
Educational Services

digitalTM

**VMS Utilities and Commands I
Student Workbook**

EY-9764E-SG-0001

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Copyright ©1988 by Digital Equipment Corporation

All Rights Reserved.
Printed in U.S.A.

The following are trademarks of Digital Equipment Corporation:

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

digital™

First Edition, May 1988

This document was prepared using VAX DOCUMENT, Version 1.0

TABLE OF CONTENTS

About This Course	xvii
MODULE 1 HARDWARE AND SOFTWARE OVERVIEW	1-1
INTRODUCTION	1-3
OBJECTIVES	1-3
COMPONENTS OF THE HARDWARE ENVIRONMENT	1-5
The Central Processing Unit (CPU)	1-5
The Console Subsystem	1-6
Main Memory	1-6
Input/Output Subsystem	1-6
THE VMS OPERATING SYSTEM	1-7
DIGITAL Command Language (DCL)	1-8
Utilities	1-9
Optional Layered Products	1-10
THE WORKING ENVIRONMENT	1-11
The Process	1-11
Components of a Process	1-12
Process Types	1-13
The System User Authorization File (SYSUAF)	1-14
SUMMARY	1-15
APPENDIX A—PERIPHERAL DEVICES	1-17
Terminals	1-17
Printers and Printer/Plotters	1-18
Disk Drives	1-20
Tape Drives	1-22
APPENDIX B—SYSTEM CONFIGURATIONS	1-24
Single Processor Configurations	1-24
Multiple-Processor Configurations	1-27
Tightly-Coupled Configurations	1-28
VAXcluster Systems	1-29
NETWORKS	1-32

MODULE 2 GETTING STARTED	2-1
INTRODUCTION	2-3
OBJECTIVES	2-4
RESOURCES	2-4
LOGGING IN TO A VMS SYSTEM	2-5
USER NAME AND PASSWORD	2-6
BEGINNING AND ENDING A TERMINAL SESSION	2-7
DCL COMMAND FORMAT	2-9
COMMAND LINE CONSTRUCTION	2-9
DCL FEATURES	2-13
EDITING A DCL COMMAND LINE	2-14
THE RECALL COMMAND	2-17
GETTING HELP	2-20
The Documentation Set	2-20
The Online Help Facility	2-22
CHANGING YOUR PASSWORD	2-24
SYSTEM MESSAGES	2-25
DISPLAYING CHARACTERISTICS OF TERMINAL, PROCESS, AND SYSTEM	2-28
Terminal Characteristics	2-28
THE SHOW TERMINAL COMMAND	2-29
THE SET TERMINAL COMMAND	2-29
SUMMARY	2-31
MODULE 3 CREATING AND EDITING TEXT FILES	3-1
INTRODUCTION	3-3
OBJECTIVES	3-4
RESOURCES	3-4
CHOOSING AN EDITOR	3-5
EDT Editing Utility	3-6
The Extensible VAX Editor (EVE)	3-7
USING THE EDT EDITOR	3-8
Invoking the EDT Editor	3-8
EDT Screen Layout	3-8
Using EDT Help	3-9
The EDT Keypad	3-11
Commonly Used Features	3-11

Ending an EDT Editing Session	3-15
EDT File Recovery	3-16
USING EVE	3-17
Invoking the EVE Interface	3-17
EVE Screen Layout	3-17
The EVE Interface	3-18
MOVING THE EVE CURSOR	3-20
INSERTING TEXT IN EVE	3-22
ERASING TEXT	3-22
DEFINING AN EDT-LIKE KEYPAD	3-23
CANCELING AN EDT-LIKE KEYPAD	3-23
Using EVE Help	3-28
Ending an EVE Session	3-29
EVE File Recovery	3-29
SUMMARY	3-30
APPENDIX A—EDT LINE-MODE EDITING	3-31
Inserting Text	3-31
Substituting Text	3-32
Moving Text from One Location to Another	3-33
Deleting Text	3-34
Using Buffers in EDT	3-35
How to Create Buffers	3-35
Copying Text from One Buffer to Another	3-36
Copying Text from a File into a Buffer	3-36
Copying Text from a Buffer into a File	3-36
Deleting Buffers	3-36
APPENDIX B—EVE	3-37
Inserting Text	3-37
Locating Text	3-38
Marking Locations in Text	3-38
Replacing Text	3-39
Restoring Text	3-40
RESTORE CHARACTER	3-40
RESTORE LINE	3-40
RESTORE WORD	3-40
Using Buffers in EVE	3-41
Using Multiple Buffers	3-42

Using Multiple Windows	3-43
DELETE WINDOW	3-44
ENLARGE WINDOW	3-44
NEXT WINDOW	3-44
PREVIOUS WINDOW	3-44
SHRINK WINDOW	3-45
SPLIT WINDOW	3-45
DEFINING KEYS	3-46
Saving Key Definitions	3-47
Using Key Definitions	3-47
Checking Spelling Errors	3-47
MODULE 4 COMMUNICATING WITH OTHER USERS	4-1
INTRODUCTION	4-3
OBJECTIVES	4-3
RESOURCES	4-3
INVOKING AND OBTAINING HELP FROM THE MAIL AND PHONE UTILITIES	4-5
THE MAIL UTILITY	4-7
Organization of Mail Messages	4-7
Using the MAIL Utility	4-8
Reading a Message	4-9
Sending a Message	4-11
Displaying a List of Messages	4-13
Deleting a Message	4-14
Exiting from the Mail Utility	4-17
THE PHONE UTILITY	4-18
The Phone Help Facility	4-19
COMMUNICATING WITH OPERATORS	4-21
The REQUEST Command	4-21
SUMMARY	4-22
MODULE 5 MANAGING FILES	5-1
INTRODUCTION	5-3
OBJECTIVES	5-3
RESOURCES	5-3
FILE SPECIFICATIONS	5-5
DEFAULTS FOR FILE SPECIFICATIONS	5-6
DEVICE SPECIFICATIONS	5-7
DIRECTORY STRUCTURE	5-9
DIRECTORY NAMES IN THE HIERARCHY	5-10

FILE MANIPULATION COMMANDS	5-11
FINDING FILES AND DETERMINING THEIR CHARACTERISTICS	5-13
Using Wildcards in File Specifications	5-15
ORGANIZING YOUR DIRECTORY STRUCTURE	5-17
CREATING A SUBDIRECTORY	5-18
USING THE SET DEFAULT COMMAND	5-19
USING THE SHOW DEFAULT COMMAND	5-19
MOVING WITHIN A DIRECTORY HIERARCHY	5-20
PROTECTING DISK AND TAPES	5-23
PROTECTING FILES IN YOUR DIRECTORY HIERARCHY	5-24
Three Levels of Disk File Protection	5-24
UIC-Based Protection	5-25
DETERMINING AND ALTERING FILE PROTECTION	5-28
DELETING A SUBDIRECTORY	5-30
Access Control Lists	5-32
Commands to Obtain ACL Information	5-33
Creating an Access Control List	5-33
Deleting an Access Control List	5-35
SUMMARY	5-36
APPENDIX A—DEVICE INFORMATION	5-37
Specifying Devices	5-37
APPENDIX B—NETWORKING INFORMATION	5-43
Managing Files on Another VMS System in Your Network	5-43
Methods of File Management in a Network	5-43
Using DCL File-Manipulation Commands in a Non-VAXcluster Network Environment	5-44
Two Node Specification Formats	5-44
Using DCL File-Manipulation Commands in a VAXcluster Environment	5-47
Two Cluster Device Specification Formats	5-47

MODULE 6 CUSTOMIZING THE USER ENVIRONMENT

INTRODUCTION	6-3
OBJECTIVES	6-3
RESOURCES	6-3
LOGICAL NAMES	6-5
Logical Name Tables	6-6
Private	6-6
Shared	6-6
Common User Operations Dealing with Logical Names	6-8

Adding Logical Names	6-9
USING LOGICAL NAMES	6-10
Logical Name Translation for Logical Names that Have Single Equivalence Strings	6-10
Sample Recursive Translation	6-11
Displaying the Contents of Logical Name Tables	6-12
Determining the Equivalence of a Logical Name	6-14
Deleting Logical Names	6-15
System-Defined Logical Names	6-17
Specifying Logical Name Access Modes	6-19
Duration of a Process-Private Logical Name Assignment	6-19
USING DCL SYMBOLS	6-20
DEFINING KEYS	6-25
SUMMARY	6-27
MODULE 7 WRITING COMMAND PROCEDURES	7-1
INTRODUCTION	7-3
OBJECTIVES	7-4
RESOURCES	7-4
DEVELOPING A COMMAND PROCEDURE	7-5
Components and Conventions	7-7
LOGIN COMMAND PROCEDURE	7-10
TERMINAL INPUT/OUTPUT	7-12
Performing Terminal Input and Output	7-13
Symbol Substitution	7-18
PASSING PARAMETERS TO COMMAND PROCEDURES	7-21
Parameters	7-21
Local Symbols P1 - P8	7-21
Passing Parameter Values to a Command Procedure	7-21
CONTROLLING PROGRAM FLOW	7-23
The IF Command	7-23
THE IF-THEN-ELSE COMMAND	7-24
Restrictions to IF-THEN-ELSE Command	7-24
The GOTO Command	7-24
LEXICAL FUNCTIONS	7-28
Format and Syntax	7-28
SUMMARY	7-32

MODULE 8 USING DISK AND TAPE VOLUMES	8-1
INTRODUCTION	8-3
OBJECTIVES	8-3
RESOURCES	8-3
CREATING PRIVATE VOLUMES: THE COMMAND SEQUENCE	8-6
MOUNTING A VOLUME WITH AN UNKNOWN LABEL	8-7
THE BACKUP UTILITY	8-8
SAVE-SET SPECIFICATIONS	8-8
SUMMARY	8-12
MODULE 9 SUBMITTING BATCH AND PRINT JOBS	9-1
INTRODUCTION	9-3
OBJECTIVES	9-4
RESOURCES	9-4
PRINTING A FILE	9-5
PRINT Command in DCL	9-5
Types of Print Queues	9-6
Qualifiers for the PRINT Command	9-8
OBTAINING STATUS OF QUEUES	9-10
Queue Status List	9-13
Modifying a Print Job Already in the Queue	9-14
Deleting a Print Job	9-15
SUBMITTING A BATCH JOB	9-16
DCL SUBMIT Command	9-16
How a Batch Job Executes	9-17
Writing a Batch Command Procedure	9-18
Qualifiers for the SUBMIT Command	9-19
OBTAINING STATUS OF BATCH QUEUES	9-22
Modifying a Batch Job Already in the Queue	9-24
DELETING A BATCH JOB	9-25
HANDLING BATCH AND PRINT JOBS	9-26
Characteristics Common to Both Batch and Print Jobs	9-26
BATCH AND PRINT QUEUES ETIQUETTE	9-26
SUMMARY	9-27

MODULE 10 DEVELOPING PROGRAMS	10-1
INTRODUCTION	10-3
OBJECTIVES	10-4
RESOURCES	10-4
PROGRAM DEVELOPMENT ON A VMS SYSTEM	10-5
THE VMS SYMBOLIC DEBUGGER UTILITY	10-10
A SAMPLE PROGRAM - GRADES	10-11
Execution of GRADES	10-12
SUMMARY	10-13
MODULE 11 EXERCISES	11-1
HARDWARE AND SOFTWARE OVERVIEW	11-3
WRITTEN EXERCISE I	11-3
WRITTEN EXERCISE II	11-5
GETTING STARTED	11-8
LABORATORY EXERCISE I	11-8
LABORATORY EXERCISE II	11-9
WRITTEN EXERCISE I	11-10
LABORATORY EXERCISE III	11-12
LABORATORY EXERCISE IV	11-13
CREATING AND EDITING TEXT FILES	11-14
INTRODUCTION TO THE LABORATORY EXERCISES	11-14
LABORATORY EXERCISE I - THE EDT EDITOR	11-15
LABORATORY EXERCISE II - THE EVE EDITOR	11-17
LABORATORY EXERCISE III - THE EVE EDITOR	11-18
COMMUNICATING WITH OTHER USERS	11-19
LABORATORY EXERCISE I	11-19
LABORATORY EXERCISE II	11-20
LABORATORY EXERCISE III	11-21
MANAGING FILES	11-22
WRITTEN EXERCISE I	11-22
WRITTEN EXERCISE II	11-23
LABORATORY EXERCISE I	11-24

LABORATORY EXERCISE II	11-25
WRITTEN EXERCISE III	11-26
LABORATORY EXERCISE III	11-27
WRITTEN EXERCISE IV	11-28
CUSTOMIZING THE USER ENVIRONMENT	11-29
WRITTEN EXERCISE I	11-29
LABORATORY EXERCISE I	11-31
LABORATORY EXERCISE II	11-32
WRITTEN EXERCISE II	11-33
LABORATORY EXERCISE III	11-34
LABORATORY EXERCISE IV	11-35
WRITING COMMAND PROCEDURES	11-36
WRITTEN EXERCISE I	11-36
INTRODUCTION TO LABORATORY EXERCISES	11-38
LABORATORY EXERCISE I	11-39
LABORATORY EXERCISE II	11-40
OPTIONAL LABORATORY EXERCISE	11-41
USING DISK AND TAPE VOLUMES	11-42
WRITTEN EXERCISE I	11-42
WRITTEN EXERCISE II	11-43
WRITTEN EXERCISE III	11-44
LABORATORY EXERCISE I	11-45
SUBMITTING BATCH AND PRINT JOBS	11-46
LABORATORY EXERCISE I	11-46
LABORATORY EXERCISE II	11-47
HARDWARE AND SOFTWARE OVERVIEW—SOLUTIONS	11-48
WRITTEN EXERCISE I	11-48
WRITTEN EXERCISE II	11-50
GETTING STARTED—SOLUTIONS	11-52
LABORATORY EXERCISE I	11-52
LABORATORY EXERCISE II	11-54
WRITTEN EXERCISE I	11-55
LABORATORY EXERCISE III	11-56

LABORATORY EXERCISE IV	11-57
CREATING AND EDITING TEXT FILES—SOLUTIONS	11-59
LABORATORY EXERCISE I - THE EDT EDITOR	11-59
LABORATORY EXERCISE II - THE EVE EDITOR	11-62
LABORATORY EXERCISE III	11-64
COMMUNICATING WITH OTHER USERS—SOLUTIONS	11-65
LABORATORY EXERCISE I	11-65
LABORATORY EXERCISE II	11-67
LABORATORY EXERCISE III	11-68
MANAGING FILES—SOLUTIONS	11-69
WRITTEN EXERCISE I	11-69
WRITTEN EXERCISE II	11-70
LABORATORY EXERCISE I	11-71
LABORATORY EXERCISE II	11-72
WRITTEN EXERCISE III	11-73
LABORATORY EXERCISE III	11-74
WRITTEN EXERCISE IV	11-75
CUSTOMIZING THE USER ENVIRONMENT—SOLUTIONS	11-77
WRITTEN EXERCISE I	11-77
LABORATORY EXERCISE I	11-79
LABORATORY EXERCISE II	11-81
WRITTEN EXERCISE II	11-82
LABORATORY EXERCISE III	11-83
LABORATORY EXERCISE IV	11-84
WRITING COMMAND PROCEDURES—SOLUTIONS	11-85
WRITTEN EXERCISE I	11-85
LABORATORY EXERCISE I	11-87
LABORATORY EXERCISE II	11-88
OPTIONAL LABORATORY EXERCISE	11-89
USING DISK AND TAPE VOLUMES—SOLUTIONS	11-91
WRITTEN EXERCISE I	11-91
WRITTEN EXERCISE II	11-92
WRITTEN EXERCISE III	11-93

LABORATORY EXERCISE I	11-94
SUBMITTING BATCH AND PRINT JOBS—SOLUTIONS	11-95
LABORATORY EXERCISE I	11-95
LABORATORY EXERCISE II	11-96
MODULE 12 TEST	12-1
TEST	12-3
ANSWERS	12-9

EXAMPLES

2-1	How to Log In and Log Out	2-8
2-2	Changing Your Password	2-24
3-1	Using the Help Facility On Line	5-10
3-2	Recovering a File After a System Interruption	3-16
4-1	Getting Help for MAIL Utility Commands	4-6
4-2	Reading a Mail Message	4-9
4-3	Sending a Mail Message	4-11
4-4	Listing Messages and Reading Old Messages	4-13
4-5	Using the REQUEST/REPLY Command	4-21
5-1	Sample Directory File	5-16
5-2	Using VMS Commands to Create and Maintain a Directory Hierarchy	5-22
5-3	Changing Your Default Protection Code	5-29
5-4	Deleting a Subdirectory from a Directory Hierarchy	5-30
5-5	Removing Subdirectories from a Directory Hierarchy	5-31
5-6	Modifying an Access Control List	5-34
6-1	Using Logical Names to Abbreviate Device and File Specifications	6-9
6-2	Displaying the Contents of the Process, Job, Group, and System Logical Name Tables	6-13
6-3	Determining the Value of a Logical Name	6-14
6-4	Assigning, Changing, and Deleting Logical Name Assignments	6-16
6-5	Using ASSIGN Command to Alter the Default Output Device of Your Process	6-19
6-6	Defining, Displaying, Using, and Deleting DCL Symbols	6-23
7-1	A Sample Command Procedure	7-8
7-2	A Sample LOGIN.COM File	7-11
7-3	A Sample of Output from a Command Procedure	7-14
7-4	Using Terminal Input and Output	7-17
7-5	Using Symbol Substitution	7-20
7-6	Passing Parameters to Command Procedures	7-22
7-7	Controlling Program Flow	7-26
7-8	Using Lexical Functions with the INFO.COM Command Procedure	7-30
7-9	Using Lexical Functions with the PRINT.COM Command Procedure	7-31
8-1	Mounting a Disk with an Unknown Label	8-7
8-2	Creating Save Sets on a Tape	8-9
8-3	Transferring Files to a Tape	8-9
8-4	Restoring Files from a Tape to a Directory	8-11
9-1	Issuing the PRINT Command	9-5
9-2	Queue Status Display Corresponding to Figure 9-1	9-12

9-3	Full Format Queue Status Display	9-13
9-4	Issuing the SUBMIT Command	9-16
9-5	Sample Batch Run of COUNT1.COM	9-21
9-6	Full Format Queue Status Display	9-22
10-1	GRADES.FOR Source File	10-11
10-2	Execution of GRADES	10-12
11-1	Process Parameters of a Sample Interactive Process	11-6

FIGURES

1	Course Map	xxvi
1-1	VAX Hardware Subsystems	1-5
1-2	Components of a Process	1-12
1-3	Sample Hardcopy and Video Terminals	1-17
1-4	Sample Printers and Printer/Plotter	1-19
1-5	Examples of Disks	1-21
1-6	Examples of Disk Drives	1-21
1-7	Examples of Tape Media	1-23
1-8	Sample Tape Drives	1-23
1-9	MicroVAX II	1-25
1-10	VAX 8600	1-26
1-11	A Tightly-Coupled System Configuration	1-28
1-12	VAXcluster System Structure	1-30
1-13	A DECnet Network	1-33
2-1	Enter a Valid User Name and Password	2-5
2-2	The Elements of a Command Line	2-9
2-3	The Elements of a System Message	2-25
3-1	EDT Screen Layout - Line Mode and Keypad Mode	3-8
3-2	EDT Keypad Definitions	3-12
3-3	EVE Screen Layout	3-17
3-4	EVE Keypad Definitions (VT100-Series Terminals)	3-18
3-5	EVE Keypad Definitions (VT200-Series Terminals)	3-19
3-6	EDT-Like Key Definitions for VT200-Series Terminals	3-24
3-7	EDT-Like Key Definitions for VT100-Series Terminals	3-26
4-1	The Relationship Between a Mail Message, Folder, and File	4-15
4-2	Using the Phone Utility	4-19
5-1	Naming Directories	5-10
5-2	File Specification in the Directory Hierarchy	5-21
5-3	File Access to Disk and Tape Volumes	5-23
5-4	Interaction of Access Categories	5-26
5-5	Elements of a Protection Code: Determines Which Users Have Access to a File	5-26
5-6	Device Specifications Used to Identify the Desired Device for a Given Operation	5-37
6-1	The Relationship Between Your Terminal, the Operating System, and the Logical Name Tables Associated with Your Process	6-7
6-2	The Relationship Between Your Terminal, the Operating System, and Your Global Symbol Table	6-21
7-1	Command Procedure Development Process	7-6
8-1	Volume Manipulation Commands	8-5
9-1	Execution and Generic Print Queues	9-7
10-1	A Flow Diagram of the Five Major Programming Steps	10-6

10-2	The Four Program Development Commands	10-7
------	---	------

TABLES

1	Course Conventions	xxvii
2-1	Elements of DCL Commands	2-10
2-2	The Three Types of DCL Qualifiers	2-12
2-3	Features of DCL	2-13
2-4	Moving the Cursor	2-14
2-5	Changing Data on the Command Line	2-15
2-6	Recalling a Previously Issued Command Line	2-16
2-7	Recalling a Previous Command Line with the RECALL Command	2-17
2-8	Controlling the Display of Information at Your Terminal	2-18
2-9	Terminating an Operation	2-19
2-10	Manuals for Locating Information About Your System	2-21
2-11	Using the DCL HELP Facility	2-23
2-12	Elements of the System Message	2-26
2-13	Severity Levels in System Error Messages	2-27
2-14	Commands for Displaying the Characteristics of Your Terminal, Process, and System	2-30
3-1	Moving the EDT Cursor	3-13
3-2	Changing the EDT Cursor Direction	3-13
3-3	Deleting Text in EDT	3-14
3-4	Restoring Text in EDT	3-14
3-5	Moving the Cursor Using Keys	3-20
3-6	Using Commands to Move the Cursor	3-21
3-7	Keys for Deleting Text	3-22
3-8	Responding to REPLACE Prompts	3-39
3-9	Creating and Manipulating Buffers	3-41
3-10	Creating and Manipulating Windows	3-43
4-1	MAIL Commands Used to Read a Mail Message	4-10
4-2	MAIL Commands Used to Send Messages	4-12
4-3	MAIL Commands Used to Maintain Messages	4-16
4-4	Phone Commands Commonly Used to Create or Reject a Terminal Link	4-20
5-1	Syntax of a Local Disk File Specification	5-5
5-2	File Specification Defaults	5-6
5-3	Naming a Device	5-8
5-4	Directory Names	5-10
5-5	File Manipulation Commands	5-11
5-6	Manipulating Files in a Directory	5-12
5-7	Using the DIRECTORY Command to Determine the Characteristics of Files	5-14
5-8	Wildcards Used to Specify File Names, Types, and Versions	5-15
5-9	Using Wildcards to Specify Files	5-16
5-10	Characters Used to Specify Directories	5-20
5-11	Summary of Effects of Access Rights to Files	5-27
5-12	Determining a User's Category by Comparing User's UIC to File Owner's UIC	5-27
5-13	Commands Used to Determine and Alter File Protection	5-28
5-14	Examples of Using Other Devices	5-38
5-15	Moving a Hierarchical File Structure from one Disk Device to Another	5-39
5-16	Codes for Some Supported Devices on a VMS System	5-40
5-17	Summary of Device Terminology	5-41

5-18	Generic Specification with the SHOW DEVICE Command	5-42
5-19	Examples of Specifying Files on Remote Nodes	5-45
5-20	DECnet-VAX DCL File-Manipulation Command Summary	5-46
5-21	Commands Used to Determine the Nodes and Devices in Your Systems Environment	5-48
6-1	Displaying the Contents of Logical Name Tables	6-12
6-2	Commands to Delete Logical Names	6-15
6-3	Process Logical Names Defined by the System	6-17
6-4	Job Logical Names Defined by the System	6-18
6-5	System Logical Names Defined by the System	6-18
6-6	Commands for Displaying and Deleting DCL Symbols	6-22
6-7	Comparison of Logical Names and DCL Symbols	6-24
7-1	System Logical Names Used with Terminal I/O	7-12
7-2	Displaying Information on the Terminal	7-13
7-3	Getting Information from the User	7-15
7-4	Redirecting Input and Output	7-16
7-5	Symbol Substitution Techniques	7-19
7-6	Relational Operators Used in Expressions	7-25
7-7	Frequently Used Lexical Functions	7-29
8-1	Commands for Creating and Accessing Private Disk and Tape Volumes	8-6
9-1	Printing Jobs with Different Characteristics	9-9
9-2	Modifying Print Jobs in a Queue	9-14
9-3	Logical Name Definitions for Interactive and Batch Processes	9-17
9-4	Submitting Batch Jobs	9-20
9-5	Displaying Batch Queue Status	9-23
9-6	Modifying a Batch Job	9-24
10-1	Languages and Associated File Types	10-8

About This Course

INTRODUCTION

The *VMS Utilities and Commands* course is a lecture/lab course designed to show you how to perform typical nonprivileged operations on a VMS system by entering commands at a terminal.

The course has been organized into a series of units, or *modules*, each designed to cover a well-organized topic, or group of topics. Each module contains its own learning objectives.

Suggested Laboratory Exercises are provided to allow you to reinforce, through practice, your knowledge of the topics covered.

COURSE DESCRIPTION

The *VMS Utilities and Commands* course describes the working environment of a VMS system and introduces frequently used operations that you can perform by entering commands at an interactive terminal.

Among the major topics covered by the course are:

- Creating, editing, and maintaining text files
- * Submitting batch and print jobs
- * Writing and using command procedures
- * Using logical names and symbols to tailor your working environment
- Using private disk and tape volumes to back up your own files
- Communicating with other users on a system and network
- Using online and printed VMS documentation to obtain information

PREREQUISITES

There are no prerequisites for this course. However, you can derive the greatest benefit from this course if you have:

- A basic knowledge of a computer system.
- The ability to work on a system using an interactive terminal.

COURSE GOALS

To effectively use the nonprivileged facilities of the VMS system, you should be able to perform the following operations:

- Use online and printed documentation to obtain information about VMS features
- Understand the basic hardware and software components of a VMS system
- * Enter syntactically correct DIGITAL Command Language (DCL) commands to obtain information from the system
- Create files using a text editor (EDT or EVE)
- Communicate with other users and system operators
- Organize files into subdirectories and maintain them
- * Use logical names, symbols, and key definitions to modify your working environment
- * Create and use command procedures to automate repetitive tasks
- * Use the printer to produce hard copies of files and use the batch processing facility to execute command procedures
- Use private disk and tape volumes to back up and store personal files
- Follow the program development steps to produce executable programs on a VMS system (OPTIONAL)

COURSE NONGOALS

This course introduces the concepts and command sequences necessary to achieve the course goals. You will *not* learn to use the following:

- The MCR command language interpreter or other RSX utilities that reside on your system.
- The syntax of BASIC, COBOL, FORTRAN, or MACRO. The module dealing with programming gives a generic overview of the various programming steps. It is recommended that students enroll in a program-specific course to obtain the greatest benefit from a programming language.
- The use of commands or utilities that require privileges beyond the most basic privileges granted to users of your system.
- The use of commands that manipulate a multiprocess environment.
- VMS programmed system services, common run-time library routines, or other features that require direct interaction with the operating system.
- Use of programmer productivity tools, databases, word processors, or any other optional software.
- References to and materials associated with VAXcluster systems.
- Advanced command procedures, such as error handling, file I/O, dynamic arrays, CALL and GOSUB commands.
- Any functions, commands, or information related to "system management."
- File applications, such as, sorting records within a file and merging files.

COURSE RESOURCES

In addition to the VMS system itself, there are three major resources available to you for completing this course:

- This *Student Workbook*
- The *manuals* of the VMS documentation set
- Your *Instructor*

DOCUMENTATION

You should have access to the following manuals to complete this course:

- *Course Student Workbook*
- *Guide to VMS Files and Devices*
- *VMS DCL Dictionary*
- *Guide to Using VMS Command Procedures*
- *VMS Mail Utility*
- *VMS Phone Utility*
- *VAX EDT Reference Manual*
- *VAX Text Processing Utility Manual*
- *VMS DCL Concepts Manual*

One complete VMS documentation set should be available for reference.

COURSE ORGANIZATION

This is a lecture course that includes structured laboratory sessions. Lecture sessions consist of instructor presentations and class discussions. Laboratory sessions consist of instructor demonstrations or directed individual study. You should try to complete the laboratory exercises during the laboratory sessions.

The material in the *Student Workbook* is divided into units, or modules of study. Each module covers one or more of the skills typically required by a nonprivileged user of a VMS system.

A module contains the following instructional elements:

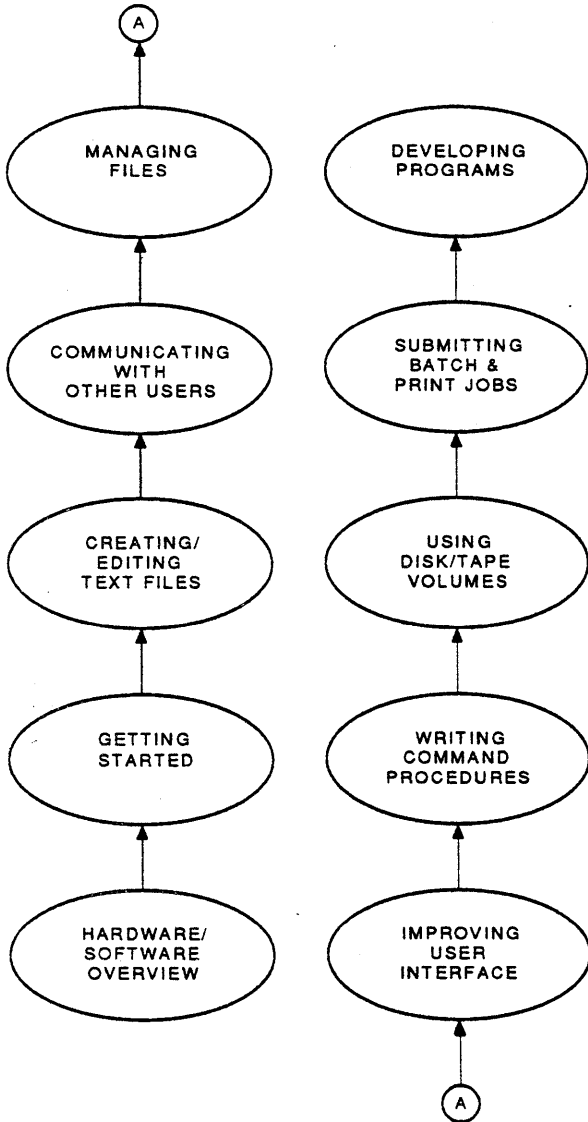
- An **introduction**, which describes the purpose of the modules, provides some motivation for mastering its objectives, and outlines its contents.
- One or more **objectives**, which describe the operations for which the module provides instruction. Objectives are designed to focus your study efforts on a selected number of skills.
- A list of **resources** you may need to complete the unit. Some of these resources are distributed with this course; others are not. Since a complete document set is distributed with each VMS operating system, you should consult your course instructor for access to materials that do not come with this course.
- The module **text**, which includes the following elements:
 - Descriptive text
 - Illustrations, which clarify the relationships among various elements of a VMS system, or summarize steps of a particular process
 - Tables, which summarize the operations covered by the modules, and list the commands needed to perform those operations
 - Examples, containing sample listings from actual terminal sessions

There is also a **Laboratory Exercises** module, which provides the practice needed to master the objectives of each module. Solutions to the exercises are also provided.

COURSE MAP DESCRIPTION

The course map shows how each module of the course is related to the other modules, and to the course as a whole. Prerequisite modules are those whose arrows in the map point into another module.

Figure 1: Course Map



TTB_X0340_88

COURSE CONVENTIONS

Table Table 1 describes the conventions used in the listings and tables of the *Student Workbook*.

Table 1: Course Conventions

Convention	Meaning
<i>new terms, prompts</i>	Terms that are introduced for the first time and system prompts are printed in italics.
< >	Angle brackets indicate that you press a key on the terminal keyboard. For example, <RET> means to press the RETURN key.
<CTRL/X>	Press and hold the key labeled CTRL while you press another key (X). Many control key sequences have special meanings.
SHOW QUEUE	Names of commands in text are shown in uppercase and bold.
\$ SHOW QUEUE/qualifier [queue-name]	Formats and command syntax are shown in bold. Words in uppercase are required, and words in lowercase represent elements that you must replace according to the description in the text.
<code>\$ SHOW QUEUE/ALL_ENTRIES SYS\$PRINT</code>	Actual examples of commands are shown in monospace type.
[]	Square brackets indicate that the enclosed item is optional. (Square brackets are not optional, however, in the syntax of some file specifications assignment statements.)
Type vs. Enter	When the word "type" is used in text, it means that you simply type a command. When the word "enter" is used, you must type the command and press the RETURN key.

MODULE 1

HARDWARE AND SOFTWARE OVERVIEW

INTRODUCTION

When you begin work on a VMS system, you enter an environment consisting of devices, programs, and data. The devices that compose the physical computer are called *hardware*. The programs that control the hardware and process the data are called the *software*. To perform job-related tasks on the system, you must use both the hardware and the software.

This module provides an introduction to VAX hardware, and an overview of the VMS software environment.

OBJECTIVES

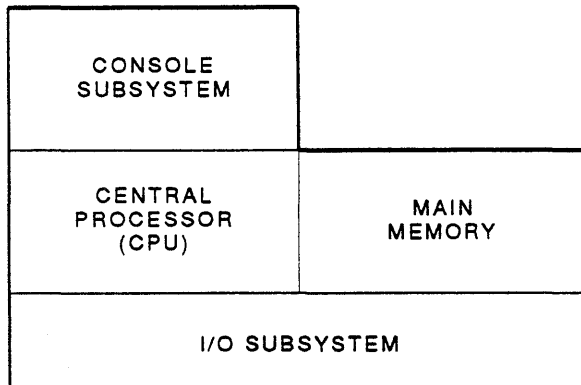
To work on a VMS system, you should be able to:

- Identify the functions of each component of the hardware environment.
- Identify and describe the functions of each component of the software environment.
- Identify elements that make up a process in the VMS environment.
- Recognize the peripheral devices supported by VAX systems.

COMPONENTS OF THE HARDWARE ENVIRONMENT

- VAX computer hardware is divided into four subsystems
- Each has a different function

Figure 1-1: VAX Hardware Subsystems



TTB_X0300_88

The Central Processing Unit (CPU)

- Executes instructions one at a time
- Some of the VAX family of processors include
 - MicroVAX II ↵
 - MicroVAX I
 - VAX-11/780 ↵
 - VAX 8200
 - VAX 8600
 - VAX 8700

The Console Subsystem →

HARD COPY TERMINALS
USED BY SYS. MANAGER

- Communicates directly with the CPU
- Is primarily used for
 - Starting up and shutting down the system
 - Installing software
 - Remote hardware diagnosis

Main Memory

- Main memory is used to store instructions and data temporarily

Input/Output Subsystem

- Provides input to and output from the system
- Consists of peripheral devices
- Common peripherals include
 - Terminals
 - Printers
 - Disk drives
 - Tape drives
- Refer to Appendix A for examples of peripheral devices

THE VMS OPERATING SYSTEM

- The VMS Operating System is a collection of programs that
 - Control the operations of the system
 - Manage the system's resources
- The operating system performs three major functions
 1. Provides the means for users to communicate with the hardware devices that make up the system
 2. Creates a working environment in which users can access the resources needed to perform tasks, without interfering with other users' activities on the system
 3. Schedules the use of the CPU, physical memory, and peripheral devices to provide equitable access for all users, while using these resources as efficiently as possible
- Typical activities of the operating system include
 - Loading programs and data into memory from storage
 - Scheduling the order of action by the CPU
 - Allocating resources, such as physical memory
 - Scheduling input and output (I/O) to other devices

DIGITAL Command Language (DCL)

- The means by which a user communicates with the system
- DCL uses common words for commands and qualifiers
- These common words make it easier to
 - Remember DCL commands
 - Enter DCL commands at a terminal
 - Recognize and correct syntax errors
- DCL commands can be used to
 - Perform file manipulation tasks
 - Display information about
 - The status of the system
 - Users on the system
 - Devices connected to your system
 - Resources available on the system
 - Execute user-written programs or system utilities
- DCL commands are translated by the Command Language Interpreter (CLI), which
 - Interprets the DCL command for correct syntax
 - Calls the VMS routines that perform the command

Utilities

Utilities are software tools that perform specified tasks. These utilities

- Are provided with the VMS software
- May have their own set of commands and command prompts
- Perform a wide variety of tasks, such as
 - Text editors
 - EDT Editor
 - Extensible VAX Editor (EVE)
 - Communication Utilities
 - MAIL
 - PHONE
 - Text Processors
 - * Digital Standard Runoff (DSR)
 - Debugging and Programming Tools
 - VMS Debugger

Optional Layered Products

- Layered products are available for VMS systems
- These optional layered products
 - Perform specific tasks that enhance the capabilities of the system
 - Are provided separately
- Types of layered products include
 - Language compilers
 - Communications software
 - Diagnostics software
 - Office automation products
 - Data management tools

THE WORKING ENVIRONMENT

The Process

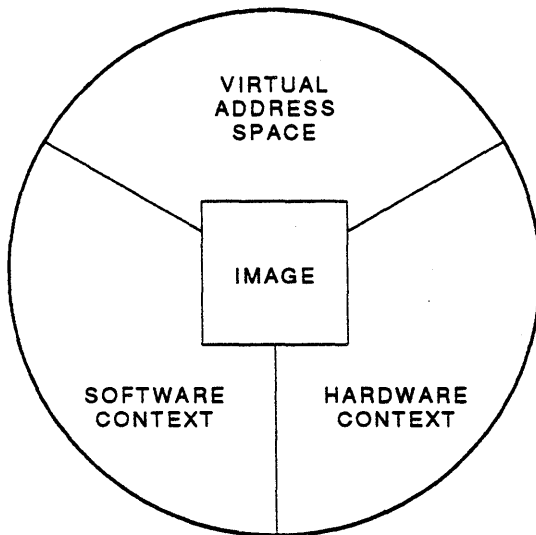
AS A USER, THE
COMPUTER SYSTEMS
YOU USE ARE A PROCESS.

- Your working environment is defined in terms of a process
- All work on a system is performed within a process
- The system uses processes to
 - Schedule the execution of programs by the CPU
 - Determine the availability of system resources
 - Allocate system resources

Components of a Process

- A process has four components
 - The *Hardware Context*—Values contained in the processor registers that describe what the process is doing *CURRENT STATUS OF HARDWARE*
 - The *Software Context*—Controls what the program is allowed to do *WHAT IS A PROGRAM DOING RIGHT NOW (FOR INTERRUPTS)*
 - *Virtual Address Space*—Process addresses are mapped to physical addresses *(ALLOW USER TO THINK HE HAS THE ENTIRE COMPUTER TO HIMSELF)*
 - The *Program (Image)*—Contains instructions to do the actual work *EXECUTING SOFTWARE*
- These components provide the environment used by the system to run an image

Figure 1-2: Components of a Process



TTB_X0258_88

Process Types

- Interactive Process
 - One of the most common process types
 - Created when a user logs in
 - Terminates when the user logs out of the system
- Subprocess
 - A process that uses some of the same resources as the parent process
 - Subprocesses allow users to have several programs executing at once
- Batch Process
 - Created by the system so that it can execute a special file called a command procedure
 - Frees up your terminal for other work

* The System User Authorization File (SYSUAF)

- SYSUAF contains the information used to create a process.
- It is used to determine all authorized users of the system.
- Information about each user is placed in this file by the system manager. Information includes:

SYSUAF.DAT

(CONTAINS PASSWORDS - ENCRYPTED!)
(NOT EVEN SYS. MAN. CAN SEE, NO ON)

- User Identification Code (UIC). The UIC is used to determine the owner of files and to determine file access.
- Default directory. This determines the disk and directory used by the user at login time.
- Privileges. Privileges determine whether the user can perform a given task.
- Priority. The process priority determines how a particular process will compete with other processes to get work done on the system.

SYSTEM MANAGER - SETS # OF TIMES

YOU CAN FAIL LOGIN

DUE TO PASSWORD

BEFORE YOU ARE

LOCKED OUT.

- CAN ALLOW USERS MORE THAN
PASSWORD.

SUMMARY

- There are four main functional subsystems of VAX computers:
 - The CPU – executes instructions.
 - The console subsystem – communicates with the CPU to monitor and control the system.
 - Main memory – stores data and instructions.
 - The I/O subsystem – consists of devices that provide input to and produce output from the system. These devices are referred to as peripherals. Peripherals include terminals, printers, disk drives, and tape drives.

