECTORY DRA0:[SMITH]
```

gives you a listing of the files in the [SMITH] directory on the device DRA0:. You can also move files from one device to another using the **COPY** command. See Table 5–15 for some examples of using other devices.

Table 5–15 Examples of Using Other Devices

Operation	Comments and Examples
Listing files in a directory on another disk	Lists all files in the directory [SMITH] located on the disk DBA2: `$ DIRECTORY DBA2:[SMITH]`
Locating a file in a directory on another disk	Searches for the file name MYFILE.TXT in the directory [SMITH] located on the disk DBA2: `$ DIRECTORY DBA2:[SMITH]MYFILE.TXT`
Copying a file from another disk to your default disk and directory	Copies the latest version of MYFILE.TXT from another disk to your default disk and directory `$ COPY DBA2:[SMITH]MYFILE.TXT MYFILE.TXT`
Listing all files on a tape device	Lists all files on a magnetic tape on device MTA2: `$ DIRECTORY MTA2:`
Finding a file on a tape device	Searches for the file MYFILE.TXT on a magnetic tape on device MTA2: `$ DIRECTORY MTA2:MYFILE.TXT`
Copying a file from tape to a disk	Copies the file MYFILE.TXT from the tape on MTA2: to your default disk and directory `$ COPY MTA2:MYFILE.TXT *.*;*`

You can also move directories from one device to another. Table 5–16 shows you how to move a hierarchical file structure from device DBA0: to device DRA2:.

Table 5–16 Moving a Hierarchical File Structure from one Disk Device to Another

Command	Comments and Examples
COPY	Copies all versions of the files in and below the SFD [SMITH.UTLCOM] on device DBA0: to the directory [JONES] on device DRA2:, preserving the hierarchical file structure. `$ COPY DBA0:[SMITH.UTLCOM...]*.*.* DRA2:[JONES...]*.*.*` Copies all versions of the files in and below the SFD [SMITH.UTLCOM] on device DBA0: to the directory [JONES.UTLCOM] on device DRA2:, preserving the hierarchical file structure. If the file UTLCOM.DIR does not exist in the directory [JONES], the COPY command fails. `$ COPY DBA0:[SMITH.UTLCOM...]*.*;* DRA2:[JONES.UTLCOM...]*.*;*`

5.16 PROTECTING DISK AND TAPES

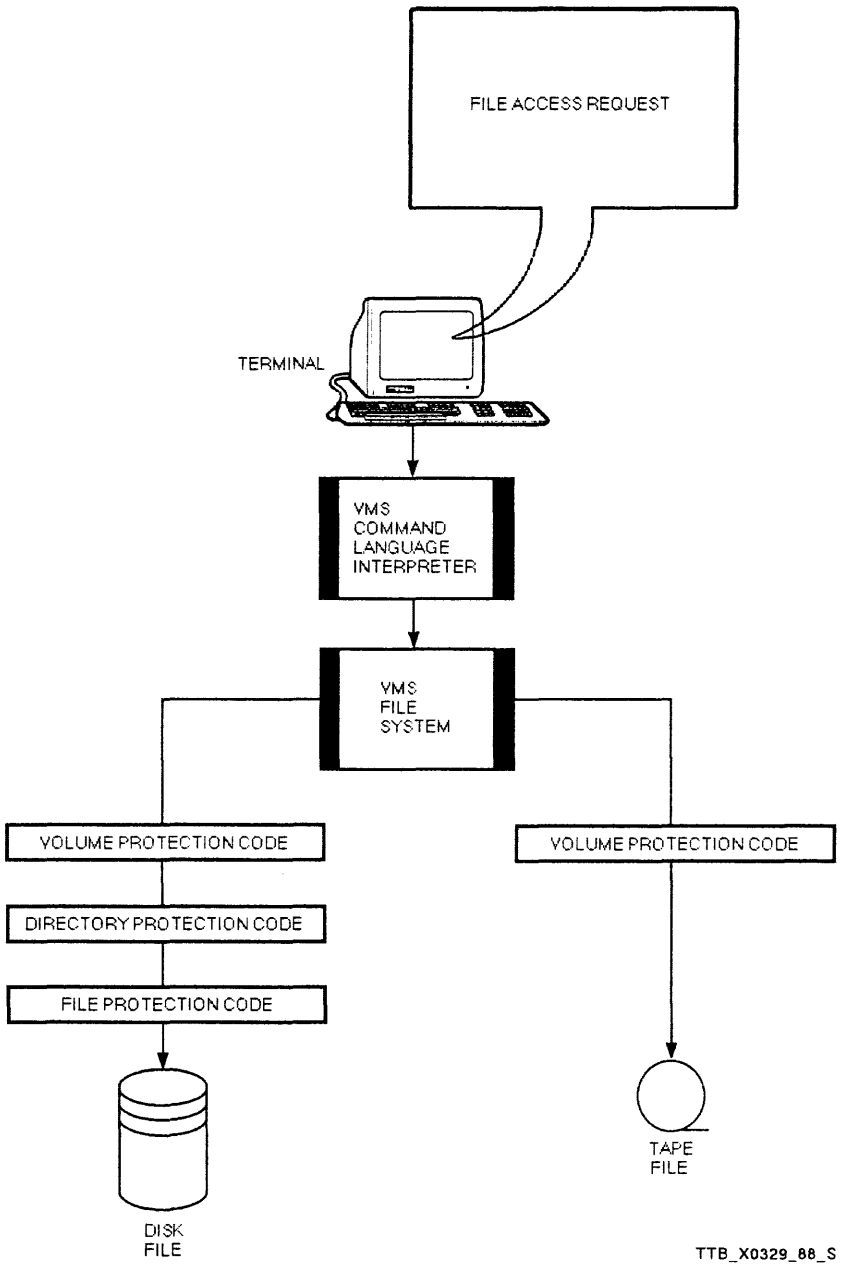


Figure 5–6 File Access to Disk and Tape Volumes

5.17 SUMMARY

Directory Type	Example	Naming Convention
Master File Directory (MFD)	[000000]	Each disk contains one MFD, named [000000].
User File Directory (UFD)	[SMITH]	Your user name is usually your UFD name.
Subdirectory (SFD)	[SMITH.PAYROLL]	You choose the names for the subdirectories you create.

Use the **DIRECTORY** command to:

- Find files on a peripheral storage device on your system
- Display the contents of directories or the characteristics of files

You may want to change file protection to:

- Restrict access to your files
- Prevent unauthorized moving or deletion of files
- Assign a special protection code for all files created in a particular directory
- Delete a subdirectory

There are two means of protecting files:

- User Identification Code (UIC–based) protection
- Access Control Lists (ACLs)

5.18 APPENDIX A—DEVICE INFORMATION

Table 5–17 Codes for Some Supported Devices on a VMS System

Code	Device Type
CS	Console Storage Device
DB	RP05, RP06 Disk
DD	TU58 Cassette Tape
DJ	RA60 Removable Disk
DL	RL02 Cartridge Disk
DM	RK06, RK07 Cartridge Disk
DQ	R80 Disk
DR	RM03, RM05, RM80, RP07 Disk
DU	RA82, RA80, RA81, RC25, RD54, RD53 Disk, RX33, RX50 Floppy Diskette
DX	RX01 Floppy Diskette
DY	RX02 Floppy Diskette
LA	LPA11–K Laboratory Peripheral Accelerator
LC	Line Printer on DMF32
LP	Line Printer on LP11
LT	Interactive Terminal or Terminal Server
MB	Mailbox
MF	TU78 Magnetic Tape
MS	TS11, TU80 Magnetic Tape
MT	TE16, TU45, TU77 Magnetic Tape
MU	TA78, TK50, TU81 Magnetic Tape

Table 5–17 (Cont.) Codes for Some Supported Devices on a VMS System

Code	Device Type
NET	Network Communication Logical Device
NL	System "Null" Device
OP	Operator's Console
RT	Remote Terminal
TT	Interactive Terminal on DZ11
TX	Interactive Terminal on DMF32
XA	DR11–W General Purpose DMA Interface
XD	DMP–11 Synchronous Communications Lines
XE	DEUNA Communication Device
XF	DR32 Interface Adapter
XG	DMF32 Synchronous Communications Lines
XJ	DUP11 Synchronous Communications Lines
XM	DMC11 Synchronous Communications Lines
XQ	DEQNA Communication Device

DCL allows you to omit the controller character and the unit number in device specifications. The effect of leaving out these parts depends on the command you use. Most DCL commands interpret an incomplete device specification as a physical device specification, or one that is meant to signify a single physical device. Other commands interpret an incomplete device specification as a generic device specification, or one that specifies a group of devices on your system.

The commands that interpret an incomplete device specification as a physical device name include: **INITIALIZE**, **DISMOUNT**, **DEALLOCATE** and all file manipulation commands. Physical device specifications always default missing controller characters to A, and missing unit numbers to 0. For example, the system interprets the incomplete device specification DB: as DBA0:.

One command that interprets an incomplete device specification as a generic device name is **SHOW DEVICE**. Table 5–18 summarizes device terminology, while Table 5–19 demonstrates the syntax you use when specifying devices. For additional information concerning commands, refer to the *VMS DCL Dictionary*.

Table 5–18 Summary of Device Terminology

Term	Definition
Peripheral Device	A unit on the system used for information input, output, or storage. A device can be classified either as a mass storage device or as a record–oriented device.
Mass Storage Device	A device used for storing information on a magnetic medium. Examples include disks and tapes.
Record–Oriented Device	A device used for reading and writing single units of data. Terminals, printers, and card readers are examples of these devices.
Physical Device Name	A specific physical device on the system. Consists of a device–type code, a controller character, and a unit number.
Logical Device Name	A synonym for a physical device name. Often used to refer to a specific volume, regardless of the device on which it is mounted. Usually the system manager sets up logical names.
Generic Device Name	A group of devices, consisting of a physical device name that does not specify the controller and the unit number.
Cluster Device Name	Name of a device on a node in a cluster, consisting of a cluster node name or allocation class and a device name separated by a dollar sign.

Table 5–19 Generic Specification with the SHOW DEVICE Command

Operation	Comments and Examples
Using Physical Device Names	
Specifying a particular device	Displays full information on the magnetic tape (MT) unit (0) on controller A `$ SHOW DEVICE/FULL MTA0:`
Using Generic Device Names	
Specifying all devices of a given type except terminals	Displays brief characteristics of all RA60 devices (DJ) `$ SHOW DEVICE DJ:`
Specifying all devices of a given type on a single controller	Shows brief characteristics of all MT magnetic tape devices on controller A `$ SHOW DEVICE MTA:`
Specifying all devices of a given type at the same position on different controllers	Displays brief characteristics of all terminals (TT) with unit number 1 on any controller `$ SHOW DEVICES TT1:`
Specifying all terminals	Displays brief characteristics of all system terminals `$ SHOW DEVICE T:`
Specifying your assigned terminal	Displays brief characteristics of your assigned terminal (TT: is a system–defined logical name equating to your terminal) `$ SHOW DEVICE TT:`

5.19 APPENDIX B—NETWORKING INFORMATION

5.19.1 Managing Files on Another VMS System in Your Network

5.19.1.1 Methods of File Management in a Network

VMS systems allow you to manage files on devices connected to other systems without compromising the VMS file security features. You have several choices for managing files located on devices connected to another system. These choices include:

- **Using the SET HOST command –**

  You can use the **SET HOST** command to connect your terminal (through the current host processor) to another processor, called the remote processor, and enter DCL file–manipulation commands. This requires that both processors run DECnet and that you know a user name and password of an account on the remote system. The account you use on the remote system supplies values to the remote system to process your DCL file–manipulation commands.

- **Using an access–control string in your DCL commands –**

  You can include an access–control string in the DCL file–manipulation commands that function across the network. (This section addresses access–control strings.) The access–control string includes both the user name and password of an account on a remote system. The account referenced by the access–control string also provides default values to the remote system to process your DCL file–manipulation commands.

- **Using a proxy account –**

  You can use the defaults supplied by a proxy account when you use DCL file–manipulation commands that function across the network. (A proxy account, if the system manager establishes one for you, associates your user name with an account on the remote system.) The account on the remote system associated with your user name provides the default values the remote system needs to process your DCL file–manipulation commands.

- **Using the DECnet defaults –**

  The system manager can establish a default DECnet account. If this account is created, it will supply default values to DCL file–manipulation commands entered across the network.

5.19.2 Using DCL File–Manipulation Commands in a Non–VAXcluster Network Environment

5.19.2.1 Two Node Specification Formats

This section first discusses using DCL file–manipulation commands in a network that does not contain a VAXcluster system. Then it looks at factors concerning managing files on a VAXcluster system.

When you manage files on a device connected to a remote node, it is necessary to include the node specification of the remote processor in the file specification of DCL file–manipulation commands. Two formats for the node specification follow.

1. **Nodename::**
2. **Nodename "access control string"::**

When you use the **Nodename::** format, the remote system takes the following action:

- It processes your file–manipulation request as though your request was issued from its default DECnet account. If there is no default DECnet account, your file–manipulation request fails.
- It uses the UIC of its default DECnet account to determine the file access rights.
- It obtains default values from its default DECnet account for fields you omit from file specifications in your file–manipulation request.

When you include an access–control string, the remote system takes a different action. The access–control string you provide consists of a user name and password in the format **user–name password**. When you use this format, the remote system takes the following action:

- It processes your file–manipulation request under a process created from the account specified in the access–control string.
- It uses the UIC of the account specified in the access–control string to determine file access rights.
- It obtains default values from the account specified in the access–control string for fields you omit from file specifications in your file–manipulation request.

Table 5–20 shows examples of specifying files on a device connected to a remote node.

Table 5–20 Examples of Specifying Files on Remote Nodes

Examples	Comments
`$ DIRECTORY DIPPER::DBA1:[SMITH]PAY.FOR;1`	Specifies the file PAY.FOR;1 in the directory [SMITH] on disk DBA1: on remote node DIPPER:.
`$ DIRECTORY DIPPER"BILL SMITH"::DBA1:[SMITH]PAY.FOR;1`	Specifies the same as the preceding example. Access to the file uses the UIC of user SMITH.
`$ DIRECTORY DIPPER"BILL SMITH"::PAY.FOR;1`	Specifies the same as the preceding example. The process supplies the defaults under the account for SMITH.
`$ DIRECTORY DIPPER"BILL SMITH"::[SMITH.DOC]PAY.DAT`	Specifies the file PAY.DAT in the subdirectory [SMITH.DOC] on the default disk of user SMITH on node DIPPER:.

The file management commands that work across a network are a subset of the commands used to manage files on an individual VAX processor. Table 5–21 illustrates how to use DECnet–VAX file management commands.

Table 5–21 DECnet–VAX DCL File–Manipulation Command Summary

Function	Comments and Examples
Adding the contents of one or more files to the end of another file (files may be local or remote)	Appends the contents of file DEMO.DAT in the directory [JAFFE] on the remote node BOSTON:: to the file TEST.DAT in your current directory on your local node. `$ APPEND BOSTON"JAFFE ANN"::DEMO.DAT TEST.DAT`
Copying one or more files to or from a remote node	Copies the file DEC12.DAT from your current directory to the directory [JANES] on the remote node WHYNOT::. Defaults on the remote node come from the UAF record specified within quotes. The same file name is retained. `$ COPY DEC12.DAT WHYNOT"JANES JIL"::*.*`
Creating a disk file on a remote node	Creates the file TEST.DAT in the directory [MODEL] on disk DBA1: of remote node TRNTO::. `$ CREATE TRNTO::DBA1:[MODEL]TEST.DAT` `Text is entered into file TEST.DAT` `CTRL/Z`
Displaying information about a file	Lists the files in the subdirectory [JANES.SUB1] located on the remote node WHYNOT::. `$ DIRECTORY WHYNOT"JANES JIL"::[JANES.SUB1]`

Table 5–21 (Cont.) DECnet–VAX DCL File–Manipulation Command Summary

Function	Comments and Examples
Displaying the contents of a file at a terminal on a remote node	Displays the file PAY.DOC;1 in the directory [GREEN] on disk DBA1: located on remote node DIPPER::. `$ TYPE DIPPER::DBA1:[GREEN]PAY.DOC;1`
Deleting one or more files at a remote terminal	Deletes all versions of the file PAY.FOR in subdirectory [JONES.SUB1] located on remote node WHYNOT::. `$ DELETE WHYNOT"JANES JIL"::[JANES.SUB1]PAY.FOR;*`

5.19.3 Using DCL File–Manipulation Commands in a VAXcluster Environment

5.19.3.1 Two Cluster Device Specification Formats

You can also manipulate files on a device connected to a system that is in a cluster. To do this, include a cluster device specification in your DCL file–manipulation commands. A cluster device specification has one of the following formats:

1. **node–name$device–name**
2. **$allocation–class$device–name**

The first format shows a cluster device specification consisting of the name of the device, preceded by the name of the node, followed by a dollar sign ($).

A sample cluster device specification of this type follows:

```
PETER$DUA1:
```

In the example above, PETER is the name of the node, DUA1 is the name of the physical device, and the dollar sign ($) is a delimiter.

The second format shows the device name of the cluster device specification prefixed by a number. The prefix number is called an *allocation class*, and is a number between 1 and 255.

A sample cluster device specification of this type follows:

```
$1$DUA0:
```

Only devices set up in a special way can use an allocation class as part of their device specifications. The system manager is responsible for setting up the devices and choosing an allocation class for each node in the cluster.

You may want to check the names of nodes and devices while you are manipulating your files. To do this, DCL provides the following commands: **SHOW NET**, **SHOW CLUSTER**, and **SHOW DEVICES**. Table 5–22 lists commands you can use to obtain information about nodes and devices in your systems environment. For more information on VAXclusters systems, refer to the *VMS VAXcluster Manual.*

Table 5–22 Commands Used to Determine the Nodes and Devices in Your Systems Environment

Operation	Command/Example	Comments
Determine the names of nodes in a network	`$ SHOW NET`	Displays a list of nodes in your network.
Determine the names of nodes in a cluster	`$ SHOW CLUSTER`	Displays a list of nodes (HSC and VAX) in your cluster.
Determine the names of devices accessible to your node	`$ SHOW DEVICES`	Displays a list of devices accessible to your node.

5.20 WRITTEN EXERCISE I

Suppose your default directory contains the following files:

```
A.DAT;1          A.FOR;2          AREA.FOR;2       AREA.FOR;1
B.DAT;3          B.FOR;1          C.DAT;4          C.FOR;1
MAILD22.DAT;2    MAILF22.DAT;2    MAILIST.COB;1    MAILJ14.DAT;1
```

1. List the files that are specified by the following file specifications (using the **DIRECTORY** command):

   a. *.FOR;2

   b. *.FOR

   c. A*.*;*

   d. A%%%.*;*

   e. %.DAT

   f. *.*;*

2. Give a single file specification that describes the following lists of files:

   a. A.DAT;1, A.FOR;2

   b. A.DAT;1, B.DAT;3, C.DAT;4

   c. MAILD22.DAT;2, MAILF22.DAT;2, MAILJ14.DAT;1

   d. A.DAT;1, MAILJ14.DAT;1

5.21 WRITTEN EXERCISE II

Next to each file maintenance operation, write the letter that corresponds to the VMS command best suited to accomplish it. Specify each command at least once.

Commands

a. APPEND
b. COPY
c. DELETE
d. DELETE/CONFIRM
e. DIFFERENCES
f. DIRECTORY
g. DIRECTORY/OUTPUT=file–specification
h. PRINT
i. PURGE
j. RENAME
k. TYPE

Operations

1. _____ Display the contents of a file at your terminal.
2. _____ Display the contents of your default directory at your terminal.
3. _____ Remove a specified file from your default directory.
4. _____ Remove all but the most recent version of a specified file from your default directory.
5. _____ Create an exact duplicate of a file in your default directory.
6. _____ List the contents of a file at the default system printer.
7. _____ Compare the contents of two files.
8. _____ Add the contents of one file to another.
9. _____ Change a directory name to a new directory name.
10. _____ Display the name of each file in your default directory and remove or retain it by entering a "Y" or an "N" at your terminal.
11. _____ List the contents of your default directory in a file for future reference.

5.22 WRITTEN EXERCISE III

Next to each directory maintenance operation, write the letter of the VMS command best suited to perform the job. You may use some commands more than once; you will not use others at all.

Commands

a. COPY
b. CREATE
c. CREATE/DIRECTORY
d. DELETE
e. DELETE/DIRECTORY
f. DIRECTORY
g. RENAME
h. SET DEFAULT
i. SET PROTECTION
j. SHOW DEFAULT
k. SHOW PROTECTION

Operations

1. _____ Display the name of your current default directory.
2. _____ Display the contents of a directory hierarchy.
3. _____ Remove a directory from a directory hierarchy.
4. _____ Add a directory to a directory hierarchy.
5. _____ Move files from one directory to another.
6. _____ Change your current default directory.
7. _____ Change the protection code of a directory file.
8. _____ Display the name of your current default device.
9. _____ Change your current default device.

5.23 WRITTEN EXERCISE IV

Each of the following questions describes an operation a user wants to perform on a given disk or tape file. Given the UIC of the user, and the owner UIC and protection code of the file, its directory, or its volume, determine whether the file system will permit the operation to occur. If the operation is permissible, write the word TRUE in the space that precedes the question; if it is not, write the word FALSE.

1. _____ A user with a UIC of [100,200] wants to delete a file on a tape volume.

```
Volume Owner UIC:           [100,200]
Volume Protection Code:     (S:RWED,O:RWED,G:RWED,W:RE)
```

2. _____ A user with a UIC of [363,2] wants to create a file on an RX33 disk volume.

```
Volume Owner UIC:           [363,0]
Volume Protection Code:     (S:RE,O:RWED,G:RE,W)
```

3. _____ A user with a UIC of [4,4] wants to read a file on an RA60 disk volume.

```
File Owner UIC:             [411,22]
File Protection Code:       (S,O:RWED,G,W:R)
```

4. _____ A user with a UIC of [100,200] wants to update a record in a file on an RA80 disk volume.

```
Volume Owner UIC:           [1,1]
Volume Protection Code:     (S:RWED,O;RWED,G:RWED,W:RWED)
Directory Owner UIC:        [100,210]
Directory Protection Code:  (S:RWE,O:RWE,G:RWE,W:RE)
File Owner UIC:             [100,210]
File Protection Code:       (S:RE,O:RWED,G:RWE,W:RE)
```

5. _____ A user with a UIC of [521,6] wants to read a file on an RA81 disk volume.

```
Volume Owner UIC:           [1,1]
Volume Protection Code:     (S:RWED,O:RWED,G:RWED,W:RWED)
Directory Owner UIC:        [521,13]
Directory Protection Code:  (S:RWE,O:RWE,G,W)
File Owner UIC:             [521,13]
File Protection Code:       (S:R,O:RWED,G:R,W:R)
```

5.24 LABORATORY EXERCISE I

1. Create a subdirectory called [.SUB1].
2. Copy some files from your login directory into [.SUB1].
3. Move yourself to that subdirectory.
4. Obtain a directory listing of all files in the subdirectory.
5. Combine two files to create a new file named NEWFILE.DAT.
6. Create another subdirectory beneath [.SUB1] and name the new subdirectory [.SUB2].
7. Copy some files from [.SUB1] into [.SUB2].
8. Obtain a directory listing of all files in the subdirectory.
9. Delete both subdirectories.

5.25 LABORATORY EXERCISE II

1. Create a file in your login directory. What protection code does this newly created file have and how did it get that protection code?
2. Change the protection code for this file to (S:R,O:R,G;R,W:R). Display the protection code to verify the change.
3. Delete this file. What happened and why?
4. Change your DEFAULT protection code to (S:R,O:RWED,G:R,W:R). Create a new file named NEWFILE.TXT. What protection code does this new file have and why?
5. Change your DEFAULT protection to give all persons in your UIC group RWED access and all persons in the WORLD category RWE access.

5.26 LABORATORY EXERCISE III

1. Choose a file in your directory. Issue a DCL command to obtain Access Control List information regarding that file.
2. Modify the UIC protection on the above file so that your group has no access.
3. Modify the ACL information to allow Read, Write, and Execute access to the file.
4. Check to see if an ACL was created. Have some of your fellow students try to access the file.
5. Delete the ACL on the above file.

5.27 WRITTEN EXERCISE I—SOLUTIONS

1. List the files that are specified by the following file specifications:

   a. *.FOR;2
   ```
   A.FOR;2, AREA.FOR;2
   ```

   b. *.FOR
   ```
   A.FOR;2, AREA.FOR;2, AREA.FOR;1, B.FOR;1, C.FOR;1
   ```

   c. A*.*;*
   ```
   A.DAT;1, A.FOR;2, AREA.FOR;2, AREA.FOR;1
   ```

   d. A%%%.*;*
   ```
   AREA.FOR;2, AREA.FOR;1
   ```

   e. %.DAT
   ```
   A.DAT;1, B.DAT;3, C.DAT;4
   ```

   f. *.*;*
   ```
   All files
   ```

2. Give a single file specification that describes the following lists of files:

   a. A.DAT;1, A.FOR;2
   ```
   A.* or A.*;*
   ```

   b. A.DAT;1, B.DAT;3, C.DAT;4
   ```
   %.DAT or %.DAT;*
   ```

   c. MAILD22.DAT;2, MAILF22.DAT;2, MAILJ14.DAT;1
   ```
   MAIL*.DAT;* or MAIL%%%.DAT
   ```

   d. A.DAT;1, MAILJ14.DAT;1
   ```
   *.DAT;1
   ```

5.28 WRITTEN EXERCISE II—SOLUTIONS

Commands

a. APPEND
b. COPY
c. DELETE
d. DELETE/CONFIRM
e. DIFFERENCES
f. DIRECTORY
g. DIRECTORY/OUTPUT=file–specification
h. PRINT
i. PURGE
j. RENAME
k. TYPE

Operations

1. __k__ Display the contents of a file at your terminal.
2. __f__ Display the contents of your default directory at your terminal.
3. __c__ Remove a specified file from your default directory.
4. __i__ Remove all but the most recent version of a specified file from your default directory.
5. __b__ Create an exact duplicate of a file in your default directory.
6. __h__ List the contents of a file at the default system printer.
7. __e__ Compare the contents of two files.
8. __a__ Add the contents of one file to another.
9. __j__ Change a directory name to a new directory name.
10. __d__ Display the name of each file in your default directory and remove or retain it by entering a "Y" or an "N" at your terminal.
11. __g__ List the contents of your default directory in a file for future reference.

5.29 WRITTEN EXERCISE III—SOLUTIONS

Commands

a. COPY
b. CREATE
c. CREATE/DIRECTORY
d. DELETE
e. DELETE/DIRECTORY
f. DIRECTORY
g. RENAME
h. SET DEFAULT
i. SET PROTECTION
j. SHOW DEFAULT
k. SHOW PROTECTION

Operations

1. __j__ Display the name of your current default directory.
2. __f__ Display the contents of a directory hierarchy.
3. __d__ Remove a directory from a directory hierarchy.
4. __c__ Add a directory to a directory hierarchy.
5. __g__ Move files from one directory to another.
6. __h__ Change your current default directory.
7. __i__ Change the protection code of a directory file.
8. __j__ Display the name of your current default device.
9. __h__ Change your current default device.

5.30 WRITTEN EXERCISE IV—SOLUTIONS

1. FALSE A user with a UIC of [100,200] wants to delete a file on a tape volume.

```
Volume Owner UIC:          [100,200]
Volume Protection Code:    (S:RWED,O:RWED,G:RWED,W:RE)
```

   Files on a tape volume cannot be deleted.

2. TRUE A user with a UIC of [363,2] wants to create a file on an RX33 disk volume.

```
Volume Owner UIC:          [363,0]
Volume Protection Code:    (S:RE,O:RWED,G:RE,W)
```

   The user is a member of the same group as the owner of the volume. Since group members have been granted EXECUTE rights, the user can create a new file.

3. TRUE A user with a UIC of [4,4] wants to read a file on an RA60 disk volume.

```
File Owner UIC:            [411,22]
File Protection Code:      (S,O:RWED,G,W:R)
```

   The user belongs to the SYSTEM user category. System users do not have READ access rights to the file. However, READ access rights have been granted to members of the WORLD category; therefore, the user will be able to read the file.

4. TRUE A user with a UIC of [100,200] wants to update a record in a file on an RA80 disk volume.

