

OpenVMS Fundamentals, Version 7.2

Student Guide

Rev. 0.30

Training



COMPAQ

OpenVMS Fundamentals, Version 7.2

Student Guide

Rev. 0.30



Training

COMPAQ

© 2000 Compaq Computer Corporation

COMPAQ and the Compaq logo, AlphaServer, DECnet, HSJ, PDP, and VAX are registered in the U.S. Patent and Trademark Office.

Alpha, OpenVMS, and Tru64 are trademarks of Compaq Information Technologies Group, L.P.

Microsoft, Windows, Windows NT, and Visual C++ are trademarks of Microsoft Corporation.

Motif, OSF/1, UNIX, the “X” Device, IT DialTone, and The Open Group are trademarks of The Open Group.

All other product names mentioned herein may be trademarks of their respective companies.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS SUBJECT TO CHANGE WITHOUT NOTICE AND IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND. THE ENTIRE RISK ARISING OUT OF THE USE OF THIS INFORMATION REMAINS WITH RECIPIENT. IN NO EVENT SHALL COMPAQ BE LIABLE FOR ANY DIRECT, CONSEQUENTIAL, INCIDENTAL, SPECIAL, PUNITIVE, OR OTHER DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, OR LOSS OF BUSINESS INFORMATION), EVEN IF COMPAQ HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND WHETHER IN AN ACTION OF CONTRACT OR TORT, INCLUDING NEGLIGENCE.

The limited warranties for Compaq products are exclusively set forth in the documentation accompanying such products. Nothing herein should be construed as constituting a further or additional warranty.

OpenVMS Fundamentals, Version 7.2

Student Guide

September 2000

Overview

Introduction	Overview-1
Course Length	Overview-1
Intended Audience	Overview-1
Prerequisites	Overview-1
Course Objectives	Overview-1
Course Goals	Overview-2
Course Outline	Overview-2

Module 1 — Hardware Overview

Objectives	1-1
Topics	1-1
System Architectures	1-2
Overview of System Architecture	1-3
VAX Compared to Alpha	1-4
Peripherals	1-5
OpenVMS Clusters	1-7
Networks	1-8
Learning Check	1-10

Module 2 — Software Overview

Objectives	2-1
Topics	2-1
OpenVMS	2-2
Login and User Name	2-3
The Process	2-4
Software Context	2-5
Software Context Information Display	2-7
Sample UAF Record	2-9
Hardware Context	2-10
Virtual Address Space	2-11
Virtual Address Space on Alpha	2-12
Process Types	2-13
Process Privileges	2-14
System Processes	2-15
Process Terms	2-16
Utilities	2-17
Layered Products	2-18
Learning Check	2-19

Module 3 — Introduction to DCL

Objectives.....	3-1
Topics.....	3-1
DCL	3-2
Passwords.....	3-3
DCL Features	3-4
DCL Session.....	3-6
Starting a DCL Session.....	3-6
\$ LOGOUT	3-6
Command Line Format	3-7
Types of Commands	3-7
Command String Elements	3-8
DCL Informational Commands.....	3-9
DCL Command Line Parsing	3-10
Verb Processing.....	3-10
Changing Verb Behavior	3-10
DCL and File-Oriented Parameters	3-11
File Specification Defaults.....	3-11
User-Defined File Names.....	3-11
DCL File Name Parsing Example	3-12
Qualifiers.....	3-13
HELP	3-14
Summary of Basic DCL Commands	3-16
SPAWN Command.....	3-17
Command Line Editing.....	3-18
Control Keys.....	3-19
Terminal Characteristics	3-20
/PAGE Qualifier	3-21
Error Messages	3-22
Sample Error Message	3-22
Severity Levels	3-23
HELP Message Utility.....	3-24
HELP/MESSAGE Example.....	3-25
DCL Commands by Function.....	3-26
Learning Check	3-30

Module 4 — Using Editors

Objectives.....	4-1
Topics.....	4-1
Editing Options.....	4-2
EVE.....	4-3
Overview	4-3
The EVE Editor.....	4-4
EVE Screen Layout	4-4

EVE Help	4-5
Basic EVE Commands.....	4-6
Sample EVE VT100 Keypad	4-7
Sample EVE EDT Keypad.....	4-8
Startup Keypads.....	4-9
Exiting EVE Editor.....	4-10
EVE File Recovery	4-11
EDT.....	4-12
Overview	4-12
The EDT Editor.....	4-13
EDT Keypad Mode Screen Layout	4-13
EDT Help	4-14
EDT Line Mode Commands	4-15
EDT Keypad Keys.....	4-16
Exiting the EDT Editor.....	4-17
EDT File Recovery	4-18
EDT Initialization File (EDTINI.EDT)	4-19
Learning Check	4-20

Module 5 — Mail and Phone

Objectives.....	5-1
Topics.....	5-1
Mail Utility.....	5-2
Sending Messages.....	5-3
Using an Editor in Mail.....	5-4
Sending a File.....	5-5
Reading Messages	5-6
Deleting Mail.....	5-7
Summary of Mail Commands	5-8
Phone Utility.....	5-9
Phone Example.....	5-10
Phone Command Summary.....	5-11
Learning Check	5-12

Module 6 — Directories and Files

Objectives.....	6-1
Topics.....	6-1
Complete File Specification.....	6-2
Directories	6-3
Directory Organization and File Specifications	6-4
Creating Directories.....	6-5
Directory Command	6-6
Deleting Subdirectories.....	6-7

Wildcards	6-8
General-Purpose Wildcards	6-8
Directory Wildcard Characters.....	6-9
File Protection	6-10
Processes, Objects, and Protections.....	6-11
Determining Your UIC	6-12
Protection Categories.....	6-13
Setting File Protection	6-15
Access Control Lists	6-16
Learning Check	6-20

Module 7 — File Manipulation

Objectives.....	7-1
Topics.....	7-1
File Management Commands.....	7-2
APPEND Command	7-3
Record Types.....	7-4
DUMP Utility	7-5
DIFFERENCES Command.....	7-7
SEARCH Command	7-8
Learning Check	7-9

Module 8 — Logical Names

Objectives.....	8-1
Topics.....	8-1
Logical Names.....	8-2
Logical Name Creation	8-6
Logical Name Tables	8-8
Process Logical Name Table.....	8-8
Job Logical Name Tables.....	8-9
Group Logical Name Tables	8-10
System Logical Name Table	8-12
Logical Name Translation.....	8-17
Process Permanent Logical Names.....	8-18
Default Process Permanent Logical Names	8-18
Redefining Process Permanent Logical Names.....	8-19
Other Logical Name Features	8-20
Learning Check	8-21

Module 9 — Symbols

Objectives.....	9-1
Topics.....	9-1

DCL Symbols	9-2
Creating Symbols	9-3
Symbol Tables.....	9-3
Viewing and Deleting Symbols.....	9-4
Using Wildcards to Abbreviate	9-5
Symbol Substitution.....	9-6
Symbol Types.....	9-8
Integer Symbol Examples	9-9
Symbols Compared to Logical Names	9-10
Defining Keys at the DCL Prompt	9-11
Define/Key Command Qualifiers	9-12
Learning Check	9-13