The software environment is made up of several components:

- The VMS Operating System
 - Controls software on the system
 - Provides the means of communication with other hardware devices on the system
 - Schedules the allocation of resources and the execution of programs
- The user interface with the VMS system is the DIGITAL Command Language (DCL)
 - The means by which a user communicates with the system
 - Uses common English-like words
 - Interpreted by the Command Language Interface (CLI)
- Utilities are software tools that perform specific tasks
 - Provided with the system software
 - Include tools such as editors, text formatters, and communication utilities
- Optional Layered Products
 - Perform tasks beyond those of the system software
 - Must be purchased and installed separately

The working environment is defined in terms of a process:

- The system creates and controls processes
- Information used to create processes is stored in the System User Authorization File (SYSUAF)

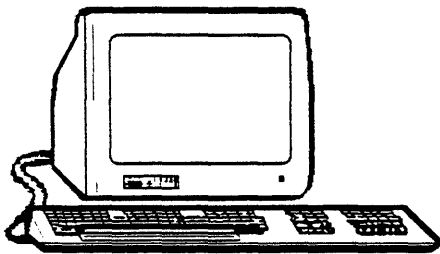
- VAX system configurations can be classified as single processors and multiple-processor configurations.
- A single-processor configuration is any single VAX processor and its peripheral devices.
- There are three types of multiple-processor configurations:
 - Tightly-coupled multiprocessors
 - Networks
 - VAXcluster systems—configured midway between multiprocessors and networks
- Multiple-processor configurations consist of:
 - Processors
 - Peripheral devices
 - Communication devices
 - Transmission media
 - Terminal servers (optional)
- A local area network spans a limited geographical area.
- A wide area network spans a larger area.
- The important difference between a network and a VAXcluster system is that the sharing of information between nodes is much faster and easier in a VAXcluster system.

APPENDIX A—PERIPHERAL DEVICES

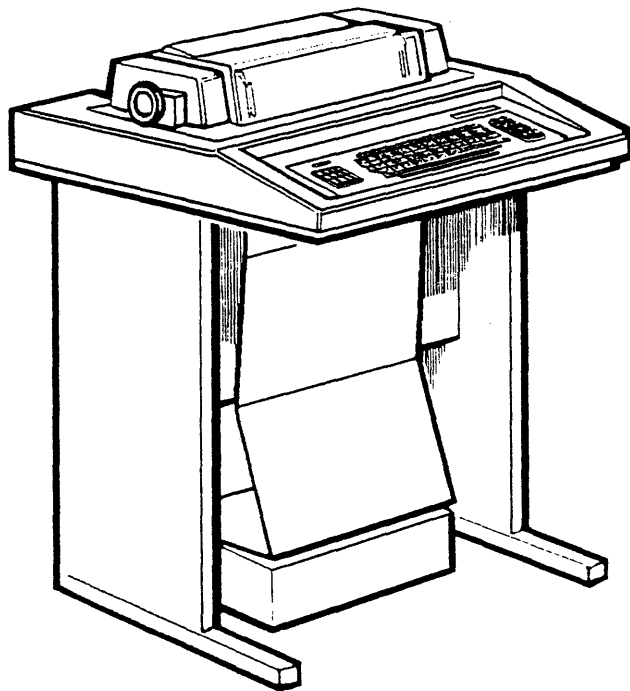
Terminals

- Used to communicate with the computer
- Two types of terminals
 - Hardcopy
 - Video

Figure 1-3: Sample Hardcopy and Video Terminals



A VIDEO
TERMINAL



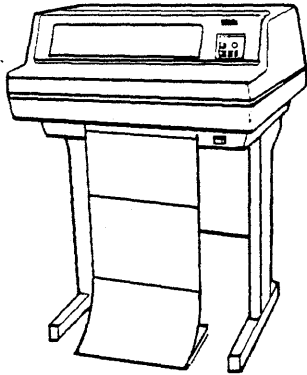
A HARDCOPY
TERMINAL

TTB_X0302_88_S

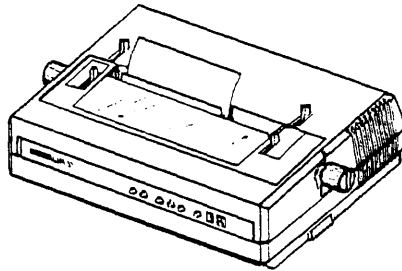
Printers and Printer/Plotters

- Printers provide output from the system
- Various sizes and types include
 - Line printers (high speed)
 - Letter quality printers (high-quality print)
 - Laser printers (high-quality print and graphics)
- Printer/plotters are used for graphic output

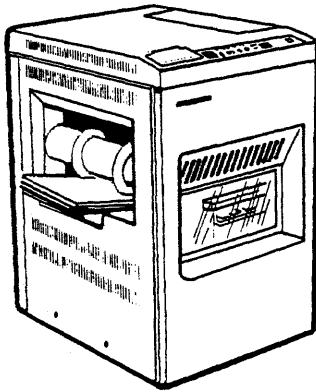
Figure 1-4: Sample Printers and Printer/Plotter



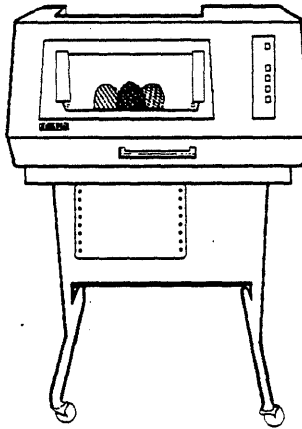
LINE PRINTER



LETTER-QUALITY PRINTER



LASER PRINTER



(OFTEN WITH PAPER ROLL)

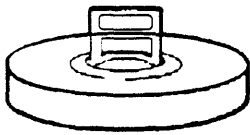
PRINTER/PLOTTER

TTB_X0303_88_S

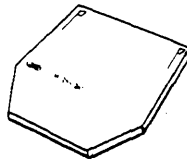
Disk Drives

- Record and read data on magnetic disks
- Are sometimes called mass storage devices
- Disks used in the drives
 - Are called storage media
 - Usually store frequently used data FASTER THAN TAPE
 - Are either removable or fixed
 - Various types of removable disks include
 - Cartridges
 - Disk packs
 - Diskettes
 - CDROM - EXPENSIVE

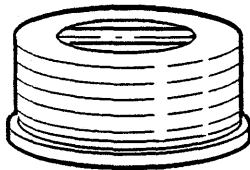
Figure 1-5: Examples of Disks



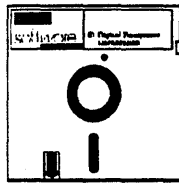
DISK CARTRIDGE
(TOP LOADING)



DISK CARTRIDGE
(FRONT LOADING)



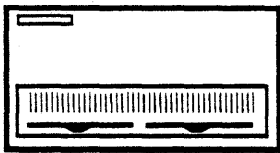
DISK PACK
(TOP LOADING)



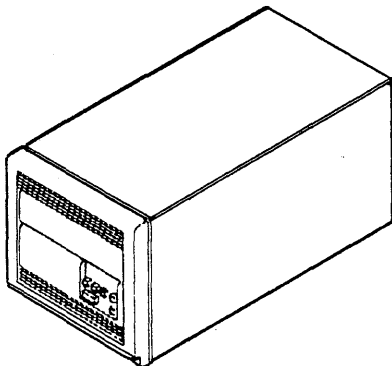
DISKETTE

TTB_X0304_88_S

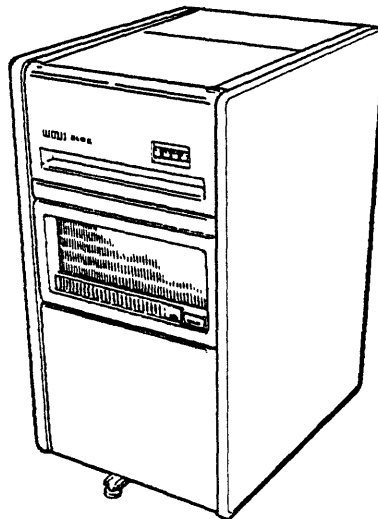
Figure 1-6: Examples of Disk Drives



DUAL DISKETTE DRIVE



DISK CARTRIDGE DRIVE



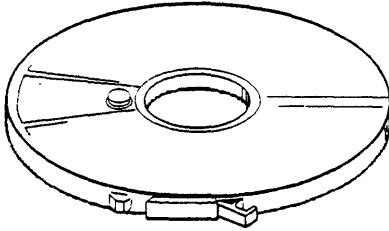
DISK PACK DRIVE

TTB_X0305_88_S

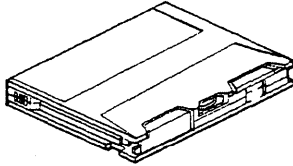
Tape Drives

- Record and read data on magnetic tapes
- Tapes usually store
 - Backup copies of data
 - Infrequently used data
- Two kinds of tapes
 - Reel Tapes
 - Tape cartridges

Figure 1-7: Examples of Tape Media



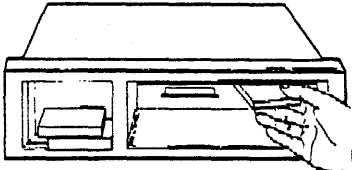
REEL TAPE



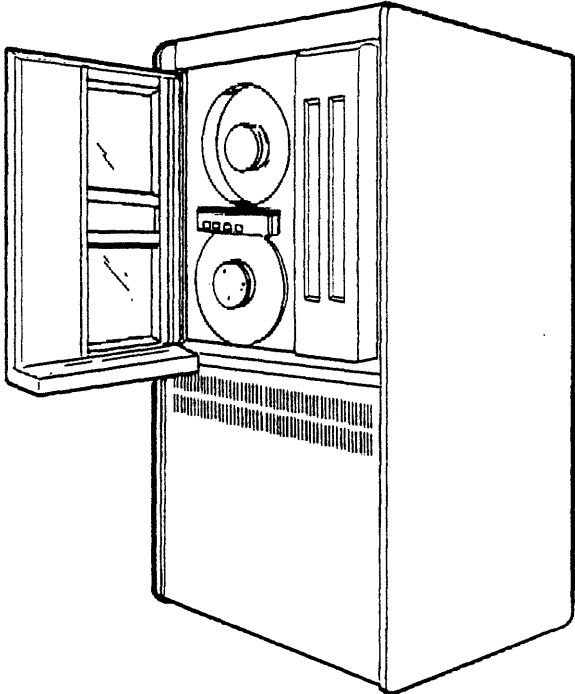
TAPE CARTRIDGE

TTB_X0306_88_S

Figure 1-8: Sample Tape Drives



CARTRIDGE TAPE DRIVE



REEL-TO-REEL TAPE DRIVE

TTB_X0307_88_S

APPENDIX B—SYSTEM CONFIGURATIONS

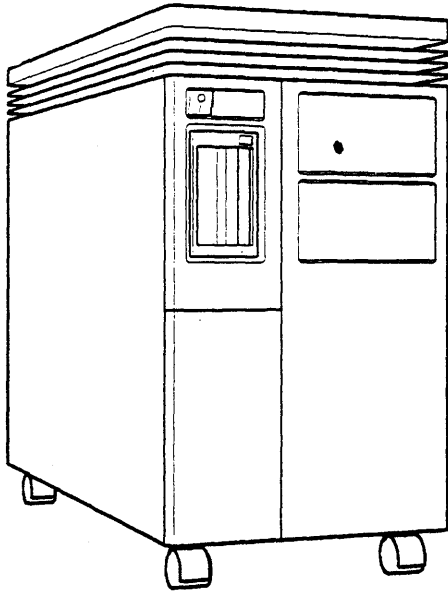
- You can build different configurations with
 - VAX processors
 - Peripheral devices
- System configurations can be classified as
 - Single processors
 - Multiple-processor configurations
- A system can be:
 - A single VAX processor and its peripheral devices
 - A collection of VAX processors

Single Processor Configurations

- Any single VAX processor and its peripheral devices
- The family of VAX processors includes
 - VAX 8700
 - VAX 8650
 - VAX 8250
 - VAX-11/785
 - VAX-11/780
 - MicroVAX 3000
 - MicroVAX II

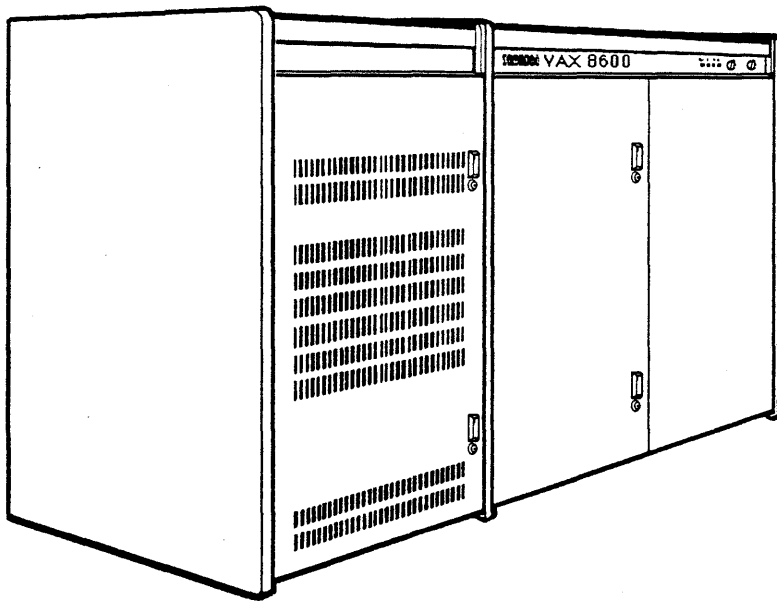
Figures 1-9 and 1-10 are not drawn to scale.

Figure 1-9: MicroVAX II



TTB_X0308_88

Figure 1-10: VAX 8600



TTB_X0309_88_S

Multiple-Processor Configurations

- Two or more communicating processors
- There are three classifications
 - Tightly-coupled multiprocessors
 - Networks
 - VAXcluster systems

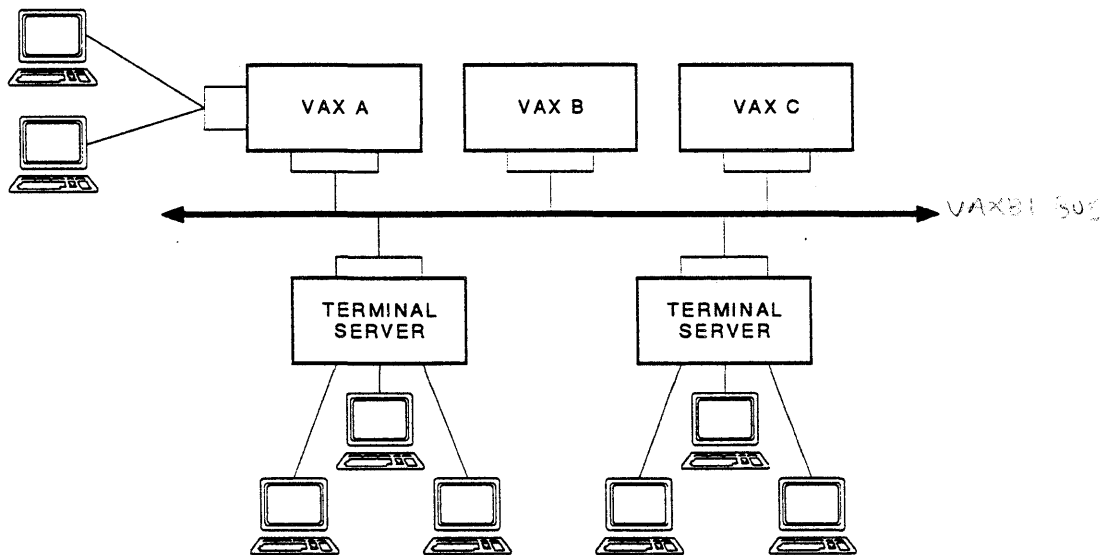
OS - NETWORK!

Tightly-Coupled Configurations

- Share operating system code
- Cannot operate independently
- Provide high performance
- Used in compute-intensive applications
- Example: VAX 8820
 - Two CPUs share memory by means of a VAXBI system bus
 - Master processor runs the VMS operating system and controls the attached processor

FAST # CPUs, BUT DIFFICULT TO USE!

Figure 1-11: A Tightly-Coupled System Configuration



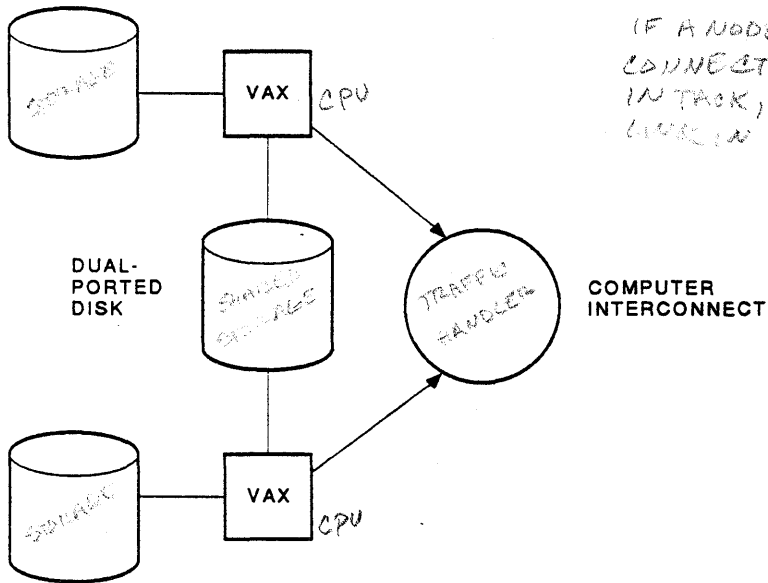
TTB_X0310_88

VAXcluster Systems

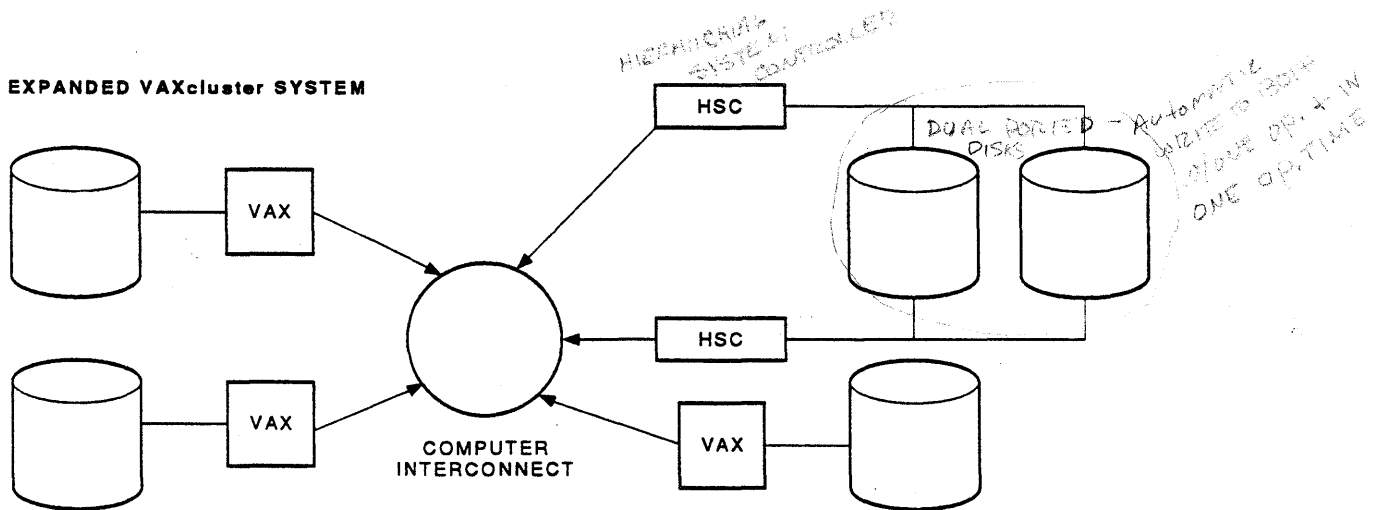
- Flexible multiprocessing system
- VAXcluster systems can share
 - Disk and tape devices
 - A common file system
- In addition to providing the functions of a network, VAXcluster systems provide
 - Higher availability of system resources
 - Faster and easier sharing of information and resources between nodes
- A VAXcluster system configuration
 - May have hardware similar to a network configuration
 - May contain the same components as a network
 - VAX processors
 - Communication devices
 - Transmission media
 - Terminal servers
 - DECnet software
 - May have other VAXcluster system specific hardware
 - Hierarchical Storage Controller (HSC)
 - Computer Interconnect (CI)
- * Major difference between a VAXcluster system and a network is VMS cluster software, which synchronizes access to shared resources.

Figure 1-12: VAXcluster System Structure

SIMPLE VAXcluster SYSTEM



EXPANDED VAXcluster SYSTEM



TTB_X0168_88

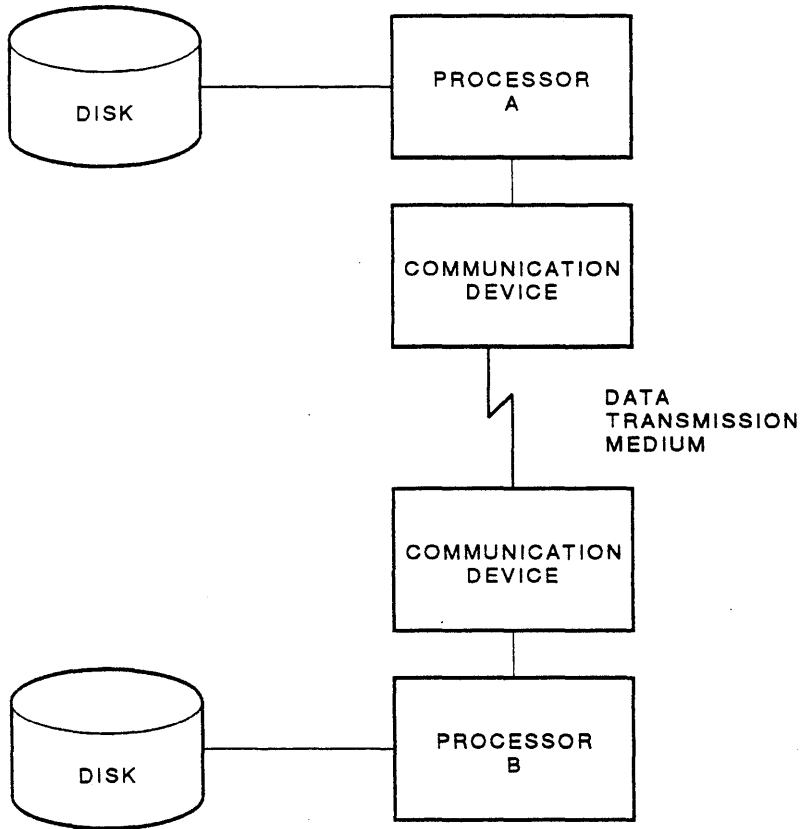
Notes on Figure 1-12:

1. The expanded VAXcluster system has added a third VAX system (with a local disk), and two HSC controllers with two dual-ported disks connected to them.
2. Any of the three VAX systems in the expanded VAXcluster system can mount the two disks that are connected to the HSC controllers. HSC disks are more available to users than local disks, because HSC disks are not dependent upon the availability of any processor.
3. The disks are dual-pathed to the HSC controllers, further increasing the disks' availability in the VAXcluster system. If one HSC fails, all traffic to connected disks automatically switches to the second HSC.

NETWORKS

- Consist of two or more communicating processors
- VMS system can be connected to
 - Other DIGITAL systems
 - Other manufacturers' systems
- DIGITAL-to-DIGITAL networks are established using
 - Two or more processors
 - Hardware communication devices
 - Data transmission media
 - Terminal servers (optional)
 - DECnet software
- DECnet software enables communication between networked systems
- A user logged in to one of these systems can
 - Communicate with a user who is logged in to another node
 - Access disk files stored on another node
 - Write programs that communicate with programs running on another node

Figure 1-13: A DECnet Network



TTB_X0312_88

Notes on Figure 1-13:

1. This network consists of two processors, or nodes. Each node has a disk drive.
2. Each processor in the network has an attached communication device. The communication devices are connected by a data transmission medium.
3. Access to disk files stored on a given node depends upon the availability of that node. For example, if Processor A is shut down, any disk files stored on Processor A's disk become inaccessible to users logged in to Processor B.

MODULE 2

GETTING STARTED

INTRODUCTION

To perform daily tasks on a VMS system, you must issue instructions written in the *DIGITAL Command Language (DCL)*. DCL consists of a vocabulary and rules of grammar, as in any language.

The DCL vocabulary includes commands, parameters, and qualifiers, all of which perform functions similar to those of verbs, nouns, adverbs, and adjectives in English. When you arrange them to form a command line, the *Command Language Interpreter (CLI)* causes images to be run to perform the requested actions.

This module introduces you to:

- Communicating with the VMS system by using the DIGITAL Command Language (DCL)
- Using both online and printed VMS documentation

OBJECTIVES

To effectively use the interactive features of the VMS system, you should be able to:

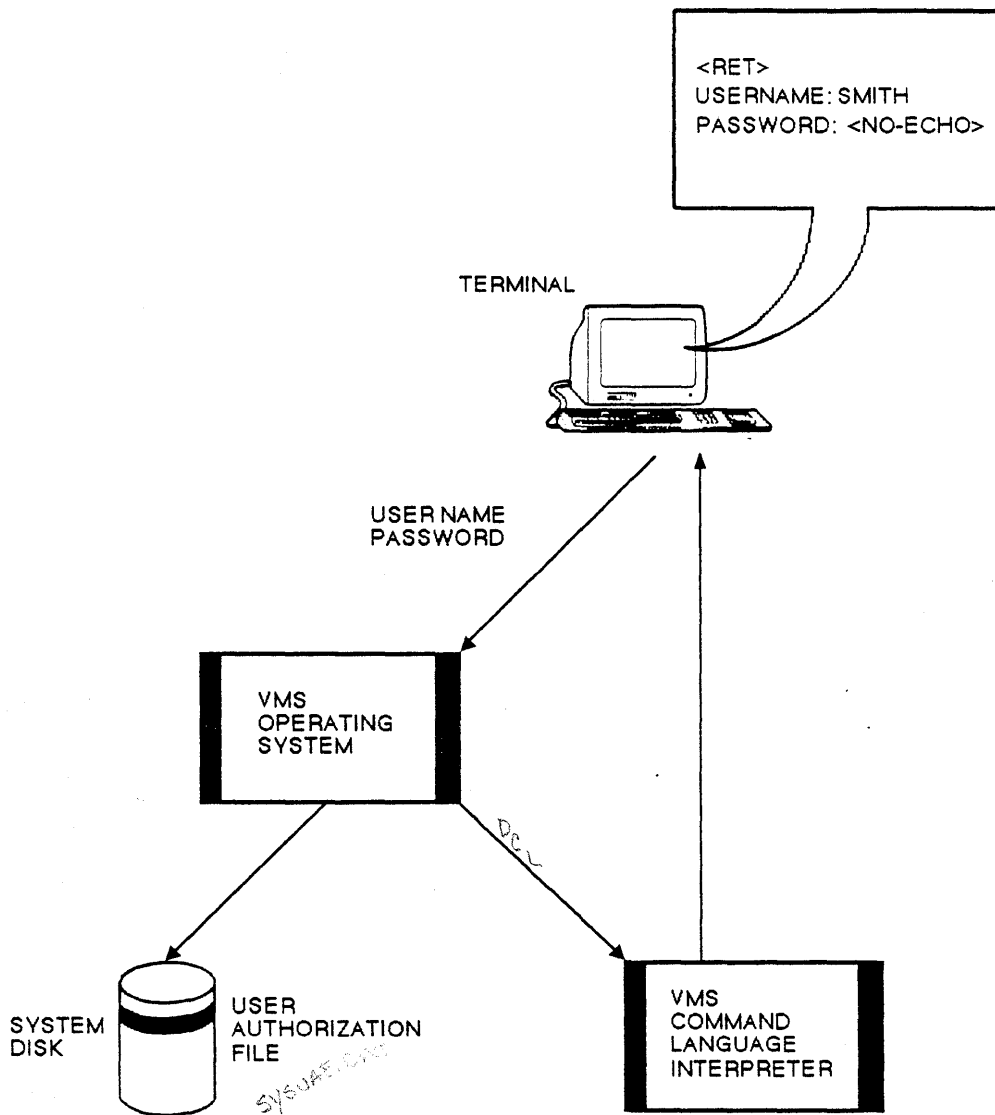
- Log in and log out of the system.
- Use DCL to perform work on the user's behalf.
- Use the VMS HELP facility and VMS documentation to obtain information about DCL commands and error messages.
- Interpret any VMS error messages and issue a corrected command by using the DCL command-line editor.
- Obtain and interpret information about the terminal, process, and system.

RESOURCES

- *VMS General User's Manual*
- *VMS DCL Dictionary*

LOGGING IN TO A VMS SYSTEM

Figure 2-1: Enter a Valid User Name and Password



TTB_X0313_88_S

USER NAME AND PASSWORD

Your User Name:

- Consists of 1 to 12 characters
- Is assigned by the system manager

Your Password:

- Consists of 1 to 31 characters
- Legal characters include:
 - A through Z
 - 0 through 9
 - \$ (dollar sign)
 - _ (underscore)

BEGINNING AND ENDING A TERMINAL SESSION

To log in to the system:

- Press the RETURN key (<RET>) on the terminal keyboard.
- In response to the prompt *Username:*, type your user name, then press <RET>.
- In response to the prompt *Password:*, type your password, then press <RET>. The system does not display your password.

To log out of the system:

- At the DCL prompt (\$), type LOGOUT and press <RET>.

Example 2-1: How to Log In and Log Out

```
VAX/VMS SUPER
Username: SMITH
Password:
Welcome to VAX/VMS SUPER
Last interactive login on Wednesday, 30-DEC-1987 10:27

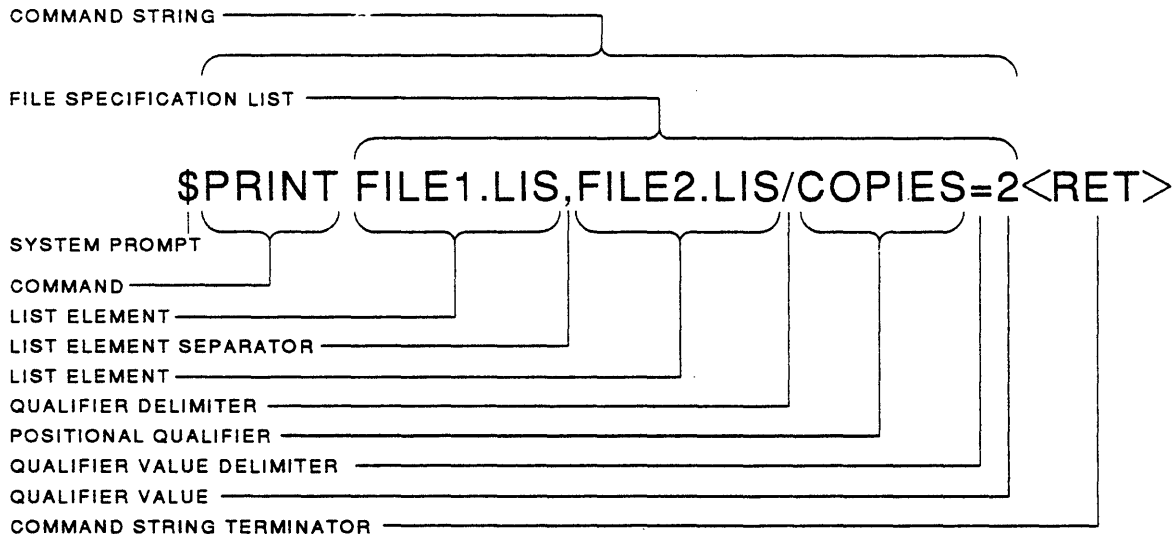
$ LOGOUT/FULL
SMITH      logged out at  5-JAN-1988 10:53:51.92
Accounting information:
Buffered I/O count:           46      Peak working set size:   333
Direct I/O count:            24      Peak page file size:    1969
Page faults:                 496     Mounted volumes:         0
Charged CPU time:            0 00:00:02.51  Elapsed time:           0 00:00:18.42
```

NOTE

The /FULL qualifier of the LOGOUT command displays a summary of the accounting information for the terminal session. \$ LOGOUT<RET> does not display any information.

DCL COMMAND FORMAT

Figure 2-2: The Elements of a Command Line



TTB_X0314_88

COMMAND LINE CONSTRUCTION

- One or more spaces or tabs separate commands, command options, and parameters from each other.
- Slash marks (/) separate qualifiers from commands and parameters.
- Commas (,) separate elements in a parameter list.
- Pressing <RET> passes the command line to the DCL CLI for execution regardless of the cursor position on the line.

Table 2-1: Elements of DCL Commands

Command Element	Definition
Command Line	<p>A command line is the complete specification of a DCL command. One command line can consist of up to 1024 characters.</p> <p>(NOTE: By entering a hyphen (-) prior to pressing <RET>, you can enter the command line in segments. Each command line segment can consist of up to 256 characters. The system concatenates the segments into one DCL command prior to interpreting the command.)</p>

Required Elements of a Command Line

Verb	<p>The verb of the command line is like the verb of a sentence in English. It specifies the action of your request. The verb usually consists of one word.</p> <p>Example: <code>\$ HELP</code></p>
Parameter	<p>Parameter(s) receive the action of the verb, much like the object does in an English sentence.</p> <p>Example: <code>\$ PRINT FILE1.TXT</code></p> <p>In the instruction <code>PRINT FILE1.TXT</code>, <code>PRINT</code> is the verb and <code>FILE1.TXT</code> is the parameter.</p>

Optional Elements of a Command Line

Qualifier	<p>The qualifier(s) of the command line describe or modify the action taken by the verb. A slash (/) precedes each qualifier. You can place qualifiers after the verb or after a parameter. (Some qualifiers accept one or more values.)</p> <p>Format: <code>/qualifier[(value[,...])]</code></p> <p>Example: <code>\$ SHOW PROCESS/ALL</code></p> <p>The qualifier <code>/ALL</code> modifies the action. (There are three types of qualifiers. For an explanation of the qualifier types, refer to Table 2-2).</p>
Value	<p>A value assigns a specific quantity to a qualifier. If you specify more than one value, you must separate the values with commas and enclose them in parentheses.</p> <p>Example: <code>\$ PRINT/COPIES=2 FILE1.TXT</code></p> <p>In the above instruction, <code>/COPIES=integer</code> is a qualifier to the verb <code>PRINT</code>. The value of the qualifier is the integer 2.</p>

Table 2-1: Elements of DCL Commands (Cont)

Command Element	Definition
Optional Elements of a Command Line	
\$	When you are in interactive mode, the DCL CLI ignores the dollar sign. However, the dollar sign must precede commands you place in files. (This technique will be discussed in the Batch and Print Jobs module and the Command Procedures module.)
!	The exclamation mark (!) indicates a comment. The system disregards anything on a command line following an exclamation mark. (The exclamation mark helps document commands you place within a file.) Examples: \$!The DCL CLI ignores this comment line \$ SHOW PROCESS !This comment is ignored
* Label:	The label is a character string that identifies a particular line in a file that contains DCL commands. Such a file is referred to as a command procedure. You should use labels only in command lines within command procedures.

* **Table 2-2: The Three Types of DCL Qualifiers**

Qualifier Type	Comments
Command Qualifiers	<p>Command qualifiers have the same meaning regardless of where they appear in the command line.</p> <p>Examples:</p> <pre>\$ PRINT/HOLD FILE1.TXT <i>or</i> \$ PRINT FILE1.TXT/HOLD</pre> <p>Since /HOLD is a command qualifier, the above two commands have the same effect. Both commands place the request in a hold state.</p>
Positional Qualifiers	<p>Positional qualifiers have different meanings depending on where they appear in the command line.</p> <p>Example:</p> <pre>\$ PRINT/COPIES=2 FILE1.TXT, FILE2.TXT</pre> <p>A positional qualifier placed after the verb, but before the first parameter, affects the entire command line. Therefore, this command requests the printing of two copies of FILE1.TXT and two copies of FILE2.TXT.</p> <pre>\$ PRINT FILE1.TXT/COPIES=2, FILE2.TXT</pre> <p>A positional qualifier placed after a parameter affects only that parameter. Therefore, this command line requests the printing of two copies of FILE1.TXT and one copy of FILE2.TXT.</p>
Parameter Qualifiers	<p>There are several types of parameter qualifiers. Refer to the command descriptions in the <i>VMS DCL Dictionary</i> for the names and types of parameter qualifiers that can be used with each command.</p>

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

Red Screen

DCL FEATURES

Table 2-3: Features of DCL

Feature	Example	Description
Continuation	<pre>\$ PRINT/COPIES=2 - _ \$ FILE1.TXT, FILE2.TXT, - _ \$ FILE3.TXT, FILE4.TXT</pre>	The hyphen continues a command line over more than one line of input.
Abbreviation	<pre>\$ LOGOUT ! These are \$ LOGO ! equivalent \$ LO</pre>	You can abbreviate commands and keywords to four or fewer characters.
Prompting	<pre>\$ PRINT <RET> _ File: FILE1.TXT</pre>	Type a command and press <RET>. DCL will prompt for the required and optional parameters.

EDITING A DCL COMMAND LINE

To accomplish various tasks, you will have to perform many of the following operations on a DCL command line:

- Move the cursor
- Add or delete data from the command line
- Recall a previously issued command line
- Control information displayed at your terminal
- Terminate an operation

Table 2-4: Moving the Cursor

Operation	Special Function Key	Comments
Moving the Cursor to the Left	LEFT ARROW CTRL/D	Moves the cursor one character to the left. Holding the LEFT ARROW key down moves the cursor until the key is released.
Moving the Cursor to the Right	RIGHT ARROW CTRL/F	Moves the cursor one character to the right. Holding the RIGHT ARROW key down moves the cursor until the key is released.
* Moving the Cursor to the Beginning of the Line	CTRL/H	Moves the cursor to the beginning of the line.
* Moving the Cursor to the End of the Line	CTRL/E	Moves the cursor to the end of the line.

Table 2-5: Changing Data on the Command Line

Operation	Special Function Key	Comments
Deleting a Character	DELETE	Deletes the character to the left of the cursor. (Note that on a hardcopy terminal the system responds by typing a backslash (\) followed by the delete character.)
*Deleting a Word	LF CTRL/J	Deletes the preceding word.
*Deleting the Line	CTRL/U	Erases all characters to the left of the cursor. When the cursor is at the end of the line, pressing the CTRL/U key sequence erases the entire command line.
*Clearing the Line and the Type-Ahead Buffer	CTRL/X	Discards the current line and deletes data in the type-ahead buffer.
Replacing a Character	text...	Pressing any keyboard character causes that character to replace the character originally at the cursor position. This is referred to as the OVERSTRIKE mode of operation. OVERSTRIKE mode is the default data entry mode.
*Inserting a Character	CTRL/A	CTRL/A changes the terminal's data entry mode from OVERSTRIKE mode to INSERT mode. When you press any keyboard key in INSERT mode, the original text moves to the right, making room for the new character(s). If you are in INSERT mode, CTRL/A changes the terminal's mode back to OVERSTRIKE.

Table 2-6: Recalling a Previously Issued Command Line

Operation	Special Function Key	Comments
Recalling the most recent commands	UP ARROW CTRL/B	Consecutively recalls the last command passed to the DCL CLI. Commands used for recalling previously entered commands from the command buffer are not retained.
Recalling recently entered commands	DOWN ARROW	Recalls recently entered commands from the command buffer. After recalling the most recently entered command, pressing the DOWN ARROW key displays a blank line.
Refreshing a DCL command	CTRL/R	Redisplays the last unentered command line on your terminal. (Note that on a hardcopy terminal, the system issues a <RET> prior to retyping the current command line.)

* THE RECALL COMMAND

The **RECALL** command displays previously-entered commands so that the user can re-use them. Up to 20 commands are stored in the RECALL buffer.

Press <RET> after you have redisplayed a command to have the system execute that command.

Table 2-7: Recalling a Previous Command Line with the RECALL Command

Operation	Command/Qualifier	Comments
* Displaying the RECALL buffer	RECALL/ALL	The /ALL qualifier displays a numbered list of the commands you have entered. (UP TO 20)
* Recalling the third most recently entered command line	RECALL 3	Adding the parameter 3 to the RECALL command recalls the third most recently entered command line.
* Recalling the most recently entered PRINT command	RECALL PRINT	The command parameter PRINT recalls the last PRINT command entered.
* Erasing the RECALL buffer	RECALL/ERASE	The /ERASE qualifier empties the contents of the RECALL buffer.