```
Volume Owner UIC:           [1,1]
Volume Protection Code:     (S:RWED,O;RWED,G:RWED,W:RWED)
Directory Owner UIC:        [100,210]
Directory Protection Code:  (S:RWE,O:RWE,G:RWE,W:RE)
File Owner UIC:             [100,210]
File Protection Code:       (S:RE,O:RWED,G:RWE,W:RE)
```

The user can access files on the volume because all access rights to the volume have been granted to all user categories. The user is a member of the same group as the owner of the file and the directory in which it is listed. Members of the GROUP category have been granted WRITE access rights; therefore, the user can update the file.

5. FALSE A user with a UIC of [521,6] wants to read a file on an RA81 disk volume.

```
Volume Owner UIC:           [1,1]
Volume Protection Code:     (S:RWED,O:RWED,G:RWED,W:RWED)
Directory Owner UIC:        [521,13]
Directory Protection Code:  (S:RWE,O:RWE,G,W)
File Owner UIC:             [521,13]
File Protection Code:       (S:R,O:RWED,G:R,W:R)
```

The user can access files on the volume because all access rights to the volume have been granted to all user categories. The user is a member of the same group as the owner of the file and the directory in which it is listed. Members of the GROUP category, however, cannot read the directory; therefore, the user will be unable to read the file.

5.31 LABORATORY EXERCISE I—SOLUTIONS

1. Create a subdirectory called [.SUB1].

```
$ CREATE/DIRECTORY [.SUB1]
```

2. Copy some files from your login directory into [.SUB1].

```
$ COPY/LOG EXISTING-FILE-NAMES [XXX.SUB1]*
```

3. Move yourself to that subdirectory.

```
$ SET DEFAULT [XXX.SUB1]
```

4. Obtain a directory listing of all files in the subdirectory.

```
$ DIRECTORY
```

5. Combine two files to create a new file named NEWFILE.DAT.

```
$ COPY FILE1,FILE2 NEWFILE.DAT
```

6. Create another subdirectory beneath [.SUB1] and name the new subdirectory [.SUB2].

```
$ CREATE/DIRECTORY [XXX.SUB1.SUB2]
or
$ CREATE/DIRECTORY [.SUB2] (Assuming you are in the subdirectory [.SUB1])
```

7. Copy some files from [.SUB1] into [.SUB2].

```
$ COPY EXISTING-FILE-NAMES [.SUB2]*
```

8. Obtain a directory listing of all files in the subdirectory [.SUB2].

```
$ DIRECTORY [.SUB2]
```

9. Delete both subdirectories.

```
$ DELETE *.*;* (Assuming you are in subdirectory [.SUB2]
$ SET DEFAULT [-.SUB1]
$ SET PROTECTION=(O:RWED) SUB2.DIR
$ DELETE *.*;*
$ SET DEF [-] (Login directory)
$ SET PROTECTION=(O:RWED) SUB1.DIR
$ DELETE SUB1.DIR;1
```

5.32 LABORATORY EXERCISE II—SOLUTIONS

1. Create a file in your login directory. What protection code does this newly created file have and how did it get that protection code?

```
$ CREATE MYFILE.TXT
Type in text
CTRL/Z
```

The protection applied to this file is the system default protection the VMS system puts on newly created files.

2. Change the protection code for this file to (S:R,O:R,G;R,W:R). Display the protection code to verify the change.

```
$ SET PROTECTION=(S:R,O:R,G:R,W:R) MYFILE.TXT
$ DIRECTORY/PROTECTION MYFILE.TXT
```

3. Delete this file. What happened and why?

```
$ DELETE MYFILE.TXT;*
```

The system issues a system message informing you that you cannot delete this file, because you changed the file protection so that the owner does not have DELETE privilege.

4. Change your DEFAULT protection code to (S:R,O:RWED,G:R,W:R).

```
$ SET PROTECTION=(S:R,O:RWED,G:R,W:R)/DEFAULT
```

Create a new file named NEWFILE.TXT. What protection code does this new file have and why?

```
$ CREATE NEWFILE.TXT
Type in text
CTRL/Z
$ DIRECTORY/PROTECTION NEWFILE.TXT
```

By changing your DEFAULT protection, you specify that all files now created will have this new default protection.

5. Change your DEFAULT protection to give all persons in your UIC group RWED access and all persons in the WORLD category RWE access.

```
$ SET PROTECTION=(G:RWED,W:RWE)/DEFAULT
```

5.33 LABORATORY EXERCISE III—SOLUTIONS

1. Choose a file in your directory. Issue a DCL command to obtain Access Control List information regarding that file.

```
$ DIRECTORY/SECURITY file-name
```

2. Modify the UIC protection on the above file so that your group has no access.

```
$ SET PROTECTION=(G) file-name
```

3. Modify the ACL information to allow Read, Write, and Execute access to the file.

```
$ EDIT/ACL file-name
(IDENTIFIER=xxxx,ACCESS=READ+WRITE+EXECUTE)
```

4. Check to see if an ACL was created. Have some of your fellow students try to access the file.

```
$ DIRECTORY/SECURITY file-name
```

5. Delete the ACL on the above file.

```
$ SET ACL/DELETE file-name
```

CUSTOMIZING THE USER ENVIRONMENT

6.1 INTRODUCTION

Previously, you learned to enter instructions to the operating system and to specify the locations of devices, directories, and files. The command strings and device and file specifications that perform these operations are sometimes lengthy. This complexity is an invitation to typographical and grammatical errors.

This module introduces *logical names* and demonstrates how to to use them in place of complicated device and file specifications in command strings. It also explains how to create and use symbols to tailor the command language.

Finally it describes how to define terminal keys to perform frequently used functions.

6.2 OBJECTIVES

- To make file references device–independent, you should be able to:
  - — Use the logical names the VMS system defines for all users
  - — Create and use logical names for file access
- To customize the command language for particular needs, you should be able to:
  - — Create and use symbols as command synonyms
  - — Use symbols as variables in DCL expressions
  - — Use symbols as an integral part of command procedures
- Finally, you should be able to define and use terminal keys to speed up execution of frequently used DCL commands.

6.3 RESOURCES

- *VMS DCL Dictionary*
- *VMS DCL Concepts Manual*

6.4 LOGICAL NAME ASSIGNMENTS

A *logical name* is a name you can use in place of all or part of a file specification. When you issue DCL commands, you can use logical names in place of the device name or file specification equated to the logical name. The system translates the logical name to the corresponding equivalence string(s).

Equivalence strings are usually the name of a device, some portion of a file specification, or another logical name.

Logical names are also used to pass data among programs, or between a command procedure and a program.

To create logical name assignments, use either the **ASSIGN** or **DEFINE** command. The format is:

$ DEFINE logical–name equivalence string[,...]

$ ASSIGN equivalence–name [,...] logical–name

Logical names and their equivalence strings can each have a maximum of 255 characters, including alphanumeric characters, the dollar sign (?), and the underscore (_). Any other characters must be enclosed in quotation marks.

6.4.1 Logical Name Tables

Logical names and their equivalence strings are stored in *logical name tables.* During a terminal session, your process has at least four logical name tables associated with it:

- *Process Logical Name Table* (LNM$PROCESS_TABLE)

  The contents of your process logical name table is known only to your current process. It contains process–private logical name assignments.

- *Job–Wide Logical Name Table* (LNM$JOB_YYYYYYYY, where YYYYYYYY is the number the system assigns to your job)

  The contents of your job–wide logical name table is known to your process tree, which is your login process and all of its subprocesses. It contains shareable logical name assignments.

- *Group Logical Name Table* (LNM$GROUP_000XXX, where XXX represents your UIC group number)

  The contents of the group logical name table is known to all users whose UIC group number matches your own. This also contains shareable logical name assignments. Privilege is needed to add logical names to this table.

- *System Logical Name Table* (LNM$SYSTEM_TABLE)

  The contents of the system logical name table is known to all processes on your system. This also contains shareable logical name assignments. Privilege is needed to add logical names to this table.

The logical names catalogued in your process logical name table are referred to as *process–private logical names*, while the logical names catalogued in the job–wide, group, and system logical name tables are known as *shareable logical names.*

Figure 6–1 illustrates the relationship between your terminal, the operating system, and the logical name tables associated with your process.

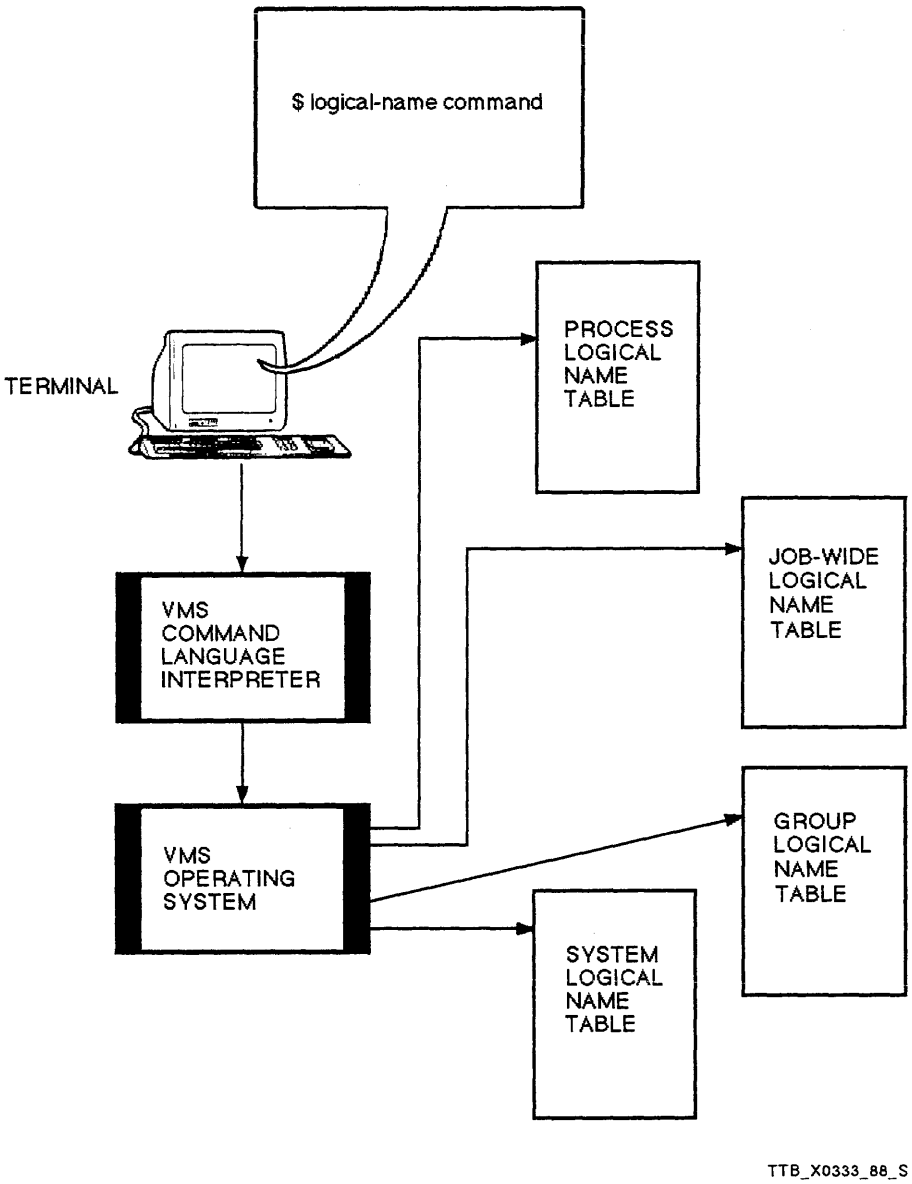


Figure 6–1 The Relationship Between Your Terminal, the Operating System, and the Logical Name Tables Associated with Your Processor

6.4.2 Common User Operations Dealing with Logical Names

As a user you may want to:

- Display the contents of logical name tables
- Determine the equivalence name of a logical name
- Add or override logical name assignments in your process logical name table
- Remove a logical name from your process logical name table

The following sections describe each of the above steps in detail.

6.4.2.1 Adding Logical Names

To add logical names, you can use either the **ASSIGN** or the **DEFINE** command. The **ASSIGN** command syntax is:

$ ASSIGN[/table-name][/mode-name]
_Device: equivalence-name[,...]
_Log name: logical-name[:]

Example:

```
$ ASSIGN DRA0:[SMITH.UANDC] MINE
```

The **DEFINE** command syntax is:

$ DEFINE[/table-name][/mode-name]
_Log name: logical-name[:]
_Equ name: equivalence-name[,...]

Example:

```
$ DEFINE MINE DRA0:[SMITH.UANDC]
```

Both examples above assign the logical name MINE to the disk DRA0, the directory name [SMITH], and the subdirectory name [.UANDC].

```
$ CREATE/DIRECTORY/LOG [SMITH.LOG]
%CREATE-I-CREATED, DISK:[SMITH.LOG] created

$ ASSIGN [SMITH.LOG] MY_LOG

$ COPY/LOG [SMITH]MYFILE.TXT MY_LOG
%COPY-S-COPIED, DISK:[SMITH]MYFILE.TXT;1 copied to
DISK:[SMITH.LOG]MYFILE.TXT;1 (1 block)

$ TYPE MY_LOG:MYFILE.TXT
This is a file for use in displaying the use of logical names
to abbreviate devices and file specifications.  This is in
the module entitled "Customizing the User Environment".
$
```

Example 6–1 Using Logical Names to Abbreviate Device and File Specifications

6.5 USING LOGICAL NAMES

6.5.1 Logical Name Translation

When the system encounters a file specification or device name in a command string, it performs logical name translation on it automatically. The following steps are performed by the system to translate logical names that have single equivalence strings.

- If any of the following conditions exist, logical name translation terminates:
  - — Ten translations have already been made (recursively).
  - — The left–most component of the specification is not delimited by a colon, a space, a comma, or an end of line.
  - — The equivalence string is a logical name that has the Terminal attribute. If a logical name as the Terminal attribute, the translation is completed after the first translation.
  - — If the logical name has the Concealed attribute, the translation normally displays the logical name for the device, rather than the physical name for the device.
- The system searches the Process, Job, Group, and System logical name tables in that order for the first logical name match. A logical name match occurs when:
  - — The part of the specification preceding the first delimiter matches an entry in a logical name table.
  - — The entire specification matches an entry in a logical name table.
- If a match occurs, the equivalence name replaces the logical name and step 1 is repeated. Otherwise, logical name translation terminates.

Both Terminal and Concealed are translation attributes. They are defined by using the **/TRANSLATION_ATTRIBUTES=** qualifier for either the **DEFINE** or **ASSIGN** commands. Refer to the *VMS DCL Dictionary* for further details.

6.6 RECURSIVE TRANSLATION

This section describes the steps that occur when the system does a recursive translation:

The following command is entered:

```
$ DIRECTORY SYS$LOGIN
```

The file specification passed to the **DIRECTORY** command consists of a single component SYS$LOGIN. The symbol contains no delimiters. When the system searches your logical name tables, it locates the following definition, which is in the format used by the **SHOW LOGICAL** command:

```
"SYS$LOGIN" = "DISK_USER:[SMITH]" (LNM$JOB_8016D000)
```

Given this assignment, the system substitutes the equivalence string, DISK_ USER:[SMITH], for the original specification SYS$LOGIN. Next, the system searches your logical name tables in an attempt to translate DISK_USER, the left–most portion of the revised specification. This search locates the physical device name DBA0: with the Terminal attribute and the Concealed attribute, as the equivalence string assigned to the logical name DISK_USER. The **SHOW LOGICAL** command displays this relationship as follows:

```
"DISK_USER" = "DBA0:" (LNM$SYSTEM_TABLE)
```

The system now substitutes the equivalence string DBA0: for the logical name DISK_USER to produce the file specification DBA0:[SMITH].

Since the logical name DISK_USER has the Terminal attribute assigned, further iterative translation is prevented. And, since the Concealed attribute is assigned, you are assured that DBA0: is the physical device name. In this case, the DIRECTORY image looks for files cataloged in DBA0:[SMITH] but uses DISK_USER:[SMITH] in the heading of its display. Concealed device names are useful when, as in this case, the logical name has more meaning to the user than the physical device name.

To complete the processing of the file specification, the system substitutes default values for fields that remain unspecified. In this example, no file name, file type, or version number has been specified. Since you have entered the **DIRECTORY** command, the system substitutes wildcard characters (*) in the remaining fields of the specification. It does this on the assumption that you want to list the contents of the directory file named [SMITH] on the device named DBA0:.

6.6.1 Sample Recursive Translation

- Command

```
$ DIRECTORY PROJECTS
```

- First table search (Looking for PROJECTS)

```
"PROJECTS" = "DISKUSER:[ELLEN]" (LNM$PROCESSTABLE)
```

- Second table search (Looking for DISKUSER)

```
"DISKUSER" = "DBA0:" (LNM$SYSTEMTABLE)
```

- Result

```
DBA0:[ELLEN] - Searched
```

Table 6–1 Commands for Displaying the Contents of Logical Name Tables

Example	Comments
`$ SHOW LOGICAL`	By default, displays logical names from the process, job–wide, group, and system logical name tables
`$ SHOW LOGICAL/FULL`	Displays all of the attributes of logical names from the process, job–wide, group, and system logical name tables
`$ SHOW LOGICAL/PROCESS`	Displays logical names from your process logical name table
`$ SHOW LOGICAL/JOB`	Displays logical names from your job–wide logical name table
`$ SHOW LOGICAL/GROUP`	Displays logical names from your group logical name table
`$ SHOW LOGICAL/SYSTEM`	Displays logical names from the system logical name table

```
$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

  "SYS$COMMAND" = "_DISK$RTA1:"
  "SYS$DISK" = "DISK:"
  "SYS$ERROR" = "_DISK$RTA1:"
  "SYS$INPUT" [super] = "_DISK:"
  "SYS$INPUT" [exec] = "_DISK$RTA1:"
  "SYS$OUTPUT" [super] = "_DISK$RTA1:"
  "SYS$OUTPUT" [exec] = "_DISK$RTA1:"
  "TT" = "RTA1:"
$ SHOW LOGICAL/JOB

(LNM$JOB_80392E4E40)

  "SYS$LOGIN" = "DISK:[HUBBARD]"
  "SYS$LOGIN_DEVICE" = "DISK:"
  "SYS$REM_ID" = "HUBBARD"
  "SYS$REM_NODE" = "WHYSO::"
  "SYS$SCRATCH" = "DISK:[HUBBARD]"
$ SHOW LOGICAL/GROUP

(LNM$GROUP_000011)

  "MY_DISK" = "DJAO:"
$ SHOW LOGICAL/SYSTEM

(LNM$SYSTEM_TABLE)

  "DBG$INPUT" = "SYS$INPUT:"
  "DBG$OUTPUT" = "SYS$OUTPUT:"
  "DISK$WHYNOT_SYS" = "DISK:"
  "SYS$ANNOUNCE" = "Welcome to WHYNOT "
  "SYS$COMMON" = "DISK:[SYS0.SYSCOMMON.]"
  "SYS$DISK" = "DISK:"
  "SYS$ERRORLOG" = "SYS$SYSROOT:[SYSERR]"
  "SYS$HELP" = "SYS$SYSROOT:[SYSHLP]"
  "SYS$MAINTENANCE" = "SYS$SYSROOT:[SYSMAINT]"
  "SYS$MANAGER" = "SYS$SYSROOT:[SYSMGR]"
  "SYS$MESSAGE" = "SYS$SYSROOT:[SYSMSG]"
  "SYS$NODE" = "WHYNOT::"
  "SYS$SYLOGIN" = "SYS$MANAGER:SYLOGIN.COM"
  "SYS$SYSDEVICE" = "DISK:"
  "SYS$SYSROOT" = "DISK:[SYS0.]"
        = "SYS$COMMON:"
  "SYS$SYSTEM" = "SYS$SYSROOT:[SYSEXE]"
  "SYS$UPDATE" = "SYS$SYSROOT:[SYSUPD]"
```

Example 6–2 Displaying the Contents of the Process, Job, Group, and System Logical Name Tables

6.7 DETERMINING THE EQUIVALENCE OF A LOGICAL NAME

There are two commands used to determine the equivalence of a logical name.

The command **SHOW LOGICAL logical–name** translates iteratively up to 10 levels until everything is resolved.

The command **SHOW TRANSLATION logical–name** displays the first equivalence string and then stops. No iteration is performed.

```
$ ASSIGN DJAO: DISK1

$ ASSIGN DISK1: MYNAME

$ SHOW TRANSLATION MYNAME
  MYNAME = "DISK1:"  (LNM$PROCESS_TABLE)
$ SHOW LOGICAL MYNAME
   "MYNAME" = "DISK1:" (LNM$PROCESS_TABLE)
1  "DISK1" = "DJAO:" (LNM$PROCESS_TABLE)
```

Example 6–3 Determining the Value of a Logical Name

6.8 DELETING LOGICAL NAMES

The **DEASSIGN** command deletes logical names from a particular logical name table. The qualifier **/ALL** can be used as well, depending upon how many logical names you want to delete.

The following table lists examples of deleting logical names.

Table 6–2 Commands for Deleting Logical Names

Format/Examples	Comments
$ DEASSIGN[/table-name] [logical-name]	
`$ DEASSIGN MYFILE`	Deletes the logical name MYFILE from your process logical name table.
`$ DEASSIGN/ALL`	Deletes all assignments that you have placed in your process logical name table.
`$ DEASSIGN/JOB`	Deletes a logical name in your job table.
`$ DEASSIGN/GROUP OURFILE`	Deletes the logical name OURFILE from your group logical name table. (This assumes you have GRPNAM privilege.)
`$ DEASSIGN/SYSTEM PAYROLL`	Deletes the logical name PAYROLL from the system logical name table. (This assumes you have SYSNAM privilege.)

Example 6–4 illustrates the manipulation of logical names.

```
$ ASSIGN DJA0: DISK1
$ ASSIGN DISK1:[HUBBARD] LOG
$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

  "DISK1" = "DJA0:"
  "LOG" = "DISK1:[HUBBARD]"
  "SYS$COMMAND" = "_DISK:"
  "SYS$DISK" = "DISK:"
  "SYS$ERROR" = "_DISK$RTA1:"
  "SYS$INPUT" [super] = "_DISK:"
  "SYS$INPUT" [exec] = "_DISK$RTA1:"
  "SYS$OUTPUT" [super] = "_DISK$RTA1:"
  "SYS$OUTPUT" [exec] = "_DISK$RTA1:"
  "TT" = "RTA1:"

$ ASSIGN DJA1: DISK1
%DCL-I-SUPERSEDE, previous value of DISK1 has been superseded

$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

  "DISK1" = "DJA1:"
  "LOG" = "DISK1:[HUBBARD]"
  "SYS$COMMAND" = "_DISK$RTA1:"
  "SYS$DISK" = "DISK:"
  "SYS$ERROR" = "_DISK$RTA1:"
  "SYS$INPUT" = "_DISK$RTA1:"
  "SYS$OUTPUT" [super] = "_DISK$RTA1:"
  "SYS$OUTPUT" [exec] = "_DISK$RTA1:"
  "TT" = "RTA1:"

$ DEASSIGN/ALL
$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

  "SYS$COMMAND" = "_DISK$RTA1:"
  "SYS$DISK" = "DISK:"
  "SYS$ERROR" = "_DISK$RTA1:"
  "SYS$INPUT" = "_DISK$RTA1:"
  "SYS$OUTPUT" [super] = "_DISK$RTA1:"
  "SYS$OUTPUT" [exec] = "_DISK$RTA1:"
  "TT" = "RTA1:"
$
```

Example 6–4 Assigning, Changing, and Deleting Logical Name Assignments

6.9 SYSTEM–DEFINED LOGICAL NAMES

When you log in, the system defines a number of logical names and stores them in your process logical name table. The system also creates a job–wide logical name table for your process and all of its potential subprocesses. A number of other logical names are defined and stored in the system logical name table at the time your system is initialized.

You can override these permanently or temporarily with the **ASSIGN** or **DEFINE** commands previously discussed.

Table 6–3 lists the logical name definitions you will use most often at your terminal.

Table 6–3 Process Logical Names Defined by the System

Logical Name	Equivalence Name
SYS$COMMAND	Original value of SYS$INPUT, equated to your terminal for interactive use and command procedures.
SYS$DISK	Default disk established at login. Can be changed by the SET DEFAULT command.
SYS$ERROR	Default device to which the system writes messages. For an interactive user, the system equates SYS$ERROR to the terminal.
SYS$OUTPUT	Default output devices. For an interactive user, SYS$OUTPUT is equated to the terminal.
SYS$INPUT	Default input device. For all interactive use, SYS$INPUT is equated to the terminal. For command procedures, it is equated to the command file.
TT	Default device name for your terminal in interactive mode and for the console in batch mode.

Table 6–4 Job Logical Names Defined by the System

Logical Name	Equivalence Name
SYS$LOGIN	Default disk and directory established at login time. This "home" directory is specified in the authorization record.
SYS$LOGIN_DEVICE	Default disk established at login. Unlike the logical name SYS$DISK, SYS$LOGIN_DEVICE is not changed by the SET DEFAULT command.
SYS$SCRATCH	Default device and directory to which temporary files are written. For all use this is equated to your default directory.

Table 6–5 System Logical Names Defined by the System

Logical Name	Equivalence Name
SYS$SYSTEM	Device and directory of operating system programs and procedures.
SYS$HELP	Device and directory name of system help files.
SYS$LIBRARY	Device and directory name of system libraries.
SYS$MESSAGE	Device and directory name of system message files.
SYS$SHARE	Device and directory name of system shareable images.
SYS$SYSDEVICE	VMS system disk, device referred to in the system logical names listed above.
SYS$NODE	Current network node name for the local system, if DECnet is active on the system.

6.9.1 SPECIFYING ACCESS MODES

You can specify a particular access mode by using **/USER_MODE** or **/SUPERVISOR_MODE** in conjunction with the **ASSIGN** or **DEFINE** commands.

User mode assignments last until the next image run in your process completes execution. (An image is a program in its executable form.)

Supervisor mode (the default mode) assignments are in effect until:

- You log out
- You assign the particular logical name to a different equivalence string
- You remove the logical name assignment by using the DCL command **DEASSIGN**

Note in Example 6–5 that SYS$OUTPUT, which by default is your terminal, has been redirected to the file named OUTPUT.LIS. The **SHOW PROCESS** command is issued, followed by the command **TYPE OUTPUT.LIS** to display the file OUTPUT.LIS at your terminal screen.