Module 10 — Introduction to Command Procedures

Objectives.....	10-1
Topics.....	10-1
Command Procedures	10-2
WRITE Command.....	10-3
Command Procedure Output.....	10-3
Command Procedure Usage Example	10-4
Positional Parameter Symbol Example.....	10-6
Command Procedure Development.....	10-7
Conventions and Rules	10-8
Conditional Execution	10-9
Determining Mode of Execution.....	10-10
F\$MODE()	10-10
Examples.....	10-10
LOGIN.COM	10-11
Characteristics	10-11
Considerations for LOGIN.COM.....	10-11
Sample LOGIN.COM.....	10-12
PIPE Command	10-16
Learning Check	10-20

Module 11 — Disk-to-Tape Activities

Objectives.....	11-1
Topics.....	11-1
File Copy from Disk to Tape	11-2
Steps to Copy a Tape	11-2
Further Tape Activities	11-3
User-Level Backup	11-4
Replicating a Directory Hierarchy.....	11-4
Backups to Tape	11-6
Processing Floppy Disks.....	11-7
Sample Floppy Copy	11-8
Learning Check	11-16

Module 12 — Queues: Batch and Print

Objectives.....	12-1
Topics.....	12-1
Queues.....	12-2
Batch Queues.....	12-3
Commonly Used Qualifiers.....	12-3
Sample Command Procedure Using Parameters.....	12-5
Process Permanent Logical Name Contrasts	12-7
Print Queues	12-8
Learning Check	12-9

Appendix A — Extending Command Procedures

Objectives.....	A-1
Topics.....	A-1
Terminal I/O	A-2
Terminal Output and Input.....	A-3
INQUIRE Example.....	A-4
Example of Using Command Parameters.....	A-5
Example of Changing Process Permanent Logical Names.....	A-6
Reading from a File	A-8
Steps to Read a File	A-8
Example of Reading from a File	A-9
Sample Use of READ with a Scratch File	A-11
Writing to a File.....	A-14
Steps to Write to a File.....	A-14
Relational Operators	A-15
Controlling Code Flow	A-16
Loops	A-17
Subroutines.....	A-21
Lexical Functions	A-23
Lexical Function Summary	A-24
Lexical Function Examples.....	A-26
Example of Using Lexicals	A-29
Using Lexicals to Determine Volume Label.....	A-30
Using Lexicals to Set Prompt.....	A-31
Using Lexicals to Determine Day of Week	A-32
Using Lexicals to Obtain Device Information	A-33
Error Handling.....	A-34
Learning Check	A-35

Appendix B — Program Development Environment

Objectives.....	B-1
Topics.....	B-1
Development Steps.....	B-2
Object Libraries	B-4
Automating Library Search Through Logical Names	B-6
Libraries and Object Modules	B-7
Library Example	B-8
Linker Search	B-10
The Debugger	B-11
Learning Check	B-12

Introduction

This course teaches you to use the OpenVMS operating system to create, manipulate, and manage files. You will learn how to tailor the user environment using logical names and DCL symbols.

The DCL command language interpreter is described, and you will learn how to use the command line interface. You will learn to develop basic command procedures and be able to create a LOGIN.COM command file to tailor your login session.

Course Length

Four days.

Intended Audience

OpenVMS users and programmers.

Prerequisites

Basic knowledge of computer use.

Course Objectives

To demonstrate a fundamental knowledge of the OpenVMS operating system, you should be able to:

- Tailor the user environment using logical names and DCL symbols.
- Describe the DCL command language interpreter and use its command line interface.
- Develop basic command procedures.
- Create a LOGIN.COM command procedure to tailor your login session.

Course Goals

Upon completing this course, you should be able to:

- Log into an OpenVMS system and use DCL.
- Create and modify files using an editor.
- Manage files and organize them into directory files.
- Set protection permissions on files.
- Create and use DCL symbols.
- Create and use logical names.
- Use the Mail utility to read and send mail.
- Write basic command procedures.
- Create a LOGIN.COM command procedure.
- Submit batch and print jobs to queues.

Course Outline

Hardware Overview

- Overview of System Architecture
- VAX Compared to Alpha
- Peripherals
- OpenVMS Clusters
- Networks
- Networking Overview

Software Overview

- LOGIN
- The Process
- Software Context
- Hardware Context
- Virtual Address Space
- Virtual Address Space on Alpha
- Process Types
- Process Privileges

- System Processes
- Process Terms
- Utilities
- Layered Products

Introduction to DCL

- DCL
- DCL Features
- Command Line Format
- Command String Elements
- DCL Informational Commands
- DCL and File Oriented Parameters
- DCL File Name Parsing Example
- Qualifiers
- Getting Started with HELP
- Basic DCL Commands
- Command Line Editing
- Control Keys
- Terminal Characteristics
- /PAGE Qualifier
- Error Messages
- HELP Message Utility

Using Editors

- Editing Options
- EVE Overview
- EVE Editor
- EVE File Recovery
- EDT Overview
- EDT Line Mode Commands
- Mail and Phone

Mail and Phone

- Mail Utility
- Phone Utility

Directories and Files

- Complete File Specification
- Directories
- Creating Directories
- Directory Command
- Deleting Subdirectories
- Wildcards
- File Protection
- Processes, Objects, and Protections
- Protection Categories
- Categories of File Access
- File Protection
- Access Control Lists

File Manipulation

- File Management Commands
- APPEND
- Record Types
- DUMP Utility
- DIFFERENCES
- SEARCH Command

Logical Names

- Logical Names
- Logical Name Tables
- Group Logical Name Tables
- System Logical Name Table
- Logical Name Translation
- Process Permanent Logical Names
- Other Logical Name Features

Symbols

- DCL Symbols
- Symbol Substitution
- Symbol Types
- Integer Symbol Examples
- Symbols Compared to Logical Names
- Defining Keys at the DCL Prompt

Introduction to Command Procedures

- Command Procedures
- WRITE Command
- Positional Parameter Symbols
- Command Procedure Development
- Conventions and Rules
- Conditional Execution
- Determining Mode of Execution
- LOGIN.COM
- PIPES

Disk-to-Tape Activities

- File Copy from Disk to Tape
- Further Tape Activities
- User-Level Backup
- Backups to Tape
- Processing Floppies

Queues: Batch and Print

- Queues
- Batch Queues
- Process Permanent Logical Name Contrasts
- Print Queues

Optional Topics

- Extending Command Procedures
- Program Development Environment

Objectives

Computer software is meaningless without computer hardware on which to run. The OpenVMS operating system operates on two different hardware architectures: VAX and Alpha. These architectures consist of some common subsections such as CPU, memory, and peripherals, which are introduced in this module. The concept of joining or connecting various VAX systems is also examined.