(or up arrow!)
(and down arrow)

EX: SOMEONE SENDS YOU A MESSAGE WHILE IN EDT, REFRESHES SCREEN.

Table 2-8: Controlling the Display of Information at Your Terminal

Operation	Special Function Key	Comments
* Redisplaying the Terminal Screen	CTRL/W	Within some programs, CTRL/W refreshes the current terminal display. (This key sequence is useful within the EDT editor.)
Suspending Terminal Output	CTRL/S	Suspends the display of information at your terminal.
	HOLD SCREEN	The HOLD SCREEN key on VT200-series terminals also suspends the display.
	NO SCROLL	The NO SCROLL key on VT100-series terminals also suspends the display.
Resuming Terminal Output	CTRL/Q	Allows the program suspended by CTRL/S to resume execution.
	HOLD SCREEN	Depressing the HOLD SCREEN key again after halting the terminal display resumes the display on VT200-series terminals.
	NO SCROLL	Depressing the NO SCROLL key again after halting the terminal display resumes the display on VT100-series terminals.
* Suppressing and Resuming Terminal Display	CTRL/O	Suppresses the current image's output. The routine that generates the display continues to execute, returning control to the terminal when it terminates. The command echoes as <i>Output off</i> . Entering the key sequence a second time, enables the terminal to receive output again. The command echoes as <i>Output on</i> .

↳ ALSO FOUND IN V.2.0

Table 2-9: Terminating an Operation

Operation	Special Function Key	Comments
Canceling a Command Line	CTRL/Y	Cancels the execution of the current image. Control returns to DCL command level. The command echoes as <i>Interrupt</i> .
	CTRL/C	Within certain applications, CTRL/C cancels command processing. CTRL/C echoes as <i>Cancel</i> . (When CTRL/C is not enabled, use CTRL/Y.)
Closing a File	CTRL/Z	Indicates the end of a file entered at the terminal (for example, a file you opened with the CREATE command). In certain utilities, the CTRL/Z key sequence is equivalent to the EXIT command (for example, the MAIL Utility). The command echoes as <i>Exit</i> .
Determining Your Current Process Operation	CTRL/T	Momentarily interrupts output to display a line of statistical information about the current process. This key sequence is only informative. It does not affect the process operation.

GETTING HELP

The Documentation Set

- Contains
 - Information about the VMS system
 - Discussions of concepts
 - Command examples
 - Definitions
 - Restrictions and problems
- Refer to Table 2-10 as an index to documentation



Table 2-10: Manuals for Locating Information About Your System

Topic	VMS Manual
Commands and Qualifiers	VMS DCL Dictionary
Descriptions	See Prompts, Command Parameters, and File Qualifiers sections of a given entry
Examples	See Examples section of a given entry
Syntax	See Format section of a given entry
Concepts of the Operating System	VMS Glossary and Concepts Manual
Definitions of Terms and Acronyms	VMS Glossary and Concepts Manual
Information and Error Messages Issued by the System	VMS System Messages and Recovery Procedures Reference Manual
Interpretations	
Suggested user actions	
Location of Major Topics in the Document Set	Overview of VMS Documentation VMS Master Index
Restrictions and Known Problems with Current Operating System Release	VMS Version V5.0 Release Notes (Current Version)
Software Available for Your Use	A VMS Operating System (Current Version) Software Product Description is included with the VMS documentation set

The Online Help Facility

To invoke the Help facility:

- Enter the **HELP** command at the VMS prompt
- Select a topic from the displayed list
 - Entering a topic printed in uppercase yields text on the DCL command of the same name
 - Entering a topic printed in lowercase yields text on a general topic
- Press <CTRL/Z> to leave the Help facility
- Table 2-11 lists commands for operating the Help facility

Table 2-11: Using the DCL HELP Facility

Operation	Command	Comments
Displaying a List of Available Help Topics	\$ HELP	The HELP command lists the topics on which you can obtain information. The system responds with the <i>Topic?</i> prompt.
Displaying Instructions on the Help Facility	\$ HELP INSTRUCTIONS	Displays detailed instructions on how to use the Help facility.
Displaying a List of Hints	\$ HELP HINTS	Produces lists of commands grouped by function.
Displaying Information About the SHOW Command	\$ HELP SHOW	The SHOW parameter used with the HELP command produces an informative display on the SHOW command. The system responds with the prompt <i>SHOW Subtopic?</i>
Displaying Information About the SHOW PROCESS Command	\$ HELP SHOW PROCESS	By including the keyword PROCESS, you can have the system produce an informative display on the SHOW PROCESS command. <i>SHOW PROCESS Subtopic?</i> is the system prompt.
* Redisplaying the Previous HELP Screen	<?>	Pressing the question mark key (?) re-displays the previous HELP message.
Returning to DCL Command Level	<RET>	Moves you one level closer to DCL level. When you are at the <i>Topic?</i> prompt, pressing <RET> returns you to DCL command level.
	* <CTRL/Z>	CTRL/Z or EXIT returns you to DCL command level regardless of the HELP prompt. Both commands echo as <i>EXIT</i> .

CHANGING YOUR PASSWORD

The DCL command **SET PASSWORD** changes your password. When changing your password, user input is not echoed at the terminal. You must enter the new password twice. If the two entries do not match, the password does not change.

Example 2-2: Changing Your Password

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

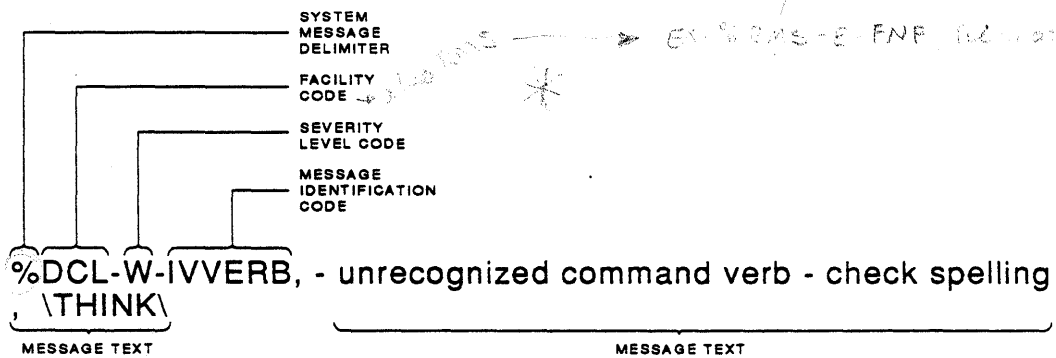
Note that in a real session, neither the old password nor the new password and its verification appear on the screen.

* SYSTEM MESSAGES — ERROR FACILITY — puts messages to screen.

A system message consists of the following parts:

- System message delimiter
- Facility code
- Severity level code
- Message identification code
- Message text

Figure 2-3: The Elements of a System Message



TTB_X0315_00

Table 2-12: Elements of the System Message

Message Element	Code	Purpose
System Message Delimiter	% -	All system messages begin with either a percent sign (%) or a hyphen (-). The percent sign precedes the first system message received, while the hyphen precedes all additional messages.
Facility Code	DCL TYPE	Names the portion of the operating system that detected an error.
Severity Level Code	S <small>SUCCESS (NOT COMMON)</small> I <small>INFO</small> W <small>WARNING</small> E <small>ERROR</small> F <small>FATAL - FATAL ERROR</small>	Describes the severity of the error. (For an explanation of each severity level code, see Table 2-13.)
Message Identification Code	FNF IVVERB	Used to locate further information about a message.
Message Text	unrecognized command verb check spelling	Gives a more detailed explanation of the error, and suggests an action to recover from the error.

Table 2-13: Severity Levels in System Error Messages

Severity Level	Abbreviation	Comments
Success	S	VMS does not usually display success messages.
Informational	I	VMS sometimes displays additional information about success of operation.
Warning	W	Some operations may have succeeded. Others may have failed.
Error	E	The operation probably failed, but some part may have succeeded.
Severe (or Fatal) Error	F	The operation failed.

* DISPLAYING CHARACTERISTICS OF TERMINAL, PROCESS, AND SYSTEM

Your working environment is defined by the characteristics assigned to:

- Your terminal
- Your process
- Your system

Terminal Characteristics

- Physical (hardware)
- Assigned by system manager
- Displayed and changed with the following commands:
 - * **SET TERMINAL**
 - * **SHOW TERMINAL**

* THE SHOW TERMINAL COMMAND

The **SHOW TERMINAL** command displays the current characteristics of a terminal. Each characteristic corresponds to an option of the **SET TERMINAL** command.

Example:

```
$ SHOW TERMINAL

Terminal: _VTA145:      Device_Type: PRO_Series      Owner: SMITH
Physical terminal: _LTA88:      Username: SMITH
Input: 9600      LFfill: 0      Width: 80      Parity: None
Output: 9600     CRfill: 0      Page: 24
Terminal Characteristics:
Interactive      Echo              Type_ahead       No Escape
No Hostsync     Ttsync           Lowercase        Tab
Wrap            Scope            No Remote        No Eightbit
Broadcast       No Readsnc       No Form          Fulldup
No Modem        No Local_echo    No Autobaud      Hangup
No Brdcstmbx   No DMA           No Altypeahd     Set_speed
Line Editing    Overstrike editing No Fallback      No Dialup
No Secure server Disconnect       No Psthru        No Syspassword
No SIXEL Graphics No Soft Characters Printer port     Application keypad
ANSI_CRT       Regis            No Block mode    Advanced_video
Edit_mode      DEC_CRT          No DEC_CRT2      No DEC_CRT3
```

* THE SET TERMINAL COMMAND

The **SET TERMINAL** command changes the system's interpretation of the terminal's characteristics.

Example:

```
$ SET TERMINAL/WIDTH=132
```

This example changes the width of the terminal screen to 132 characters.

* **Table 2-14: Commands for Displaying the Characteristics of Your Terminal, Process, and System**

Information	VMS Command and Option	Command Qualifier
Time of Day	\$ SHOW TIME	None
Terminal Characteristics	\$ SHOW TERMINAL	None
* Process Parameters		
Default Device	* \$ SHOW PROCESS	None
Default Directory	\$ SHOW PROCESS	None
User Name	\$ SHOW PROCESS	None
Priority	\$ SHOW PROCESS	None
Process Identification Code (PID)	\$ SHOW PROCESS	None
User Identification Code (UIC)	\$ SHOW PROCESS	None
Account Name	\$ SHOW PROCESS	/QUOTAS
Process Quotas and Limits	\$ SHOW PROCESS	/QUOTAS
Privileges	\$ SHOW PROCESS	/PRIVILEGES
Space Available for Your Use on Your Default Device	\$ SHOW QUOTA	None
All Processes Running on Your System	\$ SHOW SYSTEM	None
Names of All Users Currently Logged in to Your System	\$ SHOW USERS	None
Names of Devices on Your System	\$ SHOW DEVICES	None

(SHOW PROCESS /ALL)

SUMMARY

To log in to the system:

- Press <RET>.
- Type your user name <RET>.
- Type your password <RET>. Remember, your password is not displayed.

To log out of the system:

- Type **LOGOUT** <RET>.

DCL Command Elements

Command Element	Definition
Command line	A command line is the complete specification of a DCL command.

Required Elements of a Command Line

Verb	The verb specifies the action of your request.
Parameter	Parameter(s) receives the action of the verb.

Optional Elements of a Command Line

Qualifier	The qualifier(s) describes or modifies the action taken by the verb. A slash (/) precedes each qualifier.
Value	A value assigns a specific quantity to a qualifier.
\$	The dollar sign must precede commands you place in files.
!	The exclamation mark (!) indicates a comment.
Label:	The label is a character string that identifies a particular line in a command procedure.

Getting Help

The documentation set contains information about the VMS system, discussions of concepts, and command definitions and examples.

The online Help facility is invoked by entering the **HELP** command. **System Messages**

System messages consist of the system message delimiter, facility code, severity level code, message identification code and the message text.

MODULE 3

CREATING AND EDITING TEXT FILES

INTRODUCTION

One of the most common tasks for a user is the creation and modification of text files. Text files can assume a number of forms and can serve many purposes. They can be:

- Memos and letters
- Data files that are used by other programs and utilities
- Computer programs written in a language like FORTRAN, Pascal, or COBOL

VMS software provides a number of ways to create, maintain, and modify text files. The two most popular are:

- The EDT Editor
- The Extensible VAX Editor (EVE)

— MODIFIABLE EDITOR

OBJECTIVES

To create and modify text files on a VMS system, you should be able to:

- Use the proper DCL command to invoke a text editor.
 - EDT Editor
 - EVE Editor
- Identify the major features of each editor.
- Use appropriate commands and keys to perform editing tasks such as:
 - Moving the cursor
 - Adding and deleting text
 - Selecting and manipulating text strings
- Terminate an editing session.
- Use available online Help facilities.
- Recover files that were being edited at a system interruption.

RESOURCES

- *VMS Guide to Text Processing*
- *VMS Text Processing Utility Reference Manual (Appendix F)*
- *VMS EDT Reference Manual*

CHOOSING AN EDITOR

There are many reasons for choosing one editor over another. Your choice may be based on ease of editing, the ability to edit more than one file simultaneously, or using multiple buffers and windows.

EDT is the editor supplied with many DIGITAL systems. The EVE editor is an editor available only on VMS systems.

Restrictions may apply in unique situations, such as having to edit only on a hardcopy terminal.

Features of both the EDT and EVE editors follow. This listing should aid you in deciding which editor to choose.

EDT Editing Utility

- Default text editing utility supplied with a VMS system
- Available on most DIGITAL systems
- *• Allows editing on hardcopy terminals *- NOT SO W/EVE
(THIS IS WHY EDT IS KEPT AROUND)*
- More system load than EVE
- Two editing modes are available
 - Line mode *- DEFAULT MODE*
 - Keypad mode *- FULL SCREEN*
- Line mode
 - Automatically entered when EDT is invoked
 - Indicated by an asterisk prompt (*)
 - Works with the file on a line-by-line basis
 - Primarily intended for a hardcopy terminal
- Keypad mode
 - Requires a video terminal
 - Entered by using the **CHANGE** command at the line-mode prompt (*)
 - Works with the file as a unit
 - Modifications made on the screen become modifications to the file

The Extensible VAX Editor (EVE)

Using EVE, it is possible to manipulate and edit text both in newly created files and existing files.

- Features include:
 - Keypad editing
 - Insert and overstrike modes for text entry
 - Automatic word wrap
 - Multiple windows
- Can be customized to the user's needs, using the features of the VAXTPU programming language
- Primarily a VMS editor
- Less system load than EDT
- Provides more features than EDT
- Provides EDT-like keypad if desired
- Functions on VT100 and VT200-series and later terminals, and on VAX workstations

EVE was designed with both ease of learning and ease of use in mind. Testing has shown it to be easier to learn and use than EDT.

Each editor will be discussed in greater detail in the remainder of this module.

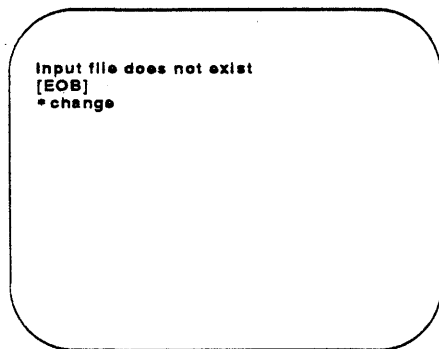
USING THE EDT EDITOR

Invoking the EDT Editor

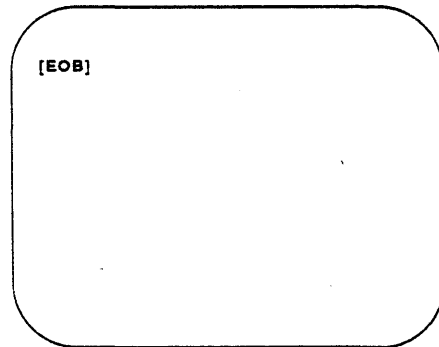
- Command format: `$ EDIT file-name` *DEFAULTS TO CMT*
- The command qualifier `/EDT` is available but not required *TO NAME EDT*

EDT Screen Layout

Figure 3-1: EDT Screen Layout – Line Mode and Keypad Mode



OR JUST "C" FOR "CHANGE"!



TTB_K0210_00

Using EDT Help

- From Line mode
 - Type **HELP** at the asterisk (*) prompt. Topics are then listed
 - Type **HELP topic-name**
 - Enter **CHANGE** to switch to Keypad mode (if desired)
- From Keypad mode
 - Press <PF2> (VT100) or <HELP> (VT200)
 - Press key on which you want help
 - Press space bar to exit

Example 3-1: Using the Help Facility On Line

```
$ EDIT MYFILE.TXT
  1  Although the computer has always been
*help
HELP
```

You can get help on a topic by typing:

```
HELP topic subtopic subsubtopic...
```

A topic can have one of the following forms:

1. An alphanumeric string (e.g. a command name, option, etc.)
2. The match-all or wildcard symbol (*)

```
Examples: HELP SUBSTITUTE NEXT
          HELP CHANGE SUBCOMMAND
          HELP CH
```

If a topic is abbreviated, HELP displays the text for all topics that match the abbreviation.

Additional information available:

CHANGE	CLEAR	COPY	DEFINE	DELETE	EXIT	FILL
FIND	HELP	INCLUDE	INSERT	JOURNAL	KEYPAD	MOVE
PRINT	QUIT	RANGE	REPLACE	RESEQUENCE	SET	SHOW
SUBSTITUTE	TAB	TYPE	WRITE			

*HELP CHANGE

The CHANGE command puts EDT in change mode. Use change mode to edit at the character level rather than the line level.

Format: CHANGE [range] [;nokeypad command(s)]

The optional range specifies the cursor position when you enter change mode. If you omit range, the current position is used.

There are three submodes of change mode. Which submode you use depends on the type of terminal you are using and whether or not you wish to use the auxiliary (numeric) keypad for editing commands. These modes are:

1. Hardcopy mode
2. Keypad mode
3. Nokeypad mode

If the CHANGE command contains a semicolon (;) it may be followed by nokeypad commands. If the last nokeypad command is EX, EDT returns to line mode for the next command line. This is the only form of the CHANGE command that may be used in a startup command file or macro.

Additional information available:

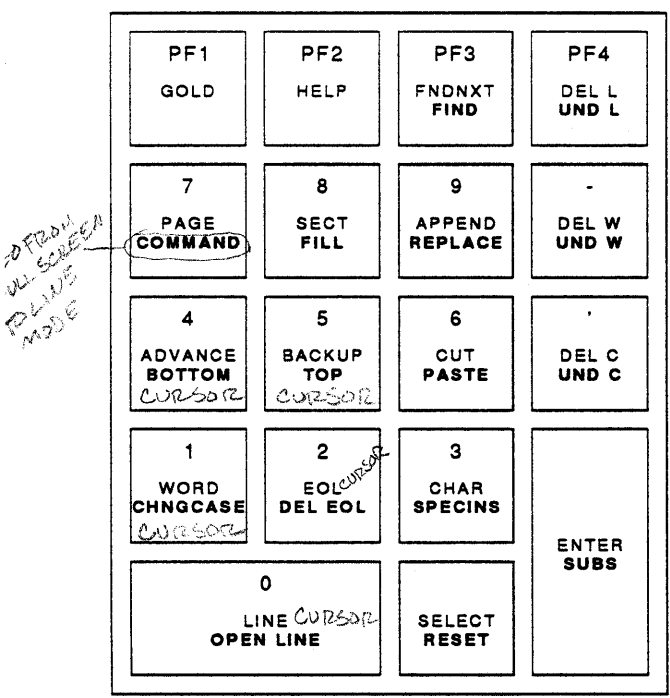
```
ENTITIES  HARDCOPY  KEYPAD  NOKEYPAD  SUBCOMMANDS
*QUIT
```


The EDT Keypad

Commonly Used Features

- GOLD key (<PF1>) activates alternate definitions for keypad keys
- Moving the cursor
 - By character, word, or line
 - To the top or bottom of files
 - In forward or reverse direction
- Deleting and undeleting text
 - By character, word, or line

* Figure 3-2: EDT Keypad Definitions



TTB_X0317_88

— "GOLD" KEY ALTERNATES BETWEEN
 LIGHT AND BOLD FUNCTIONS!
 (LIKE SECOND FUNCTION KEY ON A CALCULATOR)
 (AFFECTS ONLY ONE ACTION)

Table 3-1: Moving the EDT Cursor

Function	Key
Move one character in any direction	Arrow keys (UP, DOWN, LEFT, RIGHT)
Move to the beginning of the next line	0 (on keypad)
Move to the beginning of the next word	1 (on keypad)
Move to the end of the line	2 (on keypad)
Move to the next section of the text (16 lines)	8 (on keypad)
Move to the bottom of the buffer	GOLD key (PF1) followed by 4 (on keypad)
Move to the top of the buffer	GOLD key (PF1) followed by 5 (on keypad)

Table 3-2: Changing the EDT Cursor Direction

Direction	Key
Set cursor to forward	4 (on keypad)
Set cursor to backward	5 (on keypad)

Table 3-3: Deleting Text in EDT

Function	Key	Comments
Delete characters	⌘ (VT200-series) DELETE (VT100-series)	Deletes the character to the left of the cursor <i>VS.</i>
	* , (on keypad)	Deletes the character on which the cursor is positioned (<i>LIKE ONLY "X"</i>)
Delete words	- (on keypad)	Deletes characters from the cursor position to the beginning of the next word
Delete lines	PF4	Deletes text from the current cursor position to the beginning of the next line
	PF1 followed by 2 (on keypad)	Deletes text between the cursor and the end of the line

Table 3-4: Restoring Text in EDT

Function	Keypad Sequence
Restore the last character deleted	PF1 followed by comma (,)
Restore the last word deleted	PF1 followed by hyphen (-)
Restore the last line deleted	PF1 followed by PF4

Ending an EDT Editing Session

- From Keypad mode
- Press <CTRL/Z> to return to Line mode
- At the asterisk prompt
 - Enter **EXIT** to end the session and save changes, or
 - Enter **QUIT** to end the session without saving changes
- Or
- Press <PF1> then keypad key 7 - RE-ENTER LINE MODE!
- At the *Command:* prompt
 - Enter **EXIT** to end the session and save changes, or
 - Enter **QUIT** to end the session without saving changes

EDT File Recovery

- Journaling
 - Allows file recovery after a system interruption or failure
 - Used to reproduce the current editing session
 - Last few file modifications may not be recovered
- Journal File
 - The default file name is the same as the input file name
 - The default file type is JOU
 - Contains keystrokes and editing commands of your current terminal session
- Syntax
 - \$ EDIT/RECOVER file-name**
- * • Specify the original file type (not JOU)

Example 3-2: Recovering a File After a System Interruption

```
$ EDIT MYFILE.TXT
*CHANGE
Editing session in progress.
System interruption occurs.
System recovers.
$ EDIT/RECOVER MYFILE.TXT
```

USING EVE

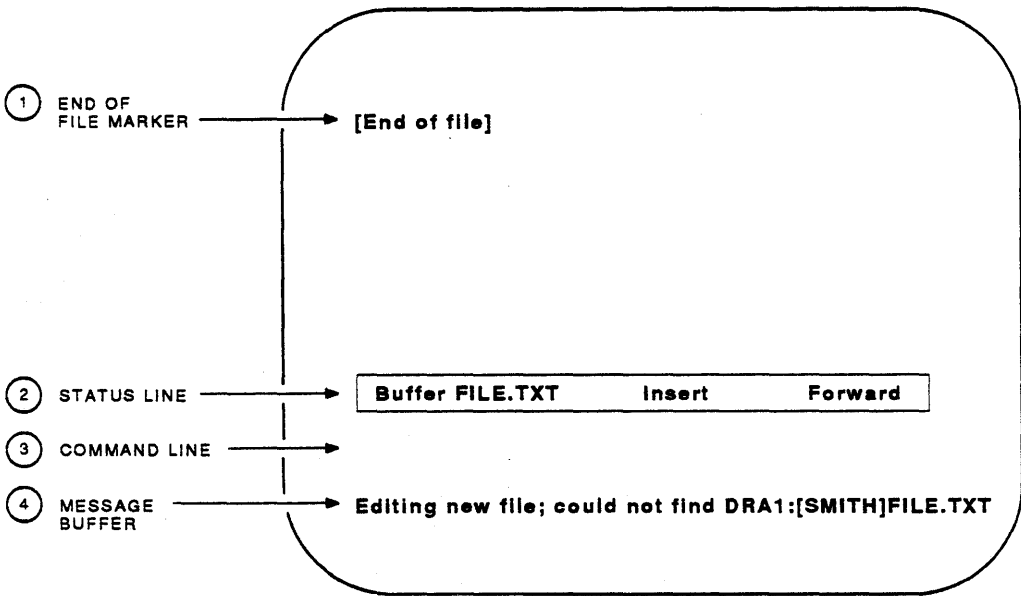
Invoking the EVE Interface

NO LINE #S!

* Command format: \$ EDIT/TPU file-name

EVE Screen Layout

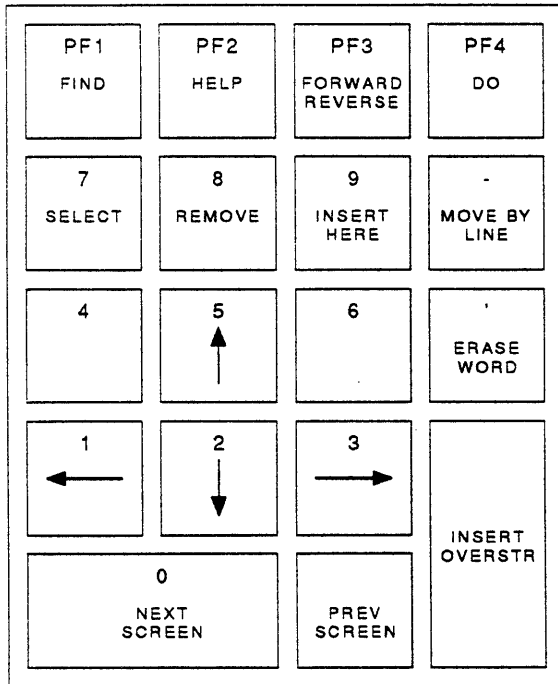
Figure 3-3: EVE Screen Layout



TTB_X0318_88

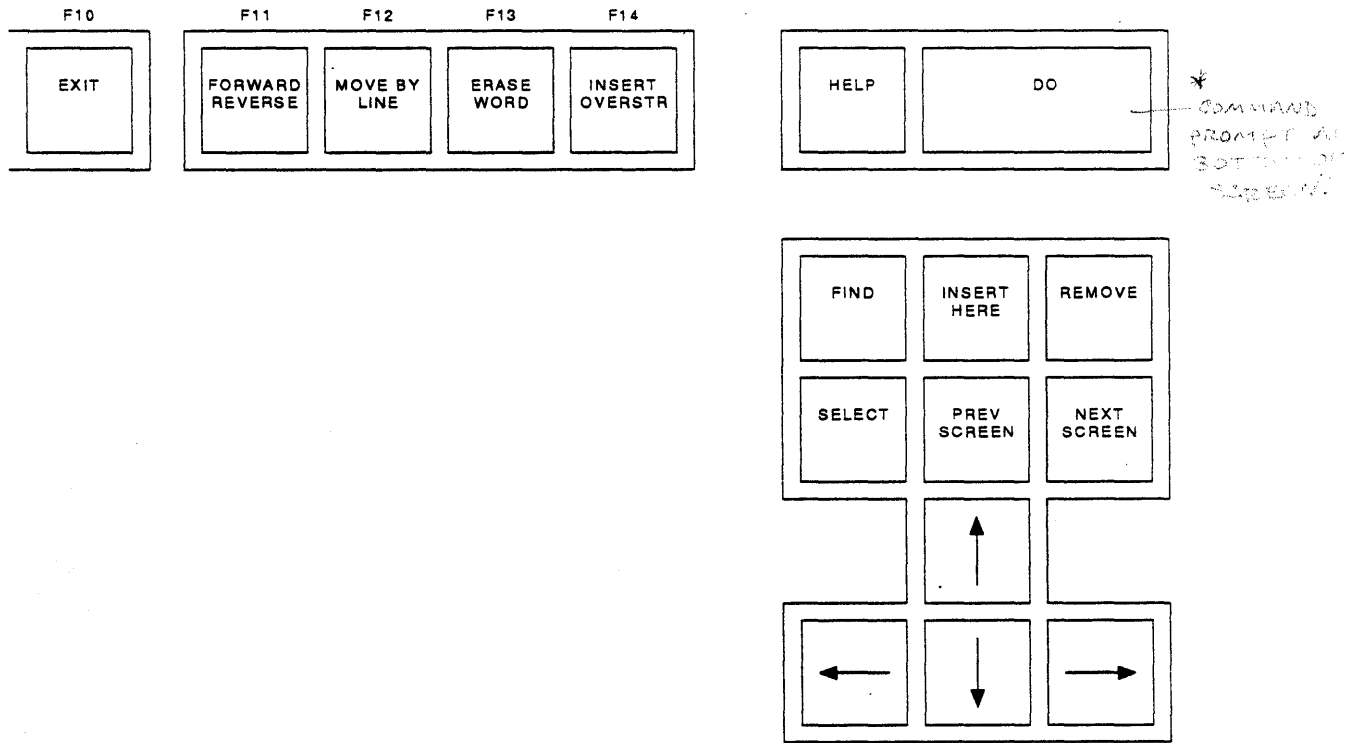
The EVE Interface

Figure 3-4: EVE Keypad Definitions (VT100-Series Terminals)



TTB_X0319_88

Figure 3-5: EVE Keypad Definitions (VT200-Series Terminals)



TTB_X0320_88

MOVING THE EVE CURSOR

The following table describes the editing keys and EVE commands that move the cursor.

* **Table 3-5: Moving the Cursor Using Keys**

Key	Cursor Destination
UP ARROW	Moves the cursor up one character.
DOWN ARROW	Moves the cursor down one character.
LEFT ARROW	Moves the cursor one character to the left.
RIGHT ARROW	Moves the cursor one character to the right.
CTRL/E	Moves the cursor to the end of the current line.
CTRL/H	Moves the cursor to the beginning of the current line.
MOVE BY LINE	Moves the cursor to the end of the current line or to the end of the next line if the cursor is already at the end of a line, when the current direction is forward. Moves the cursor to the beginning of the current line or to the beginning of the previous line if the cursor is already at the beginning of a line, when the current direction is reverse.
PREV SCREEN	Moves the cursor to the previous screen of the current buffer.
NEXT SCREEN	Moves the cursor to the next screen of the current buffer.

ENTER DO ↓

* Table 3-6: Using Commands to Move the Cursor

Command	Cursor Destination
TOP	Moves the cursor to the beginning of the current buffer.
BOTTOM	Moves the cursor to the end of the current buffer.
* BUFFER <BufferName>	Puts the specified buffer in the current window, and moves the cursor to the last location it occupied in that buffer. Creates a new buffer if the specified buffer does not exist.
* GET FILE [file-name]	Creates a new buffer that contains the text of the specified file, places the new buffer in the current window with the cursor at the beginning of the new buffer. If you specify a nonexistent file, an empty buffer is created.
LINE	Moves the cursor to the beginning of line n in the current buffer. n must be a positive integer.
MOVE BY WORD	Moves the cursor to the beginning of the next word when the current direction is forward. Moves the cursor to the beginning of the previous word when the current direction is reverse.
TWO WINDOWS OTHER WINDOW (OR OTHER SCREEN)	<i>splits buffer into two screen areas.</i> When there are two editing windows on your screen, the cursor moves to the last location it occupied in the other editing window.

*CAN HAVE DIFFERENT BUFFERS IN EACH WINDOW

WHAT LINE

gives current window position by line because no line numbers are specified on a regular basis in EVE.

ONE WINDOW

puts back to ONE window. (Make sure cursor is in correct window w/other command).

*(vs) EVE BUFFERS - contain data for modification.
EVE WINDOWS - used by the editor to display position of a cursor.*

also: NEXT WINDOW

ENLARGE WINDOW <#lines>

SHRINK WINDOW <#lines>

INSERTING TEXT IN EVE

The <INSERT OVERSTR> key changes the current editing mode. The editing mode is displayed in the status line.


Text is inserted at the current cursor position when in *Insert* mode, while text already in the file moves to the right.

Text already in the file is overwritten when in *Overstrike* mode at the current cursor position.

ERASING TEXT

Table 3-7 shows the editing keys used to erase text.

* Table 3-7: Keys for Deleting Text

Key	Effect
DELETE 	Deletes the character to the left of the cursor.
ERASE WORD	Deletes the current word or, if the cursor is not on a word, deletes the next word.
CTRL/U	Deletes all characters from the current cursor position to the beginning of the line.
SELECT	Marks text for removal from the initial cursor position to wherever you move the cursor.
REMOVE	Removes the text that was marked by the SELECT key.
INSERT HERE	Inserts the text in the INSERT HERE buffer into the file at the current cursor position.

* DEFINING AN EDT-LIKE KEYPAD

The command **SET KEYPAD EDT** defines an EDT-like keyboard. Note that this command does not enable you to enter EDT commands by using the <DO> key. You only have access to the EDT-like functions by pressing the keypad keys.

The EDT-style keypad does not fully implement EDT. The differences are:

- <CTRL/Z> makes EVE write the buffer to a file and exit to the DCL prompt.
- ^{KEYPAD 7}<GOLD/KP7> is defined as the <DO> key when the keypad is set to EDT.
- * • ^{KEYPAD 8}<GOLD/KP8> is defined as <FILL>, to reformat the currently selected text or the current paragraph. If you want this key to fill only the selected text (as in real EDT) redefine the key as <FILL RANGE>.
- EVE defines the <ENTER> key as <RET>.

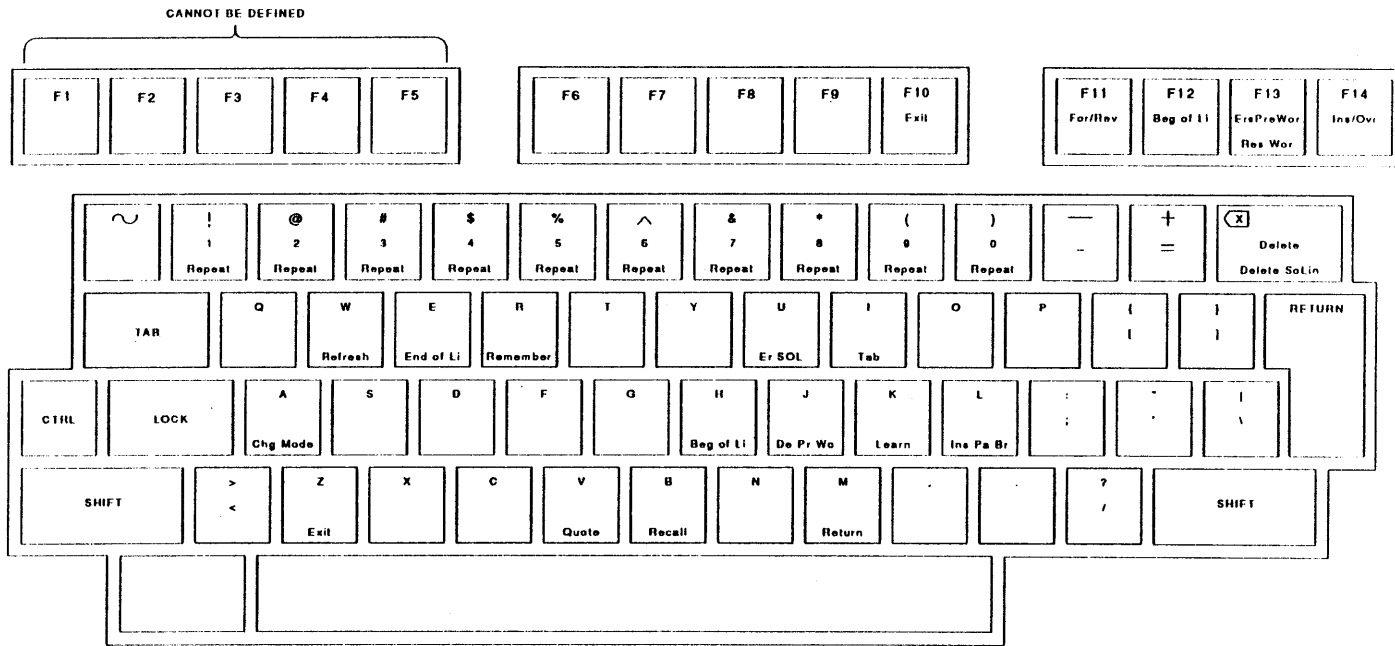
CANCELING AN EDT-LIKE KEYPAD

^{DO/KP7 first!} The command **SET KEYPAD NOEDT** cancels the EDT-like keypad setting.

For VT100-series terminals, this command sets the keypad to VT100, which is the default setting.

For VT200-series terminals, this command sets the keypad to NUMERIC, which is the default setting.

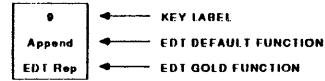
RESULT OF SET KEYPAD EDT COMMAND



SAMPLE KEYBOARD KEY



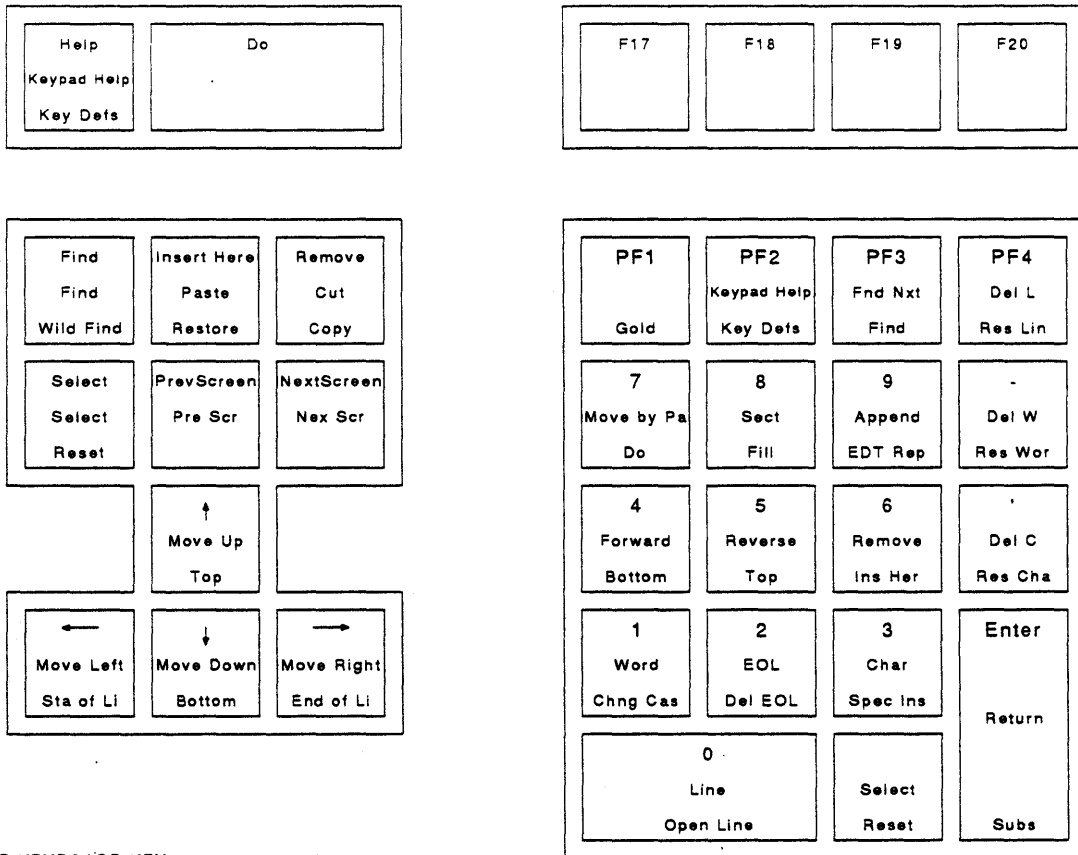
SAMPLE KEYPAD OR FUNCTION KEY



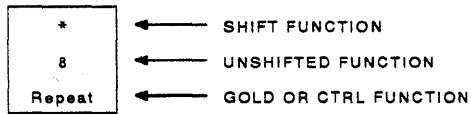
WARNING: DEFINING CTRL/X HAS UNPREDICTABLE RESULTS

Figure 3-6: EDT-Like Key Definitions for VT200-Series Terminals

Figure 3-6: EDT-Like Key Definitions for VT200-Series Terminals (Cont)

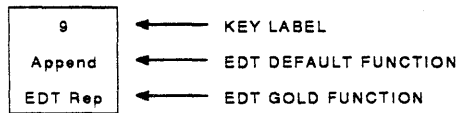


SAMPLE KEYBOARD KEY



WARNING: DEFINING CTRL/X HAS UNPREDICTABLE RESULTS

SAMPLE KEYPAD OR FUNCTION KEY



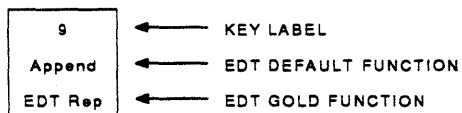
TTB_X0321_88

Figure 3-7: EDT-Like Key Definitions for VT100-Series Terminals (Cont)

PF1 Gold	PF2 Keypad Help Key Defs	PF3 Fnd Nxt Find	PF4 Del L Res Lin
7 Move by Pa Do	8 Sect Fill	9 Append EDT Rep	- Del W Res Wor
4 Forward Bottom	5 Reverse Top	6 Remove Ins Her	. Del C Res Cha
1 Word Chng Cas	2 EOL Del EOL	3 Char Spec Ins	Enter Return
0 Line Open Line		Select Reset	Subs

WARNING: DEFINING CTRL/X HAS UNPREDICTABLE RESULTS

SAMPLE KEYPAD OR FUNCTION KEY



TTB_X0324_88

Using EVE Help

- Line mode
 - Press <DO> (VT200) or <PF4> (VT100)
 - Type **HELP**
 - Enter the topic for which you want help
 - Press <RET> to exit help
- Keypad mode
 - Press <PF2> (VT100) or <HELP> (VT200)
 - Press key on which you want help
 - Press <RET> to exit

Ending an EVE Session

- Press <CTRL/Z> to exit and save modifications
- Press <DO> (VT200) or <PF4> (VT100)
 - Type **EXIT** at command prompt to save modifications, or
 - Type **QUIT** at command prompt to disregard modifications

EVE File Recovery

- Journaling facility is identical to EDT
 - Allows file recovery after a system interruption
 - Used to reproduce the editing session
 - Last few changes are not recovered
- Journal file
 - The default file name is the same as the input file name
 - The default file type is T~~J~~L
 - Contains keystrokes and editing commands of your current terminal session
- Command syntax:
 - * **\$ EDIT/TPU/RECOVER file-name** !!!
- Specify the original file type (not T~~J~~L)
- Example:

```
$ EDIT/TPU/RECOVER MYFILE.TEXT
```

SUMMARY

There are two editors available on a VMS system: EDT (the default editor) and EVE. Two editing modes are available with EDT:

- Line mode
- Keypad mode

EVE features include:

- Keypad editing
- Insert and Overstrike modes for text entry
- Automatic word wrap
- Multiple windows

The EDT Editor is invoked by:

\$ EDIT file-name

The EVE Editor is invoked by:

\$ EDIT/TPU file-name

HELP is available within both editors.