```
$ ASSIGN/USER_MODE OUTPUT.LIS SYS$OUTPUT
$ SHOW PROCESS
$ TYPE OUTPUT.LIS
22-OCT-1987 16:20:03.20   RTA1:              User: SMITH
Pid: 202001F8   Proc. name: SMITH          UIC: [GROUP11,SMITH]
Priority:   4   Default file spec: DISK:[SMITH]
Devices allocated: DISK$RTA1:
$
```

Example 6–5 Using Logical Names to Alter the Default Output Device of Your Process

6.9.2 OVERRIDING DCL TABLE NAMES

By default, both the **ASSIGN** and the **DEFINE** commands direct the system to make a logical name assignment in your process logical name table. To override this default, and make a logical name assignment in a table other than your process logical name table, use one of the following command qualifiers:

- **/PROCESS** (Default)
- **/JOB**
- **/GROUP** (requires GRPNAM privilege)
- **/SYSTEM** (requires SYSNAM privilege)

As mentioned earlier, logical name assignments in your process logical name table are known as process–private logical names. By default, logical name assignments in your user–defined logical name tables are also process–private logical names. Process–private logical names are in effect until you log out, unless you explicitly delete them. The shareable logical name assignments stored in your group and system logical name tables are not deleted when you log out.

Table 6–6 describes the commands used for defining logical names.

Table 6–6 Commands for Defining Logical Names

Operation	Format/Example	Comments
Add or modify a logical name assignment	**$ ASSIGN [/mode–name]** **_Device: equivalence–name[,...]** **_log name: logical–name[:]** `$ ASSIGN DMA0: DISK`	Assigns the equivalence string DMA0: to the logical name DISK. By default, the system stores the assignment in your process logical name table. (Note that you include the colon in the equivalence string, since it is part of the device name.)
	$ DEFINE [/table–name] [/mode–name] **_log name: logical–name[:]** **_equ name: equivalence–name[,...]** `$ DEFINE DISK DMA0:`	Assigns the equivalence string DMA0: to the logical name DISK. By default, the system stores the assignment in your process logical name table.

Table 6–7 describes the commands used for displaying logical names.

Table 6–7 Commands for Displaying Logical Names

Operation	Format/Example	Comments
Determine the value of a logical name at command level	**$ SHOW LOGICAL logical-name** `$ SHOW LOGICAL MYFILE`	Displays the translations for the logical name MYFILE found in the process, job–wide, group, and and system logical name tables.
Determine the value of a logical name using a wildcard	`$ SHOW LOGICAL SYS*`	Displays the translations for all all logical names beginning with SYS in the process, job–wide, group, and system logical name tables, because of the use of the wildcard (*).
Determine the value of a logical name during an image interrupt	**$ SHOW TRANSLATION logical-name** `$ SHOW TRANSLATION DISK`	Displays the equivalence string associated with the first occurrence of DISK in your tables. (By default, the SHOW TRANSLATION command searches process, job–wide, group, and system tables, in that order.)

6.10 USING DCL SYMBOLS

In addition to supporting logical names (which represent device, directory, and file specifications) the operating system supports a second facility for creating alternate names for commands or portions of command strings. Using a special command, called the *assignment statement (=)*, you can assign values to symbols and store these assignments in special tables called *symbol tables*. You can use symbols created in this way as variables in command procedures and as command synonyms.

The **Writing Command Procedures** module discusses the use of symbols as variables. This module covers the use of symbols as command synonyms.

The left side of the assignment statement defines the *symbol name*. The right side of the assignment statement contains an expression that can be either an integer value or a character string.

A symbol name can have a maximum of 255 characters. The symbol name must begin with a letter (A through Z), an underscore (_), or a dollar sign ($). After the first character, the name can contain any alphanumeric characters, underscores, and dollar signs. For example:

```
$ COUNT = 10
$ RESULT == 100
```

A *command synonym* is a name that represents character strings or numeric values. The portion of a command string equated to a command synonym is called its *equivalence string*.

The *global symbol table* typically stores command synonyms and their equivalence strings. Each time you log in, the system creates both a global symbol table and a local symbol table for your process. You define, display, and delete command synonyms by entering DCL commands at your terminal. Figure 6–2 shows the relationship between your terminal, the operating system, and the symbol tables associated with your process. By defining command synonyms, you can create your own command language and simplify your terminal sessions.

DCL places local symbols in the *local symbol table*. A local symbol exists as long as the command level remains active. A local symbol is defined by using a single equal sign (=) in the assignment statement:

```
$ TEST = "HELLO"
```

Global symbols are placed in the global symbol table. A global symbol exists for the duration of the process, unless it is specifically deleted. A global symbol is defined by using two equal signs (==) in the assignment statement:

```
$ RESULT == 50
```

You can also abbreviate the symbol forms by use of the asterisk (*) as the abbreviation indicator. For example, to abbreviate a local symbol to invoke the Mail utility:

```
$ M*AIL = "MAIL"
```

The Mail utility will be invoked using any of these abbreviations: "M", "MA", "MAI", and "MAIL".

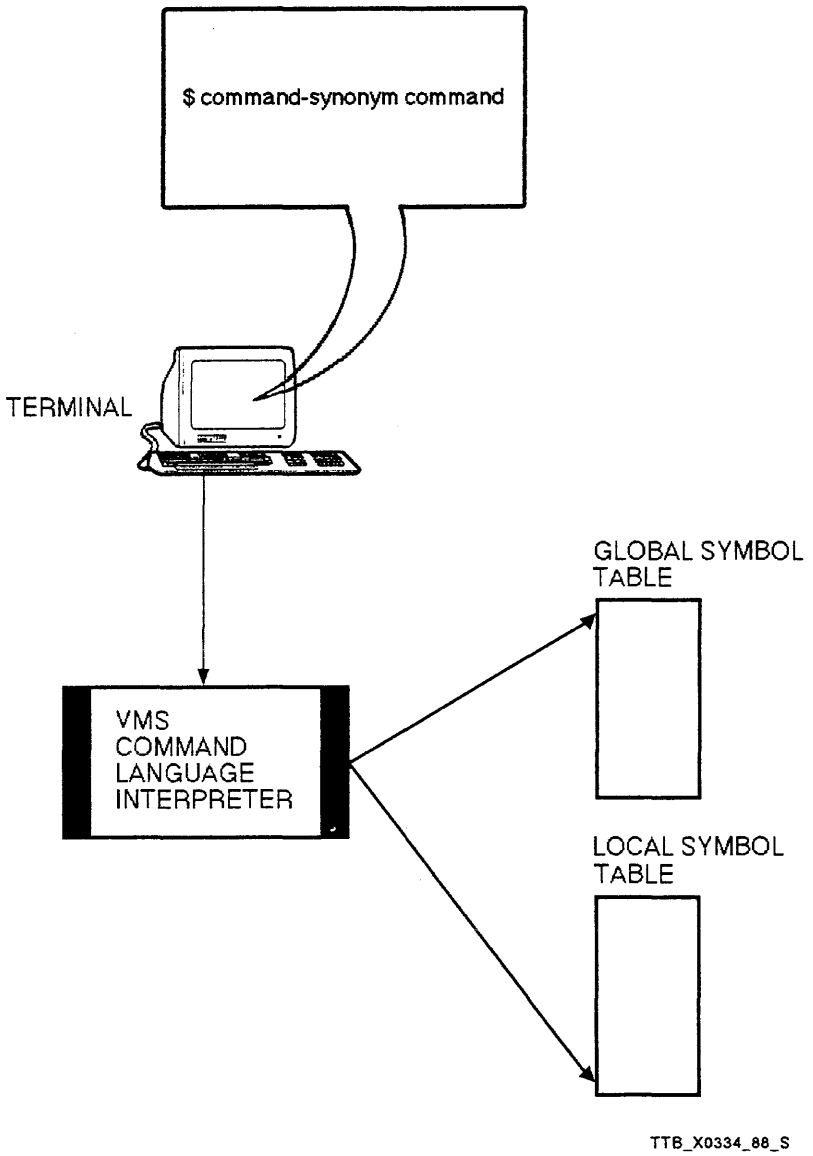


Figure 6–2 The Relationship Between Your Terminal, the Operating System, and Your Global Symbol Table

6.10.1 Deleting Symbol Definitions

Symbol definitions are deleted from a symbol table by using the **DELETE/SYMBOL [symbol–name]** command. If you do not specify the name of a symbol table the symbol definition is deleted from the local symbol table.

There are three qualifiers that can be used in conjunction with the **DELETE/SYMBOL** command. They are:

- **/LOCAL** – Specifies that the symbol name is to be deleted from the local symbol table.
- **/GLOBAL** – Specifies that the symbol name is to be deleted from the global symbol table.
- **/ALL** – Specifies that all symbol names in the specified symbol table be deleted. If neither **/LOCAL** or **/GLOBAL** are specified, all symbols defined at the current command level are deleted.

Table 6–8 describes the commands used for defining, displaying, and deleting symbols.

Table 6–8 Commands for Defining, Displaying, and Deleting DCL Symbols

Operation	Format/Example	Comments
Defining a DCL Symbol	**$ symbol-name == "equiv-string"** `$ TO == "SET DEFAULT"`	Assigns the equivalence string SET DEFAULT to the symbol TO, storing the definition in your global symbol table
Displaying Symbols		
Displaying a Single Symbol	**$ SHOW SYMBOL/GLOBAL symbol-name** `$ SHOW SYMBOL/GLOBAL TO`	Displays the value of the symbol TO
Displaying All Symbols	`$ SHOW SYMBOL/GLOBAL/ALL`	Displays the values of all symbols defined in your global symbol table
Displaying All Symbols Using a Wildcard	`$ SHOW SYMBOL/GLOBAL S*`	Displays the values of all symbols defined in your global symbol table beginning with the letter "s"
Deleting Symbols		
Deleting a Single Symbol	**$ DELETE/SYMBOL/GLOBAL symbol-name** `$ DELETE/SYMBOL/GLOBAL TO`	Deletes the symbol TO from your global symbol table
Deleting All Symbols	`$ DELETE/SYMBOL/GLOBAL/ALL`	Deletes all symbols from your global symbol table

```
$ DIRP == "DIRECTORY/OWNER/PROTECTION"
$ TO == "SET DEFAULT"
$ RETURN == "SET DEFAULT SYS$LOGIN"
$ LOCALS == "SHOW SYMBOL/LOCAL/ALL"
$ GLOBALS == "SHOW SYMBOL/GLOBAL/ALL"
$ DEL_SL == "DELETE/SYMBOL/LOCAL"
$ DEL_SG == "DELETE/SYMBOL/GLOBAL"
$ GLOBALS
  $RESTART == "FALSE"
  $SEVERITY == "1"
  $STATUS == "%X00030001"
  C*LEAR == "RUN SYS$SYSTEM:ERASE"
  DEL_SG == "DELETE/SYMBOL/GLOBAL"
  DEL_SL == "DELETE/SYMBOL/LOCAL"
  DIRP == "DIRECTORY/OWNER/PROTECTION"
  GLOBALS == "SHOW SYMBOL/GLOBAL/ALL"
  LOCALS == "SHOW SYMBOL/LOCAL/ALL"
  RETURN == "SET DEFAULT SYS$LOGIN"
  TO == "SET DEFAULT"
$ TO SYS$SYSTEM
$ DIRP DCL.EXE
Directory SYS$COMMON:[SYSEXE]

DCL.EXE;1            [SYSTEM]              (RWED,RWED,RWED,RE)

Total of 1 file.

$ RETURN
$ DEL_SG/ALL
$ GLOBALS
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \GLOBALS\
```

Example 6–6 Defining, Displaying, and Deleting Symbols

Table 6–9 compares various aspects of logical names and symbols.

Table 6–9 Comparison of Logical Names and DCL Symbols

Characteristic	Logical Names	Symbols
Function	Represent device, directory, and file specifications.	Represent commands or portions of command strings.
Usage	Used in place of any complete device, directory, or file specification. Also used in place of any contiguous group of left–hand fields in a file specification. Must occur as part of a command string parameter to be passed to the file system for translation.	Used in place of any complete command string. Also used in place of any left–hand portion of a command string. Must occur as the first word in a command string to be translated by the command language interpreter.
Storage	Stores assignments in your process, job, group, or system logical name table.	Stores assignments in your global or local symbol table.
Definition	Either the ASSIGN or DEFINE command	An assignment statement (=) or (==).
Display	Either the SHOW LOGICAL or SHOW TRANSLATION command.	The SHOW SYMBOL command.
Deletion	The DEASSIGN command.	The DELETE/SYMBOL command.

6.11 DEFINING KEYS

As you become more familiar with the VMS system, you will find that you are repeatedly entering certain commands. You can reduce the amount of typing required to one keystroke. Use the **DEFINE/KEY** command, which assigns a definition to a keyboard key. Definitions often contain part or all of a DCL command string.

The types of terminals and their associated definable keys are:

- VT52–series terminals – All definable keys located on the numeric keypad
- VT100–series terminals – All keys located on the numeric keypad plus the LEFT and RIGHT ARROW keys
- Terminals with LK201 keyboards – All keys on the numeric keypad; keys on the editing keypad (except the UP and DOWN ARROW keys); keys on the function key row across the top of the keyboard (except Function keys F1 through F5)

Keys KP0 – KP9, PERIOD, COMMA, and MINUS must be enabled for definition purposes. These keys are enabled by using either of the following commands:

```
$ SET TERMINAL/APPLICATION_KEYPAD
$ SET TERMINAL/NONUMERIC
```

Keypad keys PF1 – PF4 can also be defined.

The format is:

$ DEFINE/KEY key–name equivalence string /qualifiers

One or more of the following qualifiers can be used to alter the action of a defined key:

- **/TERMINATE** – Produces an automatic return
- **/NOECHO** – Suppresses the display of the command being invoked
- **/ERASE** – Erases the characters on the current line before displaying and executing the command invoked by the defined key
- **/NOLOG** – Suppresses the informational message you receive when you initially define a key

Example:

```
$ DEFINE/KEY PF3 "SHOW PROCESS" /TERMINATE
```

Now that key PF3 has been defined. Each time you press that key, the DCL command **SHOW PROCESS** will be executed. Note that the qualifier **/TERMINATE** is used to signify that the equivalent string (**SHOW PROCESS**) will be executed when key PF3 is pressed.

6.11.1 Displaying a Key Definition

To display a key definition, issue the DCL command:

```
$ SHOW KEY/FULL key-name
```

Example:

```
$ SHOW KEY/FULL PF3
  PF3 = "show process"
  (echo,terminate,noerase,nolock)
```

6.11.2 Removing a Key Definition

To remove a key definition, use the **DELETE/KEY** command. The following command removes the definition applied to the PF3 key:

```
$ DELETE/KEY PF3
    %DCL-I-DELKEY, HOME key PF3 has been deleted
```

You can use the qualifier **/ALL** to delete all key definitions in the current state. If you use the qualifier **/ALL**, do not specify a key name.

```
$ DELETE/KEY/ALL
```

If keys PF1 through PF4 had been defined in the DEFAULT state, you would get a message on the terminal screen stating that keys PF1 through PF4 had been deleted.

The following section describes how you can add different states to a key definition.

6.11.3 Assigning Multiple Definitions to Keys

You can assign any number of definitions to a key by assigning each definition to a different key state. For example:

```
$ DEFINE/KEY PF1 "SHOW " SETSTATE=GOLD/NOTERMINATE/ECHO
$ DEFINE/KEY PF1 " PROCESS" /TERMINATE/IF_STATE=GOLD/ECHO
```

The first time you press the PF1 key, the system echoes the string "SHOW " at your terminal and sets the state of the PF1 key to GOLD. The second time you press the PF1 key, the system:

- Tests the key state (GOLD)
- Displays the string " PROCESS" (Appends the string " PROCESS" to the string "SHOW ")
- Passes the entire command to the command language interpreter for processing

NOTE
If a key is defined and the qualifier SET_STATE is not used, then the key state is DEFAULT.

```
$ DEFINE/KEY PF1 "SHOW " /SET_STATE=GOLD/NOTERMINATE/ECHO
$ DEFINE/KEY PF1 " PROCESS" /TERMINATE/IF_STATE=GOLD/ECHO
```

Example 6–7 Defining Multiple Definitions for One Key

Notes on Example 6–7:

1. The qualifier **/SET_STATE=state-name** causes the specified state name to be set. The state name can be any alphanumeric string. In this case, the state name is GOLD.
2. The qualifier **/IF_STATE=state-name** determines whether or not a previously defined state name is in effect, in order for the key definition to be in effect.
3. The qualifier **/NOTERMINATE** allows you to create key definitions that insert text in command lines. In this particular example, using the **/TERMINATE** qualifier would have the first key definition echo the word "SHOW", but not the following word "PROCESS".
4. The qualifier **/ECHO** determines whether the equivalence string is displayed on the terminal screen when the key is pressed.

6.12 SUMMARY

- A logical name is a name you can use in place of all or part of a file specification
- They are used to:
  - — Achieve device and file independence in programs or procedures
  - — Reduce typing and improve readability (used as replacement for long file specifications)
  - — Pass data among programs, or between a command procedure and a program
- Logical names and their equivalence strings can each have a maximum of 255 characters (including alphanumeric characters, dollar signs and underscores)
- Any other characters must be enclosed in quotation marks
- Stored in logical name tables

System Defined Logical Names

When you log in, the system:

- Defines a number of logical names and stores them in your process logical name table
- Creates a job–wide logical name table for your process and all of its potential subprocesses

You may override these permanently or temporarily with the **ASSIGN** or **DEFINE** commands

DCL Symbols

- Symbols are names that represent character strings or numeric values
- Can be used as DCL command synonyms allowing the user to tailor DCL command format
- Equated to an equivalence string (which is enclosed in " ")
  - — Complete command string
  - — Portion of a command string
- Stored in one of two tables (each process has its own)
  - — LOCAL
  - — GLOBAL

Defining Keys

- Definitions often contain part or all of a DCL command string
- Reduces typing of lengthy or frequently used DCL commands

Syntax:

$ DEFINE/KEY key–name equivalence string /qualifiers

To display a key definition, issue the DCL command:

SHOW KEY/FULL key–name

To delete a key definition, issue the DCL command:

DELETE/KEY key–name

6.13 WRITTEN EXERCISE I

Write the letter of the system–defined logical name that best fits each of the device and directory descriptions on the following page. Some answers require more than one letter.

System–Defined Logical Names

a. SYS$COMMAND

b. SYS$DISK

c. SYS$ERROR

d. SYS$HELP

e. SYS$INPUT

f. SYS$LIBRARY

g. SYS$LOGIN

h. SYS$NODE

i. SYS$OUTPUT

j. SYS$SYSDEVICE

k. SYS$SYSTEM

Device and Directory Descriptions

1. _____ Specifies the default device to which the system writes output during a terminal session.
2. _____ Specifies the default device to which the system writes messages during a terminal session.
3. _____ Specifies your default disk.
4. _____ Specifies the directory in which help files are cataloged.
5. _____ Specifies the directory in which system libraries are cataloged.
6. _____ Specifies your default user file directory (UFD).
7. _____ Specifies the device from which the command language interpreter and utility programs read input during a terminal session.
8. _____ Specifies the directory in which operating system programs and procedures are cataloged.
9. _____ Specifies your terminal during an interactive process.
10. _____ Specifies the disk on which system programs and routines are stored.
11. _____ Specifies the name of the current network node.

6.14 WRITTEN EXERCISE II

Write the letter of the symbolic name type that best fits each of the following characteristics.

Symbolic Name Type

a. Command Synonym

b. Logical Name

Characteristic

1. _____ Represents device, directory, and file specifications
2. _____ Translated by the file system
3. _____ Translated by the command language interpreter
4. _____ Defined by the direct assignment statement (=)
5. _____ Deleted by the DEASSIGN command
6. _____ Displayed by the SHOW SYMBOL command
7. _____ Defined by the ASSIGN command
8. _____ Represents commands and command strings
9. _____ Deleted by the DELETE/SYMBOL command

6.15 LABORATORY EXERCISE I

Complete each of the following exercises at an interactive terminal. Display only one logical name table for each exercise.

1. Display at your terminal the contents of the logical name table used by your process. This particular logical name table contains process–private logical names.
2. Display at your terminal the contents of the logical name table used by your process and its subprocesses. This particular logical name table contains shareable logical names.
3. Display at your terminal the contents of the logical name table used by your UIC group member processes. This particular logical name table contains shareable logical names.
4. Display at your terminal the contents of the logical name table used by all system processes. This particular logical name table contains shareable logical names.
5. Create a logical name for your default directory.
    a. Check the proper logical name table to make sure your newly created logical name exists
    b. Use the logical name in conjunction with the **DIRECTORY** command to view the file names in your default directory
    c. Delete your newly created logical name after correctly performing this exercise.

6.16 LABORATORY EXERCISE II

Complete each of the following laboratory exercises at an interactive terminal.

1. Create a subdirectory
2. Create a logical name for a text file in your default directory
3. Create a logical name for your newly created subdirectory
4. Using only logical names, move the text file into your new subdirectory
5. After completing this exercise, remove the above logical names

6.17 LABORATORY EXERCISE III

Create global symbols to perform the following tasks. You can create these global symbols interactively or in the file LOGIN.COM.

1. Display a directory and size of all files in your directory
2. Show the time of day
3. Display all global symbols at your terminal
4. Move to another default directory
5. Return to your original default directory

To correct mistakes you may have made when you defined a DCL symbol, use the **DELETE/SYMBOL** command to remove the faulty definition, then enter it again.

6.18 WRITTEN EXERCISE I—SOLUTIONS

Write the letter of the system–defined logical name that best fits each of the device and directory descriptions on the following page. Some answers require more than one letter.

System–Defined Logical Names

a. SYS$COMMAND

b. SYS$DISK

c. SYS$ERROR

d. SYS$HELP

e. SYS$INPUT

f. SYS$LIBRARY

g. SYS$LOGIN

h. SYS$NODE

i. SYS$OUTPUT

j. SYS$SYSDEVICE

k. SYS$SYSTEM

Device and Directory Descriptions

1. i Specifies the default device to which the system writes output during a terminal session.
2. c Specifies the default device to which the system writes messages during a terminal session.
3. b Specifies your default disk.
4. d Specifies the directory in which help files are cataloged.
5. f Specifies the directory in which system libraries are cataloged.
6. g Specifies your default user file directory (UFD).
7. e Specifies the device from which the command language interpreter and utility programs read input during a terminal session.
8. k Specifies the directory in which operating system programs and procedures are cataloged.
9. a,c,e,i Specifies your terminal during an interactive process.
10. j Specifies the disk on which system programs and routines are stored.
11. h Specifies the name of the current network node.

6.19 WRITTEN EXERCISE II—SOLUTIONS

Write the letter of the symbolic name type that best fits each of the following characteristics.

Symbolic Name Type

a. Command Synonym

b. Logical Name

Characteristic

1. __b__ Represents device, directory, and file specifications
2. __b__ Translated by the file system
3. __a__ Translated by the command language interpreter
4. __a__ Defined by the direct assignment statement (=)
5. __b__ Deleted by the DEASSIGN command
6. __a__ Displayed by the SHOW SYMBOL command
7. __b__ Defined by the ASSIGN command
8. __a__ Represents commands and command strings
9. __a__ Deleted by the DELETE/SYMBOL command

6.20 LABORATORY EXERCISE I—SOLUTIONS

Complete each of the following exercises at an interactive terminal. Display only one logical name table for each exercise.

1. Display at your terminal the contents of the logical name table used by your process. This particular logical name table contains process–private logical names.

```
$ SHOW LOGICAL/PROCESS
```

2. Display at your terminal the contents of the logical name table used by your process and its subprocesses. This particular logical name table contains shareable logical names.

```
$ SHOW LOGICAL/JOB
```

3. Display at your terminal the contents of the logical name table used by your UIC group member processes. This particular logical name table contains shareable logical names.

```
$ SHOW LOGICAL/GROUP
```

(There may not be any logical names defined in this table.)

4. Display at your terminal the contents of the logical name table used by all system processes. This particular logical name table contains shareable logical names.

```
$ SHOW LOGICAL/SYSTEM
```

5. Create a logical name for your default directory.

```
$ ASSIGN WORK2:[SMITH] MYDIR
```

a. Check the proper logical name table to make sure your newly created logical name exists

```
$ SHOW LOGICAL MYDIR
"MYDIR" = "WORK2:[SMITH]"  (LNM$PROCESS_TABLE)
```

b. Use the logical name in conjunction with the DIRECTORY command to view the file names in your default directory

```
$ DIRECTORY MYDIR

Directory WORK2:[SMITH]

CALENDAR.EXE;1     CLASS.LIST;4      CLOCK.EXE;1      DEG.EXE;1
JOE_EVE.TPU$SECTION;1                KEYS.COM;5       LOGIN.COM;6
MAIL.DIR;1         PERSONAL.LGP;4    REMLOG.EXE;1     TODO.DAT;17
UTL.DIR;1          WEEKDAY.EXE;1

Total of 13 files.
```

c. Delete your newly created logical name after correctly performing this exercise.

```
$ DEASSIGN/ALL
```
or `$ DEASSIGN/PROCESS MYDIR`

6.21 LABORATORY EXERCISE II—SOLUTION

Compare your results with the following example.

```
$ CREATE/DIRECTORY/LOG [SMITH.TEXT]
%CREATE-I-CREATED, DISK:[SMITH.TEXT] created

$ ASSIGN [SMITH.TEXT] MY_TEXT
$ ASSIGN MYFILE.TXT;1 OUTPUT

$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

  "MY_TEXT" = "[SMITH.TEXT]"
  "OUTPUT" = "MYFILE.TXT;1"
  "SYS$COMMAND" = "_DISK$RTA1:"
  "SYS$DISK" = "DISK:"
  "SYS$ERROR" = "_DISK$RTA1:"
  "SYS$INPUT" = "_DISK$RTA1:"
  "SYS$OUTPUT" [super] = "_DISK$RTA1:"
  "SYS$OUTPUT" [exec] = "_DISK$RTA1:"
  "TT" = "RTA1:"

$ COPY/LOG OUTPUT MY_TEXT
%COPY-S-COPIED, DISK:[SMITH]MYFILE.TXT;1 copied to
 DISK:[SMITH.TEXT]MYFILE.TXT;1 (1 block)

$ DEASSIGN OUTPUT
$ DEASSIGN MY_TEXT
$
```

6.22 LABORATORY EXERCISE III—SOLUTIONS

Create global symbols to perform the following tasks. You can create these global symbols interactively or in the file LOGIN.COM.

1. Display a directory and size of all files in your directory

```
$ DS == "DIRECTORY/SIZE"
```

2. Show the time of day

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

3. Display all global symbols at your terminal

```
$ GLO == "SHOW SYMBOL/GLOBAL/ALL"
```

4. Move to another default directory

```
$ MOVE == "SET DEFAULT"
```

5. Return to your original default directory

```
$ RETURN == "SET DEFAULT SYS$LOGIN"
```

WRITING COMMAND PROCEDURES

7.1 INTRODUCTION

As you become a more experienced user, you will begin to notice that certain commands are used repeatedly, in exactly the same order. Typing in these commands at the terminal can be tedious and time–consuming.

Command procedures provide you with a means to execute these commands automatically.

Command procedures are files consisting of DCL commands. In addition to the command verbs, qualifiers and parameters commonly used at the interactive level, there are several DCL command language features that provide power and flexibility, including:

- Symbols that can be used as numeric and string variables
- Instructions that allow you to control program flow
- Lexical functions

This module presents the material needed to create, test, and run a command procedure interactively. The **Submitting Batch and Print Jobs** module shows you how to run command procedures independently of your interactive process, as *batch jobs*.

7.2 OBJECTIVES

To write DCL command procedures, you should be able to:

- Define what a command procedure is and describe why command procedures are used
- Create a command procedure, using standard DCL command elements
- Control terminal input and output in a command procedure by:
  - — Displaying messages on the terminal
  - — Accepting input from the user
  - — Redirecting input or output from the terminal to another location
- Pass data to a command procedure using parameters
- Control the flow of execution within a command procedure using:
  - — The **IF** command
  - — The **GOTO** command
- Use the proper lexical function to obtain the information needed in a command procedure

7.3 RESOURCES

- *Guide to Using VMS Command Procedures*
- *VMS DCL Dictionary*

7.4 COMMAND PROCEDURES

A *command procedure* is a file containing DCL command strings. Command strings in a command procedure are made up of DCL command *verbs*, command *qualifiers*, and *parameters*.

In addition to DCL commands, command procedures frequently make use of DCL command language features such as:

- *Symbols*, which can be used as command synonyms, or as numeric or string variables
- *Control flow statements*, similar to branching commands in programming languages
- *Lexical functions*, which provide information about the system, processes, and files

7.4.1 Common Uses

Command procedures have several uses. One use is the repeated execution of a group of instructions. Instead of entering the commands at the interactive level, you can execute the commands in a command procedure, saving time and typing. A command procedure can be used by several people, ensuring consistency of action, and reducing duplication of effort.

Command procedures are also helpful when you must use long or complicated command strings. Command procedures help assure that the syntax of these commands is correct.

Finally, command procedures are used when you want to run a job in batch mode. When you submit a command procedure to be run in batch mode, you free your terminal for other tasks. The use of batch queues is discussed in the **Submitting Batch and Print Jobs** module later in this course.

7.4.2 Developing a Command Procedure

The steps to develop a command procedure are similar to steps used to develop a computer program in any language.

1. Design the command procedure.
   - Determine what tasks the procedure should perform.
   - Decide what results the procedure should produce.
2. Create the command procedure.
   - Use the text editor of your choice.
   - Specify the file type COM for the command procedure.
3. Execute and test the command procedure.
   - Use the "at sign" (@) to execute the procedure interactively.
   - Use the DCL command **SET VERIFY** to:
     — Display each line of the procedure as it executes.
     — Help locate errors if they occur.
4. Modify and retest the command procedure, if necessary.
   - Repeat steps 2 and 3.
   - Use the DCL command **SET NOVERIFY** after the procedure has been tested and perfected.
5. Add comments to the command procedure so it is easy to read and maintain. Comments should describe:
   - The procedure in detail.
   - Any parameters that are passed to the procedure.

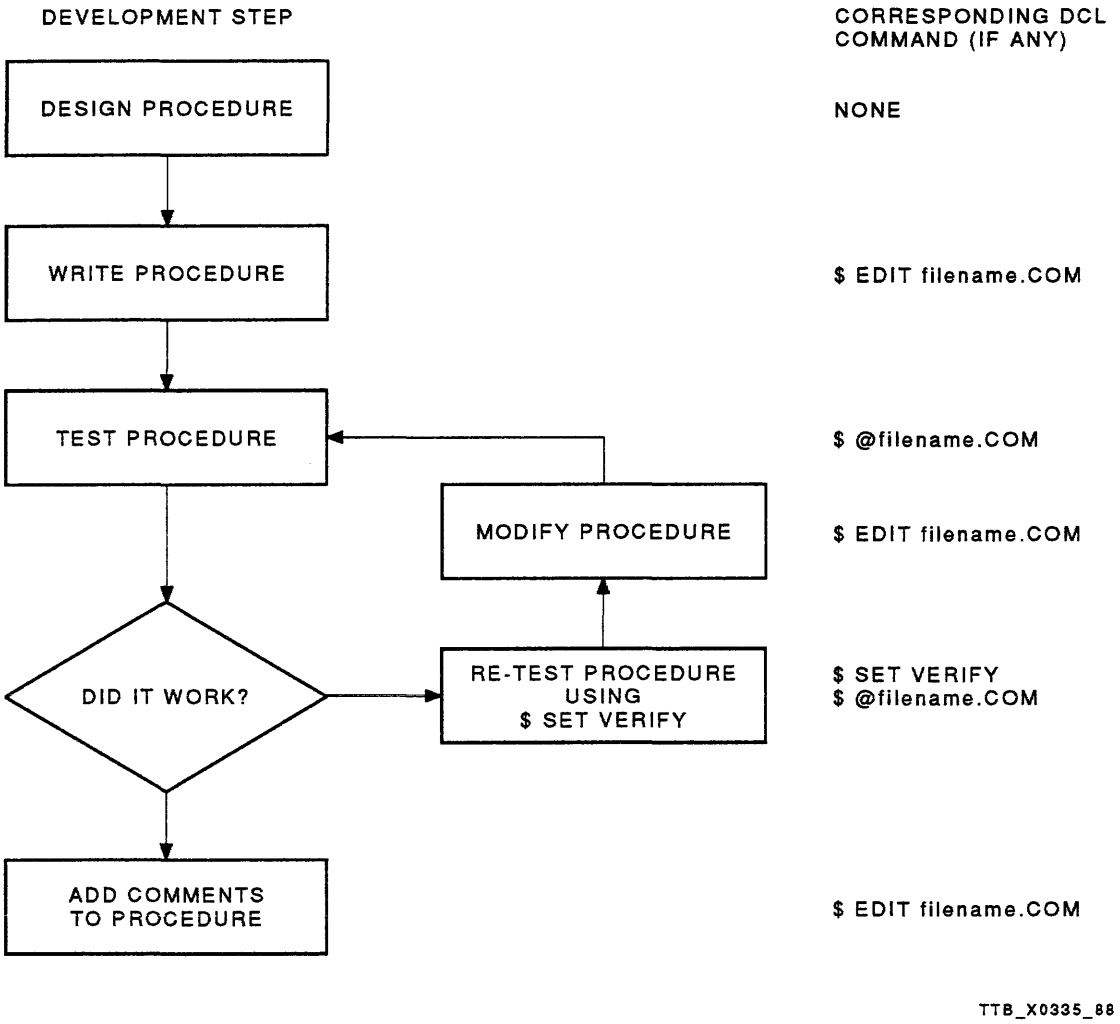


Figure 7–1 Command Procedure Development Process

7.5 COMPONENTS AND CONVENTIONS

Consistent format and clear programming style make your command procedures easy to read, test, and maintain.

7.5.1 DCL Command Lines

Always use full command verbs. Do not use abbreviations. Abbreviated commands may become ambiguous if a new DCL command verb is added.

Precede each command line with the $ prompt. Although you can change your DCL prompt at the interactive level, the dollar sign prompt is still used to indicate a command line within a procedure.

You can continue a line by placing a hyphen at the end of the line. Do NOT begin the continued line with a dollar sign.

7.5.2 Data Lines

Data lines can be used to include information used by the procedure or by another utility. When data lines are used, they are placed immediately after the command that will use them. Do NOT place a dollar sign at the beginning of a data line. Data input is terminated by the first occurrence of a dollar sign, and control is transferred back to the command procedure.

7.5.3 Comments

An exclamation point (!) indicates a comment. When the system encounters an exclamation point it immediately goes on to the next line. Everything following the exclamation point is ignored.

It is recommended that you use blank comment lines ($!) to separate blocks of instructions.

7.5.4 Labels

Labels are the names of locations within the command procedure. Like DCL commands, labels must be preceded by a $. In addition, labels must be followed by a colon (:). For better readability, no command should be placed on the same line as the label.