This module serves as a hardware platform on which to build the rest of the software details found in the following chapters. After completing this module, you will be able to:

- List the main components of an OpenVMS system.
- Describe the differences between VAX and Alpha architectures.
- Describe a typical OpenVMS Cluster system.
- List the main characteristics of a network.

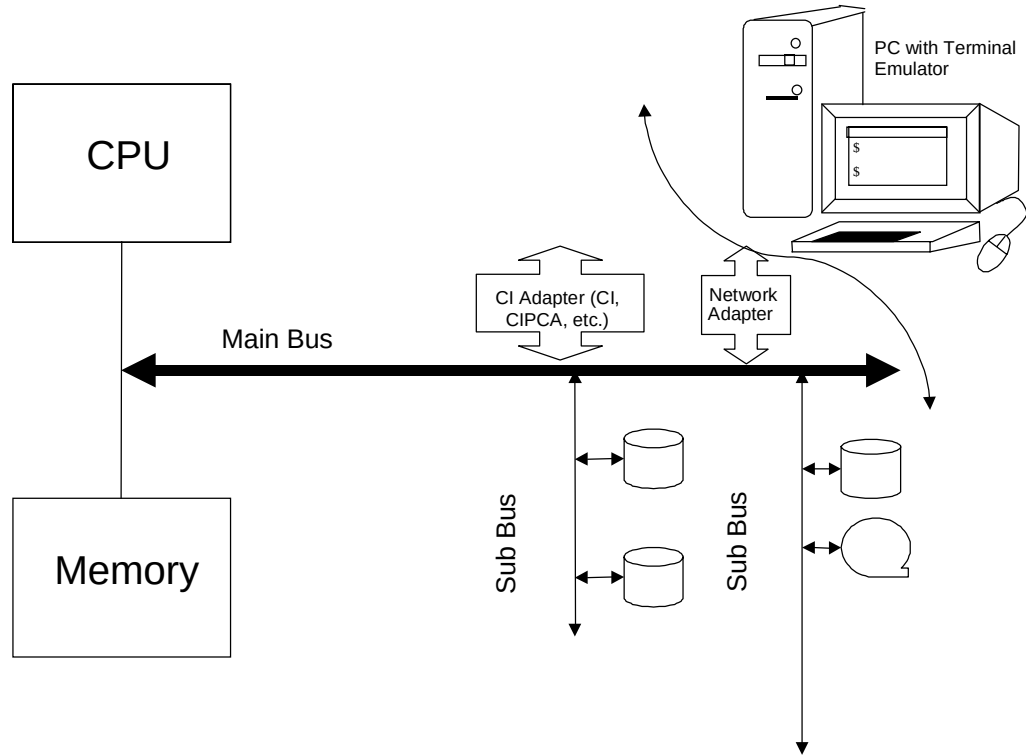
Topics

- OpenVMS System Components
- VAX Compared to Alpha
- OpenVMS Cluster Systems
- Networks

System Architectures

- A computer architecture identifies the method in which a central processing unit (CPU) operates under all processor types within the computer family, including:
 - Methods of addressing data
 - Instructions supported
 - Number and types of registers
 - Method of servicing device interrupts
 - How memory will be managed and mapped
- The CPU communicates with main memory and peripheral devices through buses. The bus structure varies depending on the type of system on which it is implemented.
- Main memory is relatively expensive, somewhat limited (tens to hundreds of megabytes normally, to gigabytes).
- Disks provide peripheral storage at a cheaper cost per byte. Normally disks have a larger capacity (gigabytes to terabytes).

Overview of System Architecture



VAX Compared to Alpha

VAX (Virtual Address eXtension)

- Design grew from the limitations imposed by the 16-bit addressing on the PDP-11.
- 32-bit architecture
- Complex instruction set, variable length instructions
- 16 general-purpose registers
- Odd byte addressable

Alpha

- Fastest processor in the computer industry
- 64-bit architecture
- 32 integer and 32 floating-point registers
- Longword and quadword addressable
- Reduced Instruction Set Computer (RISC) architecture
- VAX programs compiled under OpenVMS do not run on Alpha OpenVMS systems and vice versa.
 - VAX programs can be:
 - ◆ VESTed to run on Alpha
 - ◆ Recompiled and relinked (preferable)
- Memory allocation unit, the page size is different between architectures:
 - VAX = 512-byte fixed page size
 - Alpha = Currently 8192 bytes per page. The architecture supports 16K, 32K, and 64K byte page sizes.

Peripherals

- An OpenVMS system includes a variety of I/O (input/output) peripherals, including: disks, tapes, terminals, and so forth
- To view the peripherals connected to your system, enter:
\$ SHOW DEVICE

Example \$ SHOW DEVICE

\$ sh dev

Device Name	Device Status	Error Count	Volume Label	Free Blocks	Trans Count	Mnt Cnt
ELLIS\$DKA200:	Mounted	0	AXP71FT2	26661	367	1
ELLIS\$DKA300:	Online	0				
ELLIS\$DKA500:	Online	0				
ELLIS\$DKA600:	Online wrtlck	0				

Device Name	Device Status	Error Count
ASN0:	Online	0
FTA0:	Offline	0
LTA0:	Offline mounted	0
OPA0:	Online	0
RTA0:	Offline	0
RTB0:	Offline	0
TNA0:	Offline	0
TNA3:	Mounted	0
TTA1:	Online	0

Device Name	Device Status	Error Count
ESA0:	Online	0
ESA3:	Online	0
ESA4:	Online	0
ESA5:	Online	0
ESA6:	Online	0
ESA7:	Online	0
...		

To view more specific information on an individual device, enter:

\$ SHOW DEVICE/FULL

Example \$ SHOW DEVICE/FULL

\$ sh dev dka200/full

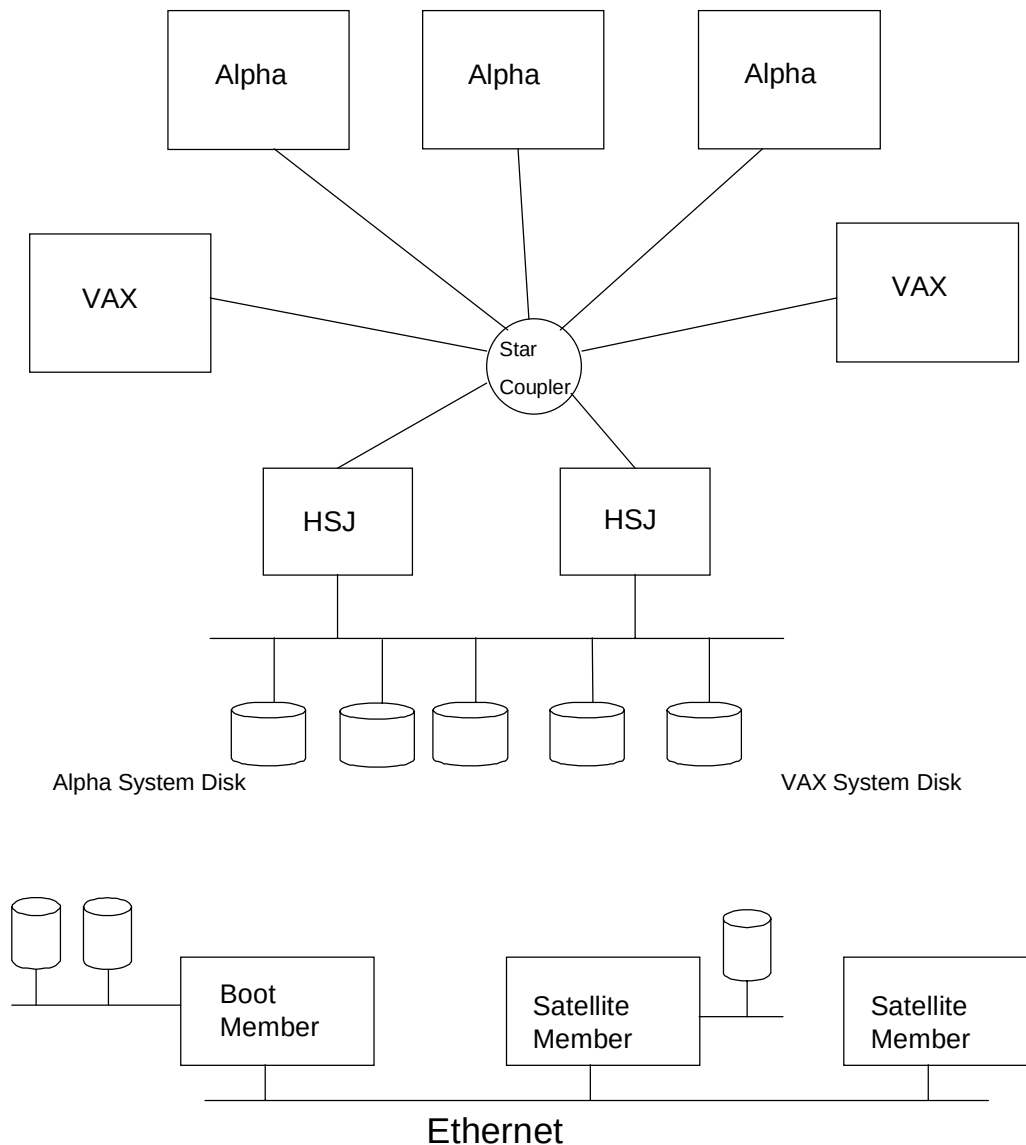
Disk ELLIS\$DKA200:, device type RZ26L, is online, mounted, file-oriented device,
shareable, available to cluster, error logging is enabled.

Error count	0	Operations completed	37555
Owner proces	""	Owner UIC	[SYSTEM]
Owner process ID	00000000	Dev Prot	S:RWPL,O:RWPL,G:R,W
Reference count	252	Default buffer size	512
Total blocks	2050860	Sectors per track	83
Total cylinders	3089	Tracks per cylinder	8
Volume label	"AXP71FT2"	Relative volume number	0
Cluster size	3	Transaction count	349
Free blocks	1034634	Maximum files allowed	256357
Extend quantity	5	Mount count	1
Mount status	System	Cache name	"_ELLIS\$DKA200:XQPCACHE"
Extent cache size	64	Maximum blocks in extent cache	103463
File ID cache size	64	Blocks currently in extent cache	103329
Quota cache size	0	Maximum buffers in FCP cache	164
Volume owner UIC	[1,1]	Vol Prot	S:RWCD,O:RWCD,G:RWCD,W:RWCD

Volume Status: subject to mount verification, protected subsystems enabled,
file high-water marking, write-through caching enabled.

\$

OpenVMS Clusters

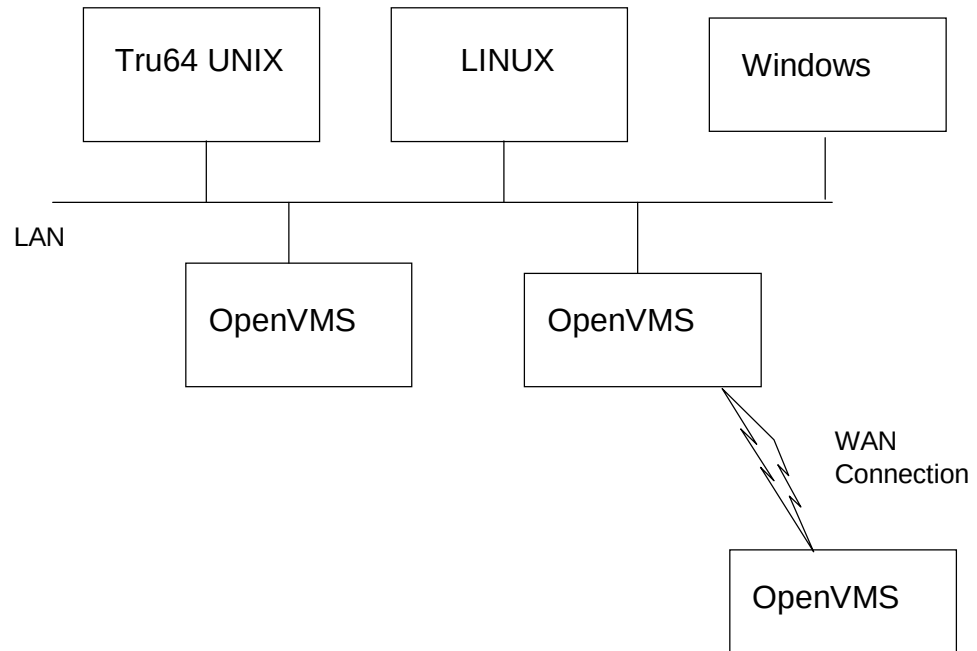


- Allow convenient incremental growth path for customers.
- Provide for ease of sharing of storage devices. Transparent device/file sharing.
- With the inclusion of HSC/HSZ/HSJ/HSG/HSD controllers, disks become available, independent of processor.
- Local Area VAXcluster (LAVC) systems allow “cluster-like” sharing for machines incapable of supporting a CI adapter.
- Provide for “diskless” systems.

Networks

- Networks consist of computers communicating over some medium (wire, microwaves, fiber optics).
- Networks include hardware (wires, communication devices) and software (DECnet Plus, TCP/IP) support.
- Users can communicate across systems.
- Users can access remote disks.
- Users can use programs and command procedures to communicate with a remote node.

Networking Overview



Learning Check

1. Describe the main hardware components of an OpenVMS system.

.....

.....

.....

.....

2. Name at least three differences between the VAX and Alpha architectures.

VAX	Alpha

3. Name at least three benefits of using an OpenVMS Cluster system.

.....