An EDT editing session is ended by pressing <CTRL/Z> then entering **EXIT** or **QUIT**, or by pressing <PF1> then keypad key 7 and entering **EXIT** or **QUIT**.

An EVE editing session is ended by pressing <CTRL/Z> or by pressing either <DO> or <PF4> then entering **EXIT** or **QUIT**.

APPENDIX A—EDT LINE-MODE EDITING

EDT's line editing facility can be used with any interactive terminal. Line editing uses the line as its point of reference. Line editing commands are useful for manipulating large blocks of text.

To aid in locating and editing text, EDT assigns line numbers. These line numbers are not part of the text and are not kept in the file when you finish an editing session.

The following commands deal with line-mode editing.

Inserting Text

Use the **INSERT** command to insert text. The cursor indents 16 spaces and waits for the text to be inserted. You can enter as many lines as you wish.

Example:

```
$ EDIT INSERT.FYI
1      This is line 1
*
2      This is line 2
*
3      This is line 3
*
4      etc.
*
[EOB]

*INSERT_2
This is the new text which is being typed
in the file. It is being inserted prior to
line 2
*EXIT
```

```
$ EDIT INSERT.FYI
1      This is line 1
*
2      This is the new text which is being typed
*
3      in the file. It is being inserted prior to
*
4      line 2
*
5      This is line 2
*
6      This is line 3
*
7      etc.
*
[EOB]

*QUIT
```

Substituting Text

Use either the **SUBSTITUTE** or **SUBSTITUTE NEXT** command to substitute strings of text.

The **SUBSTITUTE** command operates on the current line or on a specified range within the buffer.

Syntax:

***SUBSTITUTE/old-string/new-string**

Example:

```
*SUBSTITUTE/FORTRAN/PASCAL
```

To substitute a string throughout the complete buffer, use the **WHOLE** parameter in conjunction with the **SUBSTITUTE** command:

Example:

```
*SUBSTITUTE/FORTRAN/PASCAL/WHOLE
```

*The **SUBSTITUTE NEXT** command operates on the next occurrence of the specified string within the buffer.

Syntax:

***SUBSTITUTE NEXT/old-string/new-string**

Example:

```
*SUBSTITUTE NEXT/FORTRAN/PASCAL
```

Moving Text from One Location to Another

Use either the **MOVE** or **COPY** command to move one or more lines of text from one location to another.

Note that the **MOVE** command deletes the text from the original location, whereas the **COPY** command does not delete the text.

Syntax:

***MOVE first-range TO second-range**

In the following example, lines 20 through 30 are moved above line 10. Note that "second-range" always refers to a single line.

```
*MOVE 20 THRU 30 TO 10
```

CAN ALSO USE ":" FOR "THRU"

To move the current line, enter:

```
*MOVE TO 15
```

To move text without deleting the text in the original position, use the **COPY** command.

Syntax:

***COPY first-range TO second-range**

In the following example, lines 35 through 43 are moved above line 7.

```
*COPY 35 THRU 43 TO 7
```

Deleting Text

The **DELETE** command deletes lines or group of lines.

Syntax:

***DELETE range**

The following example shows how to delete line 2 in a file.

```
$ EDIT MYFILE.TXT
1  This is Line 1
* TYPE WHOLE
2  This is line 2
3  This is Line 3
4  This is Line 4
* DELETE 2
1 line deleted
3  This is Line 3
* TYPE WHOLE
1  This is Line 1
3  This is Line 3
4  This is Line 4
```

One of EDT's features is the ability to edit more than one file. To accomplish this, you must understand the relationship between buffers and files.

The following section describes the use of buffers in EDT.

* Using Buffers in EDT

Buffers are temporary storage areas for text. Buffers enable you to:

- Divide one or more files into sections.
- Move part or all of another file into your editing session.
- Create a file from part or all of the text in a buffer.

→ So ORIGINAL
IS PROTECTED
IN CASE YOU
QUIT!

When an editing session has started, a buffer called MAIN is automatically provided by EDT. The *MAIN* buffer serves as the work area for you.

How to Create Buffers

- Press the <GOLD> key, followed by the **COMMAND** function.
- At the *Command:* prompt, enter
 - The **FIND** command
 - An equal sign (=)
 - Buffer name of your choice (buffer names must begin with a letter)

Example:

Command: FIND=PASCAL

PASCAL (works same as MAIN!)

You can now insert and edit text ~~as if you were~~ in the *MAIN* buffer.

To return to the *MAIN* buffer, enter:

Command: FIND=MAIN

*ALL EXECUTE
IN LINE MODE
OF AN EDITING SESS-
* (END IN \$ DEL!)

* Copying Text from One Buffer to Another

The **COPY** command is used to copy the contents of one buffer into another buffer. In the following example, the contents of the buffer *MAIN* are copied into the buffer *FORTRAN*. The current buffer becomes *FORTRAN*.

```
*COPY =MAIN TO =FORTRAN
```

* Copying Text from a File into a Buffer

The **INCLUDE** command copies a file (outside of EDT) into a buffer. In this example, a file named *MYFILE.TXT* is copied into the *MAIN* buffer.

```
*INCLUDE MYFILE.TXT =MAIN
```

* Copying Text from a Buffer into a File

The **WRITE** command copies text from your current editing session into a file of your choice. In this example, the entire contents of the current buffer are written to a file named *MYFILE.FOR*.

```
*WRITE MYFILE.FOR
```

* Deleting Buffers

Use the line-mode command **CLEAR** to delete buffers during an editing session. In the following example, the buffer *PASCAL* is deleted.

```
*CLEAR PASCAL
```

IF YOU LEAVE THE EDITOR
YOU WILL BE PROMPTED TO
WRITE THE BUFFER TO A FILE
ELSE IT IS DISCARDED.

APPENDIX B—EVE

The following EVE commands are discussed in addition to the commands included in the previous EVE section of the module.

Inserting Text

You can insert:

- Text – Characters will be inserted into the buffer at the current cursor position by typing in the text.
- *• Files – Files can be inserted using the **INCLUDE FILE** command. The contents of the specified file are inserted into the buffer at the line before the current cursor location.

Syntax:

* **INCLUDE FILE [file-name]**

- Special Nonprinting Characters – Press <CTRL/V> and then press the special nonprinting character.

Example of inserting a form feed into the buffer:

* Press <CTRL/V>

* Press <CTRL/L> – form feed!

Locating Text

Syntax:

* **FIND search-string**

The <FIND> key is used to locate specified text strings.

If the search string is in lowercase characters, EVE disregards the case of letters and locates any occurrence of the string.

If the search string contains one or more uppercase letters, EVE locates only the occurrences of the string that match the search string exactly.

If a search string is found, EVE displays the string in highlighted type. EVE also defines the search string as a select range. While the search string is highlighted, you can perform any operation on it that requires a select range.

* **Marking Locations in Text**

The **MARK** and **GO TO** commands are used when you are editing a large file and wish to return to a specific location later on during an editing session.

Press <DO> and enter **MARK label-name**. The label name can be one or more alphanumeric characters.

Press <DO> and enter **GO TO label-name** to move the cursor to the marked location.

* Replacing Text

The **REPLACE** command allows you to replace a text string in the current buffer with another text string.

Format:

- Press <DO>
- Enter **REPLACE** (like **ERT** SUBSTITUTE)
- Enter the string to be replaced following the "Old string" prompt
- enter the new string at the "New string" prompt

EVE moves the cursor to the first occurrence of the string, and prompts: *Replace? Type yes,no,all,last or quit.* The following table lists the response and the action by EVE.

Table 3-8: Responding to REPLACE Prompts

Response	EVE's Action
YES	Replaces the string and attempts to locate another occurrence of the string in the current direction.
NO	Does not replace the string, and attempts to locate another occurrence of the string in the current direction.
ALL	Replaces the string and all other occurrences of the string in the current direction. The cursor is moved to the position where the last replacement occurred.
LAST	Replaces this occurrence of the string and stops the REPLACE procedure. The cursor does not move.
QUIT	Does not replace this occurrence of the string and stops the REPLACE procedure. The cursor does not move.

Restoring Text

The **RESTORE** command restores (or undeletes) the last word, sentence, or line erased by any of the following:

- Any EVE command or any key bound to an EVE command
- The key
- Also includes the <PF4>, <MINUS>, <KP6>, <COMMA>, and <KP2> keys when the keypad is set to EDT

RESTORE CHARACTER

The **RESTORE CHARACTER** command restores (or undeletes) the character last erased by any of the following:

- The **ERASE CHARACTER** command or any key bound to that command
- The key
- Also includes the <COMMA> key when the keypad is set to EDT

RESTORE LINE

The **RESTORE LINE** command restores (or undeletes) the line last erased by any of the following:

- The EVE commands **ERASE LINE**, **ERASE START OF LINE**, or any key bound to these commands
- The <GOLD/DEL> key sequence on a VT200 terminal
- Also includes the <PF4> or <GOLD/KP2> keys when the keypad is set to EDT

RESTORE WORD

The **RESTORE WORD** command restores (or undeletes) the word last erased by any of the following:

- The **ERASE WORD** command or any key bound to that command
- Also includes the <MINUS> key when the keypad is set to EDT

* Using Buffers in EVE

Buffers are used during an editing session as storage areas. The following table describes the commands that are used to create and manipulate buffers.

Table 3-9: Creating and Manipulating Buffers

Command	Function
BUFFER	Puts the specified buffer in the current window and moves the cursor to the last location it occupied in that buffer. Creates a new buffer if the specified buffer does not exist.
GET FILE	Creates a new buffer containing the text of the specified file, places the new buffer in the current window, and places the cursor at the beginning of the new buffer. If a file is specified that does not exist, an empty buffer is created.
SHOW	Displays a screen of information about the current buffer. If more than one buffer is active in the editing session, press the DO key to display information about the other buffers.
WRITE FILE	Writes the contents of the current buffer to a file. If a file name is not specified, EVE uses the buffer name as the file name.

Using Multiple Buffers

Use multiple buffers when you want to edit more than one file. This is very useful if you want to move text from one file to another file.

To create a new buffer

- Press <DO>
- Enter **GET FILE file-name**

To change the buffer in the current window

- Press <DO>
- Enter **BUFFER buffer-name**

When you exit from using multiple buffers, EVE writes the contents of the current buffer to a file and asks if you want to write the other buffer to a file.

*Using Multiple Windows

EVE allows you to view multiple windows on your terminal screen at the same time. You can view and edit either two sections of the same buffer (one file) or multiple buffers (multiple files) simultaneously.

The following table lists commands that are used to create and manipulate windows.

Table 3-10: Creating and Manipulating Windows

Command	Function
TWO WINDOWS	Splits the terminal screen and creates two editing windows, moving the cursor to the last position it occupied in the text of the bottom window.
OTHER WINDOW	Moves the cursor to the last position it occupied in the other window.
ONE WINDOW	Removes the other window from the screen, expanding the current window to occupy the complete screen.
GET FILE	Creates a new buffer containing the text of the specified file, places the new buffer in the current window, and places the cursor at the beginning of the new buffer. If a nonexistent file is specified, an empty buffer is created. After you create two windows on your terminal screen, use the GET FILE command to create a new buffer in one of the windows.
BUFFER	Puts a new buffer in the current window, and moves the cursor to the last position it occupied in the buffer. Creates a new buffer if the specified buffer does not exist. After you create two windows on your terminal screen, use the BUFFER command to put a different buffer in one of the windows.

DELETE WINDOW

The **DELETE WINDOW** command deletes the window in which the cursor is located, if you are using more than one window. Be aware that any edits or modifications made to the file in the current window will not be saved.

ENLARGE WINDOW

Syntax:

ENLARGE WINDOW *integer*

This command enlarges the window in which the cursor is located by the number of lines specified. EVE shrinks the other windows on the screen accordingly.

Integer is the number of lines you want to add to the current window. The minimum value is 1. The maximum value is 20 for a VT100 or VT200 screen.

NEXT WINDOW

The **NEXT WINDOW** command moves the cursor from the current window to the window below. If the cursor is already in the bottom window, EVE moves the cursor to the top window.

PREVIOUS WINDOW

The **PREVIOUS WINDOW** command moves the cursor from the current window to the window above. If the cursor is already in the top window, EVE moves the cursor to the bottom window.

SHRINK WINDOW

Syntax:

SHRINK WINDOW integer

The **SHRINK WINDOW** command reduces the size of the window the cursor is currently in by the number of lines specified. EVE enlarges the other windows accordingly.

Integer is the number of lines by which you want to shrink the window. The minimum value is 1. The maximum value is 9, which is the number of lines by which you can shrink a window if you have only two windows on the screen.

SPLIT WINDOW

Syntax:

SPLIT WINDOW integer

The **SPLIT WINDOW** command splits the window in which the cursor is located.

Integer is the number of smaller windows that you want to appear on the terminal screen. If you omit the integer, the current window is replaced with two windows.

* DEFINING KEYS

You can define keys to execute frequently used EVE commands. You may also save key definitions to be used from one editing session to the next.

EVE does not allow you to define

- The <DO> key ✓
- The <RET> key ✓
- The space bar ✓
- All printing characters on the main keyboard ✓

DIGITAL recommends that you do not define the following keys and control key sequences

- <DELETE>
- <F6> (VT200 series)
- <HELP> (VT200 series) <PF2> (VT100 series)
- <CTRL/C>
- <CTRL/R>
- <CTRL/S>
- <CTRL/T>
- <CTRL/U>
- <CTRL/Q>
- <CTRL/X>
- <CTRL/Y>

* To define a key !

- Press <DO>
- Enter **DEFINE KEY**
- Type the key to be associated with the EVE command
- A message, *Key defined*, appears if you have successfully defined a key

* Saving Key Definitions

The **SAVE EXTENDED TPU** command saves all key definitions in a section file that you specify. This command must be executed before ending an editing session.

Format:

* **SAVE EXTENDED TPU device:[directory]file-name.TPU\$SECTION**

You should include the device, directory, and file name that you choose. The section must be TPU\$SECTION.

If you specify the same file name each time you execute the **SAVE EXTENDED TPU** command, all key definitions will accumulate in the same file from all editing sessions.

* Using Key Definitions

To use this extended version of EVE, you must include the **/SECTION** qualifier when invoking EVE.

Syntax:

* **EDIT/TPU/SECTION=device:[directory]file-name.TPU\$SECTION file-name**

Example:

```
$ EDIT/TPU/SECTION=DISK:[SMITH]EVEDEFS.TPU$SECTION MYFILE.TXT
```

* Checking Spelling Errors

The **SPELL** command checks for spelling errors in a selected text or buffer if your system contains VAX-11 DECSpell.

If you do not select any text, **SPELL** checks the entire current buffer, and replaces the misspelled words with correct words.

WE DON'T HAVE!

MODULE 4

COMMUNICATING WITH OTHER USERS

INTRODUCTION

The ability to communicate with users, both on your system and on other VMS systems is invaluable. There are two VMS utilities and a DCL command that allow you to do this. They are:

- The Mail utility (**MAIL**), which allows you to send and receive messages to other users.
- * The Phone utility (**PHONE**) for communicating interactively with other users that are logged in.
- * The DCL command **REQUEST**, which displays a message sent by you at a system operator's terminal, and optionally, requests a reply.

OBJECTIVES

To effectively communicate with other users, you should be able to:

- Send and receive messages between users.
- Print and delete mail messages.
- * • Organize mail messages by using mail files.
- Place and answer calls using the Phone utility.
- * • Send messages to a system operator using the **REQUEST** command.

RESOURCES

- *VMS Mail Utility Manual*
- *VMS Phone Utility Manual*
- *VMS DCL Dictionary*

INVOKING AND OBTAINING HELP FROM THE MAIL AND PHONE UTILITIES

- Enter the command that invokes the utility
 - The name of the utility (**MAIL** or **PHONE**)
 - The screen prompt is the name of the utility (**MAIL>** or **PHONE>**)
 - Enter the **HELP** command at the utility's prompt
- >NOT REALLY, ACTUALLY %*
- `MAIL> HELP`
- Follow the online **HELP** instructions
 - To exit the utility, enter
 - The **EXIT** command at the utility's prompt, or
 - The **<CTRL/Z>** key sequence, which brings the utility's prompt to the screen, followed by another **<CTRL/Z>** key sequence

Example 4-1: Getting Help for MAIL Utility Commands

MAIL> HELP

HELP

Allows you to obtain information about the MAIL Utility.

To obtain information about all of the MAIL commands, enter the following command:

MAIL> HELP *

To obtain information about individual commands or topics, enter HELP followed by the command or topic name.

Format:

HELP [topic]

Additional information available:

/EDIT	/PERSONAL_NAME	/SELF	/SUBJECT	ANSWER	ATTACH
BACK	COMPRESS COPY	CURRENT	DEFINE	DELETE	DIRECTORY
EDIT	ERASE EXIT	EXTRACT	FILE	FIRST	Folders
FORWARD	GETTING_STARTED	HELP	KEYPAD	LAST	MAIL
MARK	MOVE NEXT	PRINT	PURGE	QUIT	READ
REMOVE	REPLY SEARCH	SELECT	SEND	SET-SHOW	SPAWN
V5_CHANGES					

Topic? READ

THE MAIL UTILITY

- Allows you to send messages to and receive messages from other users, both on your system or within a network.

Organization of Mail Messages

- By default, a file named MAIL.MAI stores mail messages. This file is automatically created for you by the system.

- MAIL organizes messages in folders. Three of these folders are named:

— NEWMAIL – Contains new messages you have not yet read.

— MAIL – Contains old messages you have already looked at.

— WASTEBASKET – Contains messages marked for deletion. (LIKE TRASH ON THE MAC)

The Wastebasket folder is emptied automatically when you exit from MAIL.

Using the MAIL Utility

- Use the **MAIL** command to invoke the utility

```
$ MAIL<RET>  
MAIL>
```

- Enter MAIL commands to
 - Read messages you have received
 - Send messages to other users
 - Obtain a list of mail messages you have received
 - Delete old mail messages
 - Learn about other MAIL commands

*Reading a Message

When you log in to the system, you are notified of any new mail messages.

If you are currently logged in, the Mail utility displays an informational message on your terminal screen.

- To read the first new message
 - Invoke the Mail utility
 - Press <RET> at the *MAIL*> prompt

- * • To read a message you receive while using the Mail utility
 - Enter the **READ/NEW** command

```
MAIL> READ/NEW
```

- The folder you are currently in is displayed in the upper right-hand corner of the screen

Example 4-2: Reading a Mail Message

```
$ MAIL
```

```
You have 1 new message.
```

```
MAIL>
```

```
#1          12-DEC-1984 09:19:25          NEWMAIL
From:      SPEEDY::JIM
To:        SMITH
Subj:      Status meeting
```

```
John,
```

```
I will be out of town so I will not be able to attend the status
meeting.  Fill me in when I get back.
```

```
Jim
```

```
MAIL>
```



Table 4-1: MAIL Commands Used to Read a Mail Message

Operation	Format/Example	Comment
Displaying the contents of a message in the current folder of the current file	READ n	Displays the message associated with the message number (n)
	READ<RET>	Displays the next page of the current message having the next-highest message number
	READ/NEW	Displays new messages that arrived while you are in MAIL,
	NEXT	Displays the first page of the message having the next-highest message number
	FIRST	Displays the contents of the message having the lowest message number
	LAST	Displays the contents of the message having the highest message number
	CURRENT	Displays the contents of the current (last-read) message
	BACK	Displays the contents of the message preceding the current message

Sending a Message

- Using the **SEND** command
 - At the *MAIL*> prompt, enter the **SEND** command
 - Enter the node (if different from your node) and user name
 - Enter the subject of the message
 - Enter the message

Press <RET> at the end of each line
Press <CTRL/Z> after the last line
Press <CTRL/C> to cancel the message

Example 4-3: Sending a Mail Message

```
$ MAIL
MAIL> SEND
To:      SMITH
Subj:    Department Meeting
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:

Jim,

There will be a department meeting on Friday at 9:00 am.
Please attend if at all possible.
John
CTRL/Z
MAIL>
```

* **Table 4-2: MAIL Commands Used to Send Messages**

Operation	Format/Example	Comment
Routing a message or the contents of a file to a user or group of users	<p>MAIL> SEND [/qualifier] [file-specification]</p> <p>MAIL> SEND * To: JONES, ALAN Subj: Today's Agenda</p> <p>MAIL> SEND/EDIT ! * (To: @DISTRIBUTION.DIS Subj: Today's Agenda</p> <p>MAIL> SEND MYFILE.LIS * (To: JONES Subj: Today's Agenda</p>	<p>Routes the contents of your message to each user listed after the <i>To:</i> prompt.</p> <p>Routes the contents of your message to each user listed in the file named <u>DISTRIBUTION.DIS</u>. You enter the message by using the EDT editor. The message is sent when you exit the editor.</p> <p>Routes the contents of the file MYFILE.LIS to the mail file of the user JONES.</p>
Routing a copy of the current (last-read) message to a user or group of users	<p>* MAIL> FORWARD To: JONES Subj: Good News!</p>	<p>Routes a copy of the current message to each user listed after the <i>To:</i> prompt.</p>
Routing a message or the contents of a file to the sender of the current (last-read) message	<p>* ANSWER or REPLY [/qualifier] [file-specification]</p> <p>MAIL> REPLY Subj: You're Right!</p>	<p>The REPLY or REPLY/EDIT command routes your message to the sender of the current message. (REPLY and ANSWER are synonyms.)</p>

EXAMPLE OF "DISTRIBUTION.DIS"

ADDE1::USER1
 ADDE2::USER2

← only one per line

* Displaying a List of Messages

- The **DIRECTORY** command displays a numbered list of your mail messages.

```
MAIL> DIRECTORY
```

- To read an old message
 - Enter the **DIRECTORY** command
 - Enter the desired message number

Example 4-4: Listing Messages and Reading Old Messages

```
MAIL> DIRECTORY
```

#	From	Date	Subject	MAIL
1	SPEEDY::JIM	12-DEC-1984	Status meeting	
2	SPEEDY::SMITH	12-DEC-1984	Schedule of meetings	
3	SPEEDY::JONES	12-DEC-1984	Party	

```
MAIL> 3
```

```
#3      12-DEC-1984 09:22:53      MAIL
From:   SPEEDY::JONES
To:     SMITH
Subj:   Party
```

John,

We're having an office party next Thursday.
Would you like to come?

Tom

```
MAIL>
```

*Deleting a Message

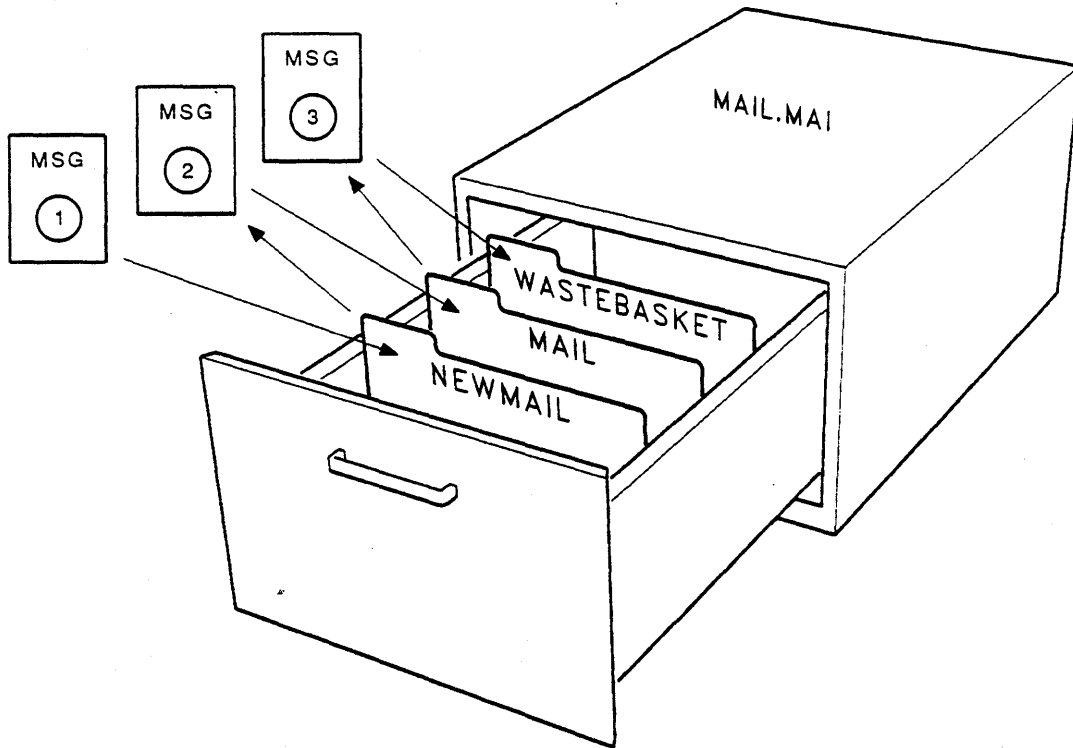
- The **DELETE** command moves a message to the Wastebasket folder. The message is not deleted until you exit Mail.
- Either a single message or a range of messages can be deleted.
- If a message number is omitted, this command marks the message you are currently reading for deletion.

```
MAIL> DELETE 3 (Deletes message number 3)
```

```
MAIL> DELETE 1,3,5-7 (Deletes message numbers 1, 3, 5, 6, 7)
```

- *• Deleted messages may be recovered from the Wastebasket folder by using the **MOVE** command.

Figure 4-1: The Relationship Between a Mail Message, Folder, and File



TTB_X0325_88

Table 4-3: MAIL Commands Used to Maintain Messages

Operation	Example	Comments
Displaying a list of folders	DIRECTORY <FOLDERNAME >	Displays a list of all folders in the current mail file.
Displaying a list of messages	DIRECTORY CALENDAR	Produces a list of all the messages in the folder. Each message contains a message number.
Moving between folders	SELECT CALENDAR	Moves you between folders of your choice.
Filing a message	MOVE CALENDAR	Moves the current message to the folder named CALENDAR.
Copying a message to a file	EXTRACT DWAYNE.TXT	Places a copy of the current message into a sequential file with the file name specified.
Printing a message	PRINT	Places a copy of the current message into the default queue for printing.
Emptying the Wastebasket folder	PURGE or EXIT	Discards all messages in the Wastebasket folder.

* FOLDER EXISTS ONLY IF IT HAS MESSAGE(S) IN IT!

*** MORE INFO IN GENERAL USER'S MANUAL -!

Exiting from the Mail Utility

- Enter the **EXIT** command, or
- Press <CTRL/Z> at the *MAIL*> prompt

* THE PHONE UTILITY

You can use the Phone utility to contact another user by:

- Entering the name of a user as a parameter to the **PHONE** command.

```
$ PHONE HARKINS
```

- Entering the name of a user from within the Phone utility.

```
$ PHONE  
% DIAL HARKINS
```

The system responds to your request by displaying a repeating message on the terminal of user HARKINS. To respond to this message, HARKINS must:

- Enter the **PHONE** command, then
- Enter the **ANSWER** command, or
- Enter the **REJECT** command at the % prompt

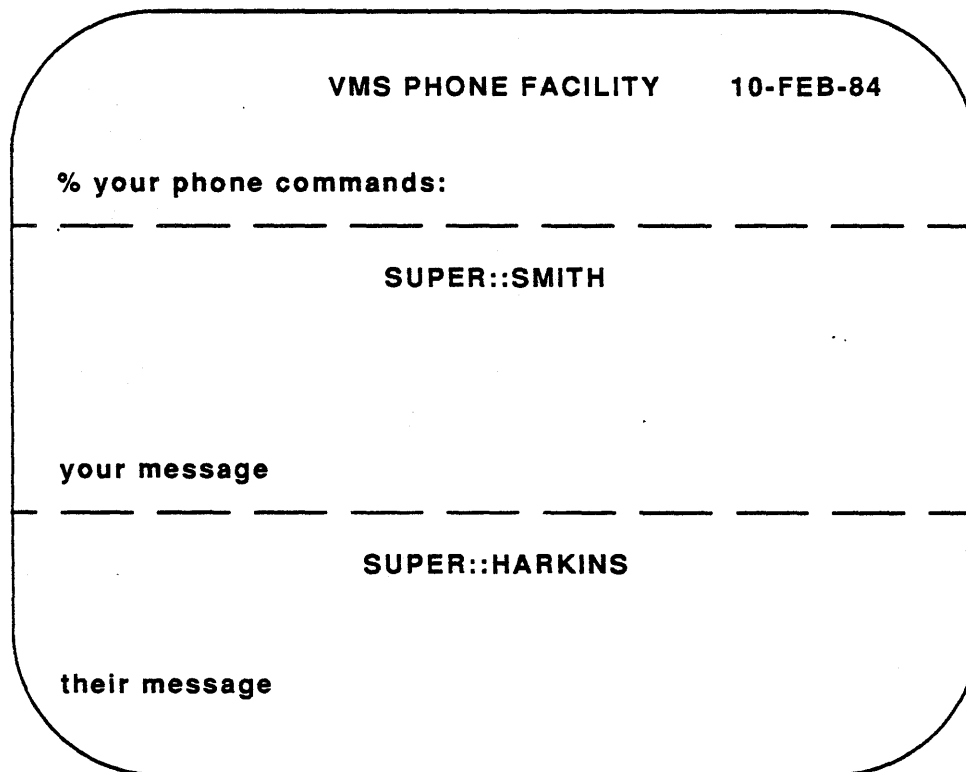
To enter a command while having a conversation, type the percent sign (%) followed by the command.

**SENDS A CHARACTER AT A TIME!
(CAN SEE EDITING OF SENT)*

The Phone Help Facility

- To display a list of available help topics, enter the **HELP** command.
- To display a particular help text, enter the command **HELP help-option**.
- After you receive the help display, type any character to refresh the Phone split screen.
- Entering either the **EXIT** command or pressing <CTRL/Z> returns you to DCL command level. (Remember, to enter the **EXIT** command during a conversation, you must first type a percent sign.)

Figure 4-2: Using the Phone Utility



TTB_X0326_88

Table 4-4: Phone Commands Commonly Used to Create or Reject a Terminal Link

Operation	Format/Example	Comment
Choosing who to call	DIRECTORY [node[::]]	Displays a list of people with whom you could talk on your system or any other system in the network.
Requesting a terminal link	DIAL user-name DIAL JONES	Displays a repeating message on the terminal JONES is logged in to, that signifies a terminal link request.
Accepting a terminal link	ANSWER	Links your terminal to the terminal of the caller. The split screen is displayed.
Rejecting a terminal link	REJECT	Terminates the repeating message caused by the DIAL command entered by another user.
Placing others on hold	HOLD	Places all other terminals in the conversation on HOLD.
Reversing the hold	UNHOLD	Reverses your previously entered HOLD command.
Terminating a terminal link	HANGUP	Terminates the links of all terminals in the conversation.
Leaving the Phone utility	EXIT <CTRL/Z>	Exits the Phone utility by first executing an automatic HANGUP command.

COMMUNICATING WITH OPERATORS

* The REQUEST Command

There may be times when you wish to communicate with a system operator. For example, you may need to have a magnetic tape mounted so you can access it.

The **REQUEST** command displays a message at a system operator's terminal and optionally requests a reply.

- Sending a message without a reply

\$ REQUEST "message-text"

```
$ REQUEST "Please mount magtape 4 on MTA0:"
```

- Sending a message that expects an operator reply

\$ REQUEST/REPLY "message-text"

```
$ REQUEST/REPLY "Please mount magtape 4 on MTA0"
```

Example 4-5: Using the REQUEST/REPLY Command

```
$ REQUEST/REPLY "Please mount magtape 4 on MTA0"  
%OPCOM-S-OPRNOTIF, operator notified waiting... 11:23:02.92  
%OPCOM-S-OPREPLY, AFTER 11:30  
13-MAR-1985 11:26:03.87, request 7 completed by operator OPA0  
$
```

SUMMARY

The Mail Utility

The Mail utility allows you to send to and receive messages from other users, both on your system and within a network. The Mail utility is invoked by entering **MAIL** at the DCL prompt.

- To read a new message, press <RET> at the *MAIL*> prompt.
- To read a message received while you are in the Mail utility, enter the **READ/NEW** command.
- To send a message, enter the **SEND** command at the *MAIL*> prompt.
 - Enter the node (if different from your node) and user name
 - Enter the subject of the message
 - Enter the message
 - Press <CTRL/Z> after the last line of the message

The Phone Utility

You can use the Phone utility to contact another user by:

- Entering the name of a user as a parameter to the **PHONE** command.
- Entering the name of a user from within the Phone utility.

To enter a command while having a conversation, type the percent sign (%) followed by the command.

The REQUEST Command

The **REQUEST** command displays a message at a system operator's terminal and optionally requests a reply.

```
REQUEST "message-text"
```

```
REQUEST/REPLY "message-text"
```

MODULE 5

MANAGING FILES

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.

OBJECTIVES

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

- Locate files stored on disks.
- Locate directories in directory trees.
- Add and remove files from a directory.
- Display contents of files.
- Protect files from being accessed by unauthorized users.

RESOURCES

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

FILE SPECIFICATIONS

- A file is a logically related collection of records.
- Use a file specification to identify a file you may wish to access.

Table 5-1 illustrates the syntax of a file specification.

Table 5-1: Syntax of a Local Disk File Specification

** \$ DCL-COMMAND DBAO: [SMITH] MYFILE .DAT; 7*

can ALSO USE % PERIOD!

Name	Reference	Rules of Naming	Example
Device	Storage device name	1 to 255 characters	DBAO:
Directory	Catalogue 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 <i>(CAN BE ANYTHING)</i>
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 (\$)

NODES: DEVICE: [DIRECTORY . SUBDIRECTORY ...] FN . FT; VERSION#

DEFAULTS FOR FILE SPECIFICATIONS

- Each part of a file specification is called a *field*
- You can omit fields and allow the system to supply defaults for those fields
- To override the default value for a field, supply a value for that field
- The following table summarizes the defaults used by the system

Table 5-2: File Specification Defaults

Part of File Specification	Default
Device	Device established at login by the system manager or specified by the last <u>SET DEFAULT</u> command
Directory	Directory established at login by the system manager or specified by the last <u>SET DEFAULT</u> command
Name	None
Type	Depends on the DCL command
Version	The highest version number

Also, SHOW DEFAULT

DEVICE SPECIFICATIONS - 3 TYPES

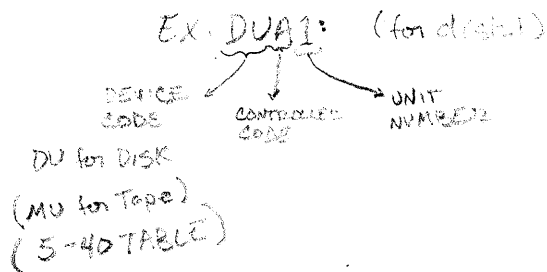
- * • Logical device name
 - Synonym for a physical device name (EX. DISK1 to represent DUA1)
 - Established by the system manager
- Physical device name
 - Refers to a specific physical system device
 - Device code
 - Controller character (Optional)
 - Unit number (Optional)
- Generic device name
 - Specifies a group of devices

Table 5-3: Naming a Device

Device Specification	Function	Value	Default Value
<u>Device type code</u>	Identifies device type	2-13 characters	None
<u>Controller character</u>	Names controller to which device is attached	One or more of the characters A-Z	A
<u>Unit number</u>	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 for further information regarding devices.



DIRECTORY STRUCTURE

→ ROOT OF DIRECTORY TREE

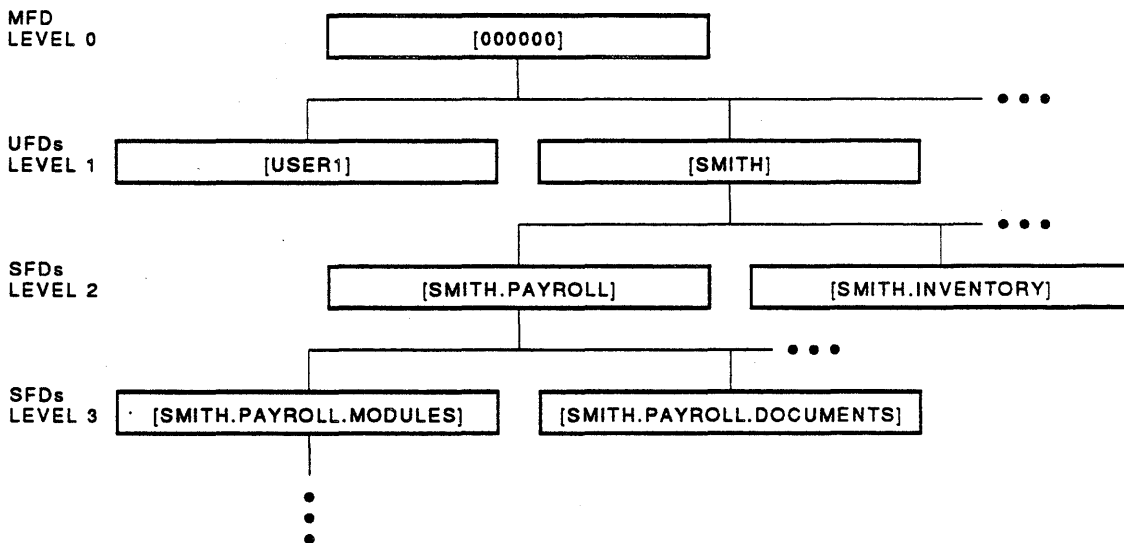
- * • Each disk contains a Master File Directory (MFD) that catalogs all User File Directories (UFDs). → DELETE THIS AND YOUR SCREWED!
- A *directory* is a file that catalogs another set of files on a disk. Each UFD contains the name of each file and pointers to where the file is located on the disk.
- Directory files have a file type of DIR.
- Users can further subdivide their files by creating *subdirectories*. Subdirectories will be discussed later in this module.