```
$ !                                              REPORT1.COM
$ !
$ !
$ ! This command procedure sets your default directory
$ ! to the REPORTS.MONDAY subdirectory, prints out a report for
$ ! Monday, returns you to your login device and
$ ! directory, then exits.
$ !
$ !
$ ! Set your default to the REPORTS.MONDAY subdirectory
$ !
$   SET DEFAULT DISK1:[REPORTS.MONDAY]
$ !
$ ! Print out the report for Monday
$ !
$   PRINT MONDAY.RPT
$ !
$ ! Return to your login device and directory
$ !
$   SET DEFAULT SYS$LOGIN
$   EXIT
```

Execution of REPORT1.COM:

```
$ @REPORT1
DISK1:[REPORTS.MONDAY]
Job MONDAY (queue SYS$PRINT, entry 44) started on WORK$TXA0
```

Example 7–1 A Sample Command Procedure

Now try it with VERIFY turned on:

```
$ SET VERIFY
$
$ @REPORT1
$ !                                          REPORT1.COM
$ !
$ !
$ ! This command procedure sets your default directory
$ ! to the REPORTS.MONDAY subdirectory, prints out a report for
$ ! Monday, returns you to your login device and
$ ! directory, then exits.
$ !
$ !
$ ! Set your default to the REPORTS.MONDAY subdirectory
$ !
$   SET DEFAULT DISK1:[REPORTS.MONDAY]
$ !
$ ! Print out the report for Monday
$ !
$   PRINT MONDAY.RPT
Job MONDAY (queue SYS$PRINT entry 46) started on WORK$TXA0
$ !
$ ! Return to your login device and directory
$ !
$   SET DEFAULT SYS$LOGIN
$   EXIT
```

Example 7–1 (Cont.): A Sample Command Procedure

Notes on Example 7–1:

1. This is a very simple command procedure. Its purpose is to display some good programming practices you can use when you write command procedures.
2. Comment lines, added to your procedure, help you and others to determine the procedure's function, and to simplify maintenance. Note that comment lines are indicated by a dollar sign followed by an exclamation point ($!).
3. Each DCL command in the procedure is fully spelled out. By avoiding abbreviations, you avoid possible ambiguities when new DCL commands are added.

7.6 LOGIN COMMAND PROCEDURE

This is a command procedure that is executed automatically each time you log in. The name of the file must be LOGIN.COM and be placed in your default login directory.

This file contains logical names, symbols, and other commands you may have defined to set up your terminal session.

The example on the following page shows a typical LOGIN.COM file.

Notes on Example 7–2:

1. This procedure runs automatically when you log in.
2. The procedure is commented, and similar commands are grouped together. This helps make the procedure easier to read and maintain.
3. Global symbols are created that can be used as command synonyms to reset the user's default directory. Instead of using the **SET DEFAULT** command, the user can enter these symbols at the DCL prompt.
4. Command synonyms are created for commonly used DCL commands, by assigning global symbol values to the selected DCL commands.
5. The **DEFINE** command is used to assign command values to keypad keys.

```
$ !                              LOGIN.COM
$ !
$ !
$ !
$ ! Logical names for common files and directories
$ !
$   ASSIGN SYS$LOGIN_DEVICE:[BLOOM.PASCAL] PASCAL
$   ASSIGN SYS$LOGIN_DEVICE:[BLOOM.GAMES] FUN
$   ASSIGN SYS$LOGIN_DEVICE:[BLOOM.PROCEDURES]CLEANUP.COM CLEANUP
$
$ !
$ ! Commonly used commands
$ !
$   SED  == "SET DEFAULT"
$   HOME == "SET DEFAULT SYS$LOGIN"
$   CLR == "SET TERMINAL/WIDTH=80"
$   EDT == "EDIT"
$   DS == "DIRECTORY/SIZE=ALL"
$   SD == "SHOW DEFAULT"
$   M == "MAIL"
$   PU == "PURGE/LOG"
$   XX == "DELETE"
$ !
$ !
$ ! Key definitions
$ !
$   SET TERMINAL/APPLICATION_KEYPAD
$ !
$   DEFINE/KEY/NOLOG/TERMINATE PF1 "SHOW USERS"
$   DEFINE/KEY/NOLOG/TERMINATE PF3 "SHOW TIME"
$   DEFINE/KEY/NOLOG/TERMINATE KP9 "SHOW QUEUE/ALL/FULL LPA0"
$   DEFINE/KEY/NOLOG/TERMINATE KP0 "LOGOUT"
$ !
$   EXIT
```

Example 7–2 Typical LOGIN.COM File

7.7 TERMINAL INPUT/OUTPUT

Several DCL commands allow you to perform terminal input and output operations from within a command procedure. Some of these commands allow you to prompt the user for information that can be used by the procedure. Other commands allow you to display messages and command output on the terminal screen.

These commands make use of predefined logical names. By redefining these logical names, you can allow the use of an interactive utility, like an editor, from within the procedure, or redirect terminal output to a file.

Table 7–1 lists the logical names used with terminal input and output commands. Notice that these logical names usually point to your terminal.

When you execute a command procedure, one logical name, SYS$INPUT, is automatically defined to point to the command procedure file, rather than your terminal. This means that the system looks for its input from the command procedure file, rather than from the terminal. To allow for input to be entered from the terminal (for example, to edit a file), you must redirect the logical SYS$INPUT so that it points to the terminal.

Table 7–2 lists commands that display information on your terminal. Table 7–3 lists commands that prompt the user for information. Table 7–4 lists commands that redirect terminal input and output.

Table 7–1 System Logical Names Used with Terminal I/O

Logical Name	Description	Associated File or Device	
		(At Login)	**(During Execution of a Procedure)**
SYS$COMMAND	Initial input stream for your process	Terminal	Terminal
SYS$INPUT	Default input stream for your process	Terminal	Command Procedure File
SYS$OUTPUT	Default output stream for your process	Terminal	Terminal
SYS$ERROR	Default file to which the system writes error messages	Terminal	Terminal

7.7.1 Performing Terminal Input and Output

Table 7–2 Displaying Information on the Terminal

Format/Example	Comments
$ WRITE SYS$OUTPUT string `$ WRITE SYS$OUTPUT "Hello."`	Character strings are enclosed in quotation marks.
$ WRITE SYS$OUTPUT symbol `$ WRITE SYS$OUTPUT P1`	The symbol's value is automatically substituted.
$ TYPE SYS$INPUT **text** **text** **text** **$** `$ TYPE SYS$INPUT` `Hello` `$`	Information to be displayed follows the TYPE command. A dollar sign marks the end of the information.

```
$ !                                              REPORT2.COM
$ !
$ !
$ ! This command procedure sets your default directory to the
$ ! [REPORTS.MONDAY] subdirectory, prints out a report for Monday,
$ ! returns you to your login device and directory, then exits.
$ !
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Changing your default directory"
$ !
$ ! Set your default to the correct subdirectory
$ !
$   SET DEFAULT DISK1:[REPORTS.MONDAY]
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Printing the Monday report"
$ !
$ ! Print out the report for Monday
$ !
$   PRINT MONDAY.RPT
$ !
$ ! Return to your login device and directory
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Changing back to your login directory"
$ !
$   SET DEFAULT SYS$LOGIN
$   EXIT
```

Execution of REPORT2.COM:

```
$ @REPORT2
Changing your default directory
Printing the Monday report
Job MONDAY (queue SYS$PRINT, entry 46) started on WORK$TXA0
Changing back to your login directory
```

Example 7–3 An Output Sample from a Command Procedure

Notes on Example 7–3:

1. The **WRITE SYS$OUTPUT** commands display the character string you specify on the terminal screen.
2. The **SET DEFAULT DISK1:[REPORTS.MONDAY]** command sets your default to the correct subdirectory.
3. The DCL command **PRINT MONDAY.RPT** prints the correct report.
4. The DCL command **SET DEFAULT SYS$LOGIN** returns you to your default device and directory and then you exit from the command procedure.

Table 7–3 Getting Information from the User

Format/Example	Comments
$ INQUIRE symbol "prompt" `$ INQUIRE NAME "Filename"`	The prompt string is optional. The user's response is converted to uppercase. Multiple blanks and tabs are replaced with a single space. The response is then assigned to the local symbol "Filename." If no prompt string is supplied, the symbol name is used as the prompt.
$ READ/PROMPT=string SYS$COMMAND symbol `$ READ/PROMPT="Filename: " SYS$COMMAND NAME`	The user's response is not converted to uppercase, and multiple spaces are not removed. The response is then stored in the local symbol "Filename."

Table 7–4 Redirecting Input and Output

Format/Example	Comments
`$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND` *or* `$ ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT`	The ASSIGN or DEFINE command redirects the input stream from the command procedure file to the terminal. The /USER_MODE qualifier specifies that the change remains in effect only while the next image is executing.
$ @commandfile-name/ OUTPUT=output-file-name	Redirects output to the file you specify.
`$ @COMFILE.COM/OUTPUT=COMSTAT.DAT`	Redirects the output stream to the file you specify while the next image is executing.
`$ DEFINE/USER_MODE SYS$OUTPUT output-file-name` *or* `$ ASSIGN/USER_MODE output-file-name SYS$OUTPUT` `$ DEFINE/USER_MODE SYS$OUTPUT COM_STAT.DAT`	

```
$!
$!                                        NOTICE.COM
$!
$!
$! This command procedure creates a text file containing
$! the message you specify, then mails it to DIST.DIS,
$! a predefined distribution list.
$!
$! First, display instructions to the user.
$!
$  WRITE SYS$OUTPUT " "
$  WRITE SYS$OUTPUT "Enter your message. Press CTRL/Z when done."
$  WRITE SYS$OUTPUT " "
$!
$! Redirect the logical SYS$INPUT from the command
$! procedure to the terminal.
$!
$  ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
$!
$! Have the user create the message.
$!
$  EDIT MESSAGE.TXT
$!
$! When the user exits the editor, the command procedure
$! continues.
$!
$!
$! Send the message.  The lines following the MAIL
$! command are data lines used by the MAIL utility.
$! The dollar sign indicates the end of the data.
$!
$  MAIL
SEND MESSAGE.TXT
@DIST.DIS
A NOTE FROM YOUR SUPERVISOR
$!
$! Leave the procedure
$!
$  EXIT
```

Example 7–4 Using Terminal Input and Output

Notes on Example 7–4:

1. The **WRITE SYS$OUTPUT** commands display the character string you specify on the terminal screen.
2. The **ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT** command redefines the logical name SYS$INPUT to equate to SYS$COMMAND. This command redirects input from the command procedure file to the terminal. Because the **/USER_MODE** qualifier is used, this redirection stays in effect for only one command. In this example, it is the **EDIT** command.
3. The **EDIT MESSAGE.TXT** command creates the file MESSAGE.TXT. The user's input on the terminal screen is placed in the file. When the message is completed, the user ends the command by pressing CTRL/Z. When the user presses CTRL/Z, control is passed back to the command procedure.
4. The Mail utility is invoked using the Mail command **SEND MESSAGE.TXT**. The distribution list (DIST.DIS) should have been created prior to this for the command procedure to work.
5. The lines following the **MAIL** command are data lines used by the Mail utility. Because they are data lines, they are not preceded by a dollar sign. When a procedure encounters a dollar sign, it automatically terminates the utility.

7.8 DCL SYMBOLS

Symbols are names to which you can assign a character string or an integer value. Symbols have two primary purposes. They can be used as command synonyms, or as variables in command procedures. For example, you can create a symbol HOME that has the value "SET DEFAULT SYS$LOGIN." HOME is a command synonym because it can be used synonymously with the command **SET DEFAULT SYS$LOGIN**. An example of a symbol as a variable might be assigning the symbol COUNT the integer value 1.

Symbols are defined using assignment statements. There are two types of symbols available: *local symbols* and *global symbols*. Local symbols remain in effect only while the command procedure is executing. Global symbols remain in effect until the process terminates.

Global symbols are often used to define command synonyms in your LOGIN.COM procedure. Global symbols defined in this way allow you to use synonyms in place of commands. When defined as global symbols, the definitions remain in effect as long as you are logged in.

The use of symbols in command procedures provide possibilities not readily available at the interactive level. Using symbols as numeric and string variables expands the power and flexibility of DCL commands.

Table 7–5 shows examples of assigning local and global symbol values.

Table 7–5 Symbol Assignment and Manipulation

Operation	Operator	Example
Assign an integer value to a local symbol	=	`COUNT = 1`
Assign an integer value to a global symbol	==	`NEWCOUNT == 100`
Assign a character string to a local symbol	=	`USER = "SMITH"`

7.8.1 Symbol Substitution

In a command procedure, symbols are frequently used as command synonyms, parameters, and as numeric and string variables.

To be useful within a command procedure, the system must translate symbols into their corresponding values. Some DCL commands replace symbols with their values automatically. Most DCL commands do not perform automatic symbol substitution.

You can tell the system to force symbol substitution. To indicate to the system that a symbol value should be substituted, enclose the symbol name in apostrophes ('). When the symbol name is contained in a character string, the symbol name must be preceded with two apostrophes (") and ended with a single apostrophe (').

Table 7–6 lists those commands and statements that perform automatic symbol substitution, as well as examples of nonautomatic substitution.

Table 7–6 Symbol Substitution Techniques

Symbol Usage	Substitution Technique	Example
Command synonym (first item after $ prompt)	Automatic	`$ XX = "DELETE"` `$ XX FILE.TXT;1`
In the right–hand side of an = or == assignment statement	Automatic	`$ COUNT = COUNT + 1` `$ FILESPEC = NAME + ".TXT"`
In an IF or WRITE command	Automatic	`$ IF COUNT .GT. 10 THEN -`
In a DCL command that does not perform automatic symbol substitution	Surround the symbol with apostrophes (')	`$ RUN 'PROGRAM'`
In a character string	Place two apostrophes in front of the symbol, and one apostrophe after it	`$ WRITE SYS$OUTPUT -` `"The file ''FILE' exists."`
Concatenating two symbols in a DCL command that does not perform automatic symbol substitution	Surround each symbol with apostrophes (do not leave a space between the symbols)	`$ PRINT 'NAME''TYPE'`

```
$ !                                          REPORT3.COM
$ !
$ !
$ ! This command procedure sets your default directory to the
$ ! [REPORTS.'DAY'] subdirectory, prints out a report for the
$ ! day of your choice, returns you to your login device and
$ ! directory, then exits.
$ !
$ ! Ask which daily report to print out
$ !
$   INQUIRE DAY "Day to print a report"
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Changing your default directory"
$ !
$ ! Set your default to the correct subdirectory
$ !
$   SET DEFAULT DISK1:[REPORTS.'DAY']
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Printing the ''day' report"
$ !
$ ! Print out the report for the correct day
$ !
$   PRINT 'DAY'.RPT
$ !
$ ! Return to your login device and directory
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Changing back to your login directory"
$ !
$   SET DEFAULT SYS$LOGIN
$   EXIT
```

Execution of REPORT3.COM:

```
$ @REPORT3
Day to print report for: TUESDAY
Changing your default directory
Printing the TUESDAY report
Job TUESDAY (queue SYS$PRINT, entry 47) started on WORK$TXA0
Changing back to your login directory
```

Example 7–5 Using Symbol Substitution

7.8.2 Passing Parameters to Command Procedures

Parameters are the objects of DCL commands. A parameter might be a keyword, file specification, or an integer or string value. Since command procedures are made up of DCL commands, they frequently use parameters. You can specify parameters for a command procedure when you execute the procedure.

The system automatically provides eight local symbols: P1 – P8. You can pass up to eight numeric or string values to the procedure. If you specify parameters, the system assigns the values you specify to these symbols at execution time. If you do not specify a parameter, the default value assigned to the symbol (the null string) remains in effect. The syntax is:

$ @command_procedure.com parameter_1 parameter_2 ... parameter_8

```
$ !                                          REPORT4.COM
$ !
$ !
$ ! This command procedure sets your default directory to the
$ ! [REPORTS.'P1'] subdirectory, prints out a report for the day of
$ ! your choice, returns you to your login device and directory,
$ ! then exits.
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Changing your default directory"
$ !
$ ! Set your default to the correct subdirectory
$ !
$   SET DEFAULT DISK1:[REPORTS.'P1']
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Printing the ''P1' report"
$ !
$ ! Print out the report for the correct day
$ !
$   PRINT 'P1'.RPT
$ !
$ ! Return to your login device and directory
$ !
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "Changing back to your login directory"
$ !
$   SET DEFAULT SYS$LOGIN
$   EXIT
```

Execution of REPORT4.COM

```
$ @REPORT4 TUESDAY
Changing your default directory
Printing the TUESDAY report
Job TUESDAY (queue SYS$PRINT, entry 47) started on WORK$TXA0
Changing back to your login directory
```

Example 7–6 Passing Parameters to Commands Procedures

7.9 CONTROLLING PROGRAM FLOW

The normal order of command execution in a procedure is sequential. That is, the first line of the procedure executes, then the second, and so on. This sequential order is sufficient in some procedures, such as a LOGIN.COM file, which usually performs unconditional tasks.

There are times, however, when you will want to alter the order of execution, depending on the results of conditional testing within the procedure. For example, you may want a file to print, but only if it has the proper file type. DCL provides several commands that allow you to alter the order of execution. These control–flow commands include the **IF** command, and the **GOTO** command.

7.9.1 The IF Command

Syntax:

$ IF conditional expression THEN command

When the command executes, the conditional expression following **IF** is tested. This conditional expression may to compare integer values, compare two string expressions, or test whether or not a logical expression is true or false. Table 7–7 lists the operators used in conditional expressions.

If the condition is met, the command(s) following **THEN** are performed.

If the condition following **IF** is not met, then the next DCL command in sequence is performed, or an optional **ELSE** statement can be performed. The optional **ELSE** statement provides command(s) to be performed when the **IF** condition is false.

The command(s) following **THEN** or **ELSE** can be:

- Another DCL command(s), or
- A **GOTO** command

The syntax of the **IF–THEN–ELSE** command is:

$ IF conditional expression THEN command ELSE command(s)

7.9.2 Notes on the IF–THEN–ELSE Command

- A command block started by a **THEN** statement must be terminated by an **ENDIF** statement.
- A **THEN** statement must be the first executable statement following an **IF** statement.
- **THEN**, **ELSE**, and **ENDIF** statements cannot be abbreviated to fewer than four characters.
- Do NOT specify labels on a **THEN** or **ELSE** statement.
- Labels are legal only on an **ENDIF** statement.
- Command procedures may branch within the current command block, but branching into the middle of another command block is not recommended.

7.9.3 The GOTO Command

Syntax:

$ GOTO label

The **GOTO** command does not perform any conditional testing. Rather, control is transferred to the label that follows the **GOTO** command. The DCL command that follows the label is then performed. This label can occur either before or after the **GOTO** command.

Table 7–7 Relational Operators Used in Expressions

Operator	Description
String Operators	
.EQS.	Tests if two character strings are equal.
.GES.	Tests if the first string is greater than or equal to the second string.
.GTS.	Tests if the first string is greater than the second string.
.LES.	Tests if the first string is less than or equal to the second string.
.LTS.	Tests if the first string is less than the second string.
.NES.	Tests if the two strings are not equal.
Numeric Operators	
.EQ.	Tests if two numbers are equal.
.GE.	Tests if the first number is greater than or equal to the second number.
.GT.	Tests if the first number is greater than the second number.
.LE.	Tests if the first number is less than or equal to the second number.
.LT.	Tests if the first number is less than the second number.
.NE.	Tests if two numbers are not equal.
Logical Operators	
.NOT.	Logically negates a number.
.AND.	Combines two numbers with a logical AND.
.OR.	Combines two numbers with a logical OR.

```
$!                          DEL_DIR.COM
$!
$!
$!
$! This command procedure deletes previously emptied
$! directories.  It assumes that the directory to be
$! deleted is owned by the procedure's user.
$!
$! Check to see if the user entered the directory name.
$! If yes, skip to the confirmation question.
$! If no, display a message and ask for the directory name
$!
$  IF P1 .NES. "" THEN GOTO CONFIRM
$!
$  WRITE SYS$OUTPUT " "
$  WRITE SYS$OUTPUT "This procedure deletes an emptied directory"
$  WRITE SYS$OUTPUT "The .DIR file extension is assumed."
$  WRITE SYS$OUTPUT " "
$  INQUIRE P1 "Directory name"
$!
$  CONFIRM:
$  INQUIRE P2 "Confirm, please (Y/N)"
$!
$! If the user answers 'No', abandon this procedure.
$!
$  IF .NOT. P2 THEN GOTO NODELETE
$!
$! Reset the directory protection so that the owner
$! can delete it, delete the directory and display
$! the system message.  Note that the procedure
$! substitutes the directory name for the symbol P1.
$!
$  SET PROTECTION=(O:RWED) 'P1'.DIR;*
$  DELETE/LOG 'P1'.DIR;*
$  GOTO END
$!
$  NODELETE:
$!
$  WRITE SYS$OUTPUT " "
$  WRITE SYS$OUTPUT "Directory file not deleted."
$!
$  END:
$  EXIT
```

Example 7–7 Controlling Program Flow in a Command Procedure

Execution of DEL_DIR.COM:

```
$ @DEL_DIR TEST
Confirm, please (Y/N): Y
%DELETE-I-FILDEL, DISK:[DENISE]TEST.DIR;1 deleted (3 blocks)
$
```

Second execution:

```
$ @DEL_DIR

This procedure deletes an emptied directory
The .DIR file extension is assumed.

Directory name: TEST2
Confirm, please (Y/N): Y
%DELETE-I-FILDEL, DISK:[DENISE]TEST2.DIR;1 deleted (3 blocks)
$
```

Third execution:

```
$ @DEL_DIR

This procedure deletes an emptied directory
The .DIR file extension is assumed.

Directory name: TEST3
Confirm, please (Y/N): N

Directory file not deleted.
$
```

Example 7–7(Cont.) Controlling Program Flow in a Command Procedure

7.10 LEXICAL FUNCTIONS

Lexical functions are features of the DCL command language that provide information similar to that of some DCL commands. The primary difference between DCL commands and lexical functions is the manner in which the information is provided.

In most instances, the information provided by a DCL command is returned to the terminal. Although well suited for display, it is not easily manipulated or changed. For example, the **SHOW TIME** command displays the current time on the terminal like this:

```
$ SHOW TIME
18-JUN-1988 10:10:89
$
```

There is no simple method to use this information in a command procedure. On the other hand, information provided by a lexical function is returned as a symbol value. This symbol and its associated value can be used in a command procedure. For example, you can assign the current date and time to a symbol, like this:

```
$ TIME = F$TIME()
```

You can now use the symbol TIME in subsequent DCL commands within the procedure.

All lexical functions return a value. This value can be an integer or a character string, depending on the lexical function. For complete information on the syntax, parameters, and use of lexical functions, refer to the *VMS DCL Dictionary*.

```
$!                              INFO.COM
$!
$!
$! This command procedure allows the user to leave a message
$! on the terminal screen, along with information about the
$! process. The time when the message was left is also displayed.
$!
$!
$! Use lexical functions to determine the current time
$! and day of the week
$  TIME = F$TIME()
$  CURR_TIME = F$EXTRACT(12,5,TIME)
$  WEEKDAY = F$CVTIME(TIME,,"WEEKDAY")  ! Returns Monday,Tuesday,etc.
$!
$! Clear the screen using the TYPE/PAGE NL: command
$  TYPE/PAGE NL:
$!
$! Display process name, the time, and the day of the week.
$  NAME= F$PROCESS()
$  WRITE SYS$OUTPUT NAME
$  WRITE SYS$OUTPUT " "
$  WRITE SYS$OUTPUT "IT IS ''CURR_TIME' ON A ''WEEKDAY'"
$  WRITE SYS$OUTPUT " "
$!
$! Leave the procedure
$!
$  END:
$   EXIT
```

Execution of INFO.COM

```
$ @INFO
DENISE

IT IS 12:23 ON A Monday
```

Example 7–8 Using Lexical Functions

All lexical functions begin with **F$**, followed by the function name.