.....

.....

.....

4. What are the main characteristics of a network?

.....

.....

.....

.....

Objectives

Computer hardware is meaningless without computer software to run on it and control it. The OpenVMS operating system and the software designed to run with it is very rich, flexible, and sophisticated. OpenVMS is one of the most thorough operating systems as far as supplying the user with utilities and built-in capabilities.

This module introduces you to the concepts underlying all of the OpenVMS software that runs on the VAX and Alpha platforms. After completing this module, you will be able to:

- Log in to an OpenVMS system.
- Describe the major components of the software.
- Describe the virtual address space.
- Identify the different types of processes that run on an OpenVMS system.
- Describe the difference between a utility and a layered product.

Topics

- OpenVMS
- Logging In
- Process Overview
- Hardware and Software Context
- Virtual Address Space
- Process Types
- Utilities
- Layered Products

OpenVMS

- OpenVMS
 - Virtual Memory System
 - Runs on VAX and Alpha hardware platforms
- Responsibilities of OpenVMS
 - Loads and executes programs
 - Accesses data as needed from memory
 - Supports the retrieval of data from I/O devices
 - Schedules the CPU's activities
 - Manages and allocates the system's resources in general
- DCL
 - Digital Command Language is the user's primary interface with OpenVMS.

Login and User Name

To get the attention of the system, you log in to the system.

- At your terminal, press Enter/carriage return <CR>, provided you have a terminal that is connected to the system.
- If you are connected through a terminal server or through a PC connection (PowerTerm, KEAterm, Excursion, and so forth.), connect to the service/system.

```
USERNAME: obrien<CR>
```

```
PASSWORD: <CR>      ! Will not echo.
```

```
$                  ! DCL prompt.
```

The system manager assigns you a user name and password.

- \$
The DCL prompt means you have successfully logged in and can now issue requests of the system.

- WELCOME TO OpenVMS

You may also see a message like this before the first \$ prompt.

- UAF

Before you see the welcome to OpenVMS message, the system checks to see if the system manager has authorized your activities. It checks USERNAME and PASSWORD against records in the User Authorization File (UAF), which is named SYSUAF.DAT.

The Process

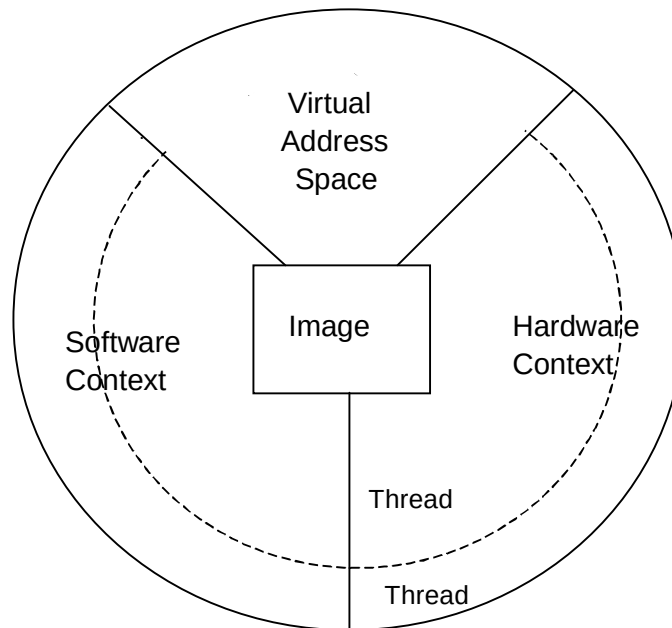
The process is the basic initial scheduling environment under OpenVMS. It defines the total environment in which an image (program) executes.

The environment is very controlled and different for each user depending on the parameters set up for you by your system manager.

All users on OpenVMS are not created equal. (The system manager is a very powerful user.)

In OpenVMS, the emphasis is on the process or the total environment in which an image (program) executes rather than the image itself.

Processes are made up of one or more threads of execution. Under OpenVMS Version 7.0 and higher, threads are the basic scheduling entity.



Software Context

Your software context is made up of information determining what your process can do and when it can do it (primarily from the UAF).

- **Account Name**
 - Identifies your process for billing purposes
 - Multiple users can share the same account name
- **Default Directory**

Indicates where your files will be created by default
- **Default I/O Device**

Your terminal (for example, TTA4:)
- **Password**
 - Validates your right to use the system
 - Is never displayed anywhere on the system (encrypted when stored)
- **Process Identification Code (PID)**
 - Uniquely identifies your process to the system
 - Consists of eight hex digits
- **Process Name**
 - Identifies your process to other users
 - Usually matches your user name
- **User Name**

Identifies the record in the UAF through which you gained access to the system

- **User Identification Code**
 - A two-part code used in file protection scenarios [group,member] (not normally visible to the user)
 - An alphanumeric identifier normally represents the UIC
- **Priority**
 - Indicates the relative scheduling importance of your process to the system
 - Ranges from 0 to 31 on VAX and 0 to 63 on Alpha OpenVMS (highest = most important)
 - Interactive priorities range from 0 to 15
 - Time-critical processes range from 16 to 31 on VAX and 16 to 63 on Alpha
 - Normal priority for standard interactive use is 4
 - Your priority can “float” depending on activity
- **Privileges**

Determine which protected system operations you can perform
- **Resource Limits/Quotas**

Determine how much of certain system resources you can use
- **Protection**

Default file/object permissions, based on UIC/held identifiers

Software Context Information Display

\$ show process/all

14-MAR-1999 23:00:43.78 User: BAE Process ID: 00000122
 Node: ELLIS Process name: "BAE"

Terminal: TNA3: (Host: 192.206.126.197 Port: 1062)

User Identifier: [BRUDEN,BAE]

Base priority: 4

Default file spec: ELLIS\$DKA200:[BAE]

Number of Kthreads: 1

Devices allocated: ELLIS\$TNA3:

Process Quotas:

Account name:

CPU limit:	Infinite	Direct I/O limit:	150
Buffered I/O byte count quota:	99808	Buffered I/O limit:	150
Timer queue entry quota:	10	Open file quota:	100
Paging file quota:	46672	Subprocess quota:	10
Default page fault cluster:	64	AST quota:	248
Enqueue quota:	2000	Shared file limit:	0
Max detached processes:	0	Max active jobs:	0

Accounting information:

Buffered I/O count:	280	Peak working set size:	2288
Direct I/O count:	43	Peak virtual size:	168848
Page faults:	463	Mounted volumes:	0
Images activated:	5		
Elapsed CPU time:	0 00:00:01.02		
Connect time:	0 01:19:30.17		

Authorized privileges:

NETMBX OPER SETPRV SYSPRV TMPMBX

Process privileges:

NETMBX	may create network device
OPER	may perform operator functions
SETPRV	may set any privilege bit
SYSPRV	may access objects via system protection
TMPMBX	may create temporary mailbox