DIRECTORY NAMES IN THE HIERARCHY

Table 5-4: Directory Names

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.

Figure 5-1: Naming Directories



TTB_X0327_88

The names given in the rectangles are directory names.

FILE MANIPULATION COMMANDS

The following table lists examples of some DCL commands used to manipulate files. There are many qualifiers that modify the action of these commands. Refer to the VMS DCL Dictionary for more details.

Table 5-5: File Manipulation Commands

Operation	Comments
Creating a file without using a text editor	The CREATE command creates a new file without using a text editor.
<pre>\$ CREATE MYFILE.TXT</pre>	
Copying a file	The COPY command creates a new file from an old file.
<pre>\$ COPY OLDFILE.TXT NEWFILE.TXT</pre>	
Changing an existing file name to new file name	The RENAME command changes all or part of an existing file name.
<pre>\$ RENAME OLDFILE.TXT NEWFILE.TXT</pre>	
Removing a file	The DELETE command removes a file. A specific version number must be used to remove a file.
<pre>\$ DELETE MYFILE.TXT;2</pre>	
Removing files on an interactive basis	The /CONFIRM qualifier initiates a system prompt to confirm whether or not the file should be deleted. A "Y" response deletes the file; an "N" response does not delete the file.
<pre>*\$ DELETE/CONFIRM MYFILE.TXT;*</pre>	
(System prompts):	
DISK: [SMITH]MYFILE.TXT;3 delete? [N]:	
DISK: [SMITH]MYFILE.TXT;2 delete? [N]:	
DISK: [SMITH]MYFILE.TXT;1 delete? [N]:	

Table 5-6: Manipulating Files in a Directory

Operation	Comments
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	
\$ PURGE MYFILE.TXT	The PURGE file-name command removes all but the highest-numbered version of a particular file.
Displaying the contents of a file at your terminal	The TYPE command displays the contents of a file at your terminal. Note that only ASCII files may be displayed.
\$ TYPE MYFILE.TXT	<i>also → PURGE /KEEP=# *.FT</i>
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.
\$ APPEND OLDFILE.TXT NEWFILE.TXT	<i>also: APPEND /NEW-VERSION FN.FT, FN.FT FN.FT ① + ② ⇒ ③</i>
Searching files for all occurrences of the specified search-string(s)	* <u>The SEARCH command searches one or more files for the specified string(s). The search string "March 13" must be enclosed in quotation marks because it contains a space character.</u>
* \$ 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. This listing goes to your terminal by default.
* \$ DIFFERENCES MYFILE.TXT YOURFILE.TXT	<i>also SEARCH /OUTPUT = FN.FT</i>
Controlling the listing output from differences	* <u>The /OUTPUT qualifier tells the system to send the listing of differences to a file.</u>
* \$ DIFFERENCES/OUTPUT=DIFF.TXT MYFILE.TXT YOURFILE.TXT	

FINDING FILES AND DETERMINING THEIR CHARACTERISTICS

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

Table 5-7: Using the DIRECTORY Command to Determine the Characteristics of Files

Operation	Comments
Listing all files in your directory	The DIRECTORY command lists all files and information about them in your directory.
<p>\$ DIRECTORY</p>	
Checking for a unique file in your directory	<u>The file specification must be included to obtain information concerning a particular file in your directory.</u>
<p>* \$ DIRECTORY MYFILE.TXT</p>	
Obtaining all information about a particular file in your directory	<u>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.</u>
<p>* \$ DIRECTORY/FULL MYFILE.TXT or * \$ DIRECTORY/FULL</p>	
Determining the size of files in your directory	<u>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.</u>
<p>* \$ DIRECTORY/SIZE [file-specification] or * \$ DIRECTORY/SIZE=ALL [file-specification]</p>	
Determining the owner and protection of a file	<u>The /OWNER qualifier determines if the owner's UIC is displayed. The /PROTECTION qualifier determines if the protection of the file is displayed.</u>
<p>* \$ DIRECTORY/OWNER/PROTECTION MYFILE.TXT</p>	

Using Wildcards in File Specifications

Wildcards are used to

- Specify more than one file
- Abbreviate a file specification
- Match one or more characters in directory names, file names, or file types

Wildcards can be used in conjunction with each other or separately

Table 5–8: Wildcards Used to Specify File Names, Types, and Versions

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

like "?" in Java!

Example 5-1: Sample Directory File

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.

Table 5-9: 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 type of FOR and file name 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

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

ORGANIZING YOUR DIRECTORY STRUCTURE

- Files can be organized into *subdirectories*
- Reasons for grouping files into subdirectories are to
 - Organize the directory structure
 - Protect them from accidental modification or loss
 - Decrease the time for the system to find them
- * • Each UFD can have a maximum of seven levels of subdirectories below it
- 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

CREATING A SUBDIRECTORY

- * The command CREATE/DIRECTORY [directory.subdirectory] creates a subdirectory
- The subdirectory name must be enclosed in brackets
- The subdirectory name includes the directory name where it is created
- Each subdirectory name is separated by a period
- The directory or subdirectory itself is a file
 - The directory or subdirectory has a file type of DIR
 - Version number of file type DIR is 1
- Example:

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

!*
VERSION LIMIT = 11
(automatic pruning!)

- The directory [SMITH] is a UFD
- The subdirectory [.DOC] is the next level below the directory [SMITH]
- The file DOC.DIR;1 now resides in [SMITH]
- * The /LOG qualifier displays on your terminal the fact that the subdirectory was created
- To create another subdirectory beneath the [.DOC] subdirectory:

\$ CREATE/DIRECTORY [directory.subdirectory.subdirectory]

- Example:

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

- The subdirectory [.FORTRAN] is listed under the subdirectory [SMITH.DOC]
- The file FORTRAN.DIR now resides in [SMITH.DOC]

VERSION NUMBER FOR
A DIRECTORY FILE IS
ALWAYS 1, CREATED
THESE FILES.

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

- Syntax:

```
$ SET DEFAULT device-name:[directory-name]
```

or

```
$ SET DEFAULT [directory-name.subdirectory-name]
```

- Examples:

```
$ SET DEFAULT DISK2:[BORGERT] (Device name and directory name change)
```

```
$ SET DEFAULT [SMITH.DOC] (Device name remains the same)
```

USING THE SHOW DEFAULT COMMAND

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

```
*$ SHOW DEFAULT
```

```
DISK:[SMITH] (System response)
```

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

```
$ SHOW DEFAULT
```

```
DISK:[SMITH.DOC] (System response)
```

* MOVING WITHIN A DIRECTORY HIERARCHY

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

- * Hyphen (-) EX. DIR [-]
- Period (.) (can never have a lone, obviously! - ONLY ONE WAY TO GO UP 1 LEVEL, MANY WAYS DOWN!)
- ! * Ellipsis (...)

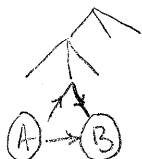
The hyphen and period characters are normally used in conjunction with the **SET DEFAULT** command to move from your current directory to another directory or subdirectory.

The ellipsis character, used with the **DIRECTORY** command, lists files in a directory and all subdirectories beneath it.

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 (<u>MUST be followed by a subdirectory name</u>)
! * ... (ellipsis)	Use current directory and all directories below it

EX. SET DEFAULT [-.NAMEB]

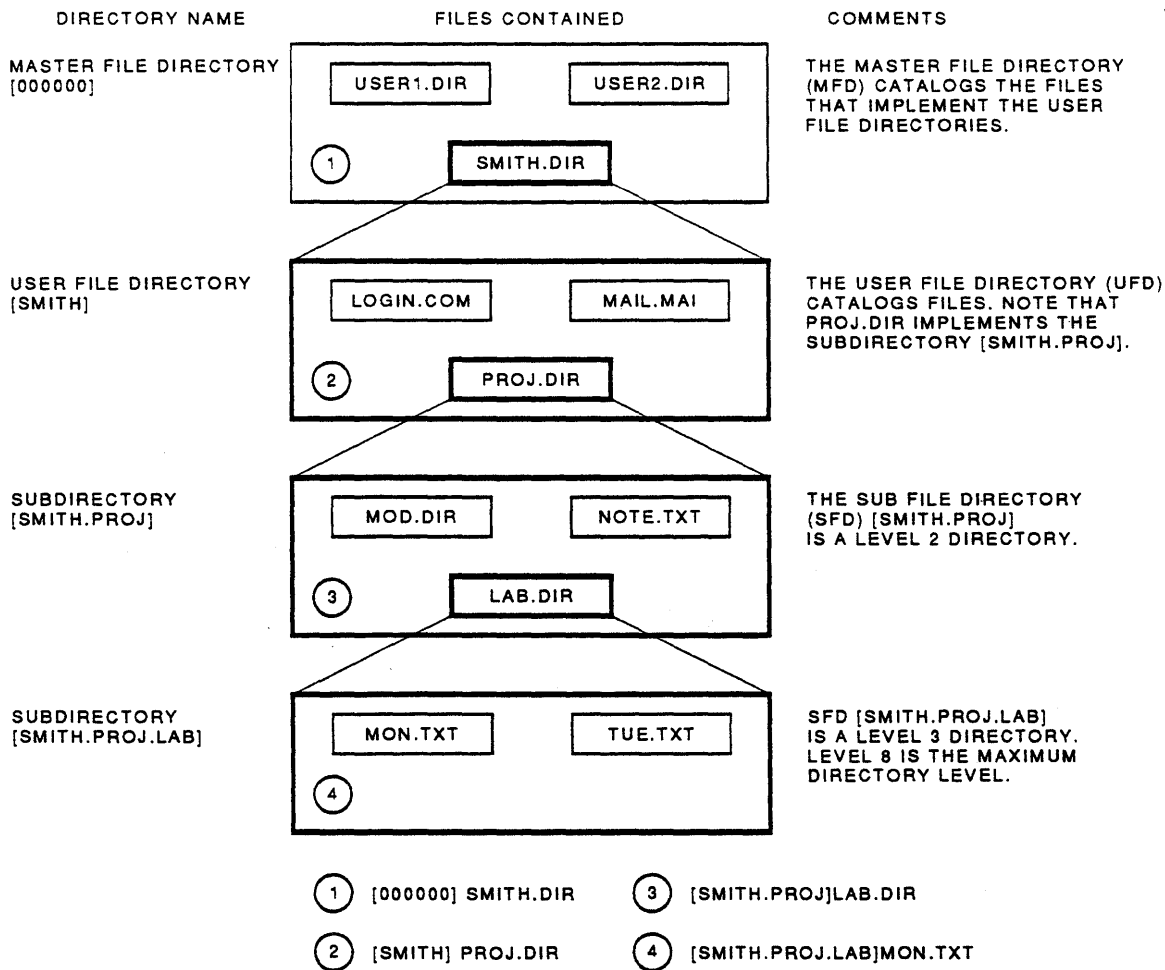


← can use it. combo!
A to B

EX. DIR [...] goes as low as you can go!

EX. DIR [... NAME] go as far down from "NAME" as possible!

Figure 5-2: File Specification in the Directory Hierarchy



TTB_X0328_88

Example 5-2: Using VMS Commands to Create and Maintain a Directory Hierarchy

```
$ SHOW DEFAULT
  DISK:[SMITH]
$ 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

$ DIRECTORY [...]

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.

*$ RENAME [SMITH]*.MAR,*.* [.UTLCOM.FIL]*.* - MOVE FILES BETWEEN DIRS!
$ SET DEFAULT [.UTLCOM.FIL]
$ DIRECTORY

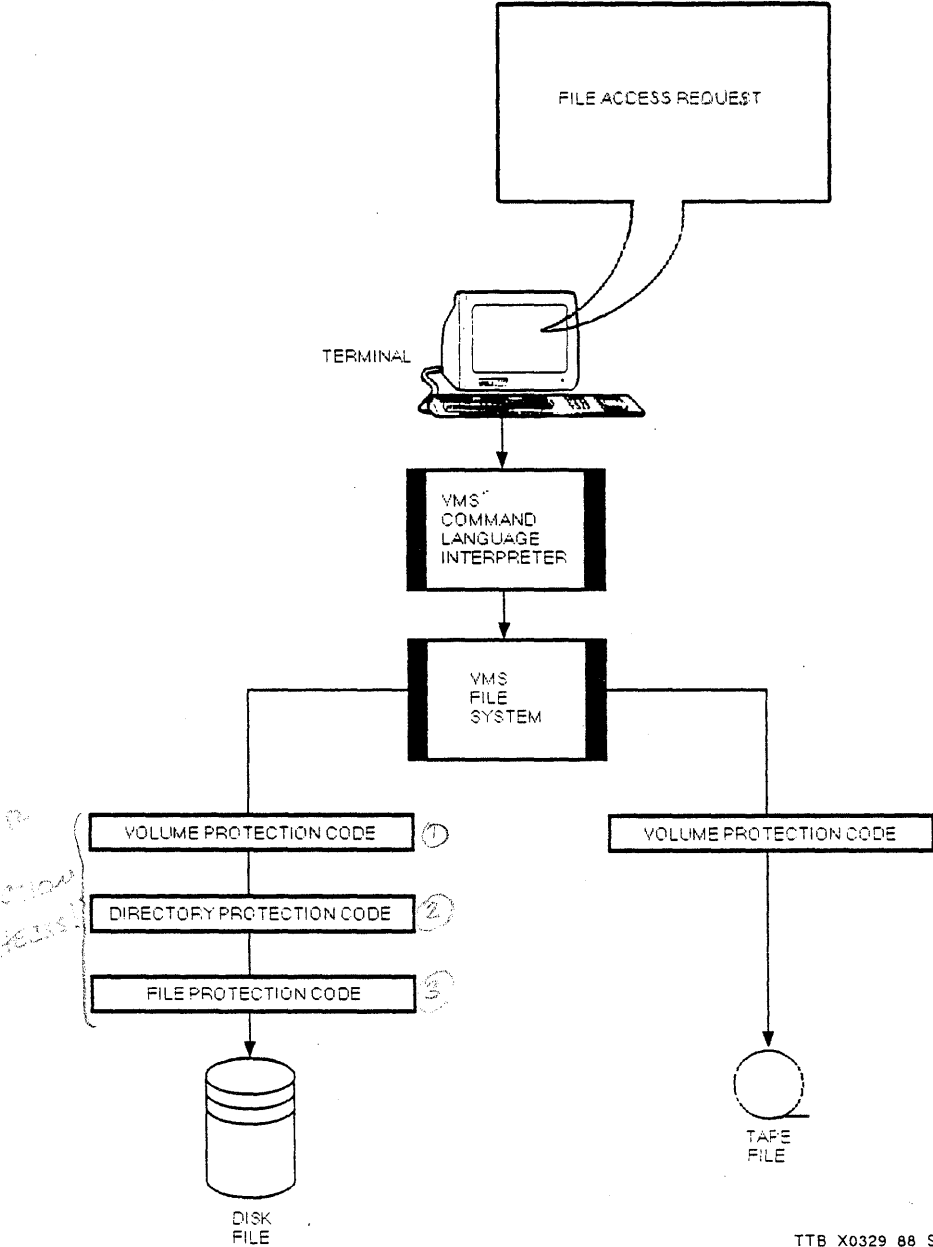
Directory DISK:[SMITH.UTLCOM.FIL]

FORCALL.MAR;1     STRPROG.TXT;1

Total of 2 files.
```

PROTECTING DISK AND TAPES

Figure 5-3: File Access to Disk and Tape Volumes



PROTECTING FILES IN YOUR DIRECTORY HIERARCHY

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

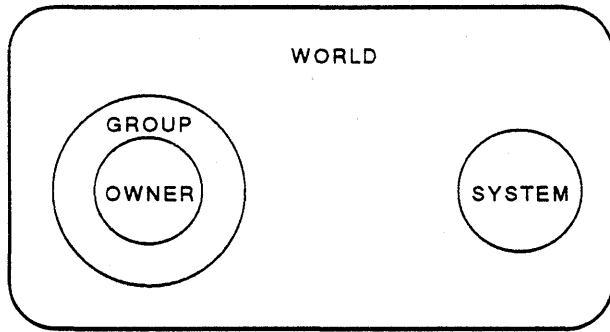
Three Levels of Disk File Protection

- Volume Protection
 - Controls who can access a particular disk volume
OR TAPE
- Directory Protection
 - Controls who can access a particular directory
- File Protection
 - Controls who can access a particular file
- * — Two means of protecting files:
 - 1) User Identification Code (UIC-based) protection
 - 2) Access Control Lists (ACLs)
SYSTEM MANAGEMENT

* UIC-Based Protection

- Format: [group,member] *→ always in square brackets! IN OCTAL!*
 - Can be either numeric or alphanumeric
 - Group = 0-37777 (octal numbering system)
 - Member = 0-17777 (octal numbering system)
- Examples:
 - Numeric: [100,30]
 - Alphanumeric: [GROUP11,SMITH]
- The default protection mechanism
- Each file is assigned a protection code when it is created
- The system manager assigns UICs to all users
- Protection codes are checked against a user's UIC before allowing them access to a file
- By default, the file UIC matches the user's UIC

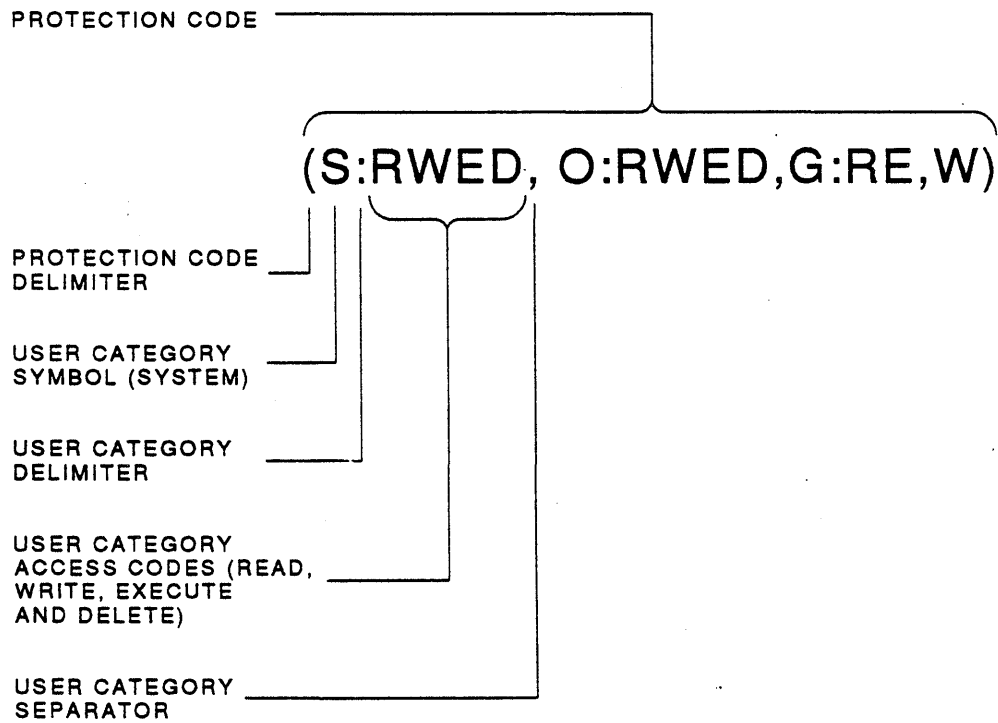
Figure 5-4: Interaction of Access Categories



TTB_X0330_88



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



TTB_X0331_88

Table 5-11: 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 <i>CAN NOT DO!</i>

Table 5-12: 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

DETERMINING AND ALTERING FILE PROTECTION

Table 5-13: Commands Used to Determine and Alter File Protection

Operation	Comments
<p>* \$ SHOW PROTECTION</p> <p>Displaying the default protection assigned to new files</p>	<p>The default protection applies to all newly created files in the current directory.</p>
<p>* \$ DIRECTORY/PROTECTION MYFILE.TXT</p> <p>Obtaining the protection code of a given file</p>	<p>Displays the current protection of an existing file.</p>
<p>* \$ SET PROTECTION=(S:RWED,O:RWED,G:RWE,W:RWE)/DEFAULT</p> <p>Changing the default protection assigned to new files</p>	<p>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.</p>
<p>* \$ SET PROTECTION=(S:RWED,O:RWE,G:RW,W) MYFILE.TXT</p> <p>Changing the protection code of an existing file</p>	<p>The protection code can be changed to allow more or less access to an existing file.</p>

NOTE

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

CAN DO SET PROTECTION=(S:RWED,O:RWE)

↑
owner has
no privileges!

Example 5-3: Changing Your Default Protection Code

```
$ 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      [100,200]          (RWED,RWED,RE,)
EDT.DIR;1           [100,200]          (RWE,RWE,RWE,RE)
Total of 1 file.
$ SET PROTECTION=(S:R,G:R)/DEFAULT same *,*;*
$ 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.
```

DELETING A SUBDIRECTORY

- Before a subdirectory can be deleted, all files cataloged in that subdirectory must be deleted.
- Set your default to the directory or subdirectory containing the subdirectory name to be deleted (SUBDIRECTORY.DIR file).
- The protection on the subdirectory to be deleted must allow the owner DELETE access. The directory protection must be changed to reflect this, since the system never assigns DELETE access to a DIR file type.
- The subdirectory can now be deleted.

Example 5-4: Deleting a Subdirectory from a Directory Hierarchy

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

Directory DISK:[SMITH.DOC]

CLASS.LIST;4          CLOCK.EXE;1          COLOR.COM;4          DEG.EXE;1
JOE_EVE.TPUSSECTION;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 5-5: Removing Subdirectories from a Directory Hierarchy

```
$ SET DEFAULT [SMITH]
$ 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.
$ SET PROTECTION=O:RWED [SMITH...]*.*;*
$ 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
$ 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.
$ 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
$ DIRECTORY [SMITH...]
Directory DISK:[SMITH]
DOC.DIR;1
Total of 1 file.
$ DELETE [SMITH...]*.*;*
$ DIRECTORY [SMITH...]
%DIRECT-W-NOFILES, no files found
```

* Access Control Lists

- An optional layer of protection
- Can be used for more control than UIC-based protection
- Usually used when access is to be provided for specific users but not all users on a system
- Based on identifiers
 - Users can have one or more identifiers
 - Files specify access rights for holders of various identifiers

ONLY - SYSTEM MANAGER!

Commands to Obtain ACL Information

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

Creating an Access Control List

- The DCL command EDIT/ACL file-name invokes the ACL editor
- Access Control List Entries (ACEs) can be added to the ACL
 - No limit to the number of ACEs contained in an ACL
 - No limit to the number of ACE characters contained in an ACL
 - ACEs are enclosed in parentheses

- Syntax:

(TYPE, [OPTIONS],[ACCESS])

- The first field indicates the group or subset of a group that will have access to files
- The second field indicates options (if any) that apply to the ACE
- The third field indicates the type of access to be granted to the file (READ, WRITE, EXECUTE, DELETE, CONTROL, NONE)

Example 5-6: Modifying an Access Control List

\$ 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) ONLY SYS.MAN
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.

BUT CAN DO

(IDENTIFIER=[GROUP, MEMBERS], ACCESS =)

*CAN
HAVE ?
ACCESS = NONE!*

*Deleting an Access Control List

- Use the SET ACL command to delete an Access Control List
- Example:

```
$ SET ACL/DELETE MYFILE.TXT
```

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

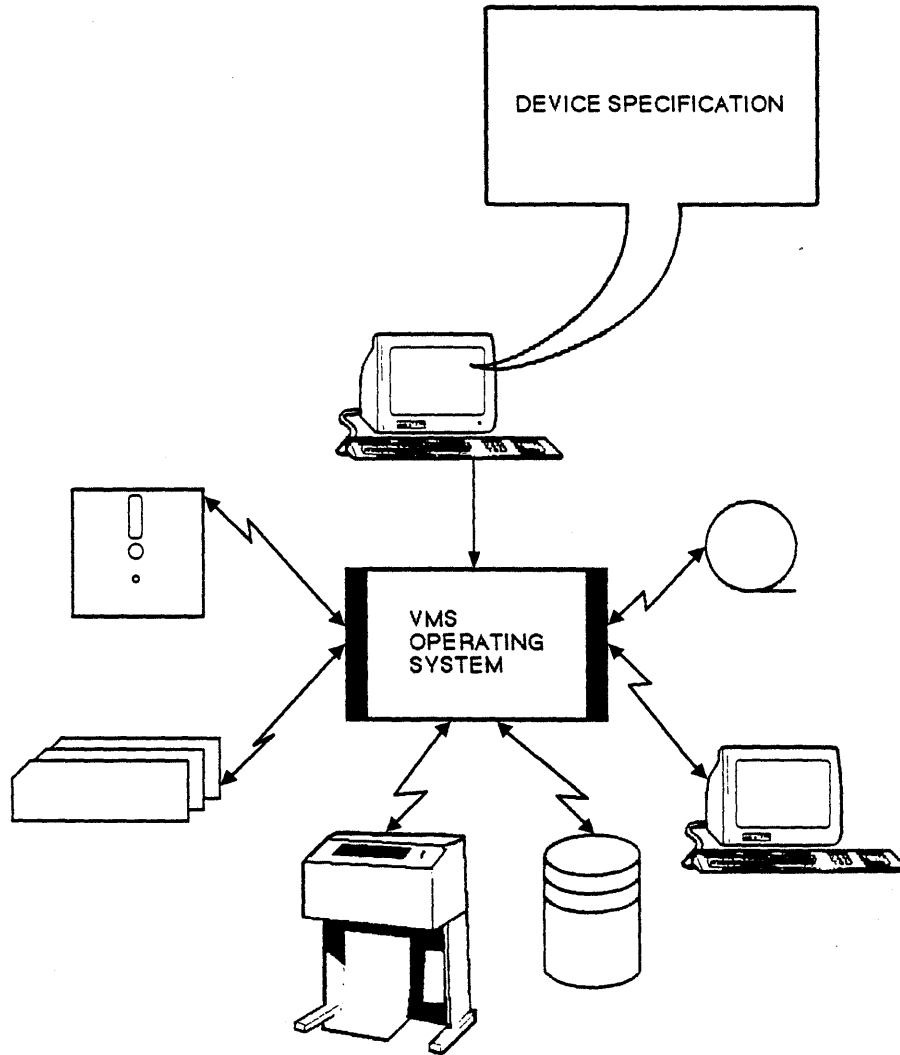
There are two means of protecting files:

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

APPENDIX A—DEVICE INFORMATION

Specifying Devices

Figure 5-6: Device Specifications Used to Identify the Desired Device for a Given Operation



VAX-11 DEVICES

TTB_X0332_88_S

Table 5-14: Examples of Using Other Devices

Operation	Comments
Listing files in a directory on another disk	Lists all files in the directory [SMITH] located on the disk DBA2:
\$ DIRECTORY device-name:[directory-name] \$ 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 device-name:[directory-name]file-name \$ 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 device-name:[directory-name]filename file-name \$ COPY DBA2 : [SMITH]MYFILE . TXT MYFILE . TXT	
Listing all files on a tape device	Lists all files on magnetic tape on device MTA2:
\$ DIRECTORY device-name: \$ DIRECTORY MTA2 :	
Finding a file on a tape device	Searches for the file MYFILE.TXT on magnetic tape on device MTA2:
\$ DIRECTORY device-name:file-name \$ 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 device-name:file-name disk:[directory-name]file-name \$ COPY MTA2 :MYFILE . TXT * . * ; *	

Table 5-15: Moving a Hierarchical File Structure from one Disk Device to Another

Command	Comment
The COPY command	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...] *.*; *
```



Table 5-16: 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
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

Table 5-17: 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 and a device name or allocation class separated by a dollar sign.

Table 5-18: Generic Specification with the SHOW DEVICE Command

Operation	Comment
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 (DU).
\$ SHOW DEVICE DM:	
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:	

APPENDIX B—NETWORKING INFORMATION

Managing Files on Another VMS System in Your Network

Methods of File Management in a Network

- **Use the SET HOST command**
 - Enter SET HOST
 - Both processors must be running DECnet
 - You must know a user name and password of an account on the remote system
 - Enter DCL file-manipulation commands
- **Use an access-control string in your DCL commands**
 - Include an access-control string in your DCL file-manipulation commands
 - A user name of an account on the remote system
 - A password for the account on the remote system
- **Use a proxy account**
 - Established by the system manager
 - Associates your user name with an account on the remote system
 - The remote account provides needed system values
- **Use the DECnet defaults**
 - The system manager can establish a default DECnet account
 - The DECnet account supplies needed system values

Using DCL File-Manipulation Commands in a Non-VAXcluster Network Environment

Two Node Specification Formats

- **Nodename::**
 - The remote system process obtains needed values from its default DECnet account
(If there is no default DECnet account, your file-manipulation request fails)
 - You have file access rights based on the DECnet account UIC
- **Nodename"access control string"::**
 - The remote system creates a process using the access control string values
 - The new remote account supplies needed system values

Table 5-19: Examples of Specifying Files on Remote Nodes

Example	Comments
<pre>\$ DIRECTORY DIPPER::DBA1:[SMITH]PAY.FOR;1</pre>	Specifies file PAY.FOR;1 in the directory [SMITH] on disk DBA1: on remote node DIPPER:.
<pre>\$ DIRECTORY DIPPER"SMITH CORONA":DBA1:[SMITH]PAY.FOR;1</pre>	Specifies the same as above example. Access to the file uses the UIC of user SMITH.
<pre>\$ DIRECTORY DIPPER"SMITH CORONA":DBA1:[SMITH]PAY.FOR;1</pre>	Specifies the same as above example. The process supplies the defaults under the account for SMITH.
<pre>\$ DIRECTORY DIPPER"SMITH CORONA":PAY.FOR;1</pre>	Specifies the file PAY.FOR in the subdirectory [SMITH.DOC] on the default disk of user SMITH on node DIPPER:.
<pre>\$ DIRECTORY DIPPER"SMITH CORONA":[SMITH.DOC]PAY.FOR</pre>	

Table 5-20: DECnet-VAX DCL File-Manipulation Command Summary

Function	Comments
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 [JAFPE] on the remote node BOSTON:: to the file TEST.DAT in your current directory on your local node.
\$ APPEND input-file[,...] output-file[,...] \$ APPEND BOSTON"JAFPE 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 SUPER::. Defaults on the remote node come from the UAF record specified within quotes. The same file name is retained.
\$ COPY Input-file[,...] output-file[,...] \$ COPY DEC12.DAT SUPER"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 file-specification \$ 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 SUPER::.
\$ DIRECTORY file-specification \$ DIRECTORY SUPER"JANES JIL"::[JANES.SUB1]	
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 file-specification \$ TYPE DIPPER::[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 SUPER::.
\$ DELETE file-specification \$ DELETE SUPER"JANES JIL"::[JANES.SUB1]PAY.FOR;*	

Using DCL File-Manipulation Commands in a VAXcluster Environment

Two Cluster Device Specification Formats

1.	Format	Example
	<code>node-name\$device-name</code>	<code>PETER\$DUA1:</code>

- Node name (name of HSC50 or VAX)
- Dollar-sign (\$)
- Device name

2.	Format	Example
	<code>\$allocation-class\$device-name</code>	<code>\$1\$DUA0:</code>

- Dollar-sign (\$)
- Allocation class (a number between 0 and 255)
- Dollar-sign (\$)
- Device name

Table 5-21: Commands Used to Determine the Nodes and Devices in Your Systems Environment

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

MODULE 6

CUSTOMIZING THE USER ENVIRONMENT

INTRODUCTION

In earlier modules, you have learned to enter commands 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 and complex, which can lead to typographical and syntactical errors.

This module introduces logical names and demonstrates how 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.

OBJECTIVES

To tailor the user environment, you should be able to:

- ① • Create and use logical names for file access.
- Use the logical names the VMS system defines for all users.
- ② • Create and use symbols as command synonyms.
- ③ • Define and use terminal keys to speed up execution of frequently used DCL commands.

RESOURCES

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

* LOGICAL NAMES!

- A *logical name* is a name you can use in place of all or part of a file specification
- It is 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

• Format:

* \$ DEFINE logical name # equivalence string[,...] !

or

* \$ ASSIGN equivalence-name [...] logical-name !

* LOGICAL IS DEFAULT
qualifier for DEFINE! For
ex. To define keys need
DEFINE/KEY!

- 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

* Logical Name Tables

▷ Private

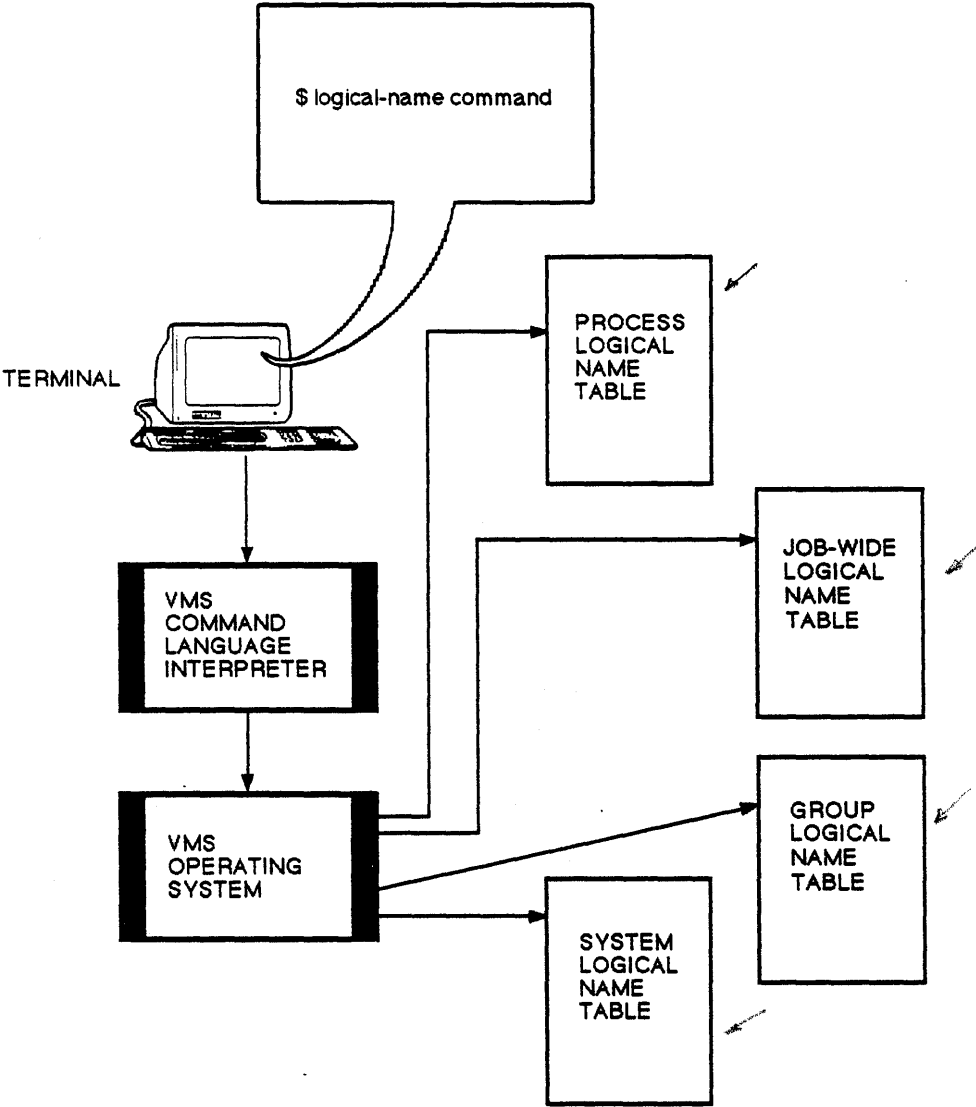
- Process logical name table
 - Used only by your process !
 - /PROCESS – DCL command qualifier

DEFAULT QUALIFIER!

▷ Shared

- Job-wide logical name table
 - Used by your process and its subprocesses !
 - /JOB – DCL command qualifier
- Group logical name table
 - Used by UIC group member processes !
 - Privilege is needed to add logical names to this table !
 - /GROUP – DCL command qualifier
- System logical name table
 - Used by all system processes !
 - Privilege is needed to add logical names to this table !
 - /SYSTEM – DCL command qualifier

Figure 6-1: The Relationship Between Your Terminal, the Operating System, and the Logical Name Tables Associated with Your Process



TTB_X0333_88_S

Common User Operations Dealing with Logical Names

- Display the contents of logical name tables
- Determine the equivalence string of a logical name
- Add or alter logical name assignments in your process logical name table
- Override system–defined logical names in your process logical name table
- Remove a logical name from your process logical name table

Adding Logical Names

- **ASSIGN** command

Format:

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

Example:

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

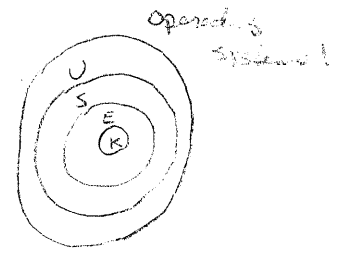
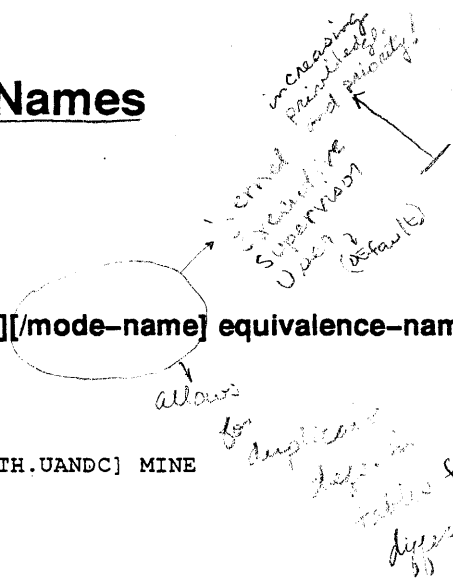
- **DEFINE** command

Format:

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

Example:

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



EX. DEFINE MINE DISK:
must specify a device

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

```
$ 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".
$
```

USING LOGICAL NAMES

Logical Name Translation for Logical Names that Have Single Equivalence Strings

- The system translates logical names automatically.
- Logical name tables are searched for the first occurrence of a logical name.

* Search order:

- 1) Process Logical Name Table
- 2) Job-Wide Logical Name Table
- 3) Group Logical Name Table
- 4) System Logical Name Table

- Translates left-most portion of all file specifications to see if it is a logical name.
- Translates:
 - Up to 10 times (recursively). ^{EX.} p.6-11
 - Until no more equivalence names to be translated.
 - Until left-most component of the specification is not delimited by a colon, a space, a comma, or an end of line.
 - Until equivalence name is a logical name that has the TERMINAL attribute. If a logical name has the TERMINAL attribute, the translation is "TERMINAL" (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.

NOTE

→ Both TERMINAL and CONCEALED are translation attributes. They are defined by using the /TRANSLATION_ATTRIBUTES= qualifier for either the DEFINE or ASSIGN DCL commands.

DEFINE /TRANSLATION_ATTRIBUTES = TERMINAL
DEFINE /TRANSLATION_ATTRIBUTES = CONCEALED

Sample Recursive Translation

- Command

```
$ DIRECTORY PROJECTS
```

- First table search (looking for PROJECTS)

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

- Second table search (looking for DISK_USER)

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

- Result

— DBAO:[ELLEN] – Searched

* Does NOT need to exist in the table in ANY PARTICULAR ORDER, (automatically ordered alphabetically). But even in the case of recursion, whole table is searched for each call!

EX. DEFINE CUSTOMER.DUA2:

```
DEFINE MONDAY CUSTOMER:[CUSTOMER*.MONTH]
```

```
EDIT MONDAY: MONDAY.TXT
```

```
DIR MONDAY
```

*notice that the colon is necessary if it is to be followed w/the rest of the file spec, else NO NEED.

Displaying the Contents of Logical Name Tables

* **Table 6-1: Displaying the Contents of Logical Name Tables**

Command String/Example	Comments
\$ <u>SHOW LOGICAL</u>	By default, displays logical names from the process, job-wide, group, and system logical name tables
\$ <u>SHOW LOGICAL/FULL</u>	Displays all of the attributes of logical names from the process, job-wide, group, and system logical name tables
\$ <u>SHOW LOGICAL/PROCESS</u>	Displays logical names from your process logical name table
\$ <u>SHOW LOGICAL/JOB</u>	Displays logical names from your job-wide logical name table
\$ <u>SHOW LOGICAL/GROUP</u>	Displays logical names from your group logical name table
\$ <u>SHOW LOGICAL/SYSTEM</u>	Displays logical names from the system logical name table

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