```
WHO = F$PROCESS()
```

All lexical functions require parentheses, even with null arguments.

Multiple arguments are separated by commas.

Optional arguments, when omitted, are indicated by commas.

Examples of arguments supplied to lexical functions:

- Integer or character strings

  ```
  WHAT = F$EXTRACT(0,3,"MAILMAN")
  ```

- Symbols

  ```
  HOWLONG = F$LENGTH(P1)
  ```

- Keywords

  ```
  WHERE = F$ENVIRONMENT("MESSAGE")
  ```

- Null arguments

  ```
  WHEN = F$TIME()
  ```

Table 7–8 Frequently Used Lexical Functions

Lexical Function	Description
F$TIME( )	Returns the current date and time string.
F$PROCESS( )	Returns the current process name.
F$MODE( )	Returns a character string indicating the mode in which a process is running (INTERACTIVE, NETWORK, or OTHER).
F$LENGTH(string)	Returns the length of a string.
F$LOCATE(substring,string)	Locates the substring in the string and returns the offset position.
F$EXTRACT(offset,number,string)	Extracts a substring from a character string expression.
F$CVTIME([input–time],[format],– [field])	Returns information about absolute, combination, or delta time strings.
F$ENVIRONMENT(item)	Returns information about the DCL command environment.
F$GETQUI( )	Returns information regarding queues and the batch and print jobs currently in those queues.
F$LOGICAL(logical–name)	Returns the equivalence string associated with the logical name. Does not perform iterative translation automatically.

```
$ !                              PRINT.COM
$ !
$ ! This procedure allows you to print multiple copies
$ ! of any file you choose.  It will ask for the file
$ ! name and number of copies if the information is
$ ! not supplied on the command line.  The procedure
$ ! will not let the user print a binary file.
$
$ NAME_FILE:
$
$ IF P1 .EQS. "" THEN INQUIRE P1 "File to be printed"
$
1  $ LENGTH=F$LENGTH(P1)
2  $ IF LENGTH .EQ. 0 THEN GOTO NAME_FILE
$
3  $ PERIOD=F$LOCATE(".",P1)
4  $ FNAME=F$EXTRACT(0,PERIOD,P1)
$
$ ! Check to see if user entered file type.  If yes, separate
$ ! filename from file type.  If no, assign .LIS type to the file
$ !
5  $ IF LENGTH .EQ. PERIOD
$    THEN FTYPE=".LIS"
6  $ ELSE FTYPE=F$EXTRACT(PERIOD,LENGTH-PERIOD,P1)
$ ENDIF
$
$ ! Check to see if user entered a binary file type. If yes, exit.
$ ! If no, see how many copies they want.
$ !
$ IF FTYPE .EQS. ".OBJ" .OR. FTYPE .EQS. ".EXE"
$       THEN WRITE SYS$OUTPUT "YOU CANNOT PRINT A ''FTYPE' FILE"
$       EXIT
$ ENDIF
$
$ NUMBER_COPIES:
$
$ IF P2 .EQS. "" THEN INQUIRE/NOPUNCTUATION P2 "HOW MANY COPIES DO YOU WANT? "
$
$ IF NUMBER .LE. 0 THEN GOTO NUMBER_COPIES
$
$ ! Print the correct number of copies then exit from the procedure
$ !
$ PRINT/COPIES='P2' 'FNAME''FTYPE'
$
$ EXIT
```

Example 7–9 Using More Detailed Lexical Functions

Notes on Example 7–9:

1 `$ LENGTH = F$LENGTH(P1)`

The F$LENGTH lexical function returns the length of the character string in the local symbol P1 and assigns this value to the symbol LENGTH.

2 `$ IF LENGTH .EQ. 0 THEN GOTO NAME_FILE`

If this statement is true, then the command procedure returns to the label NAMEFILE on the premise that the user did not type in a file name.

3 `$ PERIOD = F$LOCATE(".",P1)`

The F$LOCATE lexical function locates the period in the local symbol P1 and returns its offset within the string to the symbol PERIOD.

4 `$ FNAME = F$EXTRACT(0,PERIOD,P1)`

The F$EXTRACT lexical function extracts the file name (substring) by starting at the offset position (0) up to PERIOD in P1 and assigns it to the symbol FNAME.

The 0 lexical function IF–THEN–ELSE is employed here:

5
```
$ IF LENGTH .EQ. PERIOD
$ THEN FTYPE = ".LIS"
```

If the symbol LENGTH and the symbol PERIOD hold the same value, they automatically assign the symbol FTYPE or .LIS to the file.

6 `$ ELSE FTYPE = F$EXTRACT(PERIOD,LENGTH-PERIOD,P1)`

If a "." was found in the P1 string, we assume a file type was specified. Extract the file type from P1.

7.11 SUMMARY

- A command procedure is a file containing DCL command strings
- These command strings are made up of:
  - — DCL command verbs
  - — Command parameters
  - — Qualifiers
- Command procedures frequently make use of:
  - — DCL symbols – command synonyms, numeric and string variables
  - — Control flow commands – IF, GOTO
  - — Lexical functions
- You can perform terminal input and output functions using:
  - — INQUIRE
  - — READ SYS$COMMAND
  - — WRITE SYS$OUTPUT
  - — TYPE SYS$INPUT
- Control flow commands allow you to alter the order of command execution
  - — IF–THEN or IF–THEN–ELSE – transfers control based on the results of conditional expressions
  - — GOTO – unconditionally transfers control
- You can pass numeric and string information to the command procedure using the local symbols P1 – P8 associated with every command procedure
- Lexical functions allow you to gather and use system and process information in command procedures

7.12 WRITTEN EXERCISE I

To complete these exercises, use the following symbol definitions:

```
COUNT = 2                FILE_NAME = "PROGRAM"
P1 = "MYFILE.TXT"        FILE_TYPE = ".FOR"
P2 = "DATA.DAT"
```

Part A:

Each command below uses a symbol in some way. Indicate whether or not the symbol is used correctly. If it is used correctly, rewrite the command, replacing the symbol with its value (see above). If the symbol is used incorrectly, rewrite the command correctly.

Examples:

$ TYPE "P1"

Incorrect: $ TYPE 'P1'

$ EDIT 'P2'

Correct: $ EDIT DATA.DAT

1. $ FILE = 'FILE_NAME' + 'FILE_TYPE'
2. $ WRITE SYS$OUTPUT COUNT " copies of the file"
3. $ IF COUNT .LT. 10 THEN GOTO END
4. $ WRITE SYS$OUTPUT "The file "FILE_NAME""FILE_TYPE'"

Part B:

For the commands below, replace the underlined text with symbols, using the proper symbol substitution techniques. Use the same symbol values you used in Part A.

Example:

$ PRINT MYFILE.TXT

$ PRINT 'P1'

1. $ WRITE SYS$OUTPUT "The file is MYFILE.TXT"
2. $ TYPE PROGRAM.FOR
3. $ EDIT DATA.DAT
4. $ WRITE SYS$OUTPUT "2 copies of the file DATA.DAT exist."
5. $ FILE = "PROGRAM" + ".FOR"

7.13 INTRODUCTION TO LABORATORY EXERCISES

These lab exercises are designed to give you practice in creating, testing, and running command procedures.

The procedures in these exercises include the commonly used functions of command procedures, such as:

- Terminal input and output
- Symbol assignment and symbol substitution
- Controlling program flow
- Passing data to procedures
- Using simple lexical functions

7.14 LABORATORY EXERCISE I

LOGIN.COM is one of the most commonly used command procedures. This procedure is executed automatically each time you log in to a VMS system. It is used to tailor your working environment on the system to better suit your needs.

Write a LOGIN.COM of your own that performs the following actions:

1. Exits if the process mode is not interactive. Use the lexical function F$MODE() to test the mode of the process.
2. Defines a logical name that points to one of your subdirectories:

   ```
   disk_name:[directory_name.subdirectory_name]
   ```

   where disk_name is your default disk, and directory_name is your top–level directory.
3. Defines global symbols to be used as command synonyms. The command synonyms, when defined should perform the following actions:
   - Set default
   - Show all users currently logged in to the system
   - Display your current directory
   - Set your default to your login disk and directory
4. Displays the following information on your terminal:
   - The current date and time
   - The current default directory

7.15 LABORATORY EXERCISE II

Write a command procedure that allows you to create files that everyone on your system can access. The procedure should perform the following tasks:

1. Asks for the file name if it is not provided.
2. Displays a message that indicates the name of the file being edited.
3. Transfers control to the terminal and then allows you to edit the file.
4. Sets the protection on the file so that the WORLD has READ access.
5. Prints a copy of the file for yourself, if you choose.

The name of the file you are creating should be supplied as P1.

This exercise uses terminal input and output including:

- INQUIRE
- WRITE SYS$OUTPUT
- DEFINE/USER_MODE or ASSIGN/USER_MODE

7.16 LABORATORY EXERCISE III

The sample file ADD.COM, shown below, is intended to request two numbers, add them together, and display their sum. It doesn't behave as expected.

```
$ !                                        ADD.COM
$ ! Adds two numbers together and displays their sum.
$ ! (This command procedure doesn't work as expected.)
$
$WRITE SYS$OUTPUT "This command procedure will add two numbers together."
$INQUIRE P1 "FIRST VALUE"
$INQUIRE P2 "SECOND VALUE"
$TOT = P1 + P2
$WRITE SYS$OUTPUT "TOTAL IS ", TOT
```

Invoke ADD.COM, supply the input it requests, and determine what is wrong with it.

7.17 LABORATORY EXERCISE IV

The sample file SAVDIR.COM is shown below:

```
$!                                  SAVDIR.COM
$! Save current default directory, set default to a
$! new directory specified by the user, demonstrate
$! the new default, then reset to the original default.
$!
$! This generates errors - can you fix it?
$
$CURDIR==F$DIRECTORY()
$WRITE SYS$OUTPUT "CURRENT DIRECTORY IS ",CURDIR
$INQUIRE NEWDIR "ENTER NEW DIRECTORY SPECIFICATION"
$SET DEFAULT NEWDIR
$DIRECTORY
$SET DEFAULT CURDIR
$DIRECTORY
```

SAVDIR.COM is intended to do the following:

- Determine and display the user's current default directory
- Request a new directory specification from the user
- Set default to that new directory
- Generate a $DIRECTORY listing to demonstrate the new default
- Set the default back to the original directory

It generates an error and does not behave as expected.

Invoke SAVDIR.COM, supply the input it requests, interpret the resulting error message, and determine what is wrong with the procedure.

7.18 LABORATORY EXERCISE V

The sample files IF_THEN_1.COM and IF_THEN_2.COM, shown below, are intended to request a number from the user and determine whether it is odd or even. They do not work properly.

```
$!                                        IF-THEN_1.COM
$!      Decide whether a number is odd or even.
$!      This generates errors - can you fix it?
$
$ INQUIRE X "TYPE A NUMBER"
$ IF X THEN WRITE SYS$OUTPUT "ODD"
$       ELSE WRITE SYS$OUTPUT "EVEN"
$       ENDIF
$ WRITE SYS$OUTPUT "DONE"
```

```
$!                                        IF-THEN_2.COM
$!      Decide whether a number is odd or even.
$! This doesn't work when an odd number is given -
$! can you fix it?
$
$ INQUIRE X "TYPE A NUMBER"
$ IF X
$ THEN WRITE SYS$OUTPUT "ODD"
$ ELSE WRITE SYS$OUTPUT "EVEN"
$ WRITE SYS$OUTPUT "DONE"
```

Invoke each command procedure, supply both odd and even values, and determine what is wrong with the procedures.

7.19 OPTIONAL LABORATORY EXERCISE

Write a command procedure that displays a message on your terminal screen that states when you will return. The procedure should perform the following tasks:

1. Asks you for the number of minutes you will be away.
2. Erases the screen and then displays the following message, 12 lines from the top:

   ```
   "Back in N minutes"
   ```

   (where N is the number of minutes you supplied in Part 1).
3. It waits, and at one–minute intervals subtracts 1 from the number of minutes, erases the screen and redisplays the message with the new value.
4. When only one minute is left, it erases the screen and displays the message:

   ```
   "I'll be right back."
   ```

This exercise uses terminal input and output commands, including:

- INQUIRE/NOPUNCTUATION
- WRITE SYS$OUTPUT
- TYPE SYS$INPUT

This procedure also uses the DCL command **WAIT**. For more information on this command, refer to the *VMS DCL Dictionary*.

This procedure does NOT use lexical functions.

7.20 WRITTEN EXERCISE I—SOLUTIONS

Part A:

Each command below uses a symbol in some way. Indicate whether or not the symbol is used correctly. If it is used correctly, rewrite the command, replacing the symbol with its value. If the symbol is used incorrectly, rewrite the command correctly.

1. $ FILE = 'FILE_NAME' + 'FILE_TYPE'

   Incorrect. Correct command is: $ FILE = FILE_NAME + FILE_TYPE

   Do not use symbol substitution characters on the right–hand side of an = assignment statement.

2. $ WRITE SYS$OUTPUT COUNT " copies of the file"

   Incorrect. Correct command is: $ WRITE SYS$OUTPUT COUNT, " copies of the file"

   Separate the items in the output list with commas. The values will be concatenated. Note that the symbol COUNT is substituted automatically.

   An alternate method: $ WRITE SYS$OUTPUT "'COUNT' copies of the file"

   If you place the symbol COUNT within the quoted string, symbol substitution does not occur automatically. For symbol substitution to occur, precede the symbol with two apostrophes.

3. $ IF COUNT .LT. 10 THEN GOTO END

   Correct. $ IF 2 .LT. 10 THEN GOTO END

   DCL automatically performs symbol substitution in an **IF** command.

4. $ WRITE SYS$OUTPUT "The file ''FILE_NAME''FILE_TYPE'"

   Correct. $ WRITE SYS$OUTPUT "The file PROGRAM.FOR"

   In a character string, a symbol must be preceded by two apostrophes and followed by one.

Part B:

In the commands below, replace the underlined text with symbols, using the proper symbol substitution techniques. Use the same symbol values you used in Part A.

1. $ WRITE SYS$OUTPUT "The file is MYFILE.TXT"

   $ WRITE SYS$OUTPUT "The file is "P1'"

2. $ TYPE PROGRAM.FOR

   $ TYPE 'FILE_NAME"FILE_TYPE'

3. $ EDIT DATA.DAT

   $ EDIT 'P2'

4. $ WRITE SYS$OUTPUT "2 copies of the file DATA.DAT exist."

   $ WRITE SYS$OUTPUT "'COUNT' copies of the file "P2' exist."

5. $ FILE = "PROGRAM" + ".FOR"

   $ FILE = FILE_NAME + FILE_TYPE

7.21 LABORATORY EXERCISE I—SOLUTION

```
$!                  LOGIN.COM
$!
$!
$! Check to see if process is interactive. If not, exit.
$!
$  IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
$!
$! Define a logical name that points to the
$! COMPROC subdirectory.
$!
$  DEFINE COMPROC DISK1:[MANN.COMPROC]
$!
$! Alternately, use ASSIGN DISK1:[MANN.COMPROC] COMPROC
$!
$! Create global symbols to be used as command synonyms
$!
$  SED     == "SET DEFAULT"              ! Resets default
$  WHO     == "SHOW USERS"               ! Displays all users
$  SHD     == "SHOW DEFAULT"             ! Displays current directory
$  HOME    == "SET DEFAULT SYS$LOGIN"    ! Resets default to login values
$!
$! Display some "time and place" information on the terminal
$!
$  SHOW TIME
$!
$  SHOW DEFAULT
$!
$! Leave the procedure in an orderly manner.
$!
$  EXIT
```

7.22 LABORATORY EXERCISE II—SOLUTION

```
$!                    CREATE_FILE.COM
$!
$!
$! Expected parameters: P1 = name of file to be edited
$!
$! This command procedure allows you to edit a file, sets the
$! protection on the file so that the World has READ access,
$! then gives you the option of printing a copy of it.
$!
$! Be sure the name of the file is assigned to P1. If not, ask:
$!
$ IF P1 .EQS. "" THEN INQUIRE P1 "Filename"
$!
$! Display a message that indicates what file is being created:
$!
$ WRITE SYS$OUTPUT " "
$ WRITE SYS$OUTPUT "Editing the file ''P1'..."
$ WRITE SYS$OUTPUT " "
$!
$! Redirect SYS$INPUT so that it points to the terminal:
$!
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$!
$! Alternately, ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
$!
$! Allow the user to edit the file:
$!
$ EDIT 'P1'
$!
$! Set the required protection for the file:
$!
$ SET PROTECTION=(W:R) 'P1'
$!
$! Present the option of printing the file:
$!
$ INQUIRE/NOPUNCTUATION ANS "Print a copy of the file?  "
$ IF ANS THEN PRINT 'P1'
$ EXIT
```

7.23 LABORATORY EXERCISE III—SOLUTION

```
$ !                                          ADD_SOL.COM
$ !                                          Corrects the error in ADD.COM
$
$WRITE SYS$OUTPUT "This command procedure will add two numbers together."
$INQUIRE P1 "FIRST VALUE"
$INQUIRE P2 "SECOND VALUE"
$TOT = 0 + P1 + P2                           ! Note the correction here.
$WRITE SYS$OUTPUT "TOTAL IS ", TOT
$
$! P1 and P2 are assumed to be strings, so DCL concatenates them.
$! Force DCL to regard them as numeric symbols by preceding them with a
$! number.  Zero makes sense because it won't change the value of the sum.
```

7.24 LABORATORY EXERCISE IV—SOLUTION

```
$!                                  SAVDIR_SOL.COM
$! This corrects the errors in SAVDIR.COM.
$
$CURDIR==F$DIRECTORY()
$WRITE SYS$OUTPUT "CURRENT DIRECTORY IS ",CURDIR
$INQUIRE NEWDIR "ENTER NEW DIRECTORY SPECIFICATION"
$SET DEFAULT 'NEWDIR'                  ! Note correction here
$DIRECTORY
$SET DEFAULT 'CURDIR'                  ! and here.
$DIRECTORY
$
$! The original SAVDIR.COM did not have apostrophes around
$! the symbols NEWDIR and CURDIR, so the SET DEFAULT commands
$! assumed that they were logical names.
$! Use apostrophes to force symbol substitution in a situation
$! where symbols are not expected.
```

7.25 LABORATORY EXERCISE V—SOLUTION

```
$!                                IF-THEN_1_SOL.COM
$!      This corrects the error in IF_THEN_1.COM
$
$ INQUIRE X "TYPE A NUMBER"
$ IF X       ! Note the correction here.
$       THEN WRITE SYS$OUTPUT "ODD"
$       ELSE WRITE SYS$OUTPUT "EVEN"
$       ENDIF
$ WRITE SYS$OUTPUT "DONE"
$
$!       If you use ELSE, THEN should not be on the same line as IF.
$!       You are mixing constructs.
```

```
$!                                          IF-THEN_2_SOL.COM
$!      Decide whether a number is odd or even.
$!      This doesn't work when an odd number is given -
$!      can you fix it?
$
$! INQUIRE X "TYPE A NUMBER"
$! IF X
$!      THEN WRITE SYS$OUTPUT "ODD"
$!      ELSE WRITE SYS$OUTPUT "EVEN"
$!      WRITE SYS$OUTPUT "DONE"
$!
$!      This works fine with even input because the ELSE condition is met,
$!      so we skip over the statement(s) associated with THEN and just fall
$!      through.
$!
$!      The trouble comes when odd input is supplied.
$!      Since the THEN condition (truth for the IF test) is met, we execute
$!      the indicated statements and try to skip over the statements for ELSE.
$!      But where do we go when there's no ENDIF to finish up the IF-THEN?
$
$
$!      IF_THEN_2_SOL.COM - Corrected procedure
$
$ INQUIRE X "TYPE A NUMBER"
$ IF X
$       THEN WRITE SYS$OUTPUT "ODD"
$       ELSE WRITE SYS$OUTPUT "EVEN"
$       ENDIF    ! Here is the correction
$ WRITE SYS$OUTPUT "DONE"
```

7.26 OPTIONAL LABORATORY EXERCISE—SOLUTION

```
$ !                                          BACK_SOON.COM
$ !
$ ! This command procedure asks the user how many minutes he/she will
$ ! be away.  It erases the screen and displays the message "Back in
$ ! 'n' minutes".  It waits a minute, recalculates the value of N, and
$ ! redisplays the message.  When only one minute is left, it displays
$ ! "I will be right back".
$ !
$ ! Inquire for the number of minutes the user intends to be away.
$ WHEN:
$ INQUIRE/NOPUNCTUATION BACKSOON "How many minutes? "
$ !
$ ! If no answer, ask again.
$ IF BACKSOON .EQS. "" THEN GOTO WHEN
$ !
$ ! Top of time loop
$ LOOP:
$ IF BACKSOON .EQ. 1 THEN GOTO RIGHTBACK
$ !
$ ! Erase the screen
$ SET TERMINAL/WIDTH=80
$ !
$ ! Use the TYPE SYS$INPUT command to type eleven blank lines on
$ ! the terminal.
$ TYPE SYS$INPUT


$ !
```

```
$ ! Now use the WRITE SYS$OUTPUT command to display
$ ! the message on the screen.
$ !
$ WRITE SYS$OUTPUT "       Back in ''BACKSOON' minutes"
$ !
$ ! Wait one minute--note that the terminal is
$ ! tied up with this procedure.
$ WAIT 00:01:00.00
$ !
$ ! Subtract 1 from the number of minutes
$ BACKSOON=BACKSOON - 1
$ !
$ ! Loop until only one minute is left.
$ GOTO LOOP
$ !
$ ! The last step
$ RIGHTBACK:
$ !
$ ! Erase the Screen
$ TYPE/PAGE NL:
$ !
$ ! Use the TYPE SYS$INPUT command to type
$ ! the necessary blank lines.
$ TYPE SYS$INPUT



$ !
$ WRITE SYS$OUTPUT "       I will be right back."
$ END:
$ EXIT
```

USING DISK AND TAPE VOLUMES

8.1 INTRODUCTION

In addition to your default disk device, which stores the files you and many other users catalogue in your default directory hierarchies, your system includes a number of tape devices and smaller disk units. You can use one of these devices whenever you want to store copies of files on a private volume. Private volumes can be created on a magnetic tape reel, a floppy disk, or a smaller disk volume, such as an RK07 or RL02 pack. Once you copy files to such a volume, you can remove it and store it in some other secure location.

This module introduces the steps and commands required to create, use, and protect a private volume.

8.2 OBJECTIVES

In order to use private volumes to store private files off–line and transport them from one system to another, you should be able to perform the following operations:

- Allocate, initialize, and mount volumes
- Use the Backup Utility
- Dismount and deallocate volumes

8.3 RESOURCES

To complete this module, you should have access to the following documents:

- *VMS DCL Dictionary*
- *VMS Mount Utility Manual*
- *VMS Backup Utility Manual*

8.4 CREATING AND USING PRIVATE VOLUMES

8.4.1 The Uses of Private Disk and Tape Volumes

Private disk and tape volumes are volumes that you own exclusively. They have three major uses:

- To preserve files
- To transfer files from one system to another
- To provide a private environment for your work

8.4.1.1 Preserving Files

Your default disk, which you share with other system users, stores most of your files. This device is called a *public disk.*

Although the protection you establish for your files is normally sufficient to guard them against inadvertent destruction, they are still vulnerable to the activities of more privileged users and the operating system. For this reason, most users make copies of their most important material on backup volumes, such as magnetic tape reels or disk packs. A volume you own is called a *private volume.* A private volume has an owner user identification code (UIC) identical to your own.

8.4.1.2 Transferring Files

You may find it necessary to transfer files between systems not connected by a communications link. In such circumstances, you must be able to move your files physically from one location to another. You can conveniently do this by copying your files to a portable volume, such as a tape reel or disk pack, and then carrying that volume to the other system.

8.4.1.3 Providing a Private Environment

In certain circumstances, you may want to work on a device to which no one else has access. By creating a private volume and mounting it on a device assigned exclusively to your process, you can work without interference from other users.

In creating and using private volumes, you will use a number of VMS commands. These commands allow you to prepare and gain access to a wide range of peripheral storage devices, as Figure 8–1 suggests.

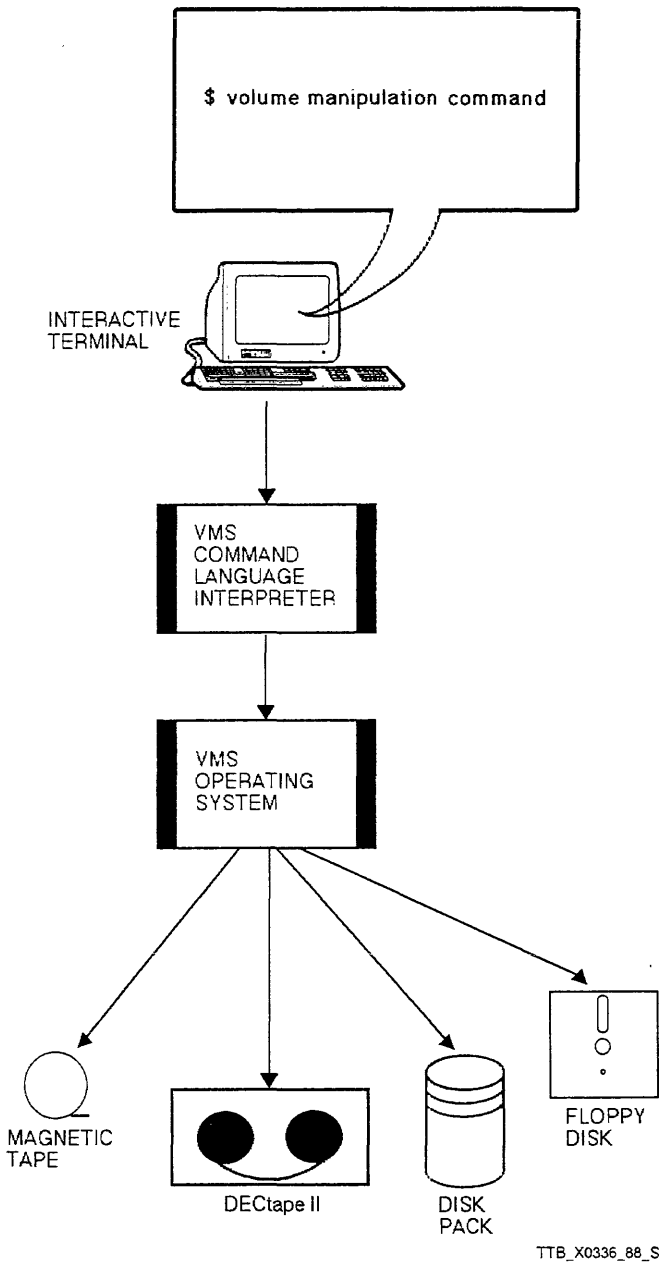


Figure 8–1 Volume Manipulation Commands

8.4.2 Creating Private Volumes: The Command Sequence

To create and use a private tape or disk volume, complete the following steps:

1. Allocate a device. This reserves a device for exclusive use by your process until you deallocate it.
2. Place a volume on the allocated device and load it. (Loading the volume physically prepares it to be accessed by commands issued to the device.)
3. Initialize the volume if it is new, or if another user has discarded it. The process of initialization builds an appropriate file structure and establishes the ownership and protection of the volume.
4. Issue the **MOUNT** command to make the volume accessible to your process. You can now create and manipulate files on the volume. You can also access the volume using a logical name.
5. When you have completed your work with the volume, issue the **DISMOUNT** command. The dismount process prohibits further access to the volume until it is remounted.
6. Unload and remove the volume from the device.
7. Deallocate the device to make it available to other users.

Table 8–1 lists the VMS commands to perform the above operations.

Example 8–1 illustrates the use of these commands to prepare a private disk volume.

Table 8–1 Commands for Creating and Accessing Private Disk and Tape Volumes

Operation	Format/Example	Comments
Allocating a Device	**$ ALLOCATE device [logical–name]**	
	`$ ALLOCATE DM DISK`	Finds the first available RK06/RK07 and assigns it to your process. The logical name DISK is placed in your process logical name table and is assigned the name of the allocated device. Other users are unable to access the device. When the ALLOCATE command is used, the DISMOUNT command does not free the device for other users.
Creating a file structure on a tape or disk	**$ INITIALIZE device label**	Builds the appropriate disk structure on the volume. Establishes volume ownership and protection. Usually used for new volumes.
	`$ INITIALIZE DMA2: TESTDISK`	The disk volume mounted on DMA2: is labeled TESTDISK. You are declared owner of the disk. All user groups are allowed all types of access (RWED). You must own the disk or possess VOLPRO privilege.
	`$ INITIALIZE MUA0: TSTTAP`	Builds ANSI level 3 tape structure on the volume located on device MUA0:. The volume receives the label TSTTAP. By default, you are declared the owner. All user groups are allowed all types of access (RWED).

Table 8–1 (Cont.) Commands for Creating and Accessing Private Disk and Tape Volumes

Operation	Format/Example	Comments
Creating a link between the volume and your process	**$ MOUNT device label [logical–name]**	
	`$ MOUNT DMA2: TESTDISK DISK`	Mounts TESTDISK on the device DMA2:. The logical name DISK is assigned the equivalence name of DMA2.
Creating a link between your process and a tape device, and specifying the volume to be placed on that device	**$ MOUNT device–name volume–label logical–name**	
	`$ MOUNT MT: XFILES TAPE`	Issuing the MOUNT command with a generic device disk specification causes the system to select an available device of the specified type for your process. Once this command is issued, someone must mount the volume or you must issue the CTRL/C sequence to cancel the request.

Table 8–1 (Cont.) Commands for Creating and Accessing Private Disk and Tape Volumes

Operation	Format/Example	Comments
Breaking the link between the volume and your process	**$ DISMOUNT[/NOUNLOAD] device**	If you want to keep the disk online when you dismount it, use the qualifier /NOUNLOAD.
	`$ DISMOUNT DMA2:`	Dismounts and automatically unloads the volume on DMA2:. Deletes the logical name assignment made by the MOUNT command. Unless the ALLOCATE command was used before the MOUNT command, this makes the device available to other users.
Deallocating a device	**$ DEALLOCATE device** `$ DEALLOCATE DMA2:`	Deallocates device DMA2:. Frees the device for use by other users. Does not delete a logical name assigned by the ALLOCATE command.