Process rights:

```
BAE                                     resource
INTERACTIVE
REMOTE
VMS$MEM_RESIDENT_USER
NET$DECLAREOBJECT
VMS$BUFFER_OBJECT_USER
SUPER_USER
```

System rights:

SYSSNODE_ELLIS

Auto-unshelve: on

Image Dump: off

Soft Affinity: off

Parse Style: Traditional

Process Dynamic Memory Area

Current Size (bytes)	131072	Current Size (pagelets)	256
Free Space (bytes)	117868	Space in Use (bytes)	13204
Largest Variable Block	117260	Smallest Variable Block	24
Number of Free Blocks	8	Free Blocks LEQU 64 Bytes	3

There is 1 process in this job:

BAE (*)

\$

Sample UAF Record

```
$ set default sys$system
```

```
$ set process/privilege=sysprv
```

```
$ run authorize
```

```
UAF> show bae
```

```
Username:  BAE                      Owner:
Account:   BRUDEN                   UIC:    [42,1]  ([BRUDEN,BAE])
CLI:       DCL                      Tables:  DCLTABLES
```

```
Default:  ELLIS$DKA200:[BAE]
```

```
LGICMD:
```

```
Flags:
```

```
Primary days:  Mon Tue Wed Thu Fri
```

```
Secondary days:                      Sat Sun
```

```
No access restrictions
```

```
Expiration:      (none)      Pwdminimum:    6      Login Fails:    0
```

```
Pwdlifetime:     999 00:00    Pwdchange:    28-JUN-1998 13:37
```

```
Last Login: 14-MAR-1999 21:41 (interactive), 19-FEB-1999 23:12 (non-interactive)
```

```
Maxjobs:        0   Fillm:    100   Byt1m:        64000
```

```
Maxacctjobs:    0   Shrfillm:  0   Pbyt1m:        0
```

```
Maxdetach:      0   BIO1m:    150   JTquota:       4096
```

```
Prclm:          8   DIO1m:    150   WSdef:         2000
```

```
Prio:           4   AST1m:    250   WSquo:         4000
```

```
Queprio:        4   TQE1m:    10   WSextent:     16384
```

```
CPU:            (none)  Enqlm:   2000   Pgflquo:     50000
```

```
Authorized Privileges:
```

```
  NETMBX      OPER      SETPRV      SYSPRV      TMPMBX
```

```
Default Privileges:
```

```
  NETMBX      OPER      SETPRV      SYSPRV      TMPMBX
```

```
Identifier                      Value      Attributes
```

```
VM$MEM_RESIDENT_USER           %X80000009
```

```
NET$DECLAREOBJECT               %X91F50005
```

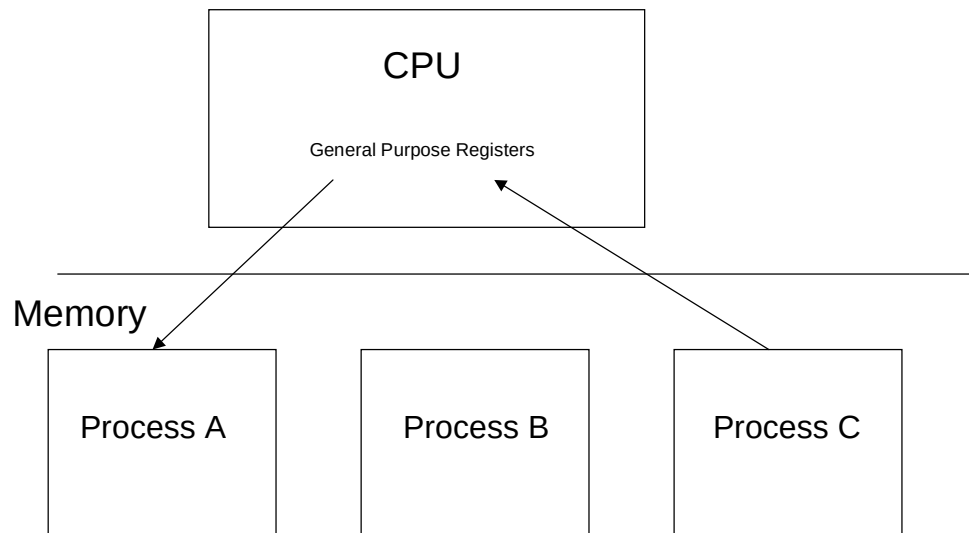
```
VM$BUFFER_OBJECT_USER          %X80000008
```

```
SUPER_USER                      %X80010002
```

```
UAF>
```

Hardware Context

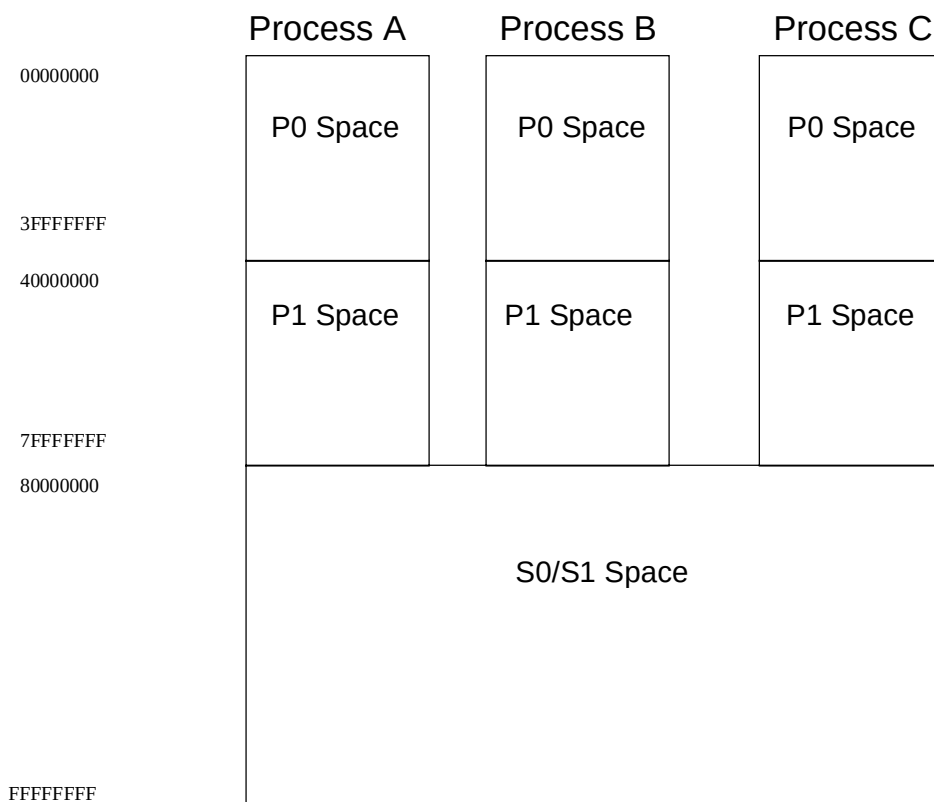
- Information held in a software data structure that holds a copy of data from the CPU registers while you are not in the CPU.
- Only one process can use the CPU at any one instant (the current process).
- Because the CPU is “time shared,” it is necessary to repeatedly remove one process from the CPU and replace it with another. This gives the users the illusion of “owning the computer.” The removed process will (typically) soon be back using the CPU.
- While a process is not in control of the CPU, the hardware context memory area is used to “remember” the information that was in the CPU registers when the process was last using the CPU.