```
$ SHOW LOGICAL/PROCESS
(LNM$PROCESS_TABLE)
"SYSS$COMMAND" = "_DISK$RTA1:"
"SYSS$DISK" = "DISK:"
"SYSS$ERROR" = "_DISK$RTA1:"
"SYSS$INPUT" [super] = "_DISK:" — Ex of MODC usage
"SYSS$INPUT" [exec] = "_DISK$RTA1:" — NOTICE LOGG. DEF: 010-9
"SYSS$OUTPUT" [super] = "_DISK$RTA1:"
"SYSS$OUTPUT" [exec] = "_DISK$RTA1:"
"TT" = "RTA1:"
$ SHOW LOGICAL/JOB
(LNM$JOB_803E4E40)
"SYSS$LOGIN" = "DISK:[SMITH]"
"SYSS$LOGIN_DEVICE" = "DISK:"
"SYSS$REM_ID" = "SMITH"
"SYSS$REM_NODE" = "SUPER:."
"SYSS$SCRATCH" = "DISK:[SMITH]"
$ SHOW LOGICAL/GROUP
(LNM$GROUP_000011)
"MY_DISK" = "DJAO:"
$ SHOW LOGICAL/SYSTEM
(LNM$SYSTEM_TABLE)
"DBG$INPUT" = "SYSS$INPUT:"
"DBG$OUTPUT" = "SYSS$OUTPUT:"
"DISK$BROWNY_SYS" = "DISK:"
"SYSS$ANNOUNCE" = ".Welcome to Brownny."
"SYSS$COMMON" = "DISK:[SYS0.SYSCOMMON.]"
"SYSS$DISK" = "DISK:"
"SYSS$ERRORLOG" = "SYSS$SYSROOT:[SYSERR]"
"SYSS$HELP" = "SYSS$SYSROOT:[SYSHLP]"
"SYSS$MAINTENANCE" = "SYSS$SYSROOT:[SYSMAINT]"
"SYSS$MANAGER" = "SYSS$SYSROOT:[SYSMGR]"
"SYSS$MESSAGE" = "SYSS$SYSROOT:[SYSMSG]"
"SYSS$NODE" = "BROWNY:."
"SYSS$SYLOGIN" = "SYSS$MANAGER:SYLOGIN.COM"
"SYSS$SYSDEVICE" = "DISK:"
"SYSS$SYSROOT" = "DISK:[SYS0.]"
= "SYSS$COMMON:"
"SYSS$SYSTEM" = "SYSS$SYSROOT:[SYSEXE]"
"SYSS$UPDATE" = "SYSS$SYSROOT:[SYSUPD]"
```

Determining the Equivalence of a Logical Name

- Two commands are available to determine the equivalence of a logical name
- Format:

* **\$ SHOW LOGICAL logical-name**

Iteratively translates the logical name up to 10 levels until everything is resolved

* **\$ SHOW TRANSLATION logical-name**

Displays the first equivalence string it finds and stops (no iteration is performed)

Example 6-3: Determining the Value of a Logical Name

```
$ ASSIGN DJAO: DISK1
$ ASSIGN DISK1: MYNAME
$ SHOW TRANSLATION MYNAME
MYNAME = "DISK1:" (LNMSPROCESS_TABLE)
$ SHOW LOGICAL MYNAME
"MYNAME" = "DISK1:" (LNMSPROCESS_TABLE)
1 "DISK1" = "DJAO:" (LNMSPROCESS_TABLE)
```

* Deleting Logical Names

Table 6-2: Commands to Delete Logical Names

Operation	Command String/Example	Comments
Delete a logical name assignment	\$ DEASSIGN [logical-name]	
	\$ DEASSIGN MYFILE	Deletes the logical name MY-FILE 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	Deletes a logical name in your group table. GRPNAM privilege is needed.
\$ DEASSIGN/SYSTEM	Deletes a logical name in the system table. SYSNAM privilege is needed.	

Example 6-4: Assigning, Changing, and Deleting Logical Name Assignments

```
$ ASSIGN DJAO: DISK1
$ ASSIGN DISK1:[SMITH] LOG
$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

"DISK1" = "DJAO:"
"LOG" = "DISK1:[SMITH]"
"SYSS$COMMAND" = "_DISK:"
"SYSS$DISK" = "DISK:"
"SYSS$ERROR" = "_DISK$RTA1:"
"SYSS$INPUT" [super] = "_DISK:"
"SYSS$INPUT" [exec] = "_DISK$RTA1:"
"SYSS$OUTPUT" [super] = "_DISK$RTA1:"
"SYSS$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:[SMITH]"
"SYSS$COMMAND" = "_DISK$RTA1:"
"SYSS$DISK" = "DISK:"
"SYSS$ERROR" = "_DISK$RTA1:"
"SYSS$INPUT" = "_DISK$RTA1:"
"SYSS$OUTPUT" [super] = "_DISK$RTA1:"
"SYSS$OUTPUT" [exec] = "_DISK$RTA1:"
"TT" = "RTA1:"

$ DEASSIGN/ALL
$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

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

* 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

Refer to the following tables for lists of system-defined logical names

Table 6-3: Process Logical Names Defined by the System

<u>Logical Name</u>	<u>Equivalence Name</u>
<u>SY\$COMMAND</u>	Original value of SYS\$INPUT, equated to your terminal for interactive use and command procedures.
<u>SY\$DISK</u>	Default disk established at login. Can be changed by the SET DEFAULT command.
<u>SY\$ERROR</u>	Default device to which the system writes messages. For an interactive user, the system equates SY\$ERROR to the terminal.
<u>SY\$OUTPUT</u>	Default output devices. For an interactive user, SY\$OUTPUT is equated to the terminal.
<u>SY\$INPUT</u>	Default input device. For all interactive use, SY\$INPUT is equated to the terminal. For command procedures, it is equated to the command file on disk.
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
* SY\$\$LOGIN	Default disk and directory established at login time. This "home" directory is specified in the authorization record.
SY\$\$LOGIN_DEVICE	Default disk established at login. Unlike the logical name SY\$\$DISK, SY\$\$LOGINDEVICE is not changed by the SET DEFAULT command.
SY\$\$SCRATCH	Default device and directory to which temporary files are written. This is always equated to your default directory. <i>EX. OF TEMP FILE IS FN.JOB</i>

* **Table 6-5: System Logical Names Defined by the System**

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

* Specifying Logical Name Access Modes

- * • /USER_MODE (temporary assignment)
- /SUPERVISOR_MODE (default – permanent assignment)

Duration of a Process-Private Logical Name Assignment

- Supervisor mode assignments last until you
 - Log out
 - Assign the particular logical name to a different equivalence string
 - Remove the logical name assignment by using the **DEASSIGN** command
- * • User mode assignments last:
 - Until the next image run in your process completes execution. (An image is a program in its executable form.)

Example 6-5: Using ASSIGN Command to Alter the Default Output Device of Your Process

```
$ 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:
$
```

EX. OF A ONE TIME
REASSIGNMENT
OF
OUTPUT!

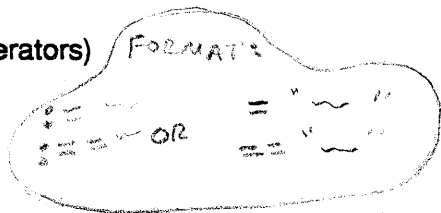
* USING DCL SYMBOLS

- 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
- * Often defined in a command file named LOGIN.COM (usually located in your default directory) for use in every terminal session

— Defined with = or == (assignment operators)

* — Local with = !!!

* — Global with ==



— Example:

```
$ SD == "SET DEFAULT"
```

* Can abbreviate symbol names using the asterisk (*) as the abbreviation character

• Example:

```
— $ M*AIL = "MAIL"
```

ex. LO*GOUT = "LOGOUT/FULL"

— The abbreviations "M", "MA", "MAI", and "MAIL" now will invoke the Mail utility

* Translation is not iterative !!!

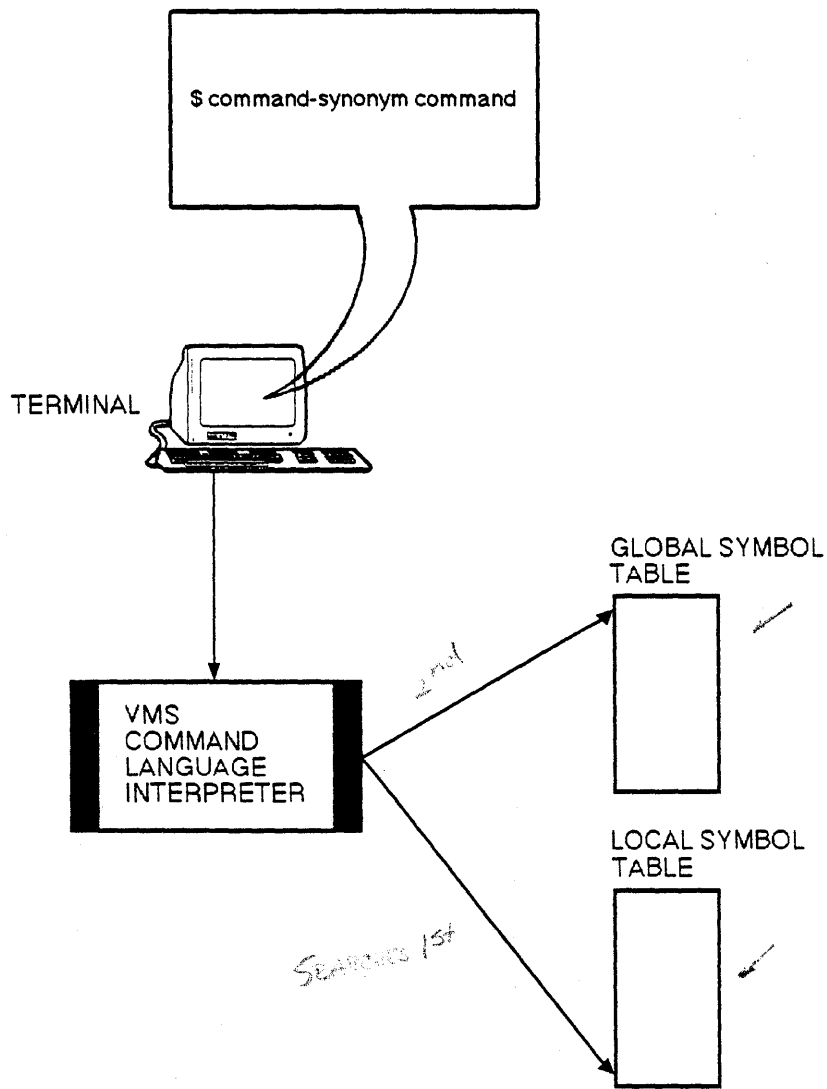
• Examples:

```
— $ PROTECT == "SET PROTECTION=(S:R,O:RWED,G:R,W)"
```

```
— $ PROTECT [SMITH] *.*;*
```

3rd son protecting
Dad's electronics!

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



TTB_X0334_88_S

Table 6-6: Commands for Displaying and Deleting DCL Symbols

Operation	Command String/Example	Comments
Displaying Symbols		
Displaying a single symbol	\$ SHOW SYMBOL/GLOBAL symbol-name \$ SHOW SYMBOL/GLOBAL GO	Displays the value of the symbol GO
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 GO	Deletes the symbol GO from your global symbol table
Deleting all symbols	\$ DELETE/SYMBOL/GLOBAL/ALL	Deletes all symbols from your global symbol table

Example 6-6: Defining, Displaying, Using, and Deleting DCL Symbols

```
$ DIRP == "DIRECTORY/OWNER/PROTECTION"
$ GO == "SET DEFAULT"
$ RETURN == "SET DEFAULT SYSS$LOGIN"
$ SHOW SYMBOL/GLOBAL/ALL
$RESTART == "FALSE"
$SEVERITY == "1"
$STATUS == "%X00030001"
DIRP == "DIRECTORY/OWNER/PROTECTION"
GO == "SET DEFAULT"
RETURN == "SET DEFAULT SYSS$LOGIN"

$ GO SYSS$SYSTEM
$ DIRP DCL.EXE
Directory SYSS$COMMON:[SYSEXE]

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

Total of 1 file.

$ RETURN

$ DELETE/SYMBOL/GLOBAL/ALL

$ SHOW /SYMBOL/GLOBAL/ALL

$RESTART == "FALSE"
$SEVERITY == "1"
$STATUS == "%X00030001"
```

Table 6-7: Comparison of Logical Names and DCL Symbols

Symbol	SYMBOLS Logical Names		LOGICAL NAME	
TO USE	Equated to all or part of a command string		Used in place of all or part of a file specification	
TO CREATE	= (LOCAL) == (GLOBAL)		ASSIGN or DEFINE	
TO DISPLAY	SHOW SYMBOL		SHOW LOGICAL or SHOW TRANSLATION	
TO DELETE	DELETE/SYMBOL		DEASSIGN	
QUALIFIERS		Used for:		Used for:
	/LOCAL	(DIS, DEL) ¹	/PROCESS	(CRE, DIS, DEL) ¹
	/GLOBAL	(DIS, DEL) ¹	/JOB	
	/ALL	(DIS, DEL) ¹	/GROUP	(CRE, DIS, DEL) ¹
			/SYSTEM	
			/ALL	(DIS, DEL) ¹ (DIS, DEL) ¹ (DIS) ¹
NOTES	/LOCAL	is the default for display and delete	/ALL	is the default for display
			/PROCESS	is the default for create and delete

¹DIS means DISPLAYING
DEL means DELETING
CRE means CREATING

SYMBOLS REPLACES → ACTIONS, VERBS
LOGICALS REPLACES → PHRASES, OBJECTS

* **DEFINING KEYS** (ONLY PER LOGIN SESSION UNLESS IN LOGIN.COM) (ONLY AT DCL LEVEL)

- Definitions often contain part or all of a DCL command string
- Reduces typing of lengthy or frequently used DCL commands
- Terminal types and associated definable keys include

- VT52-type terminals

All definable keys located on the numeric keypad

- VT100-type terminals

All keys located on the numeric keypad
<LEFT> and <RIGHT> arrow keys

- * — Terminals with LK201 keyboards (LIKE A 200)

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

\$ SET TERMINAL/NUMERIC

- Keypad keys <PF1> – <PF4> can also be defined

- Format:

EX. DEFINE/KEY PF1 "directory"

- * \$ DEFINE/KEY key-name equivalence string /qualifiers needs quotes!

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

- ! * — /TERMINATE – Produces an automatic return! else have to press Key defined and return

- * — /NOECHO – Suppresses the display of the command being invoked
(only if /TERMINATE is included else won't work!)

- /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

- To display a key definition, issue the DCL command:

* SHOW KEY/FULL key-name

Example:

```
$ SHOW KEY/FULL PF1
PF1 = "directory"
(echo, noterminal, noerase, nolock)
```

- To delete a key definition, issue the DCL command:

* DELETE/KEY key-name

Example:

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

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

MODULE 7

WRITING COMMAND PROCEDURES

INTRODUCTION

Command procedures are files consisting of DCL commands. They can be used to automatically execute command sequences that are needed repeatedly. In addition to the command verbs, qualifiers, and parameters commonly used at the interactive level, command procedures allow the use of DCL command language features that provide increased functionality 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. In a later module, you will learn how to run command procedures independently of your interactive process, as batch jobs.

*sys\$system] sylogin.com - system-wide login file, expected
login in the local login.com.*

*ⓐ - executes a command procedure!
↳ hands off to batch which loads AND INTERPRETS THE FILE!*

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.

RESOURCES

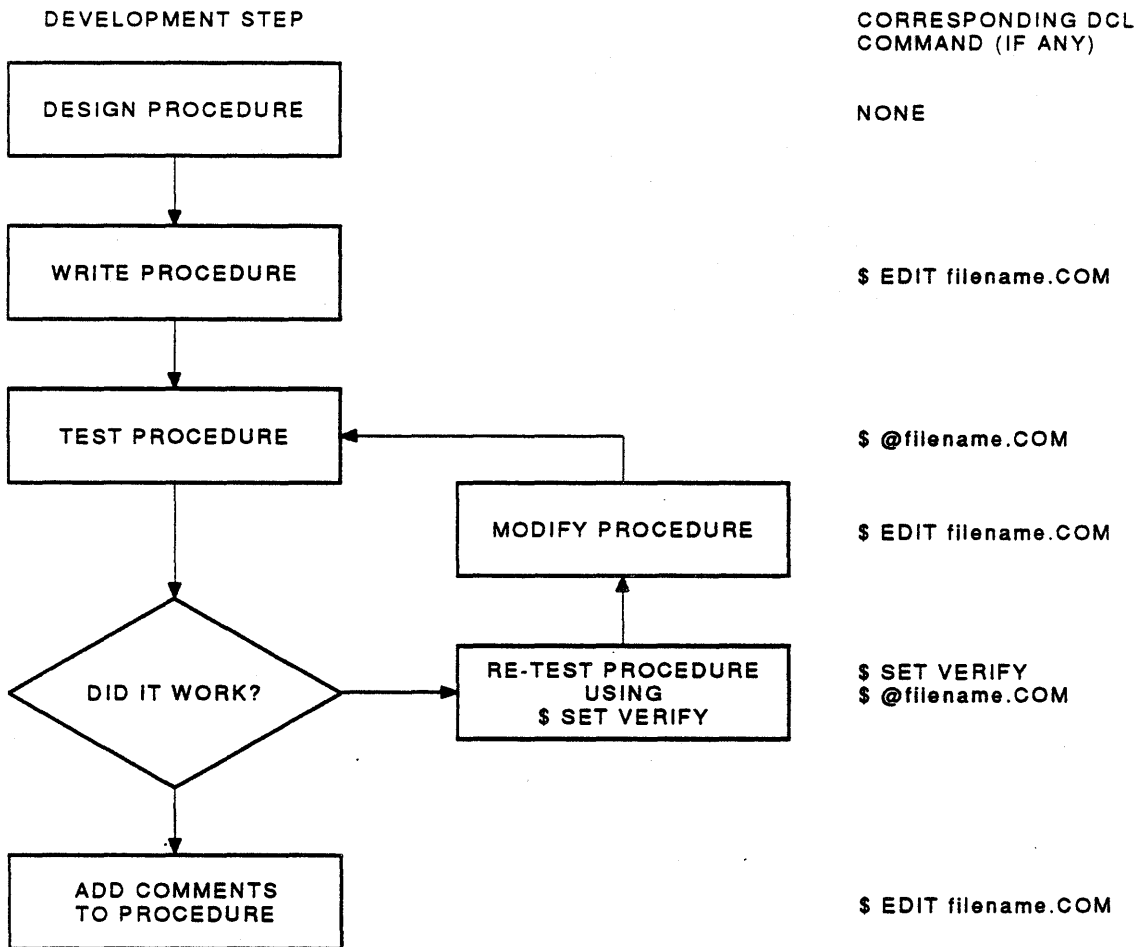
- *Guide to Using VMS Command Procedures*
- *VMS DCL Dictionary*

DEVELOPING A COMMAND PROCEDURE

The steps you take to develop a command procedure are similar to the steps you take to develop any computer program in any language. The following steps are illustrated in Figure 7-1.

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 you 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.
 - Describe any parameters that are passed to the procedure.

Figure 7-1: Command Procedure Development Process



TTB_X0335_88

Components and Conventions

Consistent formats and clear programming style make your command procedures easy to read, test, and maintain. Example 7-1 illustrates some of the conventions below.

- DCL command lines
 - Use full command names, no abbreviations
 - Precede each command line with the dollar sign (\$) prompt
 - Continue a line by placing a hyphen at the end of the line (do not begin the continued line with a dollar sign)
- Comments
 - Precede all comment lines with an exclamation mark (!)
 - Use blank comment lines to separate blocks of commands
- Labels
 - Use labels to mark locations within a procedure
 - Place the label on a line by itself
 - Follow the label with a colon (:)
- Data lines
 - Place data in a command procedure immediately after the command that will use it
 - Do **not** place a dollar sign at the beginning of a data line
(Terminated by the first occurrence of the dollar sign)

Example 7-1: A Sample Command Procedure

```
$ !                                     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]
$ !
$ ! Check to verify you are in the correct directory
$ !
$ SHOW 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:

Example 7-1: A Sample Command Procedure (Cont)

```
$ @REPORT1
Job MONDAY (queue SYSS$PRINT, entry 44) started on WORK$TXAO
```

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]
$ !
$ ! Check to verify you are in the correct directory
$ !
$ ! SHOW DEFAULT
$ ! DISK1:[REPORTS.MONDAY]
$ !
$ ! Print out the report for Monday
$ !
$ ! PRINT MONDAY.RPT
Job MONDAY (queue SYSS$PRINT entry 46) started on WORK$TXAO
$ !
$ ! Return to your login device and directory
$ !
$ ! SET DEFAULT SYSS$LOGIN
$ ! EXIT
```

LOGIN COMMAND PROCEDURE

- Is a command procedure that is executed automatically each time you log in
- *• Must be called LOGIN.COM and placed in your default login directory.
- Contains logical names, symbols, and other commands to set up your terminal session
- *• Can be disabled for a particular session by typing /NOCOMMAND after your user name at the *Username:* prompt.
- Example 7-2 shows a typical LOGIN.COM file

Example 7-2: A Sample LOGIN.COM File

```
$ !
$ !
$ !
$ !
$ ! 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 - NEEDED TO ADD FOR KPO - KPP DEF.
$ !
$ DEFINE/KEY/NOLOG/TERMINATE PF1 "SHOW USERS"
$ DEFINE/KEY/NOLOG/TERMINATE PF3 "SHOW TIME"
$ DEFINE/KEY/NOLOG/TERMINATE KP9 "SHOW QUEUE/ALL/FULL LPAO"
$ DEFINE/KEY/NOLOG/TERMINATE KPO "LOGOUT"
$ !
$ EXIT
```

TERMINAL INPUT/OUTPUT

- Several DCL commands allow you to perform terminal input and output operations
- These commands make use of predefined logical names
- Terminal input and output operations are used to:
 - Display messages and command output on the terminal screen
 - * — Prompt the user for input (like for a menu)
 - Redirect terminal output to a file
 - Allow the use of an interactive utility, such as an editor

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)
SY\$COMMAND	Initial input stream for your process	Terminal	Terminal
SY\$INPUT	Default input stream for your process	Terminal	Command Procedure File
SY\$OUTPUT	Default output stream for your process	Terminal	Terminal
SY\$error	Default file to which the system writes error messages	Terminal	Terminal

Performing Terminal Input and Output

Table 7-2: Displaying Information on the Terminal

Command/Example	Comments
\$ WRITE SYSS\$OUTPUT string \$ WRITE SYSS\$OUTPUT "Hello"	Character strings are enclosed in quotation marks.
\$ WRITE SYSS\$OUTPUT symbol \$ WRITE SYSS\$OUTPUT FILENAME	The symbol's value is automatically substituted.
\$ TYPE SYSS\$INPUT text text text \$	Information to be displayed follows the TYPE command. A dollar sign marks the end of the information.
\$ TYPE SYSS\$INPUT MENU CHOICES: 1. Add a user 2. Remove a user 3. List users \$	

Example 7-3: A Sample of Output from a Command Procedure

```
$ !                               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 SYSS$OUTPUT ""
$ WRITE SYSS$OUTPUT "Changing your default directory"
$ !
$ ! Set your default to the correct subdirectory
$ !
$ SET DEFAULT DISK1:[REPORTS.MONDAY]
$ !
$ WRITE SYSS$OUTPUT ""
$ WRITE SYSS$OUTPUT "Printing the Monday report"
$ !
$ ! Print out the report for Monday
$ !
$ PRINT MONDAY.RPT
$ !
$ ! Return to your login device and directory
$ !
$ WRITE SYSS$OUTPUT ""
$ WRITE SYSS$OUTPUT "Changing back to your login directory"
$ !
$ SET DEFAULT SYSS$LOGIN
$ EXIT
```

Execution of REPORT2.COM:

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

Table 7-3: Getting Information from the User

Command/Example	Comments
*\$ INQUIRE symbol "prompt"	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 a local symbol. If no prompt string is supplied, the symbol name is used as the prompt.
<code>\$ INQUIRE NAME "Filename"</code>	
\$ READ/PROMPT=string SYS\$COMMAND symbol	The user's response is taken as is and stored in the local symbol.
<code>\$ READ/PROMPT="Filename: " SYS\$COMMAND NAME</code>	

Table 7-4: Redirecting Input and Output

Command/Example	Comments
\$ ASSIGN/USER_MODE SYSS\$COMMAND SYSS\$INPUT <i>or</i> \$ DEFINE/USER_MODE SYSS\$INPUT SYSS\$COMMAND	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.
\$ @command_file-name/OUTPUT=output_file-name \$ @COMFILE.COM/OUTPUT=COM_STAT.DAT	Redirects output to the file you specify.
\$ ASSIGN/USER_MODE output_file_name SYSS\$OUTPUT <i>or</i> \$ DEFINE/USER_MODE SYSS\$OUTPUT output_file-name \$ DEFINE/USER_MODE SYSS\$OUTPUT COM_STAT.DAT	Redirects the output stream to the file you specify while the next image is executing.

Example 7-4: Using Terminal Input and Output

```
$!  
$!                                     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 SYSS$OUTPUT " "  
$ WRITE SYSS$OUTPUT "Enter your message. Press CTRL/Z when done."  
$ WRITE SYSS$OUTPUT " "  
$!  
$! Redirect the logical SYSS$INPUT from the command  
$! procedure to the terminal.  
$!  
$ ASSIGN/USER_MODE SYSS$COMMAND SYSS$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
```

Symbol Substitution

- In a command procedure symbols can be used as
 - Command synonyms
 - Parameters
 - Variables
- The system must translate symbols into their corresponding values
- Some DCL commands automatically replace symbols with their values
- Most DCL commands do not perform automatic symbol substitution
- To force symbol substitution
 - Enclose the symbol name in apostrophes (')
 - In a character string, precede the symbol with two apostrophes (") and end the symbol with a single apostrophe (')
- See Table 7-5 for examples

Table 7-5: Symbol Substitution Techniques

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

Example 7-5: Using Symbol Substitution

```
$ !                               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 SYSS$OUTPUT ""
$ WRITE SYSS$OUTPUT "Changing your default directory"
$ !
$ ! Set your default to the correct subdirectory
$ !
$ SET DEFAULT DISK1:[REPORTS.'DAY']
$ !
$ WRITE SYSS$OUTPUT ""
$ WRITE SYSS$OUTPUT "Printing the ''day' report"
$ !
$ ! Print out the report for the correct day
$ !
$ PRINT 'DAY'.RPT
$ !
$ ! Return to your login device and directory
$ !
$ WRITE SYSS$OUTPUT ""
$ WRITE SYSS$OUTPUT "Changing back to your login directory"
$ !
$ SET DEFAULT SYSS$LOGIN
$ EXIT
```

Execution of REPORT3.COM:

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

* PASSING PARAMETERS TO COMMAND PROCEDURES

Parameters

- Parameters are the objects of DCL commands
- Parameters can be
 - Keywords
 - File specifications
 - Integer or string values
- You can specify parameters for a command procedure at execution time

Local Symbols P1 – P8

- The system automatically provides eight local symbols: P1 through P8
- These symbols are initially assigned null values

Passing Parameter Values to a Command Procedure

- If you specify parameters when you execute the command procedure, the system
 - Assigns the values you specify to the symbols P1 – P8
 - Maintains the null value if you do not specify a parameter
- Syntax:

* \$ @command_procedure.com parameter_1 parameter_2 ... parameter_8 !

↑
PUT
IN
ACTUAL
VALUE

* RETRIEVE VALUES WITH "P1" OR "P2"
ETC. BASED ON ORDER USED IN
@COMMAND-PROCEDURE.COM

Example 7-6: Passing Parameters to Command Procedures

```

$ !
$ !                                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 SYSS$OUTPUT ""
$ ! WRITE SYSS$OUTPUT "Changing your default directory"
$ !
$ ! Set your default to the correct subdirectory
$ !
$ ! SET DEFAULT DISK1:[REPORTS.'P1']
$ !
$ ! WRITE SYSS$OUTPUT ""
$ ! WRITE SYSS$OUTPUT "Printing the 'P1' report"
$ !
$ ! Print out the report for the correct day
$ !
$ ! PRINT 'P1'.RPT
$ !
$ ! Return to your login device and directory
$ !
$ !
$ ! WRITE SYSS$OUTPUT ""
$ ! WRITE SYSS$OUTPUT "Changing back to your login directory"
$ !
$ !
$ ! SET DEFAULT SYSS$LOGIN
$ !
$ ! EXIT

```

GIVES
A BACK LINE!

TELLS US THAT WE DON'T

WANT TO PRINT THE 'P1' BUT TO SUBSTITUTE FOR THAT SYMBOL!

Execution of REPORT4.COM

```

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

```

ACTUAL PARAMETER (= P1)

DON'T NEED ".COM" BECAUSE @ IS RESERVED ONLY FOR .COM FILES!

TO PASS PARAMS. BETWEEN COM FILES:

```

FIRST.COM
$ WRITE SYSS$OUTPUT P1
$@ A.COM 'P1'

```

(using C quotes in the constant arg!)

CAN ALSO:

@A.COM STRING
(BUT OF COURSE THIS IS LESS USEFUL!)

CONTROLLING PROGRAM FLOW

- Normally commands are executed sequentially in a command procedure
- Control flow statements allow you to alter the order of execution
- Control flow commands include
 - The **IF** command
 - The **GOTO** command *- ALLOWS FOR INFINITE LOOP WHEREAS OTHERWISE THE @FN.FL ONLY CAN GO FOR 25 TIMES BEFORE AN ERROR.*

The IF Command

- Formats:

```
$ IF conditional expression THEN command
```

```
$ IF conditional expression  
$     THEN command  
$     command  
$ ENDIF
```

```
$ IF conditional expression  
$     THEN command  
$     command  
$     ELSE command  
$ ENDIF
```

EX: WITH MENU DRIVEN
\$ INQUIRE OPTION "CHOOSE AN OPTION"
\$ IF OPTION EQ. 1
\$ THEN SHOW DATA
\$ ENDIF

- Accepts multiple statements for execution when the condition is true
- The conditional expression is tested
 - If the condition is met, the command(s) following **THEN** are performed
 - If the condition is not met, the next DCL command in sequence is performed or an optional **ELSE** statement can be performed
- The command(s) following **THEN** or **ELSE** can be
 - A **GOTO** command
 - Another DCL command

THE IF-THEN-ELSE COMMAND

- The optional **ELSE** parameter provides command(s) to be performed when the **IF** condition is false
- The command(s) following **ELSE** can be any valid DCL command(s)
- Syntax: **\$ IF conditional expression THEN command ELSE command(s)**

Restrictions to 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 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

The GOTO Command

- Syntax:
\$ GOTO label
- No conditional testing is performed
- Control is transferred to the specified label

Table 7-6: 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 (collating sequence).
.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.	Tests for the opposite of a given condition.
.AND.	Tests if both of two conditions are met.
.OR.	Tests if one of a group of conditions is met.

Example 7-7: Controlling Program Flow

```
$!                               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 SYSS$OUTPUT " "
$ WRITE SYSS$OUTPUT "This procedure deletes an emptied directory"
$ WRITE SYSS$OUTPUT "The .DIR file extension is assumed."
$ WRITE SYSS$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 SYSS$OUTPUT " "
$ WRITE SYSS$OUTPUT "Directory file not deleted."
$!
$ END:
$ EXIT
```

Execution of DEL_DIR.COM:

Example 7-7: Controlling Program Flow (Cont)

```
$ @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.
$
```

LEXICAL FUNCTIONS

- Lexical functions provide information about an item or list of items
- The information is returned in a symbol that can then be used in a command procedure
- Lexical functions return integer or character strings, depending on the lexical function

Format and Syntax

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

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

- All lexical functions require parentheses, even with null arguments

- Integer or character strings:

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

- Symbols:

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

- Keywords:

```
WHERE = F$TRNLNM("SYSDISK")
```

- Null arguments:

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

- Multiple arguments are separated by commas
- Optional arguments, when omitted, are indicated by commas
- Table 7-7 describes some lexical functions
- Examples 7-8 and 7-9 demonstrate the use of some lexical functions

* **Table 7-7: Frequently Used Lexical Functions**

Lexical Function	Description
F\$TIME()	Returns the current date and time string.
F\$PROCESS()	Returns the current process name.
<u>F\$MODE()</u>	<u>Returns a character string indicating the mode in which a process is running (INTERACTIVE, BATCH, or OTHER).</u>
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\$GETSYI(item,[node])	Invokes the \$GETSYI System Service to return status and identification information about your system, or a node in your cluster.
F\$ENVIRONMENT(item)	Returns information about the DCL command environment (PRIVILEGES, DEVICE, and DIRECTORY).
F\$GETQUI()	Returns information regarding queues and the batch and print jobs currently in those queues.

Example 7-8: Using Lexical Functions with the INFO.COM Command Procedure

```
$!                               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 7-9: Using Lexical Functions with the PRINT.COM Command Procedure

```
$ !
$ !          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"
$
$ LENGTH=F$LENGTH(P1)
$ IF LENGTH .EQ. 0 THEN GOTO NAME_FILE
$
$ PERIOD=F$LOCATE(".",P1)
$ 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
$ !
$ IF LENGTH .EQ. PERIOD
$     THEN FTYPE=".LIS"
$     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 SYSS$OUTPUT "YOU CAN'T 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 the procedure
$ !
$ PRINT/COPIES='P2' 'FNAME' 'FTYPE'
$
$ EXIT
```

gets rid of colon!

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

MODULE 8

USING DISK AND TAPE VOLUMES

INTRODUCTION

In addition to your default disk device, your system includes a number of tape devices and disks. You can use one of these devices whenever you wish to store copies of files on a private volume. Private volumes can be created on disks or tapes. Private volumes are used to preserve files, transfer files from one system to another, and provide more space on a system (system quotas).

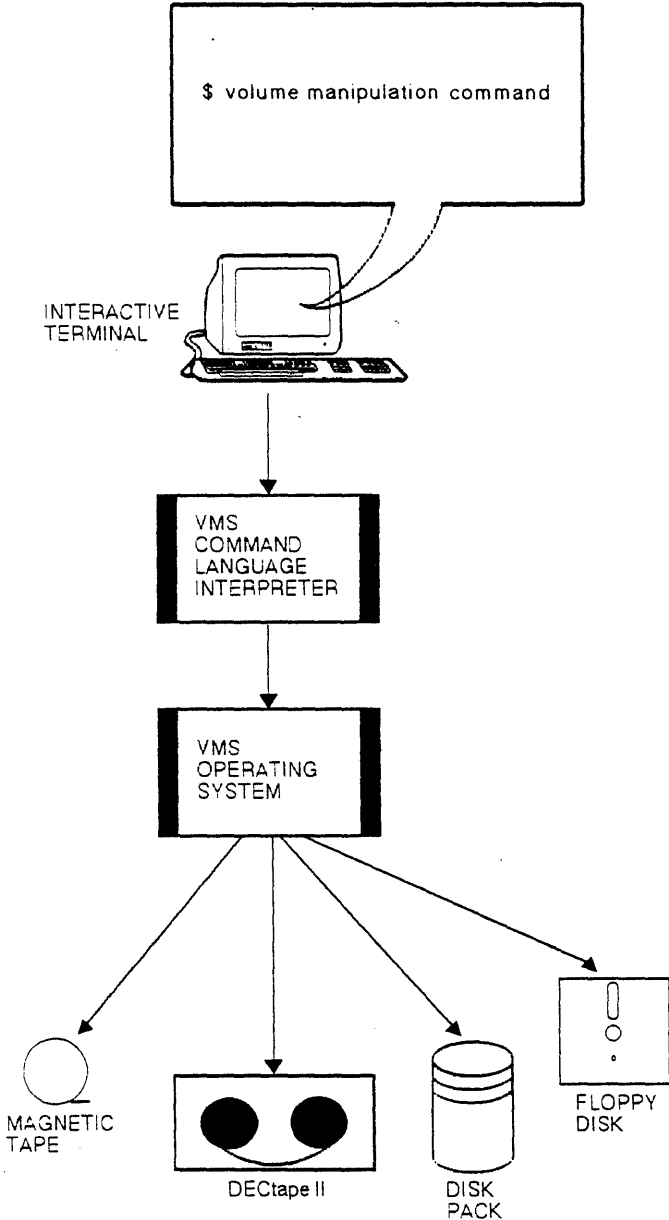
OBJECTIVES

This module introduces the steps and commands required to create and use private volumes.

RESOURCES

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

Figure 8-1: Volume Manipulation Commands



TTB_X0336_08_S

CREATING PRIVATE VOLUMES: THE COMMAND SEQUENCE

The following table lists DCL commands used to create and access disk and tape volumes.

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

Operation	Comments
Allocate a device	Allocates a device for exclusive use. The logical name DISK is placed in your process logical name table and assigned the name of the allocated device. Other users are unable to access the device.
\$ ALLOCATE device [logical-name] \$ ALLOCATE DM DISK	
Initialize a tape or disk	Builds the appropriate disk structure on the volume. Establishes volume ownership and protection. Usually used for new volumes.
\$ INITIALIZE device label \$ INITIALIZE DMA2: TEST_DISK	
Make the volume accessible to you	You can access the device as well as manipulating files on the volume. Logical names are often used.
\$ MOUNT device label [logical-name] \$ MOUNT DMA2: TEST_DISK DISK	
Prohibit further access to the volume	Closes all open files. Dismounts and unloads the volume. Deletes the logical name assignment made by the MOUNT command.
\$ DISMOUNT device \$ DISMOUNT DMA2:	
Deallocate a device	Frees the device for use by other users. Does not delete a logical name assigned by the ALLOCATE command.
\$ DEALLOCATE device \$ DEALLOCATE DMA2:	

MOUNTING A VOLUME WITH AN UNKNOWN LABEL

- MOUNT command format:
\$ MOUNT/OVERRIDE=IDENTIFICATION device-name volume-label logical-name
- Requirements are:
 - Volume ownership or
 - VOLPRO privilege

Example 8-1: Mounting a Disk with an Unknown Label

```
$ MOUNT/OVERRIDE=IDENTIFICATION DM: UNKNOWN MYDISK
%MOUNT-I-MOUNTED, MYVOL          mounted on DMA0:
$ 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.

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
- A save set can exist on a tape or disk
- When used with tape volumes, BACKUP can create and gain access to save sets only

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 can 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 demonstrates how to create a save set on a tape.

Example 8-3 shows how to transfer files from a disk to tape.

Example 8-4 illustrates how to restore files from a tape to a disk.

Example 8-2: Creating Save Sets on a Tape

```
$ SET DEFAULT [SMITH]
$ ALLOCATE MTAO:
$ INITIALIZE MTAO: SOURCE
$ MOUNT/FOREIGN MTAO:
$ BACKUP/IGNORE=LABEL_PROCESSING [...] MTAO:MY_BACKUP.BCK
$ DISMOUNT MTAO:
$ DEALLOCATE MTAO:
```

Example 8-3: Transferring Files to a Tape

```
$ ALLOCATE MUAO:
%DCL-I-ALLOC, _BROWNY$MUAO: allocated
$ INITIALIZE MUAO: SOURCE
$ MOUNT/FOREIGN MUAO:
%MOUNT-I-MOUNTED, SOURCE mounted on _BROWNY$MUAO:

$ 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 *.*;* MUAO:FOR.BCK
$ SET DEFAULT [SMITH.PASCAL]
$ BACKUP/IGNORE=LABEL_PROCESSING *.*;* MUAO:PAS.BCK
$ BACKUP/REWIND/LIST MUAO:PAS.BCK
```

Example 8-3: Transferring Files to a Tape (Cont)

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:      BROWNY::
Written on:     _BROWNY$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:      BROWNY::
Written on:     _BROWNY$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 8-4: Restoring Files from a Tape to a Directory

```
$ MOUNT/FOREIGN MUA0:
%MOUNT-I-MOUNTED, SOURCE mounted on _BROWNSMUA0:

$ DIRECTORY [SMITH.FORTRAN]
%DIRECT-W-NOFILES, no files found

$ DIRECTORY [SMITH.PASCAL]
%DIRECT-W-NOFILES, no files found

$ SET DEFAULT [SMITH.FORTRAN]

$ BACKUP/IGNORE=LABEL_PROCESSING MUA0:FOR.BCK *.*;*

$ DIRECTORY

Directory DISK:[SMITH.FORTRAN]
EXAMPLES.FOR;1      FILES.FOR;1      TEXT.FOR;1
Total of 3 files.

$ SET DEFAULT [SMITH.PASCAL]

$ BACKUP/REWIND/IGNORE=LABEL_PROCESSING MUA0:PAS.BCK *.*;*

$ DIRECTORY

Directory DISK:[SMITH.PASCAL]
EXAMPLES.PAS;1      FILES.PAS;1      TEXT.PAS;1
Total of 3 files.
```

SUMMARY

Creating Private Volumes: The Command Sequence

The following commands are used to create and access disk and tape volumes.