```
1 $ SHOW DEVICE DM:

  Device             Device       Error      Volume     Free   Trans  Mnt
  Name              Status       Count      Label     Blocks  Count  Cnt
  DMA0:              Online           0
2 $ ALLOCATE DM: DISK
  %DCL-I-ALLOC, DMA0: allocated
3 $ MOUNT/FOREIGN DISK
  %MOUNT-I-MOUNTED,              mounted on DMA0:
4 $ DISMOUNT/NOUNLOAD DISK
5 $ INITIALIZE DISK USER_DISK
6 $ MOUNT DISK USER_DISK
  %MOUNT-I-MOUNTED, USER_DISK    mounted on DMA0:
7 $ CREATE/DIRECTORY DISK:[HELP]
  $ DIRECTORY DISK:[HELP]
  %DIRECT-W-NOFILES, no files found
8 $ COPY SYS$HELP:NOTES.HLB DISK:[HELP]
  $ DIRECTORY DISK:[HELP]

  Directory DMA0:[HELP]

  NOTES.HLB;2

  Total of 1 file
9 $ SHOW DEVICE/FULL DISK

  Disk DMA0:, device type RK07, is online, allocated, deallocate on
  dismount, mounted, error logging enabled.

  Error count                16     Operations completed              1940
  Owner process         "SMITH"     Owner UIC                   [VMS,SMITH]
  Owner process ID     000000A2     Dev Prot    S:RWED,O:RWED,G:RWED,W:RWED
  Reference count            2     Default buffer size                512

  Volume label     "USER_DISK"     Relative volume no.                  0
  Cluster size               3     Transaction count                    1
  Free blocks            53691     Maximum files allowed             6723
  Extend quantity            5     Mount count                          1
  Mount status         Process     Cache name             "_DRA0:XQPCACHE"
  File ID cache size        64     Extent cache size                   64
  Quota cache size           0
  Write-thru caching enabled

  Volume is subject to mount verification, file high-water marking.

10 $ DISMOUNT DISK
11 $ DEALLOCATE DISK
  $
```

Example 8–1 Preparing and Transferring Files to a Disk Volume

Notes on Example 8–1:

The following comments are keyed to the example.

1 `$ SHOW DEVICE DM:`

The **SHOW DEVICE** command displays the status of devices on your system. In this example, you request a report on the status of all available RK06 and RK07 disk units by specifying the generic name DM as the single command parameter.

2 `$ ALLOCATE DM: DISK`

The **ALLOCATE** command causes the system to search for a free device of type DM and allocate it to your process. The message displayed at your terminal informs you that such a device was found: DMA0:. The system will assign the device name DMA0: to the logical name DISK and record it in your process logical name table.

At this point, place a disk in the device and load it.

3 `$ MOUNT/FOREIGN DISK`

The **MOUNT/FOREIGN** command makes the contents of your volume available to the system, but makes no assumptions concerning its file structure. DISK is the logical name assigned to the device name DMA0: by the **ALLOCATE** command of the preceding line.

In this case, the disk pack is a new one. Had it been used, you would have required VOLPRO privilege to mount it, unless its owner UIC matched your own.

4 `$ DISMOUNT/NOUNLOAD DISK`

Before you can build a file structure on the volume, you must first dismount it by issuing the **DISMOUNT** command. The **/NOUNLOAD** qualifier tells the system that you want to keep the device and volume in a ready state; without it, you would be required to load the unit again in the machine room. DISK is the logical name that you equated with DMA0: when you allocated the device.

5 `$ INITIALIZE DISK USER_DISK`

The **INITIALIZE** command builds a FILES–11 file structure on your new volume. By default, you are declared its owner (that is, your UIC becomes the owner UIC of the disk) and all user groups are granted all types of access (READ, WRITE, EXECUTE, and DELETE). The volume gets the label USER_DISK and DISK is the logical name of the device on which it is mounted.

6 `$ MOUNT DISK USER_DISK`

The **MOUNT** command makes the contents of the volume accessible to your process. The system compares the volume label to the specified label, USER_ DISK; a mismatch results in an error message. DISK is the logical name created when you allocate the RK07 disk unit (see note 2). Since the device is allocated to your process, no other user can access the volume. The volume protection code and the volume owner UIC determine your access to USER_DISK. In this case, you are the owner of the volume, and the volume protection code allows you unrestricted access to its contents. The message displayed at your terminal indicates that the **MOUNT** command executed successfully.

7
```
$ CREATE/DIRECTORY DISK:[HELP]
$ DIRECTORY DISK:[HELP]
```

Since you own the volume, you have the right to create user file directories. The **CREATE/DIRECTORY** command creates a directory named [HELP]. By default, your UIC becomes the owner UIC of the directory file. The **DIRECTORY** command confirms the existence of the new directory. At the moment, the directory lists no files.

8
```
$ COPY SYS$SYSHELP:NOTES.HLB DISK:[HELP]
$ DIRECTORY DISK:[HELP]
```

The **COPY** command transfers files from a directory on the system disk to the private disk you have created and mounted. In this case, you copy a file named NOTES.HLB in the directory logical name SYS$HELP to your private volume. The system lists the copied file in your newly created directory and assigns it an owner UIC and protection code equal to your current UIC and default protection code. The listing generated at your terminal by the **DIRECTORY** command confirms the success of the operation.

```
9   $ SHOW DEVICE/FULL DISK
```

To determine the characteristics of a volume, mount it and issue a **SHOW DEVICE/FULL** command, specifying the device that holds the volume as the target device. In this example, the device is the RK07 disk unit whose logical name is DISK. The protection code of the device permits anyone to load and access a volume according to the protection code of the volume. The last block of information in the display generated by the **SHOW DEVICE** command is volume information. Note the following characteristics:

- `Volume Label`: The label of the disk is USER_DISK, the one you specified in the **INITIALIZE** command.
- `Owner UIC`: The owner UIC associated with the disk is [VMS,SMITH], the default UIC of your process.
- `MOUNT Count`: Only one process has mounted the volume (your own).
- `Relative Volume Number`: The relative volume number is zero, indicating that this volume is not a member of a series of tapes. When a volume consists of more than one tape reel, it is referred to as a volume set.

```
10  $ DISMOUNT DISK
```

The **DISMOUNT** command closes access to the volume mounted on the device whose name is associated with DISK. The system automatically unloads the volume; no one can access the volume without first reloading it by depressing an appropriate switch on the device. At this point, remove the disk pack from the unit.

```
11  $ DEALLOCATE DISK
```

The **DEALLOCATE** command releases the device from the control of your process. It does not delete the logical name DISK, however, from your process logical name table.

8.5 THE BACKUP UTILITY

The Backup utility provides a means of protection against file or volume corruption. It does this by creating equivalent backup copies. **BACKUP** can be used to back up an entire volume set in a single operation, or to back up selected groups of files from a volume. Thus, if an original file or volume is lost, deleted, or corrupted, a backup copy containing the original data will be available to replace it.

When **BACKUP** saves files, it creates a special file in BACKUP format on the specified output volume. This special BACKUP file is called a *save set*. Only **BACKUP** can interpret save sets, because they are written in a unique BACKUP format that improves the efficiency of file transfer and storage.

BACKUP is used to perform the following operations:

- Copy files between disks.
- Save disk files to a BACKUP save set.
- Restore files to disk from a BACKUP save set.

The format of the Backup utility is:

$ BACKUP/qualifier input-specifier output-specifier

- Files specified are placed in a save set.
- A save set can exist on a tape or disk.
- The Backup utility can read the format of information in a save set.
- Tapes must be mounted using the **/FOREIGN** qualifier.
- When used with tape volumes, **BACKUP** can create and gain access to save sets only.

8.5.1 Save–Set Specifications

A *save–set specification* is a label for a BACKUP save set. The Backup utility creates and labels a save set and then writes files to the save set. A save–set specification may include:

- A node name
- A device specification
- A directory
- A save–set name
- A period (the mandatory delimiter after the save–set name)
- A save–set type (usually BCK or SAV)

Example 8–2 describes how to create a save set on a tape volume.

```
1 $ SET DEFAULT [SMITH]
2 $ ALLOCATE MUA0:
3 $ INITIALIZE MUA0: SOURCE
4 $ MOUNT/FOREIGN MUA0:
5 $ BACKUP/IGNORE=LABEL_PROCESSING [...] MUA0:MY_BACKUP.BCK
6 $ DISMOUNT MUA0:
7 $ DEALLOCATE MUA0:
```

Example 8–2 Creating a Save Set on a Tape Volume

Notes on Example 8–2:

1 Sets the default directory to the directory from which the files will be backed up.

2 The **ALLOCATE** command allocates tape drive MUA0: for your exclusive use.

3 Normally magnetic tapes do not have to be initialized for BACKUP operations. However, if a blank tape has never been initialized, or you are writing a save set on a tape that has a non–ANSI label, then the tape should be initialized.

4 The **MOUNT** command mounts the tape on the drive. It still must be physically mounted on the drive, either by the user or a system operator. The **/FOREIGN** qualifier indicates that the volume is not a file–structured volume.

5 The **/IGNORE=LABEL_PROCESSING** qualifier tells BACKUP not to check for the tape label. If no label is specified, the name of the save set must match the tape label. The [...] form of the **DIRECTORY** command indicates that all of the files in the default directory and any subdirectories are to be saved. They will be saved in the save set named MY_BACKUP.BCK.

6 The **DISMOUNT** command dismounts the tape on MUA0:.

7 The **DEALLOCATE** command deallocates the tape drive so other users can access the drive.

Example 8–3 shows how to transfer files to a tape volume.

```
$ ALLOCATE MUA0:
%DCL-I-ALLOC, _WHYNOT$MUA0: allocated

$ INITIALIZE MUA0: SOURCE

$ MOUNT/FOREIGN MUA0:
%MOUNT-I-MOUNTED, SOURCE mounted on _WHYNOT$MUA0:

$ DIRECTORY [...]
Directory DISK:[SMITH]
EVE.INIT;1          FORTRAN.DIR;1       INSERT.FYI;6
JOE_EVE.TPU$SECTION;1                   LOGIN.COM;21        PASCAL.DIR;1
Total of 6 files.

Directory DISK:[SMITH.FORTRAN]
EXAMPLES.FOR;1      FILES.FOR;1         TEXT.FOR;1
Total of 3 files.

Directory DISK:[SMITH.PASCAL]
EXAMPLES.PAS;1      FILES.PAS;1         TEXT.PAS;1
Total of 3 files.

Grand total of 3 directories, 12 files.

$ SET DEFAULT [.FORTRAN]

$ BACKUP/IGNORE=LABEL_PROCESSING *.*;* MUA0:FOR.BCK

$ SET DEFAULT [SMITH.PASCAL]

$ BACKUP/IGNORE=LABEL_PROCESSING *.*;* MUA0:PAS.BCK

$ BACKUP/REWIND/LIST MUA0:PAS.BCK
```

Example 8–3 Transferring Files to a Tape Volume

```
Listing of save set(s)

Save set:           PAS.BCK
Written by:         SMITH
UIC:                [000011,000051]
Date:               25-JAN-1988 13:30:10.59
Command:            BACKUP/IGNORE=LABEL_PROCESSING *.*;* MUA0:PAS.BCK
Operating system:   BACKUP version:    V5.0
CPU ID register:    08000000
Node name:          WHYNOT::
Written on:         _WHYNOT$MUA0:
Block size:         8192
Group size:         10
Buffer count:       3

[SMITH.PASCAL]EXAMPLES.PAS;1                          2  21-JAN-1988 15:17
[SMITH.PASCAL]FILES.PAS;1                             2  21-JAN-1988 15:18
[SMITH.PASCAL]TEXT.PAS;1                              2  21-JAN-1988 15:17

Total of 3 files, 6 blocks
End of save set

$ BACKUP/REWIND/LIST MUA0:FOR.BCK
Listing of save set(s)

Save set:           FOR.BCK
Written by:         SMITH
UIC:                [000011,000051]
Date:               25-JAN-1988 13:31:37.89
Command:            BACKUP/IGNORE=LABEL_PROCESSING *.*;* MUA0:FOR.BCK
Operating system:   VAX/VMS version X5.0
BACKUP version:     V5.0
CPU ID register:    08000000
Node name:          WHYNOT::
Written on:         _WHYNOT$MUA0:
Block size:         8192
Group size:         10
Buffer count:       3

[SMITH.FORTRAN]EXAMPLES.FOR;1                         2  21-JAN-1988 15:16
[SMITH.FORTRAN]FILES.FOR;1                            2  21-JAN-1988 15:16
[SMITH.FORTRAN]TEXT.FOR;1                             2  21-JAN-1988 15:16

Total of 3 files, 6 blocks
End of save set

$ DISMOUNT MUA0:
$ DEALLOCATE MUA0:
```

Example 8–3 (Cont.): Transferring Files to a Tape Volume

Table 8–2 shows commands for displaying the characteristics of devices and volumes.

Table 8–2 Commands for Displaying Device and Volume Characteristics

Operation	Format/Example	Comments
Displaying the characteristics of a device	**$ SHOW DEVICE/FULL device** `$ SHOW DEVICE/FULL DMA2:`	Displays the characteristics of device DMA2:
Displaying the characteristics of a volume	**$ SHOW DEVICE/FULL device** `$ SHOW DEVICE/FULL MYDISK`	Displays the characteristics of the volume currently mounted on the device whose logical name is MYDISK

8.6 USING PRIVATE VOLUMES

When you want to use a volume that you have created and used previously, complete the following steps to make the volume available to your process:

1. Issue a **MOUNT** command using a generic device specification. (If you want to use the same device to mount several successive volumes, you can allocate it first.)
2. Load the volume on the device specified by the VMS system in response to your **MOUNT** request.
3. Use the volume to perform the needed functions.
4. Issue the **DISMOUNT** command to break your link with the volume and free the device for other users.
5. Unload the volume from the disk or tape device.

Example 8–4 describes how to restore files from a tape to a directory.

```
1 $ MOUNT/FOREIGN MUA0:
  %MOUNT-I-MOUNTED, SOURCE mounted on _WHYNOT$MUA0:

  $ DIRECTORY [SMITH.FORTRAN]
  %DIRECT-W-NOFILES, no files found
2
  $ DIRECTORY [SMITH.PASCAL]
  %DIRECT-W-NOFILES, no files found

3 $ SET DEFAULT [SMITH.FORTRAN]

4 $ BACKUP/IGNORE=LABEL_PROCESSING MUA0:FOR.BCK *.*;*

5 $ DIRECTORY

  Directory DISK:[SMITH.FORTRAN]

  EXAMPLES.FOR;1      FILES.FOR;1         TEXT.FOR;1

  Total of 3 files.

6 $ SET DEFAULT [SMITH.PASCAL]

7 $ BACKUP/REWIND/IGNORE=LABEL_PROCESSING MUA0:PAS.BCK *.*;*

8 $ DIRECTORY

  Directory DISK:[SMITH.PASCAL]

  EXAMPLES.PAS;1      FILES.PAS;1         TEXT.PAS;1

  Total of 3 files.
```

Example 8–4 Restoring Files from a Tape to a Directory

Notes on Example 8–4:

Example 8–4 describes the procedure to restore the two save sets named FOR.BCK and PAS.BCK from a magnetic tape back to two directories on your default disk. These save sets contain backups of the directories named [SMITH.FORTRAN] and [SMITH.PASCAL].

The following comments are keyed to the example.

1 `MOUNT/FOREIGN MUA0:`

The **MOUNT** command creates a link between a tape device and your process. The system responds with a message saying the volume is loaded onto the drive. Some systems inform an operator that you need to have a volume loaded, and the operator will load the volume. On other systems, you must load the volume yourself.

2 The two **DIRECTORY** commands show that both the subdirectories [.FORTRAN] and [.PASCAL] do not contain any files.

3 The **SET DEFAULT [SMITH.FORTRAN]** command moves you to the subdirectory into which the files should be restored.

4 **BACKUP/IGNORE=LABEL_PROCESSING MUA0:FOR.BCK *.*;*** tells the system to ignore any tape label checking (via the use of the **/IGNORE=LABEL_PROCESSING** qualifier). The device and save–set name are given (MUA0:FOR.BCK). The wildcard asterisk syntax requests that all file names, file types, and version numbers in the save set (FOR.BCK) be copied to the subdirectory [.FORTRAN].

5 The **DIRECTORY** command is issued to confirm that all files were transferred from the tape device to the subdirectory [.FORTRAN]. In this example, three files were copied to the subdirectory.

6 The **SET DEFAULT [SMITH.PASCAL]** command is given to move to the subdirectory [.PASCAL].

7 The **BACKUP** command is again issued, the only difference being that the save–set name is PAS.BCK.

8 The **DIRECTORY** command is issued to confirm that all files were transferred from the tape device to the subdirectory [.PASCAL]. In this example, three files were copied to the subdirectory.

8.7 MAINTAINING, SHARING, AND EXTENDING PRIVATE VOLUMES

In addition to the basic operations described previously, you can perform a number of more specialized tasks on disk and tape volumes. This section examines a number of these, including:

- Protecting the contents of a volume
- Sharing access to a disk volume
- Mounting a volume whose label is unknown

8.7.1 Protecting and Sharing Access to Volumes

You control access to disk and tape volumes by the values you assign to the following parameters:

- The volume owner UIC
- The volume protection code

The system sets these parameters when you initialize the volume, and overrides them when the volume is mounted.

For you to initialize a volume, one of the following conditions must exist:

- The volume is blank
- The owner UIC matches your own
- You have VOLPRO privilege

Note that there are separate owner UICs and protection codes for:

- Volumes (disk and tape)
- Directories (disk only)
- Files (disk only)

When you want to override the protection or owner UIC set during the initializing phase, or extend volume access to other users, you can use qualifiers to the **MOUNT** command. Values set in the mounting phase stay in effect until the volume is dismounted.

8.7.2 Mounting a Volume with an Unknown Label

From time to time, you may forget the name you assigned to a volume. To determine the label name, mount the volume, using the following command syntax:

$ MOUNT/OVERRIDE=IDENTIFICATION device–name volume–label – logical–name

The **MOUNT/OVERRIDE** command allows you to successfully mount a volume without knowing its label, providing that you own it or have VOLPRO privilege. Example 8–5 illustrates how to use the **MOUNT/OVERRIDE** command to mount a disk whose label is unknown.

```
1 $ MOUNT/OVERRIDE=IDENTIFICATION DM: UNKNOWN MYDISK
  %MOUNT-I-MOUNTED, MYVOL        mounted on DMA0:
2 $ SHOW DEVICE/FULL MYDISK

  Disk DMA0:, device type RK07, is online, allocated, deallocate on dismount,
  mounted, error logging enabled.

  Error count                    33    Operations completed              3891
  Owner process              "SMITH"   Owner UIC                       [100,0]
  Owner process ID          000000A2   Dev Prot    S:RWED,O:RWED,G:RWED,W:RWED
  Reference count                2     Default buffer size                512

  Volume label               "MYVOL"   Relative volume no.                  0
  Cluster size                   3     Transaction count                    1
  Free blocks                53703     Maximum files allowed             6723
  Extend quantity                5     Mount count                          1
  Mount status               Process   Cache name              "DRA0:XQPCACHE"
  File ID cache size            64     Extent cache size                   64
  Quota cache size               0
  Write-thru caching enabled

  Volume is subject to mount verification, file high-water marking.
```

Example 8–5 Mounting a Disk with an Unknown Label

Notes on Example 8–5:

The following comments are keyed to the example.

1 `$ MOUNT/OVERRIDE=IDENTIFICATION DM: UNKNOWN MYDISK`

The **MOUNT/OVERRIDE** command successfully mounts a disk for which a label is not specified. UNKNOWN is chosen as the volume label. You can specify anything for the volume label parameter or you can omit it; the **MOUNT** command ignores whatever you enter. MYDISK is the logical name of the device on which the volume is loaded. The message returned to your terminal reports the label value.

2 `$ SHOW DEVICE/FULL MYDISK`

The **SHOW DEVICE/FULL** command confirms that the label of the volume is identical to the one reported in the MOUNT message at your terminal.

8.8 SUMMARY

Creating Private Volumes: The Command Sequence

Table 8–3 lists the commands that are used to create and access disk and tape volumes.

Table 8–3 Creating and Accessing Private Volumes

Operation	Comments/Format
Allocating a Device	Allocates a device for exclusive use.
	$ ALLOCATE device [logical–name]
Initializing a tape or disk	Establishes volume ownership and protection.
	$ INITIALIZE device label
Making the volume accessible to you	You can access the device as well as manipulating files on the volume.
	$ MOUNT device label [logical–name]
Prohibiting further access to the volume	Closes all open files. Dismounts and unloads the volume.
	$ DISMOUNT device
Deallocating a device	Frees the device for use by other users.
	$ DEALLOCATE device

The Backup Utility

The Backup utility performs the following operations:

- Copies disk files
- Saves disk files to a BACKUP save set
- Restores files to disk from a BACKUP save set

Format:

$ BACKUP/qualifier input-specifier output-specifier

- Tapes must be mounted using the **/FOREIGN** qualifier to the **MOUNT** command.
- Files specified are placed in a save set, which can be on tape or disk.
- When used with tape volumes, **BACKUP** can create and gain access to save sets only.

8.9 WRITTEN EXERCISE I

The list below contains the major steps that you must complete to create and use a private volume. Indicate the order of these steps by writing the appropriate number in the space that precedes each one.

1. _____ Allocate device
2. _____ Deallocate device
3. _____ Dismount volume
4. _____ Initialize volume
5. _____ Load volume
6. _____ Mount volume
7. _____ Unload volume

8.10 WRITTEN EXERCISE II

Choose the VMS command best suited to perform each of the following operations and write its letter in the preceding space.

VMS Commands

a. ALLOCATE

b. DEALLOCATE

c. DISMOUNT

d. INITIALIZE

e. MOUNT

f. SHOW DEVICE/FULL

Operations

1. _____ Build a FILES–11 structure on a disk or an ANSI Level 3 structure on a tape.

2. _____ Terminate access by your process to the contents of a volume.

3. _____ Display the owner UIC and protection code of a volume.

4. _____ Initiate access by your process to the contents of a volume.

5. _____ Release a device from exclusive use by your process.

6. _____ Reserve a device for exclusive use by your process.

8.11 WRITTEN EXERCISE III

Write a VMS command string to perform each of the following operations.

1. Allocate any available RK06/RK07 device to your process and assign the logical name RL_DISK to it.
2. Dismount the disk volume on RL_DISK without unloading it.
3. Allocate any available magnetic tape unit to your process and assign the logical name TAPE to it.
4. Initialize a tape volume you have loaded on TAPE. Assign the label TAP_BK to the unit.
5. Mount TAP_BK on the tape device that you have allocated to your process so that the Backup utility can process it.
6. Back up the most recent version of each file in the hierarchy associated with your default UFD to a save set on TAP_BK.
7. List the contents of the save set on TAP_BK at your terminal.
8. Terminate access to TAP_BK, allowing the system to automatically unload the volume.
9. Release the tape device so others on your system can use it.
10. Delete the logical name TAPE from the logical name table that stores it.

8.12 WRITTEN EXERCISE IV

Write a VMS command string to perform each of the following operations.

1. Mount a volume whose label is unknown to you on device MTA2:.
2. Initialize a volume located on device DMA1:. Assign it the label MYVOL. Set its owner UIC to that of your current process. Extend all access rights to members of the OWNER and SYSTEM categories; deny all access rights to members of the GROUP and WORLD categories.
3. Mount the volume you created in the preceding example. Set the volume UIC to [100,200] and allow unrestricted access to members of all user categories.
4. Create a user file directory on MYVOL named PUBLIC, declaring the owner to have a UIC of [100,200].
5. Copy all files listed in [PUBLIC] on MYVOL to MYTAPE.
6. Another user on your system has mounted a volume on a RL02 device. The volume is enabled for sharing. The volume label is SHARE_DISK. Mount the volume so that you can access its contents from your own process. Assign the logical name RL_DISK to the RL02 device.
7. Dismount the shared disk volume on RL_DISK. If your process is the last to use the volume, unload it.

8.13 LABORATORY EXERCISE I

Complete the following exercises at an interactive terminal.

1. Allocate the tape.
2. Initialize the tape, giving it a label name of MYTAPE.
3. Mount the tape, so that BACKUP can be used.
4. Obtain a listing of the files in your directory.
5. Transfer all files from your directory to the tape.
6. Confirm that all files transferred successfully to the tape.
7. Dismount the tape.
8. Deallocate the tape.

8.14 WRITTEN EXERCISE I—SOLUTIONS

The list below contains the major steps that you must complete to create and use a private volume. Indicate the order of these steps by writing the appropriate number in the space that precedes each one.

1. 1 Allocate device
2. 7 Deallocate device
3. 5 Dismount volume
4. 3 Initialize volume
5. 2 Load volume
6. 4 Mount volume
7. 6 Unload volume

8.15 WRITTEN EXERCISE II—SOLUTIONS

Choose the VMS command best suited to perform each of the following operations and write its letter in the preceding space.

VMS Commands

a. ALLOCATE

b. DEALLOCATE

c. DISMOUNT

d. INITIALIZE

e. MOUNT

f. SHOW DEVICE/FULL

Operations

1. __d__ Build a FILES–11 structure on a disk or an ANSI Level 3 structure on a tape.
2. __c__ Terminate access by your process to the contents of a volume.
3. __f__ Display the owner UIC and protection code of a volume.
4. __e__ Initiate access by your process to the contents of a volume.
5. __b__ Release a device from exclusive use by your process.
6. __a__ Reserve a device for exclusive use by your process.

8.16 WRITTEN EXERCISE III—SOLUTIONS

1. Allocate any available RL02 device to your process and assign the logical name RL_DISK to it.

```
$ ALLOCATE DL: RL_DISK
```

2. Dismount the disk volume on RL_DISK without unloading it.

```
$ DISMOUNT/NOUNLOAD RL_DISK
```

3. Allocate any available magnetic tape unit to your process and assign the logical name TAPE to it.

```
$ ALLOCATE MT: TAPE
```

4. Initialize a tape volume that you have loaded on TAPE. Assign the label TAP_BK to the unit.

```
$ INITIALIZE TAPE TAP_BK
```

5. Mount TAP_BK on the tape device that you have allocated to your process so that the Backup utility can process it.

```
$ MOUNT/FOREIGN TAPE
```

6. Back up the most recent version of each file in the hierarchy associated with your default UFD to a save set on TAP_BK.

```
$ BACKUP/IGNORE=LABELPROCESSING [...]*.* TAPE:TAP_BK.BCK
```

   This answer assumes that your current default directory is your UFD.

7. List the contents of TAP_BK at your terminal.

```
$ BACKUP/LIST TAPE:TAP_BK.BCK
```

8. Terminate access to TAP_BK, allowing the system to automatically unload the volume.

```
$ DISMOUNT TAPE
```

9. Release the tape device so others on your system can use it.

```
$ DEALLOCATE TAPE
```

10. Delete the logical name TAPE from the logical name table that stores it.