Operation	Comments
Allocating a Device	Allocates a device for exclusive use.
\$ ALLOCATE device [logical-name]	
Initialize a tape or disk	Establishes volume ownership and protection.
\$ INITIALIZE device label	
Make the volume accessible to you	You can access the device as well as manipulating files on the volume.
\$ MOUNT device label [logical-name]	
Prohibit 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.

MODULE 9

SUBMITTING BATCH AND PRINT JOBS

INTRODUCTION

The **PRINT** command allows you to obtain a hardcopy version of a file. Usually your print job must wait in an orderly list of print requests called a *queue*. The system uses factors such as the priority and size of your job to determine how long your job waits before printing. This does not affect your terminal session because as soon as you issue the **PRINT** command your terminal is freed up so you can do other jobs. Commands are provided so you can check on your job's progress in the queue and determine when it has completed.

The VMS system provides a similar facility for queuing command procedures for execution. Until now, you have run command procedures interactively. They process as though you were typing in each command. However, you can create a *batch process* to execute a command procedure independently of your interactive process. The **SUBMIT** command allows you to do this. The VMS system also determines when sufficient system resources are available for processing a job from the *batch queue*, and begins to process one or more jobs from that queue. You do not need to be logged in for your batch job to execute.

This module discusses the **PRINT** and **SUBMIT** commands and their qualifiers. These commands have different functions. However, they have several concepts and qualifiers in common.

OBJECTIVES

To effectively handle batch and print jobs, you should be able to perform the following operations:

- Print one or more files.
- Submit command procedures to be executed as a batch job.
- Display and modify the status or characteristics of print and batch jobs.
- Delay processing of batch or print jobs.
- Delete a batch or print job from its queue.

RESOURCES

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

PRINTING A FILE

PRINT Command in DCL

- The PRINT command uses a default file type of LIS
- Job numbers indicate the order in the queue - SUBMIT ORDER AND PRINT ORDER IS DETERMINED BY PRIORITY ✓
- The print queue, named SYS\$PRINT, handles print requests by default
- The first available printer prints the job

Example 9-1: Issuing the PRINT Command

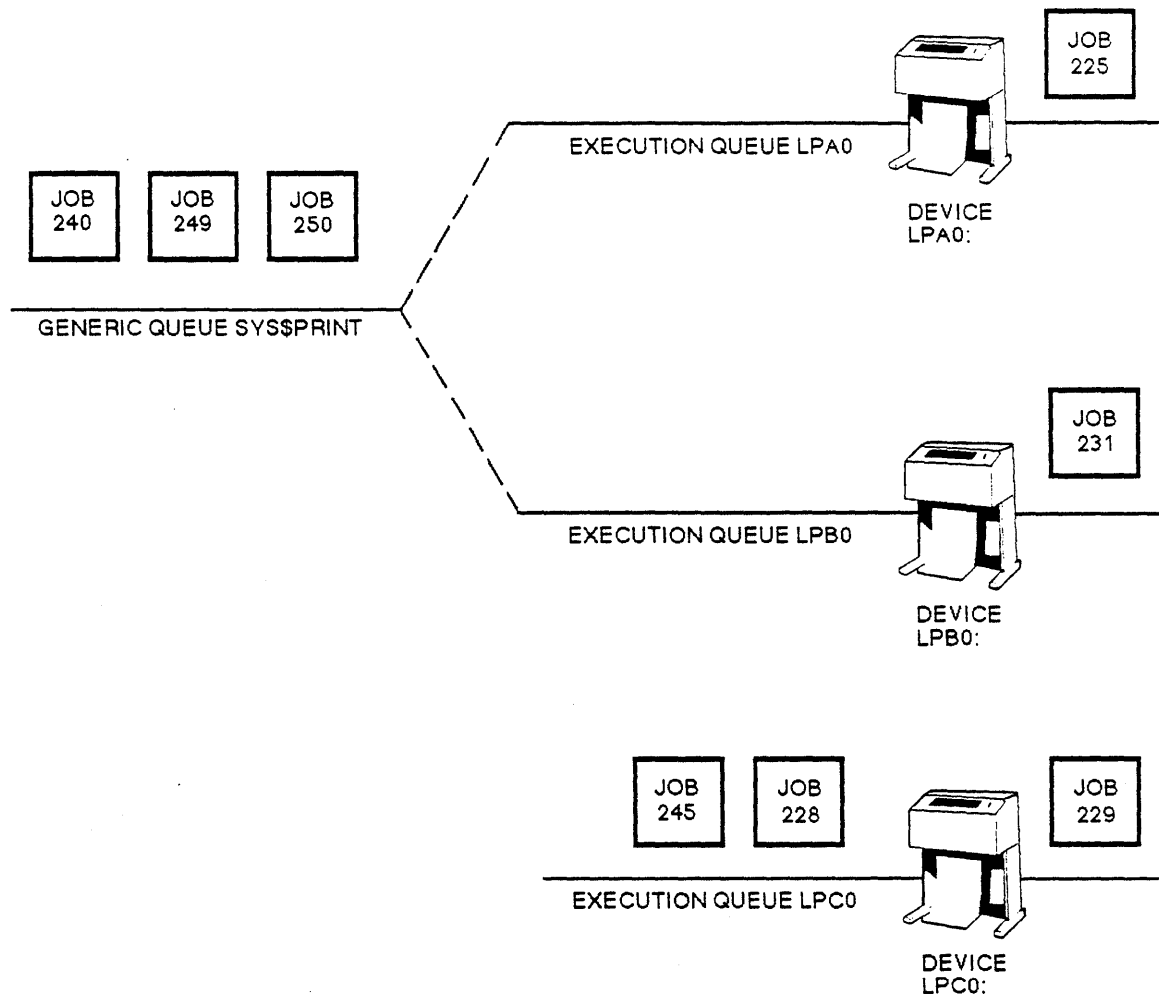
```
$ PRINT MYFILE.TXT  
Job MYFILE (queue SYS$PRINT, entry 456) started on LPA0  
$
```

Types of Print Queues

- Execution queue
 - Associated with each printer
 - Usually has the same name as the physical device name
 - Responsible for the actual printing of jobs
- Generic queue
 - Responsible for the distribution of print jobs to printers with similar characteristics
 - Holds jobs until the first available execution queue is free
- To specify a particular queue, use the **/QUEUE** qualifier

PRINT/QUEUE = LPBS SM.SI

Figure 9-1: Execution and Generic Print Queues



TTB_X0338_88_S

Qualifiers for the PRINT Command

- Number of copies

/COPIES

- Defaults to one copy
- Number of copies can be 1 – 255

- Spacing of the print job

/[NO]SPACE

- For single spacing use **/[NO]SPACE** (Default)
- For double spacing use **/SPACE**

- Whether the system notifies you when the job is completed or aborted

/NOTIFY

- **/[NO]NOTIFY** is the default

- Number of times your complete job is printed

/JOB_COUNT

- A value from 1 to 255
- Default is one printing

- Number of pages to print

/PAGES=([lowlim,],uplim**)**

- **lowlim** = First page to be printed
- **uplim** = Last page to be printed

- Time job is released to print

/AFTER=time

- Default is current date and time
- Time can be specified as absolute time, or a combination of absolute and delta time

Table 9-1: Printing Jobs with Different Characteristics

Operation	Example
Requests that two copies of the file MEMO.TXT be printed.	\$ PRINT/COPIES=2 MEMO.TXT
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
Requests three printings of the file MEMO.TXT.	\$ PRINT/JOB_COUNT=3 MEMO.TXT
Requests a double-spaced copy of the file MEMO.TXT.	\$ PRINT MEMO.TXT/SPACE
Prints pages 6 through 8.	\$ PRINT/PAGES=(6,8) MYFILE.TXT
Prints page 6 through the end of file.	\$ PRINT/PAGES=(6,"") MYFILE.TXT
Releases the file MEMO.TXT for printing at 6 p.m. on the current date.	\$ PRINT/AFTER=18:00 MYFILE.TXT

OBTAINING STATUS OF QUEUES

- Format:

\$ SHOW QUEUE/qualifier [queue-name]

- Example:

```
$ SHOW QUEUE/ALL_ENTRIES SYSS$PRINT
```

```
Terminal queue SYSS$PRINT, on BROWNY::$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
```

- Format:

\$ SHOW QUEUE/BY_JOB_STATUS[=keyword[,...]] [queue-name]

Keywords for the **BY_JOB_STATUS** qualifier include:

- **EXECUTING** (Displays executing jobs)
- **HOLDING** (Displays jobs on hold)
- **PENDING** (Displays pending jobs)
- **RETAINED** (Displays jobs retained in queue after execution)
- **TIMED_RELEASE** (Displays jobs on hold until a specified time)

- Example:

```
$ SHOW QUEUE/BY_JOB_STATUS=TIMED_RELEASE SYSS$PRINT
```

```
Terminal queue SYSS$PRINT, on BROWNY::$PRINTER, mounted form DEFAULT
Jobname      Username      Entry      Blocks      Status
-----      -
MYFILE       SMITH         96         1         Holding until 2-DEC-1987 15:00
```

- Format:

\$ SHOW QUEUE/DEVICE=[keyword[,...]] [queue-name]

- Keywords for /DEVICE qualifier include:

- **PRINTER** (Displays all print queues)
- **SERVER** (Displays all server queues)
- **TERMINAL** (Displays all terminal queues)

- Example:

```
$ SHOW QUEUE/DEVICE=SERVER
Server queue BROWNY$NARROW, stopped, on BROWNY::, mounted form DEFAULT
Jobname      Username      Entry  Blocks  Status
-----      -
MYFILE       SMITH         97     1      Holding until 2-DEC-1987 15:00
Server queue BROWNY$WIDE, stopped, on BROWNY::, mounted form DEFAULT
```

- Format:

\$ SHOW ENTRY [entry-number] /[qualifier]

- Example:

```
$ SHOW ENTRY 96
Jobname      Username      Entry  Blocks  Status
-----      -
MYFILE       SMITH         96     1      Holding until 2-DEC-1987 15:00
On terminal queue SYS$PRINT
```

- Example:

```
$ SHOW ENTRY 96/FULL
Jobname      Username      Entry  Blocks  Status
-----      -
MYFILE       SMITH         96     1      Holding until 2-DEC-1987 15:00
On terminal queue SYS$PRINT
Submitted 2-DEC-1987 09:18 /FORM=DEFAULT /PRIORITY=100
_DISK:[SMITH]MYFILE.TXT;1
```

Example 9-2: Queue Status Display Corresponding to Figure 9-1

\$ 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

Queue Status List

Example 9-3: Full Format Queue Status Display

```
$ SHOW QUEUE/DEVICES/FULL/ALL_ENTRIES
```

```
Terminal queue COMP, on SUPER::SUPER$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 SUPER::SUPER$LPA0:, mounted form DEFAULT  
/BASE_PRIORITY=4 /DEFAULT=(FEED,FORM=DEFAULT)  
/LIBRARY=SYSDEVCTL_LN01 Lowercase /OWNER=[GROUP1,SYSTEM]  
/PROTECTION=(S:E,O:D,G:R,W:W) /SEPARATE=(FLAG,RESET=(ANSI$RESET))
```

```
Server queue NMSQUE01, on SUPER::, mounted form DEFAULT  
/BASE_PRIORITY=4 /DEFAULT=(FEED,FORM=DEFAULT)  
/OWNER=[GROUP1,SYSTEM] /PROCESSOR=NMS$DAEMON /PROTECTION=(S:E,O:D,G:R,W:R)  
/RETAIN=ERROR
```

```
Generic printer queue NMSQUEUE  
/GENERIC=(NMSQUE01,NMSQUE02) /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-1987 11:26
Submitted 24-NOV-1987 11:16 /PRIORITY=100				
_ \$1\$DUA0:[SYSCOMMON.NMAIL]NMAIL\$1987112217065820.WRK;1				

Modifying a Print Job Already in the Queue

- Can change characteristics of your print job if it is not currently printing
- Can move your job to another queue
- Use the **SET ENTRY** command
- See the following table for examples

Table 9-2: Modifying Print Jobs in a Queue

Command	Comments
\$ SET ENTRY 100/COPIES=5	Changing the number of copies to five.
\$ SET ENTRY 80/RELEASE	Releasing a job that was previously held.
\$ SET ENTRY 95/REQUEUE=FASTJOBS	Moving a print job to another printer.

Deleting a Print Job

- Can delete a print job while it is printing or while it is pending in a queue
- May need to do this if you accidentally print a file with non-ASCII characters, such as an EXE or OBJ file
- Use the **DELETE/ENTRY** command
- Example:

```
$ DELETE/ENTRY=120 FASTJOBS
```

SUBMITTING A BATCH JOB

DCL SUBMIT Command

- The **SUBMIT** command uses a default file type of 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

Example 9-4: Issuing the SUBMIT Command

```
$ SUBMIT ACTION.COM  
Job ACTION (queue SYS$BATCH, entry 136) pending  
$
```

How a Batch Job Executes

- Batch job's relationship to job that submitted it
 - Runs independently
 - Uses same UAF characteristics
 - Executes same LOGIN.COM file
 - Uses same default device and directory for file access
- Batch job's SYS\$OUTPUT assigned to batch log file
 - Created in login default directory
 - File name is the same as the name of batch command procedure
 - File type is LOG
 - File is printed, then deleted on completion of batch job
- Logical name assignments for batch processes

Table 9-3: 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

Writing a Batch Command Procedure

- Two ways to run a command procedure
 - Interactive
 - Batch
- For command procedures running in batch, consider
 - The system's login command procedure is executed
 - Your login command procedure is executed

Use the **F\$MODE()** lexical function

Bypass symbol definitions (not used in command procedures)

Bypass commands that require a terminal, such as

INQUIRE
SET TERMINAL

- By default, severe errors terminate batch job execution
- The batch process's default directory is the one specified as **SYS\$LOGIN**
- Verification is on in batch process by default

Qualifiers for the SUBMIT Command

The **SUBMIT** command:

- The **/QUEUE** qualifier overrides the default system queue.
- The **/PARAMETERS** qualifier passes parameters to the command procedure.
- The **/LOGFILE** qualifier renames the log file. The default is the command file name with a file type of LOG.
- The **/PRINTER** qualifier redirects where the log file is printed. The default queue is SYS\$PRINT.
- The **/KEEP** qualifier retains a copy of your log file in your directory. The default action is to print the log file and then delete it.
- The **/AFTER** qualifier delays the execution of the job until a later time. The default is to place the job in the queue immediately.
- The **/NOTIFY** qualifier notifies you when the job completes or aborts. The default is **/NONOTIFY**.
- Refer to Table 9–4 for examples.

Table 9-4: Submitting Batch Jobs

Operation	Comments
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.
<pre>\$ SUBMIT ACTION</pre>	
Submitting a job to a specified queue	By default, the system batch queue SYS\$BATCH is used.
<pre>\$ SUBMIT/QUEUE=SLOWBATCH ACTION</pre>	
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.
<pre>\$ SUBMIT/AFTER=19:00 MYFILE.TXT</pre>	
Submitting a job that requires parameters	Up to eight parameters may be specified using symbols P1-P8. The symbols are local to the specified command procedures. P1 is 3 and P2 is SUM.
<pre>\$ SUBMIT/PARAMETERS=(3, SUM) MATH</pre>	
Changing the log file name	The log file is called WENDY.LOG instead of MYFILE.LOG.
<pre>\$ SUBMIT/LOGFILE=WENDY MYFILE</pre>	
Keeping the log file	The log file is queued to printer LPB0 instead of SYS\$PRINT. The log file is printed and retained in the user's login directory.
<pre>\$ SUBMIT/KEEP MYFILE</pre>	

Example 9-5: Sample Batch Run of COUNT1.COM

```
$ TYPE COUNT1.COM
$! COUNT1.COM
$!
$ SHOW TIME
$ SHOW LOGICAL/PROCESS/JOB
$ EXIT

$ SUBMIT COUNT1.COM
Job COUNT1 (queue SYSS$BATCH, entry 366) started on SYSS$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$COMMAND" = "_BROWNY$RTA1:"
"SYS$DISK" = "BROWNY$DJAO:"
"SYS$ERROR" = "_BROWNY$RTA1:"
"SYS$INPUT" [super] = "BROWNY$DJAO:"
"SYS$INPUT" [exec] = "_BROWNY$RTA1:"
"SYS$OUTPUT" [super] = "_BROWNY$RTA1:"
"SYS$OUTPUT" [exec] = "_BROWNY$RTA1:"
"TT" = "RTA1:"

(LNM$JOB_803E1730)

"SYS$LOGIN" = "BROWNY$DJAO:[SMITH]"
"SYS$LOGIN_DEVICE" = "BROWNY$DJAO:"
"SYS$REM_ID" = "SMITH"
"SYS$REM_NODE" = "SUPER:."
"SYS$SCRATCH" = "BROWNY$DJAO:[SMITH]"
```

OBTAINING STATUS OF BATCH QUEUES

- Format:

\$ SHOW QUEUE/qualifier [queue-name]

Example 9-6: Full Format Queue Status Display

```
$ SHOW QUEUE/BATCH/FULL/ALL_ENTRIES HARDY_BATCH
```

```
Batch queue HARDY_SYSTEM, on HARDY::
```

```
  /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-1987 18:00
DRAFT	SMITH	1425	Holding until 4-DEC-1987 01:00
TEST	JONES	1352	Holding until 7-DEC-1987 00:00

```
Batch queue HARDY_BATCH, on HARDY::
```

```
  /BASE_PRIORITY=2 /JOB_LIMIT=3 /OWNER=[GROUP1,SYSTEM]
```

```
  /PROTECTION=(S:E,O:D,G:R,W:W)
```

Table 9–5: Displaying Batch Queue Status

Operation	Comments
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 <code>/ALL_ENTRIES</code> to the <code>SHOW QUEUE</code> command. For more job information, add the qualifier <code>/FULL</code> to either the <code>SHOW QUEUE</code> or <code>SHOW ENTRY</code> command.
<pre>\$ SHOW QUEUE/BATCH \$ SHOW QUEUE/BATCH/ALL_ENTRIES \$ SHOW QUEUE/BATCH/FULL \$ SHOW QUEUE/BATCH/FULL/ALL_ENTRIES \$ SHOW ENTRY/BATCH \$ SHOW ENTRY/BATCH/FULL</pre>	
Displaying a list of batch jobs on a particular queue	In any <code>SHOW QUEUE</code> command, you can specify a queue name instead of <code>/BATCH</code> . You can also use the qualifiers <code>/FULL</code> and <code>/ALL_ENTRIES</code> .
<pre>\$ SHOW QUEUE FASTJOBS \$ SHOW QUEUE/ALL_ENTRIES FASTJOBS</pre>	

Modifying a Batch Job Already in the Queue

- Can change job characteristics if it is not currently executing
- Can move a job to another queue
- Privileges required to affect jobs
 - Queued by you—None
 - Queued by processes in your UIC group—GROUP
 - Queued by anyone—WORLD or OPER

Table 9-6: Modifying a Batch Job

Operation	Comments
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.
<pre>\$ SET ENTRY 100/COPIES=5</pre>	
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.
<pre>\$ SET ENTRY 90/REQUEUE=WIDE</pre>	

DELETING A BATCH JOB

- Can delete a batch job while it is executing or while it is pending in the queue
- Use the **DELETE/ENTRY** command
- Example:

```
$ DELETE/ENTRY=120 WIDE
```

HANDLING BATCH AND PRINT JOBS

Characteristics Common to Both Batch and Print Jobs

- The name used to identify the job
- The VAX 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 system deletes the log file after the job completes

BATCH AND PRINT QUEUES ETIQUETTE

The following suggestions are given to insure that the VMS 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.
- Pick up your completed job promptly. Do not allow your finished jobs to sit in the printer area endlessly.

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

MODULE 10

DEVELOPING PROGRAMS

INTRODUCTION

This module presents a general discussion of the steps in developing a program on a VMS 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.

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 and 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.

System libraries contain a large number of predefined routines that user programs (such as routines that manipulate strings or generate random numbers) can call.

OBJECTIVES

Most of the programming languages available on a VMS system, involve 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

RESOURCES

For more detailed explanations of developing programs, refer to the following documents:

- *Guide to VMS Programming Resources*
- *VMS DCL Dictionary*

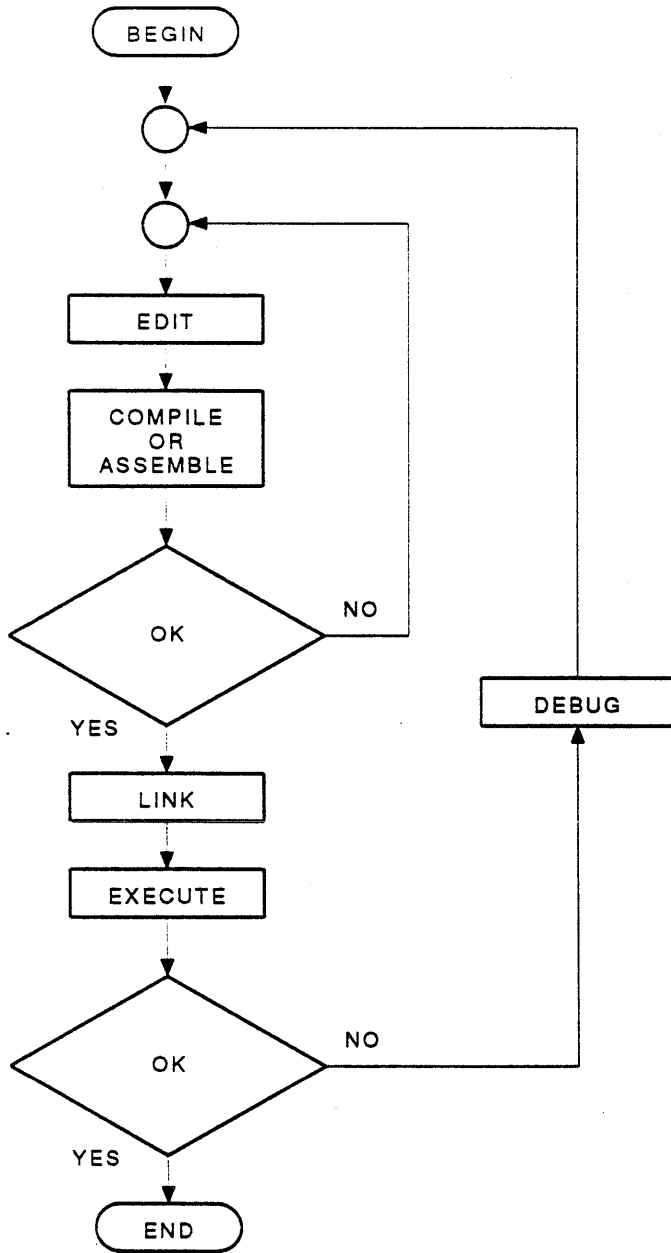
PROGRAM DEVELOPMENT ON A VMS SYSTEM

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.

Figure 10–1 illustrates the orderly flow of these five program development steps.

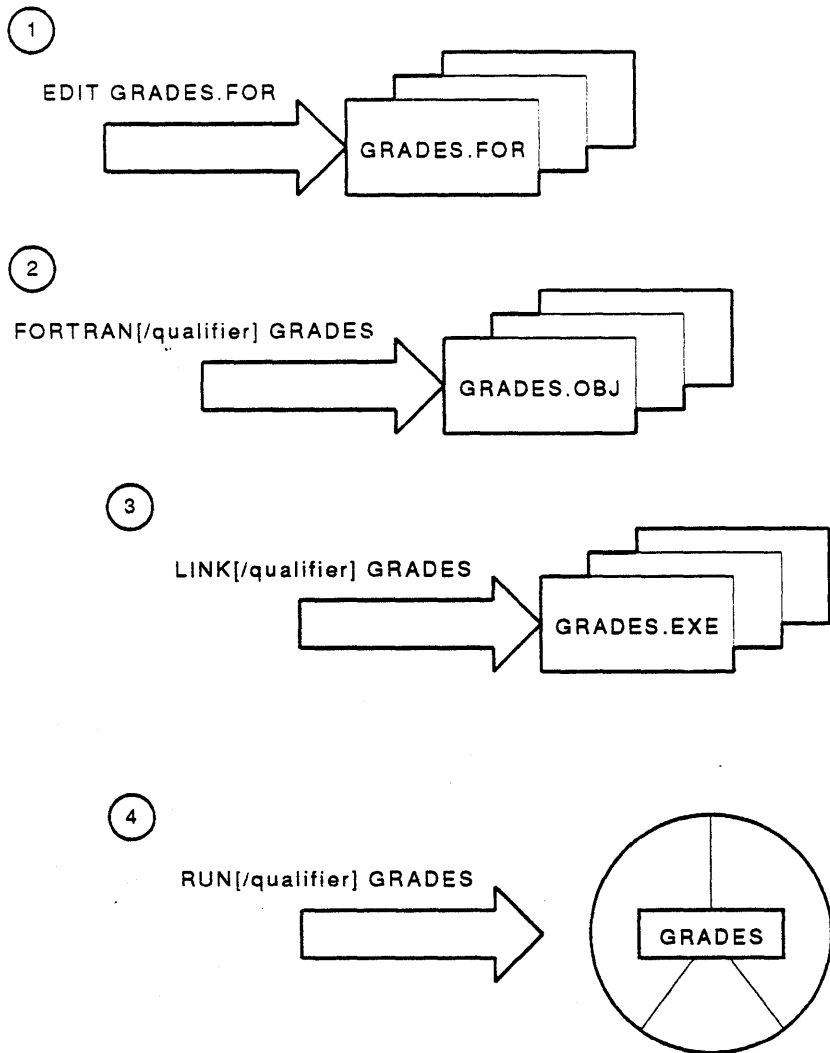
Figure 10-1: A Flow Diagram of the Five Major Programming Steps



TTB_X0261_88

Each of the five program development steps is discussed on the pages that follow. As you read each step, refer to Figure 10-2.

Figure 10-2: The Four Program Development Commands



TTB_X0339_88

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 a number of languages.

Table 10-1: Languages and Associated File Types

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 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

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.

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.

5. Debug the program to correct errors.

THE VMS SYMBOLIC DEBUGGER UTILITY

The VMS Symbolic Debugger simplifies the 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
- 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.
2. Halt the program and invoke the debugger with the DCL command **DEBUG**.
3. Run the program with the debugger.

To use the Help facility of the Debugger, invoke the symbolic debugger and enter the **HELP** command.

A SAMPLE PROGRAM – GRADES

The **GRADES** program (written in FORTRAN), contains the names of students and their grade averages 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: GRADES.FOR Source File

```
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
```

Execution of GRADES

Example 2 depicts a sample run of the GRADES program.

Example 10-2: Execution of GRADES

```
$ 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
$
```

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*

MODULE 11

EXERCISES

HARDWARE AND SOFTWARE OVERVIEW

WRITTEN EXERCISE I

In the exercise below, match each description with the appropriate component of the hardware environment. Components of the hardware environment may be used once, more than once, or not at all.

Hardware Components

- a. CPU
- b. Console Subsystem
- c. Main Memory
- d. I/O Subsystem

Descriptions

- 1. C Stores instructions and data
- 2. B Used to monitor and control the system
- 3. D Consists of peripherals
- 4. A Executes instructions
- 5. B Used for starting up and shutting down the system

Write the letter of the term that best completes each of the following statements.

1. C are used to connect the various subsystems of the computer.
 - a. Peripheral devices
 - b. Network communication devices
 - c. Interconnect devices
 - d. Storage devices

2. B have a screen for displaying information.
 - a. Hardcopy terminals
 - b. Video terminals
 - c. Laser printers
 - d. Mass storage devices

3. C is NOT a peripheral device.
 - a. Terminal
 - b. Printer
 - c. CPU
 - d. Disk drive

4. D are high-speed machines that are usually used for large quantities of stored output.
 - a. Hardcopy terminals
 - b. Disk drives
 - c. Laser printers
 - d. Line printers

5. A is NOT a type of disk.
 - a. Reel
 - b. Cartridge
 - c. Diskette
 - d. Disk pack

6. A, B record data on magnetic media.
 - a. Disk drives
 - b. Tape drives
 - c. Terminal servers
 - d. VAXcluster systems

WRITTEN EXERCISE II

The example on the following page displays the characteristics of a privileged process on your system. Using the information displayed in the example, determine the value of each of the following parameters:

1. VMS Account name
2. DSK1:SMITH Default Device and Directory Specification
3. VTA:5: Interactive Terminal Specification
4. NEVER SHOWS Password
5. 20400140 Process Identification Code
6. SMITH Process Name
7. (2040014, SMITH) User Identification Code 010
8. SMITH User Name
9. 4 Priority

Privileges (list them)

10. GRPNAM
11. GROUP
12. TMPMBX
13. NETMBX
14. —
15. —
16. —
17. —
18. INFINITE CPU Limit
19. 63 Open File Quota
20. 2 Subprocess Quota

SHOW PROCESS/ALL

Example 11-1: Process Parameters of a Sample Interactive Process

```
31-DEC-1987 13:45:39.54   VTA15:           User: SMITH
Pid: 20400140   Proc. name: SMITH           UIC: [GROUP11,SMITH]
Priority: 4     Default file spec: DISK:[SMITH]
Devices allocated: VTA15:
Process Quotas:
  Account name: VMS
  CPU limit:           Infinite   Direct I/O limit:           18
  Buffered I/O byte count quota: 12192   Buffered I/O limit:       18
  Timer queue entry quota:      10     Open file quota:         63
  Paging file quota:           9063   Subprocess quota:       2
  Default page fault cluster:   64     AST limit:              22
  Enqueue quota:              40     Shared file limit:      0
  Max detached processes:      1     Max active jobs:        0

Accounting information:
  Buffered I/O count:      21298   Peak working set size:    1500
  Direct I/O count:      11639   Peak virtual size:       1789
  Page faults:           26172   Mounted volumes:         0
  Images activated:       112
  Elapsed CPU time:      0 00:11:33.90
  Connect time:          0 04:58:58.78

Process privileges:
  GRPNAM               may insert in group logical name table
  GROUP                may affect other processes in same group
  TMPMBX               may create temporary mailbox
  NETMBX               may create network device

Process rights identifiers:
  INTERACTIVE
  LOCAL
  VMS
  SYS$NODE_SUPER

Process Dynamic Memory Area
  Current Size (bytes)      25600   Current Total Size (pages)  50
  Free Space (bytes)       21184   Space in Use (bytes)       4416
  Size of Largest Block    21072   Size of Smallest Block     56
  Number of Free Blocks     3       Free Blocks LEQU 32 Bytes  0
```

Match each of the following operations with the parameter that controls your ability to perform it. Some operations are controlled by more than one parameter.

Parameters

- a. Password
- b. Priority
- c. Privilege
- d. Process Identification Number (PID)
- e. Resource Limit
- f. User Identification Code (UIC)
- g. User Name

Operations

- 1. G, A Logging in to your system
- 11-6 2. F, C Deleting a file that belongs to another user
- 11-6 3. C Creating a group logical name
- 4. e Opening a large number of files

GETTING STARTED

LABORATORY EXERCISE I

If you have not already done so, obtain your user name and password from your instructor. Complete the following activities at an interactive terminal:

1. Log in to the system, using the user name and password assigned to you.
2. Type the following command lines at your terminal. After each command, press <RET>.
 - **SHOW TIME**
 - **SHOW USERS**
 - **SHOW TERMINAL**
3. Log out of the system.

LABORATORY EXERCISE II

1. Enter the following command line at your terminal:

```
PRODUCE NONESUCH.FIL
```

Since neither the command nor the parameter of the preceding command line exists, the operating system will display one or more error messages at your terminal.

- a. How severe was this error? *WARNING*
 - b. What part of the system produced this error message? *DCL*
2. Use the command line editor to recall the **PRODUCE** command and change it to the **TYPE** command. Now execute the command and observe the results.
 - a. How severe was this error? *ERROR*
 - b. What part of the system produced this error message? *RMS*
 - c. Did the message text differ from the previous exercise? *NO TEXT*

WRITTEN EXERCISE I

Match the letter of a special function key with each of the operations described below. You may not use every letter in the list.

Special Function Keys

- a. <CTRL/B>
- b. <CTRL/O>
- c. <CTRL/Q>
- d. <CTRL/R>
- e. <CTRL/S>
- f. <CTRL/U>
- g. <CTRL/Y>
- h. or <RUBOUT>
- i. <RET>

Operations

- *1. B You have logged in to your system. A long string of messages, all of which you have seen before, scrolls past on your screen. Suppress the messages, without stopping or aborting the program that produces them.
- 2. H You have just typed the string TYPE FILE&. The cursor is positioned immediately after the ampersand (&). Delete the ampersand (&).
- 3. G You have entered the **SHOW SYSTEM** command. A listing of users on your system scrolls past on your screen. Abort further execution of the command and return control to your terminal.
- *4. F You have entered the following command lines at your terminal:

```
$ DIFFERENCES/IGNORE=BLANK_LINES --<RET>  
_ $ FILE1 --<RET>  
_ $ FILE8
```

The cursor is immediately to the right of the number eight on the last line. Delete the last line, without deleting the preceding lines of the command string.

5. E You have entered the following string at your terminal:

```
$ SHOW PROCESS/ALL<RET>
```

Lines of information scroll past on your terminal screen. Stop the display and halt, but do not abort, the program that generates it.

6. C Resume generation of the display that you stopped in the preceding operation.

- * 7. D You have made extensive corrections to a command line at a hardcopy terminal. The output looks like this:

```
$ PRYNT\TNY\NT9\9\ FILIN\NI\
```

Display the line without the echoed corrections.

- * 8. A You have just issued a command line. Recall this command.

LABORATORY EXERCISE III

Log in to your system and use the online Help facility to obtain the information listed below. When you have finished your work, log out.

1. A listing of all topics available through the Help facility
2. A description of the login procedure
3. A description of the **/FULL** qualifier of the **LOGOUT** command
4. A description of the **TIME** option of the **SHOW** command

LABORATORY EXERCISE IV

1. Determine the following characteristics of your terminal:
 - a. Number of characters displayed in an output Line 80
 - b. Receive speed 9600
 - c. Transmit speed 9600
 - d. Terminal type (LA34, VT100, and so on) VT200
2. Determine the value of each of the following process parameters:
 - a. Account Name UC
 - b. CPU Time Limit WFM100
 - c. Default Directory Specification DISK\$STUDENT[UC13]
 - d. Default Device for Input and Output ?
 - e. Priority 4
 - f. Privileges TMPMBX, NETMBX
 - g. Process Identification Code 22C00933
 - h. Process Name UC13
 - i. User Identification Code [UC13]
 - j. User Name UC13 (UC1310)
3. Display the names of all processes running on your system. ?
4. Display the names of all users on your system. SHOW USERS
5. Display the names of all devices on your system. SHOW DEVICES
6. Log out of your system.

CREATING AND EDITING TEXT FILES

INTRODUCTION TO THE LABORATORY EXERCISES

Students should feel free to choose either the EDT Editor exercises or the EVE Editor exercises. They should choose the editor that they will be primarily using.

However, if they would like practice in both editors, they can complete all the exercises for this module.

LABORATORY EXERCISE I – THE EDT EDITOR

1. Use the EDT editor to create a text file named EXERCISE1.TEXT.
 - a. Invoke the EDT editor, using the appropriate DCL command.
 - b. Notice the message displayed on the terminal screen.
 - c. Change from Line mode to Keypad mode.
2. Before you begin entering text, you should become familiar with the Help facility that is a part of the EDT editor.
 - a. Invoke the Help facility by pressing the appropriate key.
 - b. Display information about specific keypad keys.
 - c. Exit from Help.
3. Type in the following text:

The purpose of this exercise is to allow
you practice using the basic capabilities
of the EDT Editor Utility.
4. Leave the editing session normally.
 - a. Return to Line mode.
 - b. Type in the command that ends the session and saves your actions.
 - c. Notice the system message displayed on the terminal screen.

5. Begin another editing session.
 - a. Invoke the EDT editor, using the appropriate DCL command.
 - b. Notice that the first line of the file is displayed on the terminal screen.
 - c. Change from Line mode to Keypad mode. The file's contents are displayed on the screen.
6. Modify the text.
 - a. Using the appropriate keys, move the cursor to the beginning of the word "basic" in the second line.
 - b. Delete the words "basic capabilities" and modify the line so that it reads:

```
you practice using the simpler functions
```
7. End the editing session.

LABORATORY EXERCISE II – THE EVE EDITOR

1. Use the EVE editor to create a file called EXERCISE3.TEXT.
 - a. Notice the messages that are displayed on the terminal screen.
2. Type in the lines listed below. DO NOT press the RETURN key while you are typing. The automatic word wrap feature causes new lines to begin when text reaches the right margin of the terminal screen.

Notice the automatic word wrap feature of EVE.

```
This is an exercise that uses the EVE editor.  
This editor allows you to type text into a file.  
The word wrap feature will automatically wrap lines as you type,  
so that you do not have to press the RETURN key at the end of each line.
```

3. End the editing session and save your work.
 - a. Use the appropriate key sequence or line-mode command to end the editing session.
 - b. Notice the system messages displayed on the terminal screen.
4. Begin another editing session.
 - a. Notice the system messages displayed on the terminal screen.
5. Modify the text of the first line to read:

This is an example that uses the EVE editor.

- a. Move the cursor to the beginning of the word "RETURN."
 - b. Use the appropriate key to delete the word "RETURN." *FIX*
 - c. Move the cursor to the beginning of the word "example."
 - d. Use the appropriate key to switch from Insert mode to Overstrike mode. *FIX*
 - e. Type over the words "example that uses" with the words "exercise that uses."
6. End the editing session normally.

LABORATORY EXERCISE III – THE EVE EDITOR

This exercise lets you practice editing more than one file on your terminal screen.

1. Edit a file of your choice.
2. Split your terminal screen into two windows. *TWO WINDOWS*
3. Edit another file of your choice. *GETFILE FN.FT*
4. Move text from one file into the other file. *INCLUDE FN.FT*
5. Exit the file so the moved text is saved.

COMMUNICATING WITH OTHER USERS

LABORATORY EXERCISE I

1. Invoke the Mail utility and send several mail messages to someone in your class.
2. Have your mail recipient send mail messages to you.
3. Read your messages.
4. Obtain a list of your mail messages.
5. Read only the second mail message.
6. Delete the fourth mail message.
7. Create a text file in your default directory. Send this file as a mail message.
8. Pick a message and move it to a folder named Test. Select the Test folder and check to see if the message is there.
9. List the folders you have. In addition to the folder you just created, what other folders do you have?
10. Create a distribution list for Mail. Include several members of your class. Send a short message to the people on the distribution list.

LABORATORY EXERCISE II

1. Invoke the Phone utility.
2. Obtain a list of available users. *IN PHONE TYPE "DIRECTORY"*
3. Establish a phone connection with one of the users.
4. Terminate all conversations. *% HANG-UP*

LABORATORY EXERCISE III

Send a request to a system operator using the **REQUEST** command.

MANAGING FILES

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 A.FOR;2 AREA.FOR;2
- b. *.FOR A.FOR;2 AREA.FOR;2 AREA.FOR;1 C.FOR;1 B.FOR;1
- c. A*;* A.DAT;1 A.FOR;2 AREA.FOR;2 AREA.FOR;1
- d. A%%%.** AREA.FOR;1 AREA.FOR;2
- e. %.DAT A.DAT;1 B.DAT;3 C.DAT;4
- f. *.* all!

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

- a. A.DAT;1, A.FOR;2 A.*
- b. A.DAT;1, B.DAT;3, C.DAT;4 *.DAT
- c. MAILD22.DAT;2, MAILJ14.DAT;1, MAILF22.DAT;2 MAIL*.DAT
- d. A.DAT;1, MAILJ14.DAT;1 *.DAT;1

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. K Display the contents of a file at your terminal.
2. F Display the contents of your default directory at your terminal.
3. COPY 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 file name to a new file 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.

LABORATORY EXERCISE I

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

p. 5-30
ex. 5-4

O.K. }

- 1) Directory must be empty
- 2) protection set appropriately
- 3) Delete file in parent directory!

LABORATORY EXERCISE II

Dir/200 FN,ft

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? *(RWED,RWED,RC,)*, Default
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) FN,ft*
3. Delete this file. What happened and why? *DENIED ACCESS, NO PRIVILEGE*
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?
SET PROTECTION=(S:R,O:RWED,G:R,W:R) FN,ft
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

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 each command more than once; you may 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. 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.

LABORATORY EXERCISE III

1. Choose a file in your directory. Issue a DCL command to obtain Access Control List information regarding that file. *SHOW ACL \$N.\$T*
2. Modify the UIC protection on the above file so that your group has no access. *SET PROTECTION=(O) \$N.\$T*
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. *EDIT/ACL P.5-34*
5. Delete the ACL on the above file.

SET ACL/DELETE \$N.\$T.

WRITTEN EXERCISE IV

Each of the following questions describes an operation a user wishes 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. F A user with a UIC of [100,200] wishes to delete a file on a tape volume.

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

NOT POSSIBLE!

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

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

SAME GROUP

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

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

success!

4. T A user with a UIC of [100,200] wishes 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. F A user with a UIC of [521,6] wishes 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)

*P. 11-75
see EXPLANATIONS!*

CUSTOMIZING THE USER ENVIRONMENT

WRITTEN EXERCISE I

Write the letter of the system-defined logical name below 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. BK 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. ACRF 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.

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. *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*
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. *DEFINE NAME [UCR]*
 - 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.

DEASSIGN NAME

LABORATORY EXERCISE II

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

1. Create a subdirectory.
2. Create a logical name for your newly created subdirectory.
`DEFINE LOGNAME [LOGS.SUBDIR]`
3. Create a logical name for a text file in your default directory.
`DEFINE NAME LOGNAME:FN.FI`
4. Check the process logical name table to see if your new logical names exist.
5. Using only logical names, move the text file into your new subdirectory.
6. After completing this exercise, remove the above logical names.

```
DEASSIGN NAME  
DEASSIGN LOGNAME
```

WRITTEN EXERCISE II

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

Symbolic Name Types

- a. Command Synonym
- b. Logical Name

Characteristics

- 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

LABORATORY EXERCISE III

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

1. Display a directory listing along with sizes 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. Try the symbols to see if they work. How can you get rid of the symbols without using the **DELETE/SYMBOL** command?

LABORATORY EXERCISE IV

1. Define <KP2> to be the **SHOW TIME** command. Try it first without the **/TERMINATE** qualifier, then with the **/TERMINATE** qualifier. What is the difference?
2. Define <KP3> to be the **SET DEFAULT** command. Create a subdirectory. Try to move to the subdirectory by using your newly defined key.
3. Delete the key definition for <KP2>. See if it worked by displaying all your key definitions again.

WRITING COMMAND PROCEDURES

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 SYSS\$OUTPUT COUNT " copies of the file"
3. \$ IF COUNT .LT. 10 THEN GOTO END
4. \$ WRITE SYSS\$OUTPUT "The file ''FILE_NAME'' 'FILE_TYPE'"

Part B:

In the commands below, replace the underlined text with symbols, using the proper symbol substitution techniques.

To complete these exercises, use the following symbol definitions:

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

Example:

```
$ PRINT MYFILE.TXT  
$ PRINT 'P1'
```

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

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 will 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

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 procedure of your own that performs the following actions:

1. Exit if the process mode is not interactive. Use the lexical function F\$MODE() to test the mode of the process.

IF F\$MODE().NES. "INTERACTIVE" THEN EXIT

2. Define 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. Define global symbols to be used as command synonyms. These command synonyms should perform the following actions:

- a. Set default ✓

- b. Show all users currently logged in to the system ✓

- c. Display your current directory ✓

- d. Set your default to your login disk and directory ✓

4. Display the following information on your terminal:

- a. The current date and time ✓

- b. The current default directory ✓

LABORATORY EXERCISE II

Write a command procedure that allows you to create files that everyone on your system can access. The procedure performs 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

OPTIONAL LABORATORY EXERCISE

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

1. Asks you for the number of minutes you will be away.
2. Erases the screen and then displays the 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 redisplay 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 SYSS\$OUTPUT
- TYPE SYSS\$INPUT

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

This command procedure does not use lexical functions.

USING DISK AND TAPE VOLUMES

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 preceding each step.

1. _____ Allocate device
2. _____ Deallocate device
3. _____ Dismount volume
4. _____ Initialize volume
5. _____ Load volume
6. _____ Mount volume
7. _____ Unload volume

WRITTEN EXERCISE II

Choose the VMS command best suited to perform each of the following operations and write its letter in the space provided.

VMS Commands

- a. ALLOCATE
- b. DEALLOCATE
- c. DISMOUNT
- d. INITIALIZE
- e. MOUNT
- f. SHOW DEVICE/FULL

Operations

- 1. _____ Build the appropriate structure on a disk (usually used for a new 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.

WRITTEN EXERCISE III

Write a VMS command string to perform each of the following operations.

1. Allocate any available tape unit to your process and assign the logical name TAPE to it.
2. Initialize a tape volume that you have loaded on TAPE. Assign the label TAP_BK to the unit.
3. Mount TAP_BK on the tape device so that the Backup utility can process it.
4. Back up all files in your default directory to a save set on TAP_BK.
5. List the contents of the save set TAP_BK at your terminal.
6. Terminate access to TAP_BK, allowing the system to automatically unload the volume.
7. Release the tape device so others on your system can use it.
8. Delete the logical name TAPE from the logical-name table that stores it.

LABORATORY EXERCISE I

Complete the following exercises at an interactive terminal. Note that your device names may differ from the device and directory names given in the solutions.

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.

SUBMITTING BATCH AND PRINT JOBS

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 `SY$PRINT`.
2. Use a single **PRINT** command to print two copies of the same file, giving the print job priority 3 on `SY$PRINT`.
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 physical 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 print 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.

LABORATORY EXERCISE II

1. Display at your terminal screen all of the batch queues on the system.
2. Submit a command procedure to the batch queue 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:
 - Find the entry for your batch job from the **SHOW SYSTEM** command. What was its process ID?
 - Did your LOGIN.COM file execute? Did the system-wide login procedure execute?
 - How much CPU time did your batch job use to execute?
 - How much elapsed time did your batch job use to execute?

HARDWARE AND SOFTWARE OVERVIEW—SOLUTIONS

WRITTEN EXERCISE I

In the exercise below, match each description with the appropriate component of the hardware environment. Components of the hardware environment may be used once, more than once, or not at all.

Hardware Components

- a. CPU
- b. Console Subsystem
- c. Main Memory
- d. I/O Subsystem

Descriptions

- 1. c Stores instructions and data
- 2. b Used to monitor and control the system
- 3. d Consists of peripherals
- 4. a Executes instructions
- 5. b Used for starting up and shutting down the system

Write the letter of the term that best completes each of the following statements.

1. c are used to connect the various subsystems of the computer.
 - a. Peripheral devices
 - b. Network communication devices
 - c. Interconnect devices
 - d. Storage devices

2. b have a screen for displaying information.
 - a. Hardcopy terminals
 - b. Video terminals
 - c. Laser printers
 - d. Mass storage devices

3. c is NOT a peripheral device.
 - a. Terminal
 - b. Printer
 - c. CPU
 - d. Disk drive

4. d are high-speed machines that are usually used for large quantities of stored output.
 - a. Hardcopy terminals
 - b. Disk drives
 - c. Laser printers
 - d. Line printers

5. a is NOT a type of disk.
 - a. Reel
 - b. Cartridge
 - c. Diskette
 - d. Disk pack

6. b record data on magnetic media.
 - a. Disk drives
 - b. Tape drives
 - c. Terminal servers
 - d. VAXcluster systems

WRITTEN EXERCISE II

Compare your answers with those shown below. For additional information, consult your instructor.

1. VMS Account name
2. DISK:[SMITH] Default Device and Directory Specification
3. VTA15: Interactive Terminal Specification
4. (Not Displayed) Password
5. 20400140 Process Identification Code
6. SMITH Process Name
7. [GROUP11,SMITH] User Identification Code
8. SMITH User Name
9. 4 Priority

Privileges (list them)

10. GRPNAM
11. GROUP
12. TMPMBX
13. NETMBX
14. Infinite CPU Limit
15. 63 Open File Quota
16. 2 Subprocess Quota

Match each of the following operations with the parameter that controls your ability to perform it. Some operations are controlled by more than one parameter.

Parameters

- a. Password
- b. Priority
- c. Privilege
- d. Process Identification Number (PID)
- e. Resource Limit
- f. User Identification Code (UIC)
- g. User Name

Operations

1. g,a Logging in to your system

To log in to a system, you must know the user name of a record in the UAF. You must also know the password that corresponds to that user name.

2. c,f Deleting a file that belongs to another user

Your ability to delete a file depends on your UIC.

3. c Creating a group logical name

4. e Opening a large number of files

A resource limit (FILLM) determines the number of files that you can open simultaneously.

GETTING STARTED—SOLUTIONS

LABORATORY EXERCISE I

1. Log in to the system, using the user name and password assigned to you.

Press <RET> on the terminal keyboard

At the *Username:* prompt, type your user name <RET>

At the *Password:* prompt, type your password <RET>

If typed successfully, you should get a "Welcome" message from the system. If typed unsuccessfully, the system will output an error message to your terminal, notifying you that either your Username or Password was illegal. Re-typing your Username and Password will correct this situation.

2. Type the following command lines at your terminal:

- **SHOW TIME**

```
$ SHOW TIME<RET>
31-DEC-1987 14:10:32
```

- **SHOW USERS**

```
$ SHOW USERS<RET>
VAX/VMS Interactive Users          31-DEC-1987 13:30:33.86
Total number of interactive users = 6
Username      Process Name      PID      Terminal
BECKER        BECKER            20200332 VTA115:      LTA76:
CHAPUT        Mary              20200342 VTA111:      LTA57:
COVERDALE     COVERDALE         20200390 VTA131:      LTA74:
GOEHRING      Judy              202003A7 VTA139:      LTA82:
SMITH         SMITH             20200331 VTA145:      LTA88:
JOHNSTON      JOHNSTON          20200352 VTA124:      LTA67:
```

• **SHOW TERMINAL**

```
$ SHOW TERMINAL<RET>
```

```
Terminal: _VTA145:   Device_Type: PRO_Series   Owner: SMITH
Physical terminal: _LTA88:   Username: SMITH
Input:   9600   Lfifill: 0   Width: 80   Parity: None
Output:  9600   CRfill: 0   Page: 24
Terminal Characteristics:
Interactive      Echo           Type_ahead     No Escape
No Hostsync     Ttsync        Lowercase      Tab
Wrap            Scope         No Remote      No Eightbit
Broadcast       No Readsycn   No Form        Fulldup
No Modem        No Local_echo No Autobaud    Hangup
No Brcdstmbx   No DMA        No Altypeahd   Set_speed
Line Editing    Overstrike editing No Fallback    No Dialup
No Secure server Disconnect     No Psthru      No Syspassword
No SIXEL Graphics No Soft Characters Printer port    Application keypad
ANSI_CRT       Regis         No Block_mode  Advanced_video
Edit_mode      DEC_CRT       No DEC_CRT2    No DEC_CRT3
```

3. Log out of the system.

```
$ LOGOUT<RET> or
$ LOGOUT/FULL<RET>
```

LABORATORY EXERCISE II

1.

```
$ PRODUCE NONESUCH.FIL<RET>
%DCL-W-IVVERB, unrecognized command verb - check validity
and spelling \PRODUCE\
```

a. How severe was the error?

It was a warning.

b. What part of the system produced this error message?

DCL was the part of the system that was upset.

2. Use the command line editor to recall the **PRODUCE** command and change it to the **TYPE** command. Now execute the command and observe the results.

```
$ TYPE NONESUCH.FIL
%TYPE-W-SEARCHFAIL, error searching for DISK:[SMITH]NONESUCH.FIL;
-RMS-E-FNF, file not found.
```

a. How severe was this error?

Severity levels of warning and error.

b. What part of the system produced this error message?

Messages came from the TYPE program and RMS.

c. Did the message text differ from the previous exercise?

The first exercise was an Unrecognized Command while the second was a Nonexistent File.

The system notifies you that after searching for the file it cannot be found since it does not exist. To correct the error, re-issue the **TYPE** command followed by a legal file name.

WRITTEN EXERCISE I

1. b You have logged in to your system. A long string of messages, all of which you have seen before, scrolls past on your screen. Suppress the messages, without stopping or aborting the program that produces them.
2. h You have just typed the string TYPE FILE&. The cursor is positioned immediately after the ampersand (&). Delete the ampersand (&).
3. g You have entered the **SHOW SYSTEM** command. A listing of users on your system scrolls past on your screen. Abort further execution of the command and return control to your terminal.
4. f You have entered the following command lines at your terminal:

```
$ DIFFERENCES/IGNORE=BLANK_LINES --<RET>
_ $ FILE1 --<RET>
_ $ FILE8
```

The cursor is immediately to the right of the number eight on the last line. Delete the last line, without deleting the preceding lines of the command string.

5. e You have entered the following string at your terminal:

```
$ SHOW PROCESS/ALL<RET>
```

Lines of information scroll past on your terminal screen. Stop the display and halt, but do not abort, the program that generates it.

6. c Resume generation of the display that you stopped in the preceding operation.
7. d You have made extensive corrections to a command line at a hardcopy terminal. The output looks like this:

```
$ PRYNT\TNY\NT9\9\ FILIN\NI\
```

Display the line without the echoed corrections.

8. a You have just issued a command line. Recall this command.

LABORATORY EXERCISE III

Use the following commands:

1. A listing of all topics available through the Help facility

```
$ HELP
```

2. A description of the login procedure

```
$ HELP LOGIN
```

3. A description of the **/FULL** qualifier of the **LOGOUT** command

```
$ HELP LOGOUT/FULL
```

4. A description of the **TIME** option of the **SHOW** command

```
$ HELP SHOW TIME
```

LABORATORY EXERCISE IV

1. To display the characteristics of your terminal, enter the **SHOW TERMINAL** command.
 - a. The **WIDTH** setting of your terminal determines the number of characters displayed in an output line.
 - b. The receive speed of your terminal is the first number specified for the **SPEED** setting of your terminal.
 - c. The transmit speed of your terminal is the second number specified for the **SPEED** setting of your terminal.
 - d. Your terminal type is the first value that appears following the terminal name in the **SHOW TERMINAL** display. If the terminal type (**/VT52**, **/VT100**, **/LA36**, **/LA120**, or some other value) does not match the physical characteristics of your terminal, consult your system manager.

2. To display the specified process parameters, enter the command lines shown below and look for the information specified within the parentheses.

- a. `$ SHOW PROCESS/QUOTAS` (Account Name)
- b. `$ SHOW PROCESS/QUOTAS` (CPU Limit)
- c. `$ SHOW PROCESS` (Default File Specification)
- d. `$ SHOW TERMINAL` (or `$ SHOW PROCESS`)
- e. `$ SHOW PROCESS` (Priority)
- f. `$ SHOW PROCESS/PRIVILEGES` (Privileges)
- g. `$ SHOW PROCESS` (PID)
- h. `$ SHOW PROCESS` (Process Name)
- i. `$ SHOW PROCESS` (UIC)
- j. `$ SHOW PROCESS` (User)

3. Display the names of all processes running on your system.

```
$ SHOW SYSTEM
```

4. Display the names of all users on your system.

```
$ SHOW USERS
```

5. Display the names of all devices on your system.

```
$ SHOW DEVICES
```

6. Log out of your system

```
$ LOGOUT or  
$ LOGOUT/FULL
```


CREATING AND EDITING TEXT FILES—SOLUTIONS

LABORATORY EXERCISE I – THE EDT EDITOR

1. To create the file:

a. Enter the command:

```
$ EDIT EXERCISE1.TEXT
```

b. The system displays the following:

```
Input file does not exist  
[EOB]  
*
```

The message indicates that the file EXERCISE1.TEXT did not previously exist in your directory. The [EOB] marker indicates the end of the buffer. The asterisk indicates that you are in Line mode.

c. To enter Keypad mode, type the **CHANGE** command at the line-mode prompt, then press <RET>:

```
*CHANGE<RET>
```

The screen display erases, and the [EOB] marker appears in the upper left corner of the screen.

2. To become familiar with the available Help:

a. Invoke the Help facility from Keypad mode:

Press <PF2> on the keypad.

or

Press <HELP> (on the VT200 keyboard).

The keypad diagram is displayed on your terminal screen.

b. You can display an explanation of each defined key by pressing the key while in HELP. You might want to begin with the following keys:

- HELP (PF2)
- DELETE
- DOWN ARROW
- GOLD (PF1)
- DELETE/UNDELETE LINE (PF4)

Examine any of the key definitions you wish.

c. To leave Help:

Press the SPACE BAR

3. Type in the text as indicated.

Note that EDT does not automatically wrap at the end of a line. You must explicitly press <RET> to insert carriage returns into the text.

4. To end the session and save your work:

- a. Press <CTRL/Z>. This returns you to Line mode.
- b. At the line-mode prompt, type the **EXIT** command and press <RET>.

*EXIT<RET>

c. The system displays the full file specification and the file's length in lines. The system then returns you to the DCL level.

```
$DISK:[SMITH]EXERCISE1.TEXT;1 3 lines
$
```

5. To edit the file:

a. Enter the command:

```
$ EDIT EXERCISE1.TEXT
```

b. The system displays the following:

```
1 The purpose of this exercise is to allow  
*
```

The first line of text is displayed on the screen.

c. At the line-mode prompt, type **C** then press <RET>. The contents of the file EXERCISE1.TEXT are displayed on the screen.

6. To modify the text:

a. Move the cursor to the beginning of the word "basic." You can do this by using either the arrow keys, or the keypad keys 0 (zero) and 1 (one). The keypad key 0 moves the cursor from line to line; the keypad key 1 moves the cursor from word to word.

b. To delete a word, press the MINUS (-) key (below <PF4> on the keypad). The word to the right of the cursor is deleted. If you press it again, the next word to the right of the cursor is deleted.

Type in the modifications to the text as shown.

7. End the editing session normally by pressing <CTRL/Z>, and then entering the **EXIT** command at the line-mode prompt.

LABORATORY EXERCISE II – THE EVE EDITOR

1. To create the file, enter the command:

```
$ EDIT/TPU EXERCISE3.TEXT
```

2. You should see messages at the bottom of the screen indicating that the file EXERCISE3.TEXT did not previously exist in your directory. The cursor should be positioned at the top of the screen, next to the end of file marker.

The status line appears at the bottom of the screen. It contains the name of the buffer. In this case, the buffer name is the same as the file name. In addition, the status line indicates the editing mode and search direction. The default values for these are Insert and Forward.

3. Type in the text as indicated.

Note that the EVE editor automatically wraps at the end of a line. You need not press <RET> to insert carriage returns into the text.

4. To end the editing session:

- a. Press <CTRL/Z>. This automatically ends the editing session.

You can also end an EVE editing session using a line-mode command.

Press <PF4> or <DO>. At the *Command:* prompt, type the **EXIT** command and press <RET>.

- b. An informational message is displayed that includes the full file specification and the number of line in the file. The system then returns you to the DCL level.

NOTE

If you wish to end an EVE editing session without saving changes, you must exit using Line mode. Press <PF4> or <DO>, and at the prompt, type **QUIT** and press <RET>. You will be asked if you wish to continue the quitting process. Type **Y** and press <RET>.

5. Enter the command:

```
$ EDIT/TPU EXERCISE3.TEXT
```

A message is displayed indicating that three lines were read from the file, and the contents of the file appear on the terminal screen. The cursor is at the top of the file.

6. To modify the file:
 - a. Use the arrow keys to move the cursor to the beginning of the word "Extensible."
 - b. There are two ways to delete words, depending on the terminal you are using:
 - VT100—Press the <COMMA> (,) key on the keypad. The word to the right of the cursor is deleted. Repeat this step three times.
 - VT200—Press the <F13> key along the top of the keyboard. The word to the right of the cursor is deleted. Repeat this step three times.
 - c. To delete single characters, use the key (near the <RET> key) on the keyboard.
 - d. Use the arrow keys to move the cursor to the beginning of the word "exercise."
 - e. There are two ways to switch editing modes, depending on the terminal you are using:
 - VT100—Press <ENTER> on the keypad.
 - VT200—Press <F14> along the top of the keyboard.The status line indicates that the editing mode is now Overstrike.
 - f. Type in the words "example that uses." Note that when you are in Overstrike mode, the new characters replace existing characters.
7. End the editing session normally by pressing <CTRL/Z>, and then entering the **EXIT** command at the line-mode prompt.

LABORATORY EXERCISE III

This exercise lets you practice editing more than one file on your terminal screen.

1. Edit a file of your choice.

```
$ EDIT/TPU FILE1.TXT
```

2. Split your terminal screen into two windows.

```
TWO WINDOWS
```

3. Edit another file of your choice.

```
GET FILE FILE2.TXT
```

```
OR
```

```
GET FILE2.TXT
```

4. Move text from one file into the other file.

5. Exit the file so the moved text is saved.

```
EXIT
```

NOTE

The editor will prompt you as to whether or not you wish the second file on the screen to be written. Answer Y for Yes or N for No.

COMMUNICATING WITH OTHER USERS—SOLUTIONS

LABORATORY EXERCISE I

1. Invoke the Mail utility and send several mail messages to someone in your class.

```
$ MAIL<RET>
MAIL> SEND [messages]
```

2. Have your mail recipient send mail messages to you.

3. Read your messages.

```
MAIL> READ or
MAIL><RET> or
MAIL> 1
```

4. Obtain a list of your mail messages.

```
MAIL> DIRECTORY
```

5. Read only the second mail message.

```
MAIL> READ 2
```

6. Delete the fourth mail message.

```
MAIL> DELETE 4 or
MAIL> DELETE (if it is the current message on your screen)
```

7. Create a text file in your default directory. Send this file as a mail message.

```
$ CREATE file-name
CTRL/Z
$ MAIL<RET>
MAIL> SEND file-name
```

8. Pick a message and move it to a folder named Test. Select the Test folder and check to see if the message is there.

```
MAIL> READ 1
MAIL> MOVE TEST
Folder TEST does not exist.
Do you want to create it (Y/N, default is N)? Y
MAIL> SELECT TEST
MAIL> DIRECTORY
```

9. List the folders you have. In addition to the folder you just created, what other folders do you have?

```
MAIL> DIRECTORY/FOLDERS
```

You could see three folders named MAIL, NEWMAIL, and WASTEBASKET.

10. Create a distribution list for Mail. Include several members of your class. Send a short message to the people on the distribution list.

```
$ EDIT DISTRIBUTION.DIS
USER1
USER2
.
.
CTRL/Z
EXIT
$ MAIL
MAIL> SEND
TO:@DISTRIBUTION.DIS
```


LABORATORY EXERCISE II

1. Invoke the Phone utility.

```
$ PHONE  
PHONE>
```

2. Obtain a list of available users.

```
%DIRECTORY
```

3. Establish a phone connection with one of the users.

```
USER 1          USER 2  
%DIAL user2     %ANSWER
```

4. Terminate all conversations

```
%HANGUP or CTRL/Z
```

LABORATORY EXERCISE III

Send a request to a system operator using the **REQUEST** command.

```
$ REQUEST "Please mount magtape"
```

MANAGING FILES—SOLUTIONS

WRITTEN EXERCISE I

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

MAIL*.DAT.* or MAIL%%.DAT

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

*.DAT;1

WRITTEN EXERCISE II

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 file name to a new file 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.

LABORATORY EXERCISE I

1. Create a subdirectory called [.SUB1]

```
$ CREATE/DIRECTORY [XXX.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
```

9. Delete both subdirectories.

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

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?

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

The protection applied to this file is the 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). Create a new file named NEWFILE.TXT. What protection code does this new file have and why?

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

In changing your default protection, you have specified that files now created should have this new default protection. You can check this by issuing the command:

```
$ DIRECTORY/PROTECTION NEWFILE.TXT
```

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
```

WRITTEN EXERCISE III

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.

LABORATORY EXERCISE III

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
```

NOTE

Check with your instructor to see what your GROUP identifier is.

WRITTEN EXERCISE IV

1. FALSE A user with a UIC of [100,200] wishes 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] wishes 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] wishes 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] wishes 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] wishes 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.

CUSTOMIZING THE USER ENVIRONMENT—SOLUTIONS

WRITTEN EXERCISE I

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.

LABORATORY EXERCISE I

Compare your results with those described below. For additional help, consult your instructor.

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]" (LNMS$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.TPUSSECTION;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 MYDIR  
OR  
$ DEASSIGN/PROCESS MYDIR
```

LABORATORY EXERCISE II

Compare your results with the example specified. For additional help, consult your instructor.

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

1. Create a subdirectory.

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

2. Create a logical name for your newly created subdirectory.

```
$ ASSIGN [SMITH.TEXT] MY_TEXT
```

3. Create a logical name for a text file in your default directory.

```
$ ASSIGN MYFILE.TXT;1 OUTPUT
```

4. Check the process logical name table to see if your new logical names exist.

```
$ SHOW LOGICAL/PROCESS
(LNMSPROCESS_TABLE)
"MY_TEXT" = "[SMITH.TEXT]"
"OUTPUT" = "MYFILE.TXT;1"
"SYSSCOMMAND" = "_DISK$RTA1:"
"SYSSDISK" = "DISK:"
"SYSSERROR" = "_DISK$RTA1:"
"SYSSINPUT" = "_DISK$RTA1:"
"SYSSOUTPUT" [super] = "_DISK$RTA1:"
"SYSSOUTPUT" [exec] = "_DISK$RTA1:"
"TT" = "RTA1:"
```

5. Using only logical names, move the text file into your new subdirectory.

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

6. After completing this exercise, remove the above logical names.

```
$ DEASSIGN OUTPUT
$ DEASSIGN MY_TEXT
$
```

WRITTEN EXERCISE II

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

Symbolic Name Types

- a. Command Synonym
- b. Logical Name

Characteristics

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

LABORATORY EXERCISE III

Create global symbols to perform the following tasks. You may create these global symbols interactively or in the file LOGIN.COM. Your global symbols may differ from the exercise answers.

1. Display a directory listing along with sizes 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"
```

6. Symbols disappear when you log out.

LABORATORY EXERCISE IV

1. Define <KP2> to be the **SHOW TIME** command. Try it first without the **/TERMINATE** qualifier, then with the **/TERMINATE** qualifier. What is the difference?

```
$ SET TERMINAL/NONUMERIC
$ DEFINE/KEY KP2 "SHOW TIME"
$ DEFINE/KEY KP2 "SHOW TIME"/TERMINATE
```

The difference is the first time you have to press <RET> after pressing <KP2>. The second time the return is automatically supplied.

2. Define <KP3> to be the **SET DEFAULT** command. Create a subdirectory. Try to move to the subdirectory by using your newly defined key.

```
$ DEFINE/KEY KP3 "SET DEFAULT "
$ CREATE/DIRECTORY [SMITH.TEMPORARY]
```

Press <KP3> key, type in [SMITH.TEMPORARY] and press <RET> key. To see if it worked, issue the **SHOW DEFAULT** command.

3. Delete the key definition for <KP2>. See if it worked by displaying all your key definitions again.

```
$ SHOW KEY/ALL (Displays all key definitions)
$ DELETE/KEY KP2
$ SHOW KEY/ALL
```

WRITING COMMAND PROCEDURES—SOLUTIONS

WRITTEN EXERCISE I

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 SYSS$OUTPUT COUNT " copies of the file"`

Incorrect. Correct command is: `$ WRITE SYSS$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 SYSS$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 SYSS$OUTPUT "The file 'FILE_NAME' 'FILE_TYPE'"`

Correct. `$ WRITE SYSS$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 SYSS\$OUTPUT "The file is MYFILE.TXT"
\$ WRITE SYSS\$OUTPUT "The file is 'P1'"
2. \$ TYPE PROGRAM.FOR
\$ TYPE 'FILE_NAME' 'FILE_TYPE'
3. \$ EDIT DATA.DAT
\$ EDIT 'P2'
4. \$ WRITE SYSS\$OUTPUT "2 copies of the file DATA.DAT exist."
\$ WRITE SYSS\$OUTPUT "'COUNT' copies of the file 'P2' exist."
5. \$ FILE = "PROGRAM" + ".FOR"
\$ FILE = FILE_NAME + FILE_TYPE

LABORATORY EXERCISE I

```
$! 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
$!
$! Set up global symbols to be used a 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
```

LABORATORY EXERCISE II

```

$!                               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

```

OPTIONAL LABORATORY EXERCISE

```
$ !
$ !                                     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
$ ! redisplay 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 SYSS$INPUT command to type eleven blank lines on
$ ! the terminal.
$ TYPE SYSS$INPUT
```

```
$ !
```

```
$ ! Now use the WRITE SYSS$OUTPUT command to display
$ ! the message on the screen.
$ !
$ WRITE SYSS$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 SYSS$INPUT command to type
$ ! the necessary blank lines.
$ TYPE SYSS$INPUT
```

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


USING DISK AND TAPE VOLUMES—SOLUTIONS

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. 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

WRITTEN EXERCISE II

Choose the VMS command best suited to perform each of the following operations and write its letter in the space provided.

VMS Commands

- a. ALLOCATE
- b. DEALLOCATE
- c. DISMOUNT
- d. INITIALIZE
- e. MOUNT
- f. SHOW DEVICE/FULL

Operations

- 1. d Build the appropriate structure on a disk (usually used for a new 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.

WRITTEN EXERCISE III

1. Allocate any available tape unit to your process and assign the logical name TAPE to it.

```
$ ALLOCATE MT: TAPE
```

2. Initialize a tape volume that you have loaded on TAPE. Assign the label TAP_BK to the unit.

```
$ INITIALIZE TAPE TAP_BK
```

3. Mount TAP_BK on the tape device so that the Backup utility can process it.

```
$ MOUNT/FOREIGN TAPE
```

4. Back up all files in your default directory to a save set on TAP_BK.

```
$ BACKUP/IGNORE=LABELPROCESSING [...] *.*;* TAPE:TAP_BK.BCK
```

5. List the contents of the save set TAP_BK at your terminal.

```
$ BACKUP/LIST TAPE:TAP_BK.BCK
```

6. Terminate access to TAP_BK, allowing the system to automatically unload the volume.

```
$ DISMOUNT TAPE
```

7. Release the tape device so others on your system can use it.

```
$ DEALLOCATE TAPE
```

8. Delete the logical name TAPE from the logical name table that stores it.

```
$ DEASSIGN TAPE
```

LABORATORY EXERCISE I

Complete the following exercises at an interactive terminal. Note that your device names may differ from the device and directory names given in the solutions.

1. Allocate the tape.

```
$ ALLOCATE MTAO:
```

2. Initialize the tape, giving it a label name of MYTAPE.

```
$ INITIALIZE MTAO: MYTAPE
```

3. Mount the tape, so that BACKUP can be used.

```
$ MOUNT/FOREIGN MTAO:
```

4. Obtain a listing of the files in your directory.

```
$ DIRECTORY
```

5. Transfer all files from your directory to the tape.

```
$ BACKUP/IGNORE=LABEL_PROCESSING *.*;* MTAO:JAN1.BCK
```

6. Confirm that all files transferred successfully to the tape.

```
$ BACKUP/REWIND/LIST MTAO:JAN1.BCK
```

7. Dismount the tape.

```
$ DISMOUNT MTAO:
```

8. Deallocate the tape.

```
$ DEALLOCATE MTAO:
```

SUBMITTING BATCH AND PRINT JOBS—SOLUTIONS

LABORATORY EXERCISE I

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 the same file, queuing it directly to the physical queue.

```
$ PRINT/QUEUE=LPA0 filename
```

(LPA0 may or may not be the name of your physical 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 print 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=ENTRY-NUMBER SYS$PRINT
```

LABORATORY EXERCISE II

1. Display at your terminal screen 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.COM
```

5. Examine the log file created in Step 2. Answer the following questions:

- Find the entry for your batch job from the **SHOW SYSTEM** command. What was its process ID?
- Did your LOGIN.COM file execute? Did the system-wide login procedure execute?
- How much CPU time did your batch job use to execute?
- How much elapsed time did your batch job use to execute?

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.

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.

Both the CPU time and elapsed time are in the accounting information in the last lines of the log file.

MODULE 12
TEST

... of the ...

... you ...

... and ...

... and ...

... and ...

... and ...

... and ...

...

... file from ...

...

...

...

... creates the

...

... JOHNSTON ...
... DIRECTOR ...

...

... DIRECTOR ...

... DIRECTOR ...

... DIRECTOR ...

... DIRECTOR ...

TEST

Underline the best answer to each of the following questions.

1. When logging in to a VMS system, you typically need to supply your:
 - a. User identification code and user name
 - b. User name and password
 - c. User identification code and password
 - d. User identification code, user name, and password

2. Which DCL command displays a text file on the terminal screen?
 - a. EXAMINE
 - b. TYPE
 - c. SHOW
 - d. DIRECTORY

3. Which command can move a file from one disk to another?
 - a. RENAME
 - b. COPY
 - c. CREATE
 - d. CONVERT

4. Your default directory is [JOHNSON]. Which of the following DCL commands creates the subdirectory [JOHNSON.BUDGET]?
 - a. CREATE/SUBDIRECTORY [JOHNSON.BUDGET]
 - b. CREATE/DIRECTORY [JOHNSON.BUDGET]
 - c. CREATE/DIRECTORY [JOHNSON]BUDGET.DIR
 - d. CREATE/SUBDIRECTORY BUDGET.DIR

5. What must you do before the VMS system will allow you to delete a subdirectory?
 - a. Delete all the files in the subdirectory.
 - b. Log in to the SYSTEM account.
 - c. Make backup copies of the files in the subdirectory.
 - d. Notify other users on the system that you are deleting the subdirectory.

6. Which DCL command do you use to put a job in a batch queue?
 - a. SEND/BATCH
 - b. BATCH
 - c. SUBMIT
 - d. QUEUE/BATCH

7. Which DCL command do you use to put a job in a print queue?
 - a. SEND/PRINT
 - b. PRINT
 - c. SUBMIT
 - d. QUEUE/PRINT

8. Which of the following requires you to use the ASSIGN command?
 - a. Defining a DCL symbol
 - b. Creating a new user name
 - c. Defining a logical name
 - d. Setting a file's protection

9. If a logical name is already defined, which command do you use to assign a new value to the logical name?
- a. RENAME
 - b. DEASSIGN
 - c. CREATE
 - d. ASSIGN
10. If a file already exists, which command do you use to assign a new name to the file?
- a. RENAME
 - b. DEASSIGN
 - c. CREATE
 - d. ASSIGN
11. Which logical name refers to the disk and directory that are the default when you log in?
- a. SYSS\$COMMAND
 - b. SYSS\$LOGIN
 - c. SYSS\$INPUT
 - d. SYSS\$OUTPUT
12. Which of the following DCL commands defines the symbol GO as the DCL command SET DEFAULT?
- a. SET DEFAULT == "GO"
 - b. "GO" == "SET DEFAULT"
 - c. GO == "SET DEFAULT"
 - d. "SET DEFAULT" == GO

13. Which of the following do you use to assign a global value to a symbol name?
- a. =
 - b. ==
 - c. DEFINE
 - d. DEFINE/GLOBAL
14. Which of the following operators would you use to test if two character strings are equal following an IF command in a DCL command procedure?
- a. =
 - b. ==
 - c. .EQ.
 - d. .EQS.
15. In a DCL command procedure, a label is followed by which of the following characters?
- a. -
 - b. \$
 - c. :
 - d. space
16. Which DCL command do you use to display your password?
- a. EXAMINE PASSWORD
 - b. SHOW PASSWORD
 - c. SHOW PROCESS/PASSWORD
 - d. There is no command to do this

17. Which DCL command do you use to change your password?

- a. RENAME PASSWORD
- b. CHANGE PASSWORD
- c. SET PASSWORD
- d. There is no command to do this

18. The DCL command that terminates a process is:

- a. LOGOUT
- b. EXIT
- c. DELETE
- d. QUIT

Match each of the following DCL commands with its function. You will not use all of the DCL commands.

DCL Command

- a. START
- b. @
- c. SUBMIT
- d. CREATE
- e. RUN

Function

- 1. _____ Execute a compiled and linked program
- 2. _____ Execute a command procedure in batch mode
- 3. _____ Execute a command procedure interactively

Match the VMS system component with its function. You may select a system component once, more than once, or not at all.

VMS System Component

- a. CPU
- b. I/O Interface
- c. Disk Drive
- d. Physical Memory
- e. Virtual Memory

Function

- 1. _____ The system component typically used by the hardware when referring to memory
- 2. _____ The system component typically used by a programmer when referring to memory
- 3. _____ The pathway through which data is transferred to other hardware devices
- 4. _____ The only system component that performs computations
- 5. _____ The only system component that is conceptual, and not a piece of hardware

ANSWERS

Underline the best answer to each of the following questions.

1. When logging in to a VMS system, you typically need to supply your:
 - a. User identification code and user name
 - b. User name and password
 - c. User identification code and password
 - d. User identification code, user name, and password

2. Which DCL command displays a text file on the terminal screen?
 - a. EXAMINE
 - b. TYPE
 - c. SHOW
 - d. DIRECTORY

3. Which command can move a file from one disk to another?
 - a. RENAME
 - b. COPY
 - c. CREATE
 - d. CONVERT

4. Your default directory is [JOHNSON]. Which of the following DCL commands creates the subdirectory [JOHNSON.BUDGET]?
 - a. CREATE/SUBDIRECTORY [JOHNSON.BUDGET]
 - b. CREATE/DIRECTORY [JOHNSON.BUDGET]
 - c. CREATE/DIRECTORY [JOHNSON] [BUDGET] DIR
 - d. CREATE/SUBDIRECTORY BUDGET.DIR

5. What must you do before the VMS system will allow you to delete a subdirectory?
- Delete all the files in the subdirectory.
 - Log in to the SYSTEM account.
 - Make backup copies of the files in the subdirectory.
 - Notify other users on the system that you are deleting the subdirectory.
6. Which DCL command do you use to put a job in a batch queue?
- SEND/BATCH
 - BATCH
 - SUBMIT
 - QUEUE/BATCH
7. Which DCL command do you use to put a job in a print queue?
- SEND/PRINT
 - PRINT
 - SUBMIT
 - QUEUE/PRINT
8. Which of the following requires you to use the ASSIGN command?
- Defining a DCL symbol
 - Creating a new user name
 - Defining a logical name
 - Setting a file's protection

9. If a logical name is already defined, which command do you use to assign a new value to the logical name?

- a. RENAME
- b. DEASSIGN
- c. CREATE
- d. ASSIGN

10. If a file already exists, which command do you use to assign a new name to the file?

- a. RENAME
- b. DEASSIGN
- c. CREATE
- d. ASSIGN

11. Which logical name refers to the disk and directory that are the default when you log in?

- a. SYS\$COMMAND
- b. SYS\$LOGIN
- c. SYS\$INPUT
- d. SYS\$OUTPUT

12. Which of the following DCL commands defines the symbol GO as the DCL command SET DEFAULT?

- a. SET DEFAULT == "GO"
- b. "GO" == "SET DEFAULT"
- c. GO == "SET DEFAULT"
- d. "SET DEFAULT" == GO

13. Which of the following do you use to assign a global value to a symbol name?

- a. =
- b. ==
- c. DEFINE
- d. DEFINE/GLOBAL

14. Which of the following operators would you use to test if two character strings are equal following an IF command in a DCL command procedure?

- a. =
- b. ==
- c. .EQ.
- d. .EQS.

15. In a DCL command procedure, a label is followed by which of the following characters?

- a. -
- b. \$
- c. :
- d. space

16. Which DCL command do you use to display your password?

- a. EXAMINE PASSWORD
- b. SHOW PASSWORD
- c. SHOW PROCESS/PASSWORD
- d. There is no command to do this

17. Which DCL command do you use to change your password?

- a. RENAME PASSWORD
- b. CHANGE PASSWORD
- c. SET PASSWORD
- d. There is no command to do this

18. The DCL command that terminates a process is:

- a. LOGOUT
- b. EXIT
- c. DELETE
- d. QUIT

Match each of the following DCL commands with its function. You will not use all of the DCL commands.

DCL Command

- a. START
- b. @
- c. SUBMIT
- d. CREATE
- e. RUN

Function

- 1. e Execute a compiled and linked program
- 2. c Execute a command procedure in batch mode
- 3. b Execute a command procedure interactively

Match the VMS system component with its function. You may select a system component once, more than once, or not at all.

VMS System Component

- a. CPU
- b. I/O Interface
- c. Disk Drive
- d. Physical Memory
- e. Virtual Memory

Function

- 1. d The system component typically used by the hardware when referring to memory
- 2. e The system component typically used by a programmer when referring to memory
- 3. b The pathway through which data is transferred to other hardware devices
- 4. a The only system component that performs computations
- 5. e The only system component that is conceptual, and not a piece of hardware