```
$ DEASSIGN TAPE
```

8.17 WRITTEN EXERCISE IV—SOLUTIONS

1. Mount a volume whose label is unknown to you on device MTA2:

```
$ MOUNT/OVERRIDE=IDENTIFICATION MTA2:
```

2. Initialize a volume located on device DMA1:. Assign it the label MYVOL. Set its owner UIC to that of your current process. Extend all access rights to members of the OWNER and SYSTEM categories; deny all access rights to members of the GROUP and WORLD categories.

```
$ INITIALIZE/NOSHARE DMA1: MYVOL
```

3. Mount the volume that you created in the preceding example. Set the volume UIC to [100,200] and allow unrestricted access to members of all user categories.

```
$ MOUNT/OWNER_UIC=[100,200]/PROTECTION=(W:RWED) DMA1: MYVOL
```

4. Create a user file directory on MYVOL named PUBLIC, declaring the owner to have a UIC of [100,200].

```
$ CREATE/DIRECTORY/OWNER=[100,200] DMA1:[PUBLIC]
```

5. Copy all files listed in [PUBLIC] on MYVOL to MYTAPE.

```
$ COPY DMA1:[PUBLIC]*.*;* MYTAPE
```

6. Another user on your system has mounted a volume on an RL02 device. The volume is enabled for sharing. The volume label is SHARE_DISK. Mount the volume so that you can access its contents from your own process. Assign the logical name RL_DISK to the RL02 device.

```
$ MOUNT/SHARE RL: SHARE_DISK RL_DISK
```

7. Dismount the shared disk volume on RL_DISK. If your process is the last to use the volume, unload it.

```
$ DISMOUNT RL_DISK
```

8.18 LABORATORY EXERCISE I—SOLUTIONS

Complete the following exercises at an interactive terminal. Note that your device names will differ from the device and directory names given in the solutions.

1. Allocate the tape.

```
$ ALLOCATE MUA0:
```

2. Initialize the tape, giving it a label name of MYTAPE.

```
$ INITIALIZE MUA0: MYTAPE
```

3. Mount the tape, so that BACKUP can be used.

```
$ MOUNT/FOREIGN MUA0:
```

4. Obtain a listing of the files in your directory.

```
$ DIRECTORY
```

5. Transfer all files from your directory to the tape.

```
$ BACKUP/IGNORE=LABELPROCESSING *.*;* MUA0:JAN1.BCK
```

6. Confirm that all files transferred successfully to the tape.

```
$ BACKUP/REWIND/LIST MUA0:JAN1.BCK
```

7. Dismount the tape.

```
$ DISMOUNT MUA0:
```

8. Deallocate the tape.

```
$ DEALLOCATE MUA0:
```

SUBMITTING BATCH AND PRINT JOBS

9.1 INTRODUCTION

When you issue the **PRINT** command to print a file, all the system printers may already be in use. For this reason, the VMS system maintains a list of all print requests. This ordered list is called a *print queue*, and the requests are called *print jobs*. The position of an entry in the queue depends on its priority, size, and length of time in the queue, respectively. When the job moves up to the front of the queue, the VMS system passes it to the first available printer, which prints the file.

The **PRINT** command finishes executing as soon as it enters the print job in the print queue. The system then displays the DCL prompt, so you can issue other DCL commands. You do not have to wait until the job is printed to continue your work. Your job is printed when it reaches the front of the queue.

When you execute a command procedure interactively, the VMS system carries out each DCL command exactly as if you had typed it at your terminal. Because it executes these commands in the context of your *interactive process*, you cannot issue any additional DCL commands until the entire command procedure has completed.

The VMS system creates a separate process to execute your command procedure, called a *batch process*. Because a batch process is independent of your interactive process, it does not prevent you from issuing DCL commands interactively. Each batch process is, in effect, another user of your system. To specify the command procedure you want executed in a batch process, you use the **SUBMIT** command. The VMS system maintains a *batch queue* to handle batch jobs. The **SUBMIT** command places your request, called a *batch job*, in the queue. When the batch job moves up to the front of the queue, a batch process is created to execute the job.

Although batch jobs and print jobs have different functions, batch queues and print queues have much in common. The system manager customizes these queues for each VMS system. A queue structure that matches the resources and desired uses of the system can improve system performance.

9.2 OBJECTIVES

To effectively use facilities for handling batch and print jobs, you should be able to perform the following operations by entering commands at a terminal:

- Print one or more files.
- Submit command procedures to be executed as a batch job.
- Display and modify the status or characteristics of a print or batch job.
- Delay processing of batch or print jobs.
- Delete a batch or print job from its queue.

9.3 RESOURCES

- *VMS DCL Dictionary*
- *Guide to Using VMS*

9.4 PRINTING A FILE

When you issue the **PRINT** command, the VMS system assigns a number to your print job. Your terminal displays this number, as Example 9–1 shows. *Job numbers* record the order in which jobs are queued. The **PRINT** command uses a default file type of LIS if you do not specify another type.

By default, the system places your job in the standard system print queue, SYS$PRINT. Jobs on SYS$PRINT are printed on the first available print device. In Example 9–1, the first available print device was LPA0:.

```
$ PRINT MYFILE.TXT
Job MYFILE (queue SYS$PRINT, entry 456) started on LPA0
$
```

Example 9–1 Issuing the PRINT Command

9.4.1 Using a Particular Printer

You can use the **/QUEUE** qualifier of the **PRINT** command to request that your job be printed on a particular printer. Each printer in your system has an *execution queue* associated with it. If you use the **/QUEUE** qualifier to specify an execution queue, only the device associated with that execution queue can print your job.

The name of each execution queue is the same as the name of the associated device, without the colon. For example, the VMS system associates device LPA0: with execution queue LPA0.

The standard system print queue, SYS$PRINT, is called a *generic queue*. As a device becomes available, the VMS system takes a job from the front of this generic queue and places it on the execution queue of the available device. The device then prints the job.

The system manager determines which execution queues receive print jobs from the generic print queue, and which do not. Execution queues that receive print jobs from the generic print queue are said to have generic printing *enabled*. Execution queues on which generic printing is not enabled cannot receive jobs from SYS$PRINT.

Usually, the system manager enables generic printing on a group of similar printers in a common location. A printer in a different location, or one loaded with a different ribbon or type of paper, would not have generic printing enabled. Figure 9–1 represents a system with three printers, LPA0:, LPB0:, and LPC0:. In the figure, the square to the right of each printer represents the print job being printed on that printer. All other squares represent print jobs waiting to be printed.

Table 9–1 shows how to use the **PRINT** command and the **SHOW QUEUE** command to queue print jobs and display status information on them. Later examples demonstrate the format of this status information.

Table 9–1 Queuing a Print Job

Operation	Format/Example	Comments
Printing a single–file job	**$ PRINT file–specification** `$ PRINT MYTEXT`	The PRINT command uses a default file type of LIS. The file printed is MYTEXT.LIS in this example.
Printing a multifile job	**$ PRINT file–specification[,...]** `$ PRINT MYTEXT,MEMO.TXT`	The files MYTEXT.LIS and MEMO.TXT are printed as a single print job.
Printing a job on a specified printer	**$ PRINT/QUEUE=que–name file–spec** `$ PRINT/QUEUE=LPA0 MEMO.TXT`	Either the /QUEUE or /DEVICE qualifier can be used. They are equivalent.
Printing a job at a specified time	**$ PRINT/AFTER=time file–spec** `$ PRINT/AFTER=18:00 MYTEXT.TXT`	The job will be processed after the time specified in the /AFTER= qualifier.

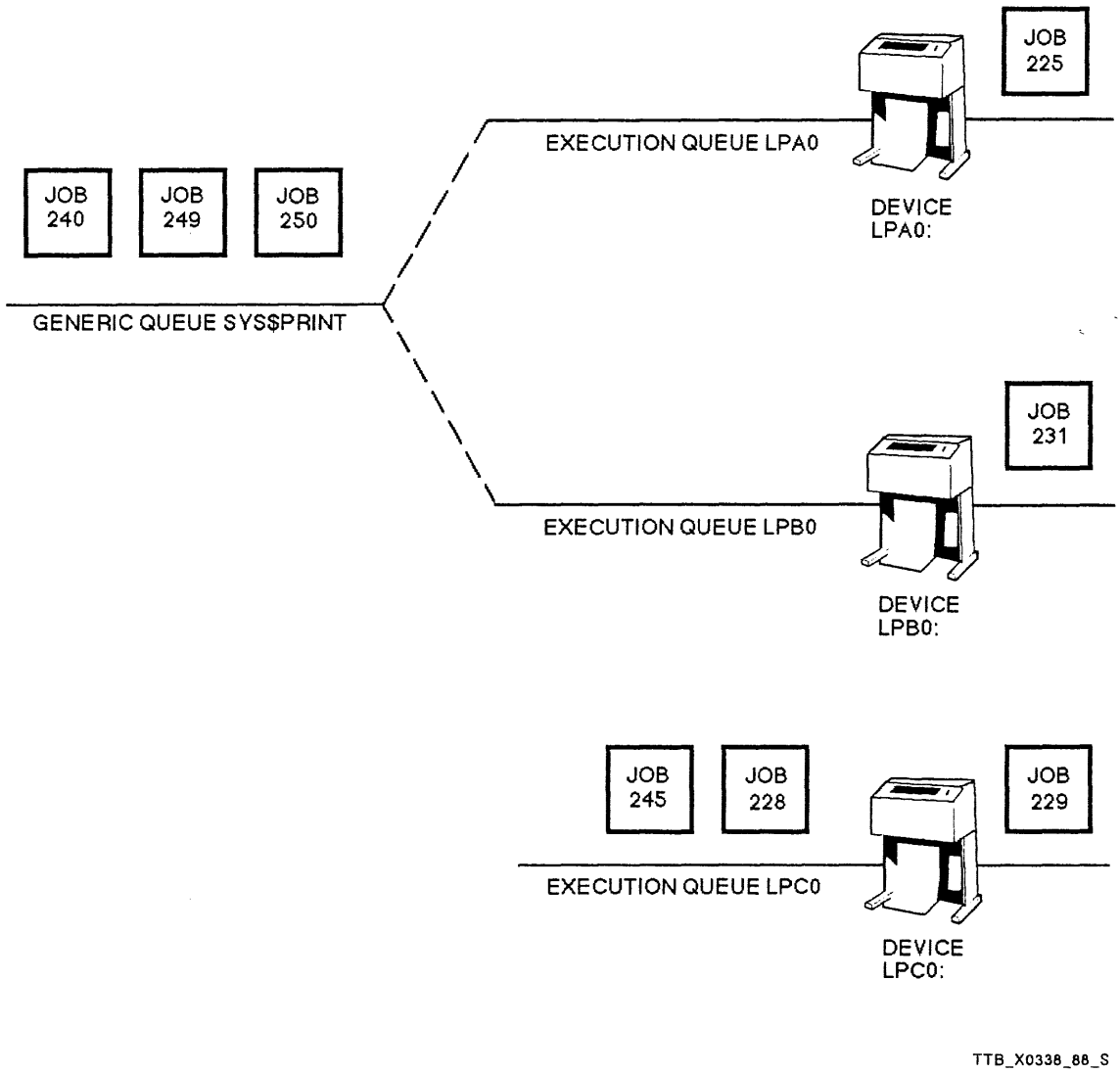


Figure 9–1 Execution and Generic Print Queues

Notes on Figure 9–1:

1. A printer can print only one job at a time. The job being printed is called the *current job*. Job 225 is the current job on queue LPA0. Similarly, job 231 is the current job on queue LPB0, and job 229 is the current job on queue LPC0.

2. Jobs waiting their turn to be printed are called *pending jobs*. Jobs 250, 249, and 240 are pending jobs on queue SYS$PRINT. Similarly, jobs 228 and 245 are pending jobs on queue LPC0.

3. SYS$PRINT is a generic queue.

   Because there are no execution printers associated with a generic queue, a generic queue cannot have a current job. Only execution queues have current jobs.

4. Execution queues LPA0 and LPB0 have generic printing enabled. Generic queue SYS$PRINT passes its jobs to LPA0 and LPB0 to be printed.

   Generic queues do not release a job to an execution queue until the associated device is ready to print it. That is why all pending jobs are on SYS$PRINT, not on LPA0 and LPB0.

   In this figure, if the VMS system queued a print job directly to LPA0, it would be a pending job on queue LPA0. However, this job would not necessarily be printed before the pending job on SYS$PRINT. The VMS system would base its selection of a current job for LPA0 on the order in which it queued the jobs.

5. Execution queue LPC0 does not have generic printing enabled. Jobs 229, 228, and 245 have been queued directly to this queue. They have not passed through SYS$PRINT.

9.4.2 Specifying the Characteristics of Print Jobs

When you queue a print job, you can also control the appearance of the job, by using the following positional qualifiers to the **PRINT** command.

- **/JOB_COUNT**

  Specifies the number of times your job is printed. A job can be printed from 1 to 255 times. The default is one printing.

- **/COPIES**

  Specifies the number of copies to print. The number of copies can be 1 – 255. By default, the **PRINT** command prints a single copy of a file.

- **/[NO]SPACE**

  Controls whether output is to be double–spaced. The default is **/[NO]SPACE**, which results in single–spaced output.

- **/PAGES=([lowlim,],uplim)**

  Specifies the number of pages to print. **Lowlim** is the first page printed and **uplim** is the last page printed.

- **/AFTER=time**

  Specifies the time to print. Time can be specified as absolute time, or a combination of absolute and delta time. The default is the current date and time.

- **/NOTIFY**

  Controls whether the system notifies you when the job is completed or aborted. The default is **/[NO]NOTIFY**.

Table 9–2 shows how to specify certain characteristics of print jobs.

Table 9–2 Setting the Characteristics of a Print Job

Operation	Comments and Examples
Printing multiple copies	**/JOB_COUNT** is a command qualifier that specifies how many times to duplicate the entire job. This example requests three printings of the file MEMO.TXT: **$ PRINT/JOB_COUNT=number file-specification** `$ PRINT/JOB_COUNT=3 MEMO.TXT` **/COPIES** is a command qualifier that can be different for each file in a job. This example requests that two copies of the file MEMO.TXT be printed: **$ PRINT/COPIES=number file-specifications** `$ PRINT/COPIES=2 MEMO.TXT` This example requests that two copies of the file MEMO.TXT and three copies of the file MYFILE.TXT be printed: `$ PRINT MEMO.TXT/COPIES=2,MYFILE.TXT/COPIES=3` This example requests that the entire job print twice. /COPIES=3 requests that the second file (FILE2.TXT) prints three times within each job count, giving a total of six copies for FILE2.TXT. `$ PRINT/JOB_COUNT=2 FILE1.TXT,FILE2.TXT/COPIES=3`
Controlling the spacing between lines and pages	**/SPACE** is a command qualifier signifying double–spaced output. The default of /NOSPACE results in single–spaced output. `$ PRINT MEMO.TXT/SPACE`
Specifying number of pages to print	By default, all pages of a job are printed. If the lowlim parameter is omitted, the job begins printing on the first page. **$ PRINT/PAGES=([lowlim,]uplim) file-specification** In this example, pages 6 through 8 are printed: `$ PRINT/PAGES=(6,8) MYFILE.TXT` In this example, pages 6 through end of file are printed: `$ PRINT/PAGES=(6,"") MYFILE.TXT`

9.5 OBTAINING STATUS OF QUEUES

There are several qualifiers you can use with the **SHOW QUEUE** command to obtain the status of print jobs at any given time. The format for using these qualifiers is:

$ SHOW QUEUE/qualifiers [queue–name]

where queue–name is the name of the printer.

Some qualifiers also have keywords that can be used to obtain additional information. The format for using keywords is:

$ SHOW QUEUE/qualifiers[=keyword[,...]] [queue–name]

To see all the entries on a particular printer, use the **/ALL_ENTRIES** qualifier with the **SHOW QUEUE** command. For example:

```
$ SHOW QUEUE/ALL_ENTRIES SYS$PRINT
```

shows you the status of all the print jobs on the SYS$PRINT queue:

```
Terminal queue SYS$PRINT, on WHYNOT::$PRINTER, mounted form DEFAULT
Jobname         Username       Entry     Blocks       Status
-------         --------       -----     ------       ------
MYFILE          SMITH           45         60       Printing
LICENSES        SMITH           48         78       Pending
TAGS            SMITH           49         88       Pending
OFFICERS        SMITH           52         90       Pending
```

(Example 9–2 gives you a detailed explanation of the information obtained here.)

The **BY_JOB_STATUS** qualifier is used with the following keywords.

- **EXECUTING** (Displays executing jobs)
- **HOLDING** (Displays jobs on hold)
- **PENDING** (Displays pending jobs)
- **RETAINED** (Displays jobs retained in the queue after execution)
- **TIMED_RELEASE** (Displays jobs on hold until a specified time)

Example:

```
$ SHOW QUEUE/BY_JOB_STATUS=TIMED_RELEASE SYS$PRINT
Terminal queue SYS$PRINT, on WHYNOT::$PRINTER, mounted form DEFAULT
Jobname         Username     Entry  Blocks  Status
-------         --------     -----  ------  ------
MYFILE          SMITH         96       1  Holding until 2-DEC-1988 15:00
```

The **DEVICE** qualifier is used with the following keywords.

- PRINTER (Displays all print queues)
- SERVER (Displays all server queues)
- TERMINAL (Displays all terminal queues)

Example:

```
$ SHOW QUEUE/DEVICE=SERVER
Server queue WHYNOT$NARROW, stopped, on WHYNOT::, mounted form DEFAULT
Jobname         Username     Entry  Blocks  Status
-------         --------     -----  ------  ------
MYFILE          SMITH         97       1  Holding until  2-DEC-1988 15:00
Server queue WHYNOT$WIDE, stopped, on WHYNOT::, mounted form DEFAULT
```

The **SHOW ENTRY** command gives you the status of a print job by number. This is the number assigned by the print queue and displayed by the **SHOW QUEUE** command. For example:

```
$ SHOW ENTRY 96
Jobname         Username     Entry  Blocks  Status
-------         --------     -----  ------  ------
MYFILE          SMITH         96       1  Holding until 2-DEC-1988 15:00
On terminal queue SYS$PRINT
```

shows you the status of print job number 96.

The **/FULL** qualifier of the **SHOW ENTRY** command gives you additional information about your print job, such as when it was submitted and what the priority is. For example:

```
$ SHOW ENTRY 96/FULL
Jobname         Username     Entry  Blocks  Status
-------         --------     -----  ------  ------
MYFILE          SMITH         96       1  Holding until  2-DEC-1988 15:00
  On terminal queue SYS$PRINT
  Submitted  2-DEC-1988 09:18 /FORM=DEFAULT /PRIORITY=100
    _DISK:[SMITH]MYFILE.TXT;1
```

Example 9–2 shows a queue status list, displayed by the command **SHOW QUEUE/DEVICE/ALL_ENTRIES**, that corresponds to Figure 9–1. The **/ALL_ENTRIES** qualifier is used because, by default, the **SHOW QUEUE** command displays only current jobs and pending jobs owned by the current process.

```
$ SHOW QUEUE/DEVICE/ALL_ENTRIES

Printer queue LPA0

Jobname         Username      Entry     Blocks    Status
-------         --------      -----     ------    ------
MYFILE.TXT      JONES           225         10    Printing at block 6

Printer queue LPB0

Jobname         Username      Entry     Blocks    Status
-------         --------      -----     ------    ------
USELESS.MEM     JONES           231        233    Printing at block 34

Printer queue LPC0

Jobname         Username      Entry     Blocks    Status
-------         --------      -----     ------    ------
SCHEDULE        SMITH           229        109    Printing at block 88
PAYROLL         JONES           228        144    Pending
SPREAD          JONES           245        156    Pending

Generic printer queue SYS$PRINT

Jobname         Username      Entry     Blocks    Status
-------         --------      -----     ------    ------
FILE.LOG        SMITH           250        198    Pending
TYPE.COM        JONES           249        206    Pending
CHECK           ANDERSON        240        220    Pending
```

Example 9–2 Queue Status Display Corresponding to Figure 9–1

Notes on Example 9–2

The following comments describe what is in each column of the display.

1. **Jobname** – Usually, this is the file name of the first file in the job. Only information relating to your jobs is displayed.
2. **Username** – Name of user who queued the job.
3. **Entry** – In this example, the entry numbers or job numbers correspond with the ones in Figure 9–1. Note that the list is kept in order by job size, not by job number.
4. **Blocks** – Size of the job in blocks (one block is 512 bytes).
5. **Status** – A job must have *active status* to compete in the queue for the printer. ***Holding*** means the job has *inactive status*in the queue, while ***pending*** means the job is actively competing in the queue. The string "Printing at block 108" means the job is the printer's *current job*.

Example 9–3 shows a queue status list, displayed by the command **SHOW QUEUE/DEVICES/FULL/ALL_ENTRIES**. Note that, because of the **/FULL** qualifier, the list contains more information than the status list shown in Example 9–2. Additional lines describing the file to be printed follow each line describing a holding print job.

```
$ SHOW QUEUE/DEVICES/FULL/ALL_ENTRIES

Terminal queue COMP, on WHYNOT::WHYNOT$TTA2:, mounted form DEFAULT
/BASE_PRIORITY=4 /DEFAULT=(FEED,FORM=DEFAULT) Lowercase
/OWNER=[GROUP1,SYSTEM] /PROTECTION=(S:E,O:D,G:R,W:W)

Printer queue LN01, on WHYNOT::WHYNOT$LPA0:, mounted form DEFAULT
/BASE_PRIORITY=4 /DEFAULT=(FEED,FORM=DEFAULT)
/LIBRARY=SYSDEVCTL_LN01 Lowercase /OWNER=[GROUP1,SYSTEM]
/PROTECTION=(S:E,O:D,G:R,W:W) /SEPARATE=(FLAG,RESET=(ANSI$RESET))

Server queue NM$QUE01, on WHYNOT::, mounted form DEFAULT
/BASE_PRIORITY=4 /DEFAULT=(FEED,FORM=DEFAULT)
/OWNER=[GROUP1,SYSTEM] /PROCESSOR=NM$DAEMON /PROTECTION=(S:E,O:D,G:R,W:R)
/RETAIN=ERROR

Generic printer queue NM$QUEUE
/GENERIC=(NM$QUE01,NM$QUE02) /OWNER=[GROUP1,SYSTEM]
/PROTECTION=(S:E,O:D,G:R,W:R) /RETAIN=ERROR

Jobname         Username     Entry  Blocks  Status
-------         --------     -----  ------  ------
NMAIL           SMITH       1630     146  Holding until 24-NOV-1988 11:26
Submitted 24-NOV-1988 11:16 /PRIORITY=100
_$1$DUA0:[SYSCOMMON.NMAIL]NMAIL$1988112217065820.WRK;1
```

Example 9–3 Full Format Queue Status Display

9.6 MODIFYING A PRINT JOB

You can change the characteristics of your print job if it is not currently printing. Table 9–3 shows how to use the **SET ENTRY** command to modify the characteristics of a print job.

Consult the documentation on the **SET ENTRY** command for a list of characteristics that you can change. This command can also move a job from one queue to another.

9.6.1 Deleting a Print Job

Table 9–3 also shows how to use the **DELETE/ENTRY** command to delete a job from a queue. You may need to do this if you accidentally print a file with non–ASCII characters, such as EXE or OBJ files. You can delete a job even if it is the current job.

Table 9–3 Modifying a Batch or Print Job

Operation	Comments and Examples
Changing the characteristics of a job	The entry–number (or job number) parameter specifies the number of the job you want to change. In this example, the number of copies for entry–number 100 is being changed to five. **$ SET ENTRY entry–number/qualifier** `$ SET ENTRY 100/COPIES=5`
Moving a job to another queue	In this example, the job MYFILE.TXT (entry number 90) is being moved from a printer using narrow paper to a printer using wide paper. **$ SET ENTRY entry–number/REQUEUE=queue–name** `$ SET ENTRY 90/REQUEUE=WIDE`
Deleting a job on a queue	Current jobs on a queue can be deleted as well as all other jobs. If a current job is deleted, its processing stops immediately. **$ DELETE/ENTRY=job–number** `$ DELETE/ENTRY=120`

9.7 SUBMITTING A BATCH JOB

When you issue the **SUBMIT** command, the VMS system assigns a number to your batch job. Your terminal displays this number, as Example 9–4 shows.

By default, your job enters the standard system batch queue, SYS$BATCH. Each job in this queue consists of a command procedure the system will execute in a batch process. (The **Writing Command Procedures** module covers command procedures.)

```
$ SUBMIT ACTION.COM
Job ACTION (queue SYS$BATCH, entry 136) pending
$
```

Example 9–4 Issuing the SUBMIT Command

9.7.1 How a Batch Job Executes

Although a batch process runs independently of your interactive process, it is like your interactive process in many respects. For example, the VMS system uses the information about you in the system user authorization file to create your batch process. The batch process has the same UIC, privileges, and quotas that you have when you log in. Also, the batch process executes your own LOGIN.COM file before executing the DCL commands in the submitted command procedure.

As it executes the DCL commands in your batch job, the VMS system writes output to a file called the *batch log*. This file is created in your login directory. It usually has the same file name as the batch command file, and a file type of LOG. The log file is printed and then deleted automatically on completion of your batch job.

The values of four process logical names the system defines for you are different for batch processes than they are for interactive processes. Table 9–4 shows the standard definitions. When batch jobs contain nested command procedures, the VMS system redefines these logical names for each command level.

Table 9–4 Logical Name Definitions for Interactive and Batch Processes

Logical Name	Definition When Interactive Process Begins to Execute	Definition When Batch Process Begins to Execute
SYS$INPUT	Interactive terminal	Batch command file
SYS$OUTPUT	Interactive terminal	Batch log file
SYS$COMMAND	Interactive terminal	Batch command file
SYS$ERROR	Interactive terminal	Batch log file

Table 9–5 shows how to use qualifiers to the **SUBMIT** command to tell the system how to handle your batch log file. In particular, you can specify:

- The name of the batch log file
- Whether or not to print the batch log file if your job completes successfully
- Whether or not to delete the batch log file after it is printed

Table 9–5 Controlling the Batch Log File

Operation	Comments and Examples
Naming the batch log file	By default, the log file has the same file name as the command file, and has the file type LOG. In this example, the log file is named OUTPUT.LOG instead of the default MYFILE.LOG. **$ SUBMIT/LOG_FILE=file–specification** `$ SUBMIT/LOG_FILE=OUTPUT MYFILE`
Printing the batch log file	By default, the log file is queued to SYS$PRINT when the batch job completes execution. In this example, the log file is queued to printer LPB0. **$ SUBMIT/PRINTER=queue–name file–name** `$ SUBMIT/PRINTER=LPB0 MYFILE`
Saving the batch log file	By default, the log file is deleted after being printed. In this example, the log file is retained in the login directory. The qualifier /NOPRINTER implies the /KEEP qualifier. **$ SUBMIT/KEEP file–specification** `$ SUBMIT/KEEP MYFILE`

9.7.2 Writing a Batch Command Procedure

You can run a command procedure in either of two ways: *interactively*, or as a *batch job* in a batch procedure. Certain differences between these two cases are discussed below.

- A batch process executes the system manager's login command procedure and your own LOGIN.COM file before executing the DCL commands in your batch command file.

  You can use the **F$MODE()** lexical function in your LOGIN.COM file to determine whether the process is interactive or not. If the process is not interactive, bypass commands that assume there is an interactive terminal, such as **INQUIRE** and **SET TERMINAL**.

  Also, because good programming practice requires you to spell out DCL commands in command procedures, you can bypass symbol definitions that define command abbreviations you use interactively.

- An error or severe error halts the execution of a batch job. You can use the **ON** command in your batch command file to modify the handling of errors.

- When the VMS system creates a batch process, its default directory is the one specified in the user authorization file. When you refer to files in your batch command files, make sure you know what your default directory is.

- By default, verification is on in batch processes. You can use the **SET VERIFY** command and the **F$VERIFY** lexical function in your batch command files to change verification.

9.7.3 Using a Particular Batch Queue

Table 9–6 shows how to use the **SUBMIT** command to queue batch jobs, and Table 9–7 shows how to use the **SHOW QUEUE** command to display status information on these jobs. Example 9–5 shows the full format of this status information.

When you execute a command procedure interactively, you can include up to eight parameters on the DCL command line to define the symbols P1 through P8. Note that you must use the **/PARAMETERS** qualifier of the **SUBMIT** command to pass parameters to a command procedure you submit as a batch job.

Table 9–6 Submitting Batch Jobs

Operation	Comments and Examples
Submitting a job with no parameters	The SUBMIT command uses a default file type of COM. The file submitted is ACTION.COM in this example. **$ SUBMIT file-specification** `$ SUBMIT ACTION`
Submitting a job to a specified queue	By default, the system batch queue SYS$BATCH is used. **$ SUBMIT/QUEUE=queue-name file-specification** `$ SUBMIT/QUEUE=SLOWBATCH ACTION`
Submitting a job after a specified time	The file MYFILE.TXT will be held until the specified time (19:00), after which it will be processed. **$ SUBMIT/AFTER=time file-specification** `$ SUBMIT/AFTER=19:00 MYFILE.TXT`
Submitting a job that requires parameters	Up to eight parameters can be specified using symbols (P1–P8). The symbols are local to the specified command procedures. **$ SUBMIT/PARAMETERS=(P1,[,...]) file-specification** `$ SUBMIT/PARAMETERS=(3,SUM) MATH`

Example 9–5 shows a run of the command procedure COUNT1.COM.

```
$!  COUNT1.COM
$!
$ SHOW TIME
$ SHOW LOGICAL/PROCESS/JOB
$ EXIT

$ SUBMIT COUNT1.COM
Job COUNT1 (queue SYS$BATCH, entry 366) started on SYS$BATCH
$
```

Output from the system's LOGIN procedure:

```
$! COUNT1.COM
$!
$ SHOW TIME
  13-JAN-1988  09:31:22
$ SHOW LOGICAL/PROCESS/JOB

(LNM$PROCESS_TABLE)

  "EVE$INIT" = "SYS$LOGIN:EVE.INIT"
  "SYS$COMMAND" = "_WHYNOT$RTA1:"
  "SYS$DISK" = "WHYNOT$DJA0:"
  "SYS$ERROR" = "_WHYNOT$RTA1:"
  "SYS$INPUT" [super] = "_WHYNOT$DJA0:"
  "SYS$INPUT" [exec] = "_WHYNOT$RTA1:"
  "SYS$OUTPUT" [super] = "_WHYNOT$RTA1:"
  "SYS$OUTPUT" [exec] = "_WHYNOT$RTA1:"
  "TT" = "RTA1:"

(LNM$JOB_803E1730)

  "SYS$LOGIN" = "WHYNOT$DJA0:[SMITH]"
  "SYS$LOGIN_DEVICE" = "WHYNOT$DJA0:"
  "SYS$REM_ID" = "SMITH"
  "SYS$REM_NODE" = "WHYSO::"
  "SYS$SCRATCH" = "WHYNOT$DJA0:[SMITH]"
```

Example 9–5 Sample Batch Run of COUNT1.COM

Table 9–7 Displaying Batch Queue Status

Operation	Comments and Examples
Displaying a list of batch jobs	By default, the only jobs displayed other than your own are those currently executing. To display all jobs, add the qualifier /ALL_ENTRIES to the SHOW QUEUE command. For more job information, add the qualifier /FULL to either the SHOW QUEUE or SHOW ENTRY command. `$ SHOW QUEUE/BATCH` `$ SHOW QUEUE/BATCH/ALL_ENTRIES` `$ SHOW ENTRY/BATCH` `$ SHOW ENTRY/BATCH/FULL`
Displaying a list of batch jobs on a particular queue	In any SHOW QUEUE command, you can specify a queue name instead of /BATCH. You can also use the qualifiers /FULL and /ALL_ENTRIES. **$ SHOW QUEUE queue–name** `$ SHOW QUEUE SYS$BATCH` **$ SHOW QUEUE/qualifier que–name** `$ SHOW QUEUE/ALL_ENTRIES SYS$BATCH`

Example 9–6 shows a queue status list, displayed by the command **SHOW QUEUE/BATCH/FULL/ALL_ENTRIES**.

```
$ SHOW QUEUE/BATCH/FULL/ALL_ENTRIES

Batch queue WHYNOT_SYSTEM, on WHYNOT::
    /BASE_PRIORITY=3 /JOB_LIMIT=4 /OWNER=[GROUP1,SYSTEM]
    /PROTECTION=(S:W,O:W,G,W)

Jobname         Username     Entry  Status
-------         --------     -----  ------
MYFILE          SMITH         1388  Holding until  3-DEC-1988 18:00

DRAFT           SMITH         1425  Holding until  4-DEC-1988 01:00

TEST            JONES         1352  Holding until  7-DEC-1988 00:00

Batch queue WHYNOT_BATCH, on WHYNOT::
    /BASE_PRIORITY=2 /JOB_LIMIT=3 /OWNER=[GROUP1,SYSTEM]
    /PROTECTION=(S:E,O:D,G:R,W:W)
```

Example 9–6 Full Format Queue Status Display

9.8 HANDLING BATCH AND PRINT JOBS

The VMS system handles some aspects of batch job and print job processing in exactly the same way. In these cases, the **PRINT** and **SUBMIT** commands use the same qualifiers.

Table 9–8 shows how to specify certain characteristics common to both batch and print jobs, including:

- The name used to identify the job
- The node on which the job is processed
- Whether the system displays the job number when the job is queued
- Whether the system notifies you when the job completes
- Whether the input file is deleted after the job completes

Table 9–8 Specifying the Characteristics of Batch and Print Jobs

Characteristic	Comments and Examples
Name of job	By default, the VMS system uses the file name of the first file in the job. The job name appears on the flag and in the queue status list as the default file name of the batch log. **$ SUBMIT/NAME=job–name** `$ SUBMIT/NAME=MYFILE.TXT` **$ PRINT/NAME=job–name** `$ PRINT/NAME=MYFILE.TXT`
Notification that job has been queued	By default, a message is displayed notifying the user of the job entry number and the name of the queue in which the job was entered. **$ PRINT/IDENTIFY job–name** `$ PRINT/IDENTIFY MYFILE.TXT`
Notification that job processing has completed	Notifies the user when the job has been completed or aborted. By default, there is no notification. **$ SUBMIT/NOTIFY file–name** `$ SUBMIT/NOTIFY MYFILE`

9.9 BATCH AND PRINT QUEUES ETIQUETTE

The following suggestions are given to insure that system batch and print queues flow efficiently and smoothly, with no "time lags" or "backups."

- Check the size of your print jobs before submitting them.
- If feasible, submit large print or batch jobs after hours.
- Set up a file size limit (in blocks) over which a job should be submitted after hours.
- If submitting a large job, verify that the paper supply is sufficient to handle that job, or have an operator check on the paper supply.
- Do not print files that are not compatible for the particular device.
- If possible, wait until the queue(s) are empty or handling a minimum of jobs before submitting your job.
- Pick up your completed job promptly. Do not allow your finished jobs to sit in the printer area endlessly.

9.10 SUMMARY

Printing a File

- The **PRINT** command uses a default file type of LIS.
- Job numbers indicate the order in the queue.
- The print queue, named SYS$PRINT, handles print requests by default.
- The first available printer prints the job.

Submitting a Batch Job

- The **SUBMIT** command uses a default file type COM unless another file type is specified.
- Each job in the queue consists of a command procedure.
- Job numbers indicate the order in the queue.
- SYS$BATCH is the default system batch queue.
- The VMS system creates a batch process to execute the command procedure.

Writing a Batch Command Procedure

- There are two ways to run a command procedure.
  - — Interactive
  - — Batch
- By default, severe errors terminate batch job execution.
- The batch process's default directory is the one specified as SYS$LOGIN.

Deleting a Batch or Print Job

- Can delete a batch or print job while it is executing or while it is pending in the queue.
- Use the **DELETE/ENTRY** command.

9.11 LABORATORY EXERCISE I

NOTE
Several of the laboratory exercises in this module ask you to create command procedure files.

Complete the following exercises at an interactive terminal.

1. Choose a text file and print it, using the generic print queue SYS$PRINT.
2. Use a single PRINT command to print two copies of the same file.
3. Display a list of all queues on your system and all jobs in the queues.
4. Select an execution queue from the queue display. (An execution queue will have the same name as its associated device, without the colon.) Print the same file, queuing it directly to the execution queue.
5. Choose two text files. Print these two files so that you get two copies of the first file and three copies of the second file.
6. Send a text file to the printer queue, requesting that the file not be printed until an hour from now.
7. Display the queue status of the job waiting to be printed. Delete this job from the queue.

9.12 LABORATORY EXERCISE II

1. Display, at your terminal, all of the batch queues on the system.
2. Submit a command procedure to batch that displays the time, displays all processes on the system, and shows all logical names on the system. Save the log file. You will need to examine it shortly.
3. Submit the above command procedure to batch so that the log file will not be printed.
4. Submit the above command procedure to batch so that the log file will not be created.
5. Examine the log file created in Step 2. Answer the following questions:
   a. Find the entry for your batch job from the SHOW SYSTEM command. What was its process ID?
   b. Did your LOGIN.COM file execute? Did the system–wide login procedure execute?
   c. How much CPU time did your batch job use to execute?
   d. How much elapsed time did your batch job use to execute?

9.13 LABORATORY EXERCISE I—SOLUTIONS

1. Choose a text file and print it, using the generic print queue SYS$PRINT.

```
$ PRINT FILENAME
```

   (FILENAME is the name of your file in all solutions.)

2. Use a single PRINT command to print two copies of the same file.

```
$ PRINT/COPIES=2 FILENAME
```

3. Display a list of all queues on your system and all jobs in the queues.

```
$ SHOW QUEUE/ALL_ENTRIES
```

4. Select an execution queue from the queue display. (An execution queue will have the same name as its associated device, without the colon.)

```
$ PRINT/QUEUE=LPA0 FILENAME
```

   (LPA0 may or may not be the name of your execution queue, depending upon how the system is set up.)

5. Choose two text files. Print these two files so that you get two copies of the first file and three copies of the second file.

```
$ PRINT FIRSTFILENAME/COPIES=2,SECONDFILENAME/COPIES=3
```

6. Send a text file to the printer queue, requesting that the file not be printed until an hour from now.

```
$ PRINT/AFTER=TIME FILENAME
```

7. Display the queue status of the job waiting to be printed. Delete this job from the queue.

```
$ SHOW QUEUE SYS$PRINT
$ DELETE/ENTRY=n     (where "n" is the entry number)
```

9.14 LABORATORY EXERCISE II—SOLUTIONS

1. Display, at your terminal, all of the batch queues on the system.

```
$ SHOW QUEUE/BATCH
```

2. Submit a command procedure to batch that displays the time, displays all processes on the system, and shows all logical names on the system. Save the log file. You will need to examine it shortly.

```
$! NAME OF .COM FILE
$!
$ SHOW TIME
$ SHOW SYSTEM
$ SHOW LOGICAL
$ EXIT
```

3. Submit the above command procedure to batch so that the log file will not be printed.

```
$ SUBMIT/NOPRINTER FILENAME.COM
```

4. Submit the above command procedure to batch so that the log file will not be created.

```
$ SUBMIT/NOLOG FILENAME.CO
```

5. Examine the log file created in Step 2. Answer the following questions:

   a. Find the entry for your batch job from the SHOW SYSTEM command. What was its process ID?

      Your batch name entry should have a name similar to *BATCHXXX (XXX would be the ID number of your job)*. Also in the right margin of the SHOW SYSTEM display, you should see the letter B.

   b. Did your LOGIN.COM file execute? Did the system–wide login procedure execute?

      The entries marked with a *B* are *batch jobs*. Both your *LOGIN.COM file* and the *system–wide login procedure* should have executed, assuming they exist. You may see some of your LOGIN.COM file commands in the log file.

   c. How much CPU time did your batch job use to execute?

   d. How much elapsed time did your batch job use to execute?

      Both the *CPU time* and *elapsed time* are in the accounting information in the last lines of the log file.

DEVELOPING PROGRAMS

10.1 INTRODUCTION

This module presents a general discussion of the steps in developing a program on a VAX system as well as an introduction to a sample program.

It does not provide details regarding any of the programming languages, such as FORTRAN or PASCAL.

A number of tools that significantly decrease the time spent developing VMS programs include:

- Interactive Text Editor (EDT)
- Compilers
- VAX MACRO Assembler
- VMS Linker
- VMS Librarian
- VMS Symbolic Debugger
- System supplied routines

The *editors, assembler, compilers,* and *linker* are utilities that prepare source programs for execution. The *VMS Symbolic Debugger* detects logic errors in executable image files.

The librarian enables you to store frequently used segments of code, such as procedures or functions, in specially indexed files called *libraries.* You can reference procedures or functions stored in a library with a program. The linker combines the code from the library with your source code to produce an *executable image file.*

For the MACRO language, you store *macros* (definitions) in a different type of library. The assembler accesses libraries containing these macros to add them to a program.

System libraries contain a large number of predefined routines that user programs (such as routines that manipulate strings or generate random numbers) can call.

Refer to the *Introduction to VMS System Routines* manual for more information regarding system–supplied routines.

10.2 OBJECTIVES

To use most of the programming languages, you should be familiar with the following program development steps:

- Creating a text file containing the source statements of the program
- Compiling or assembling the text file to create a file containing object code
- Linking the object file or files to produce a file containing executable code
- Running the executable image produced from the linker
- Debugging the program to correct errors

10.3 RESOURCES

For more detailed explanations of developing programs, refer to the following documents:

- *VMS DCL Dictionary*
- *Guide to VMS Programming Resources*

10.4 PROGRAM DEVELOPMENT ON A VMS SYSTEM

To develop a program written in a VMS language, you must complete the following steps:

- Create a text file that contains the source statements of your program.
- Compile or assemble the text file to produce a file containing object code.
- Link the object file or files to produce an executable image file.
- Run the executable code produced by the linker.
- Debug the program to correct errors.

Figure 10–1 illustrates the orderly flow of these five program development steps.

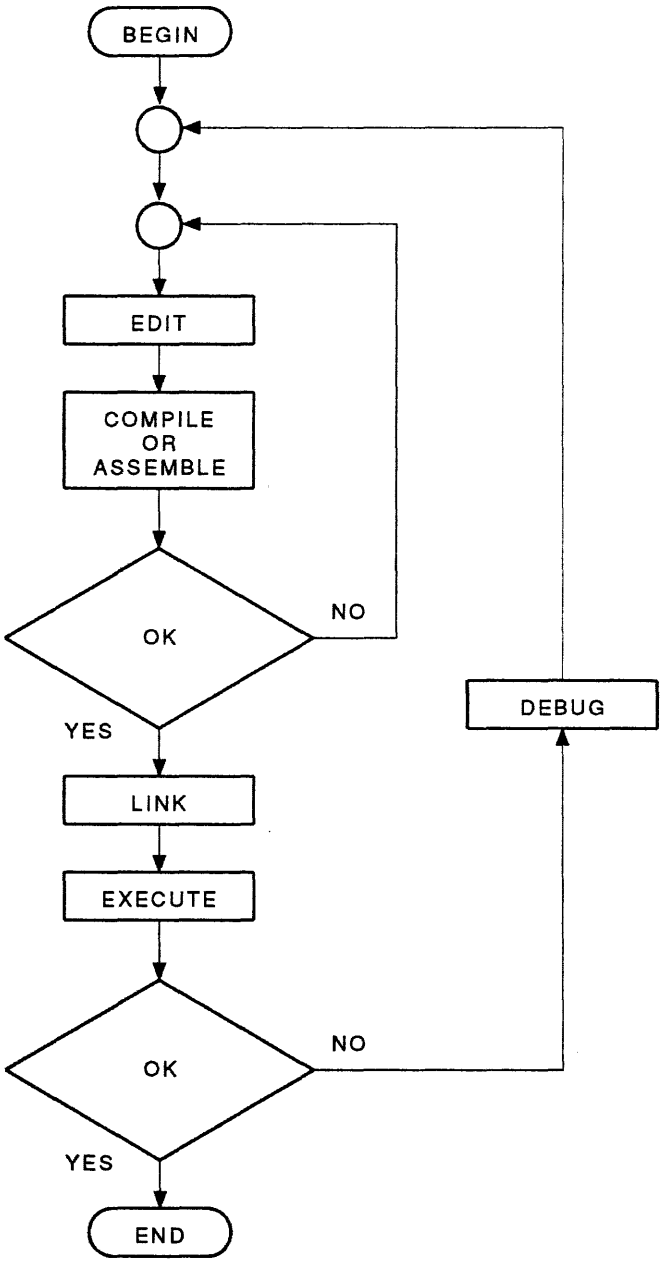


Figure 10–1 A Flow Diagram of the Five Major Programming Steps

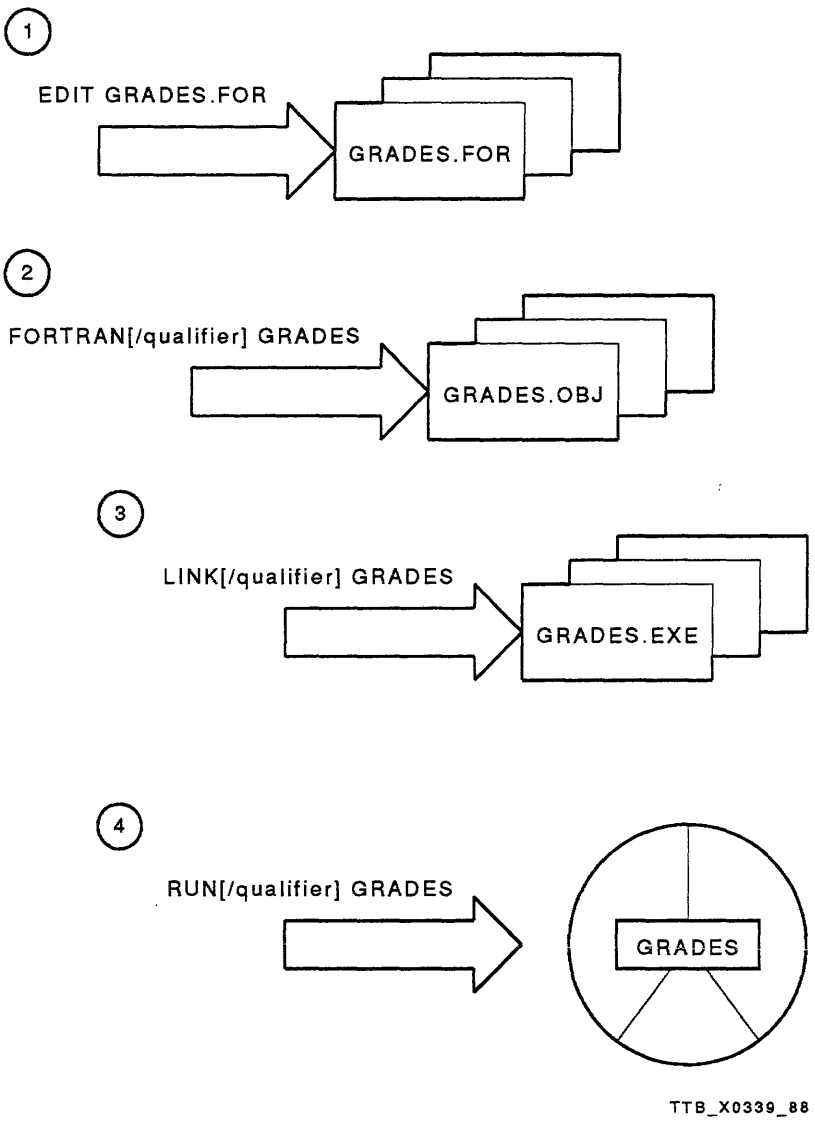


Figure 10–2 The Four Program Development Commands

Each of the five program development steps is discussed in detail below. As you read each step, refer to Figures 10–1 and 10–2, which is keyed to the steps being discussed.

1. Create a text file that contains the source statements of your program.

   Name the source file using the file type that relates to the source code programming language. Below are the default file types for the languages covered in this module.

Language	File Type
BASIC	BAS
C	C
COBOL	COB
FORTRAN	FOR
MACRO	MAR
PASCAL	PAS
PL/I	PLI

2. Compile or assemble the text file you created with an editor to produce a file containing object code.

   The compiler or assembler translates the source statements of each input file into object code, producing one or more object files of type OBJ.

   To compile or assemble the code, you must use the DCL command related to the language of the source code in the text file. The following are examples of compile and assemble commands.

Language	Compiler/Assembler Command
BASIC	$ BASIC file–specification
C	$ CC file–specification
COBOL	$ COBOL file–specification
FORTRAN	$ FORTRAN file–specification
MACRO	$ MACRO file–specification
PASCAL	$ PASCAL file–specification
PL/I	$ PLI file–specification

You can list more than one input file as a parameter. The way you list these parameters determines how many object files the compiler or assembler creates. To specify your request, use either the comma (,) or the plus sign (+) as a parameter separator. The results of your choice follow:

- If you separate input file specifications by plus signs, the compiler creates one object file containing the code from all input files.
- If you separate them by commas, it creates a separate object file for each input file.

If the compiler finds syntax errors in the source code, the system displays an appropriate message at your terminal. You can translate the message by referencing the appropriate language documentation.

Use an editor to correct the source code, and submit the new version of the text file to the compiler or assembler for translation.

The DCL Help facility gives you information about qualifiers when you enter the following command:

HELP language_name

3. Link the object file or files to produce an executable image.

   The linker searches personal and system libraries for external procedures and functions that it cannot find in the specified input files.

   To link the object file(s), invoke the VMS Linker with the DCL command **LINK**. You can specify the names of the files to be linked, such as object code files or modules from libraries, after the command. Separate names with commas. The linker assumes that the file type of input files is OBJ.

   The linker's file output contains executable code assigned the file type of EXE.

   The *VMS Linker Utility Reference Manual* describes linker errors and recommended solutions.

4. Invoke the image activator to run the executable code produced by the linker.

   To execute a program, enter the DCL command **RUN** followed by the name of a single executable image file. The **RUN** command assumes that the file type field of the input file specification is EXE.

   You should not attempt to execute a program without correcting compiler and linker errors first.

   If you have corrected all obvious errors, errors output at run time can indicate logical errors. A logical error occurs because the statements in the program do not do the intended job. A logical error could produce error messages, or simply the wrong result. Check your results carefully. If the program receives input, you should execute it several times with various types of input to be sure it does the required job in all given situations.

   To correct the program, you must debug it to find out where the error occurs. When you find the error, you must modify the source program and submit it to the compiler or assembler and linker again. Then you can execute the new executable file to see if the error was corrected.

5. Debug the program to correct errors.

   To find the cause of a logical error, you must examine the program carefully, looking at the source code one line at a time. Keep lists of variables and their contents on paper, as well as comments on loops and output to peripherals. Often, in larger programs, you can isolate the problem to a particular area of the program, saving the time of looking at every line.

   If you can isolate the problem, or if the program is not very large, it is not difficult to examine a program using paper; you can easily find errors. As you write larger programs involving more I/O, more variables, and more loops, debugging becomes more complicated, and the contribution the debugger makes increases.

10.5 THE VMS SYMBOLIC DEBUGGER UTILITY

The VMS Symbolic Debugger simplifies your debugging job. Debug commands implement many of the same debugging techniques used on paper.

The VMS Symbolic Debugger allows you to observe and manipulate your program interactively as it executes. By issuing debugger commands at the terminal, you can:

- Start, stop, and resume the execution of the program
- Trace the execution path of the program
- Monitor selected locations, variables, or events
- Examine and modify the contents of variables, or force events to occur
- Use breakpoints, tracepoints, and watchpoints in variables
- Test the effect of modifications without having to edit the source code, recompile, and, in some cases, relink

There are three ways to invoke the debugger:

1. Include the debugger in the executable image.

   The debugger is included in the executable image if you enter the **/DEBUG** qualifier with the **LINK** command. When the system subsequently executes your program, it automatically invokes the debugger, and displays the debug prompt (DBG>).

   Unless you also include the **/DEBUG** qualifier in the compiler or assembler command (**/ENABLE=DEBUG** with the **MACRO** command), the system will not include local symbol tables in the object file. The symbol tables contain the names and addresses of variables used in your program. For example, if you use the variable named **ICOUNT** in your FORTRAN program, by including the **/DEBUG** qualifier in the compiler command, you instruct the compiler to store this variable's name and address in the local symbol table it produces. Later, you can use the debugger to examine the contents of the variable **ICOUNT** and other variables stored in the local symbol table.

   Not all debug commands rely on the existence of local symbol tables. For example, the debug commands **GO**, **STEP**, and **SET TRACE** work without this information. But if you intend to examine the contents of variables, be sure to include the **/DEBUG** qualifier in your compiler or assembler command.

2. Halt the program and invoke the debugger with the DCL command **DEBUG**.

   You can halt a program by entering CTRL/Y or CTRL/C, then invoke the debugger by entering the DCL command **DEBUG**.

   Use this method to halt a "hung" program, one that will not run to completion. The debugger can determine where the program is hung.

   This method also works for a program that is already executing in the debugger if you want to display the debug prompt to input further debugging commands.

3. Run the program with the debugger.

   To run a program with the debugger, enter the **/DEBUG** qualifier with the **RUN** command. (Again, if you do not include the **/DEBUG** qualifier in the compiler or assembler command, the debugger will not be able to reference address locations by symbol names.)

The VMS Symbolic Debugger utility has an extensive Help facility. To use this facility, invoke the symbolic debugger and enter the debug command **HELP**.

10.6 A SAMPLE PROGRAM – GRADES

The FORTRAN program **GRADES** creates a file containing the names of students and the average of their grades for a particular course. The program obtains the names and grades from you, computes the average of the grades, and outputs the results to the terminal and to a designated file, ENGLISH.DAT. Example 10–1 shows the source file for GRADES.FOR and Example 10–2 shows the execution of GRADES.

```
PROGRAM GRADES
CHARACTER STUDENT_NAME*30, DONE*4
REAL AVERAGE

OPEN (UNIT=1, FILE='English', STATUS='New')

10      TYPE 20
20      FORMAT (/' Student name? ',$)
ACCEPT 30, STUDENT_NAME
30      FORMAT (1A30)

CALL COMPUTE (AVERAGE)

TYPE 40,STUDENT_NAME,AVERAGE
WRITE (1,40) STUDENT_NAME,AVERAGE
40      FORMAT (/' Student: ',A30,'Average: ',F10.1)

TYPE 50
50      FORMAT (/' Are you done ? (Yes/No) ',$)
ACCEPT 60, DONE
60      FORMAT (1A4)
IF (DONE.NE.'Y' .AND. DONE.NE.'y') GOTO 10

CLOSE (UNIT=1)
END

SUBROUTINE COMPUTE (AVERAGE)

INTEGER ICOUNT
REAL TOTAL, GRADE
ICOUNT = 0
TOTAL = 0

10      TYPE 20
20      FORMAT (' Input grade (or 0 to end input): ',$)
ACCEPT 30, GRADE
30      FORMAT (F10.0)

IF (GRADE.NE.0) THEN
ICOUNT = ICOUNT + 1
TOTAL = TOTAL + GRADE
GO TO 10
ENDIF

40       IF (ICOUNT.NE.0) AVERAGE = TOTAL/ICOUNT

RETURN
END
```

Example 10–1 GRADES.FOR Source File

10.7 EXECUTION OF GRADES

The following example depicts a sample run of the **GRADES** program, using FORTRAN.

```
$ FORTRAN GRADES
$ LINK GRADES
$ RUN GRADES
Student name?  JOHN SMITH
Input grade (or 0 to end input): 45
Input grade (or 0 to end input): 80
Input grade (or 0 to end input): 99
Input grade (or 0 to end input): 0

Student: JOHN SMITH                    Average:     74.7

Are you done ? (Yes/No) N

Student name?  MARY HAGERTY
Input grade (or 0 to end input): 82
Input grade (or 0 to end input): 69
Input grade (or 0 to end input): 94
Input grade (or 0 to end input): 0

Student: MARY HAGERTY                  Average:     81.7

Are you done ? (Yes/No) N

Student name?  HOSIAH HOWER
Input grade (or 0 to end input): 90
Input grade (or 0 to end input): 78
Input grade (or 0 to end input): 81
Input grade (or 0 to end input): 0

Student: HOSIAH HOWER                  Average:     83.0

Are you done ? (Yes/No) Y
$
$
$ TYPE ENGLISH.DAT

Student:  JOHN SMITH                   Average:     74.7

Student:  MARY HAGERTY                 Average:     81.7

Student:  HOSIAH HOWER                 Average:     83.0
$
```

Example 10–2 Sample Run of GRADES

10.8 SUMMARY

Program Development on a VMS System

A user must complete the following steps to develop a program:

- Create a text file that contains the source statements of your program.
- Compile or assemble the text file to produce a file containing object code.
- Link the object file or files to produce an executable image file.
- Run the executable code produced by the linker.
- Debug the program to correct errors.

For more detailed explanations of developing programs, refer to the following documents:

- *Guide to VMS Programming Resources*
- *VMS DCL Dictionary*

There are no Exercises for this module.