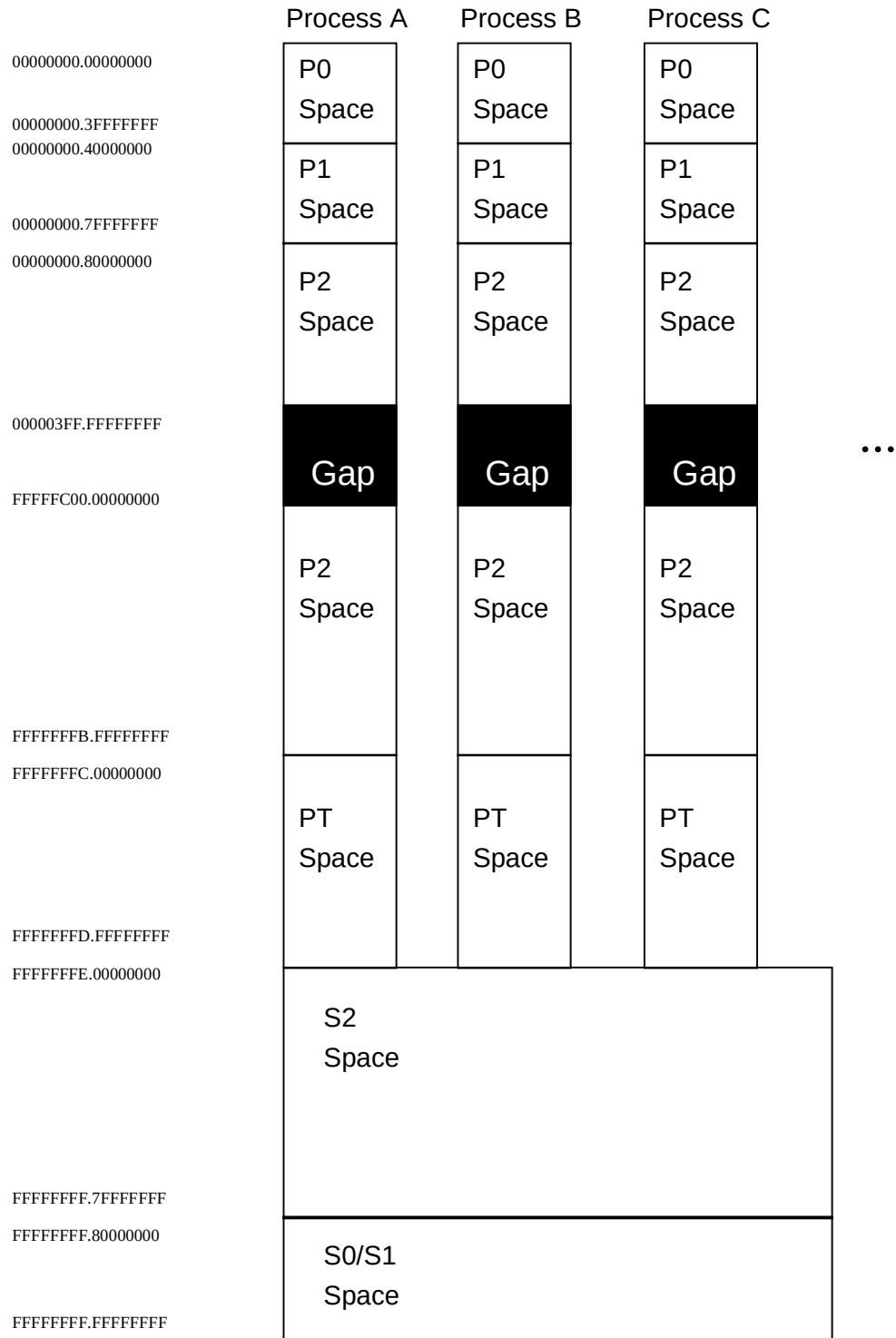
Virtual Address Space

The VAX system's 32-bit addressing enables 4.2 gigabytes of virtual memory. Virtual memory on a VAX is made up of three address spaces: P0, P1, and S0/S1 space. Physical memory is machine and site dependent.

- The Alpha system's 64-bit addressing enables access to terabytes of virtual memory. Virtual memory on an Alpha running OpenVMS supports P0, P1, P2, PT, S2, and S0/S1 spaces.
- The virtual address space allows the “mapping” of program and data storage which can be in memory and/or on disk.



Virtual Address Space on Alpha



Not to scale

Process Types

- **Interactive**

A terminal-oriented dialog between a user and OpenVMS

- **Batch**

A file of commands “submitted” to a batch queue

- **Subprocess**

A process sharing some quotas with a parent process

```
$ SPAWN                !Creates a subprocess.
```

```
% DCL-S-SPAWNED, process OBRIEN_1 spawned
```

```
% DCL-S-ATTACHED, terminal now attached to process
OBRIEN_1
```

```
$                !This $ is from the subprocess.
```

```
$LOGOUT            !Ends the subprocess.
```

```
$                !This $ is from the parent.
```

- **Network**

Created to implement support access activity, such as the File Access Listener (FAL)

```
$ ! From remote node, creates network process on our
```

```
$ ! node
```

```
    $ DIRECTORY OUR_NODE"user pwd"::
```

- **Detached**

A process that has no parent, such as JOB_CONTROL or OPCOM

Process Privileges

\$ show process/privilege

14-MAR-1999 23:51:27.04 User: BAE Process ID: 00000122
 Node: ELLIS Process name: "BAE"

Authorized privileges:

NETMBX OPER SETPRV SYSPRV TMPMBX

Process privileges:

NETMBX	may create network device
OPER	may perform operator functions
SETPRV	may set any privilege bit
SYSPRV	may access objects via system protection
TMPMBX	may create temporary mailbox

Process rights:

BAE resource
INTERACTIVE
REMOTE
VMS\$MEM_RESIDENT_USER
NET\$DECLAREOBJECT
VMS\$BUFFER_OBJECT_USER
SUPER_USER

System rights:

SY\$NODE_ELLIS

\$

System Processes

\$ show system

OpenVMS V7.2 on node ELLIS 14-MAR-1999 23:52:17.47 Uptime 0 03:07:19

Pid	Process Name	State	Pri	I/O	CPU	Page flts	Pages
00000101	SWAPPER	HIB	16	0	0 00:00:00.23	0	0
00000106	LANACP	HIB	13	65	0 00:00:00.23	135	113
00000108	FASTPATH_SERVER	HIB	10	9	0 00:00:00.09	69	82
00000109	IPCACP	HIB	10	10	0 00:00:00.06	31	43
0000010A	ERRFMT	HIB	8	365	0 00:00:01.15	87	101
0000010C	OPCOM	HIB	6	205	0 00:00:02.78	84	43
0000010D	AUDIT_SERVER	HIB	10	58	0 00:00:00.29	137	163
0000010E	JOB_CONTROL	HIB	10	40	0 00:00:00.13	52	71
0000010F	QUEUE_MANAGER	HIB	9	223	0 00:00:00.64	127	162
00000110	SECURITY_SERVER	HIB	10	32	0 00:00:01.28	312	342
00000111	DNS\$ADVER	LEF	5	738	0 00:00:02.03	427	241
00000112	LESS\$ACP_V30	HIB	9	127	0 00:00:00.41	78	90
00000113	NET\$ACP	HIB	4	77	0 00:00:00.46	130	152
00000114	REMACP	HIB	8	10	0 00:00:00.03	29	16
00000115	NET\$EVD	HIB	6	20	0 00:00:00.44	247	241
00000116	DTSS\$CLERK	LEF	12	181	0 00:00:00.73	337	108
00000117	TP_SERVER	HIB	9	748	0 00:00:06.74	76	94
00000118	LATACP	HIB	14	8	0 00:00:00.20	73	58
00000119	TCPIP\$INET_ACP	HIB	8	84	0 00:00:00.31	250	145
0000011A	TCPIP\$PWIP_ACP	HIB	10	53	0 00:00:00.25	151	136
0000011B	TCPIP\$NTP	LEF	8	11199	0 00:00:11.50	343	202 N
0000011D	VPM_SERVER	HIB	15	14	0 00:00:00.28	215	186
0000011F	DECW\$SERVER_0	HIB	8	796	0 00:00:05.53	663	872
00000120	DTLOGIN	LEF	5	143	0 00:00:00.90	325	244
00000121	DTGREET	LEF	4	461	0 00:00:08.59	804	366 S
00000122	BAE	CUR	0 4	590	0 00:00:01.80	773	126

\$

Process Terms

- **Job**

Consists of a parent process and its subprocesses. The job is used to account for usage of system resources.

- **Image**

- Only one image within one process can be executed at any given time by a CPU (though it seems like many are executing at once). Processes will compete for the CPU based on priorities.
- An image is a binary file that contains code and data. Normally the term image refers to a program.

- **Thread**

Threads may be created by explicitly or implicitly using the DECthreads manager. If Kernel threads have been enabled, the threads are treated as separate schedulable entities that share the same virtual address space.

Utilities

Utilities are software tools designed to perform specific tasks for users. Utilities differ from commands in that they commonly have their own internal command syntax and prompts.

- Utilities for user communication
 - Mail
 - Phone
- Utility for program debugging
 - OpenVMS Debugger
- Editors
 - EDT (\$ EDIT/EDT)
 - EVE (\$ EDIT)
 - Others
- Data File Manipulation
 - \$ CONVERT
 - \$ ANALYZE/RMS
 - \$ EDIT/FDL
 - Others

Layered Products

- Layered products are software tools available to be added to OpenVMS.
- They are created by Compaq or by third parties.
- A major difference between utilities and layered products is that the utilities are packaged with OpenVMS and layered products are purchased separately from OpenVMS. Layered products, typically, must be licensed separately, using the License Management Facility (LMF).
- Examples:
 - Database tools (ORACLE)
 - Compilers (FORTRAN, C, COBOL)
 - Office automation options (ALL-IN-1)
 - Software development tools (CMS, MMS, DTM)
 - Disk defragmenters (DFO, DISKKEEPER)

Learning Check

1. What is software context?

.....

.....

.....

2. What is hardware context?

.....

.....

.....

3. What is the purpose of virtual address space?

.....

.....

.....

4. Name the five types of processes.

.....

.....

.....

5. A major difference between utilities and layered products is:

.....

.....

.....

Objectives

Any operating system needs a method for the users to interact with it (to talk to it). In the OpenVMS arena, that method is the Digital Command Language (DCL). DCL is a complete and flexible command language that gives the nonprivileged and privileged user alike some hooks into OpenVMS.

This module introduces you to the parts of DCL that are used most often from the user's perspective. After completing this module, you will be able to:

- List the three parts of a DCL command.
- List the three types of DCL commands.
- Describe how a DCL command is parsed.
- Use the Help utility.
- List at least 10 basic DCL commands.
- Use DCL's command line editing capabilities.
- Describe the parts of an error message.

Topics

- DCL Features
- Command Types
- Command Qualifiers
- The Command String
- Logging In/Out
- Command Samples
- Command Line Editing
- \$RECALL
- Control Keys
- \$HELP
- Error Messages
- Common Errors

DCL

Digital Command Language (DCL) provides the language through which we communicate with the operating system.

DCL includes commands, parameters, and qualifiers that can be thought of as similar to verbs, nouns, and adverbs and adjectives in the English language:

- **Verb (command)**
Specifies the action you want DCL to perform
- **Parameter**
Object of the command (file, device, or keyword)
- **Qualifiers**
 - Modify the default action of the verb
 - Preceded by a slash (/).

The first thing you may want to do with DCL is change your password.

```
$SET PASSWORD
old password:  !system prompts
new password:  !no echo
verification:
```

Passwords

- Can be up to 31 characters long.
- Cannot be found if you forget it. If forgotten, contact the your system manager to have it changed.
- Can be generated, for example:
\$ set password/generate=10
Old password:

duckerband
choexculls
comoldence
thertensms
reentanted

Choose a password from this list, or press RETURN to get
a new list

New password:

Verification:

\$

DCL Features

DCL provides the medium for communication with OpenVMS.

- English-like:
 - To print something
\$ PRINT
 - To view something
\$ SHOW
 - To delete something
\$ DELETE
 - To type contents on terminal
\$ TYPE
 - To list file names
\$ DIRECTORY
- DCL prompts you for left out information, for example:
\$PRINT<CR>
What file?

\$COPY<CR>
From:
To:

- Commands can be abbreviated.
 - DCL, normally, only looks at the first four letters of a command.
 - Can shorten to fewer than four letters if the character sequence is unique.
\$ DIRECTORY
\$ DIRE
\$ DIREX
\$ DIR
 - These all work, but \$DI will cause an error because it is not unique.
- Command line continuation is implemented by the hyphen (-).
 - Command string can have up to 1024 characters.
 - Continuation prompt is _\$.
\$ PRINT/HEADER/BURST/FLAG/JOB COUNT=4/TRAIL -
_ \$ NOTIFY/NOTE="MY STUFF!" FRED.DAT
- Comments can be included in command lines (usually for command procedures).
 - Comments are begun by the exclamation point (!).
 - Can be inserted at the start of a line or to the right of a DCL command.
\$PRINT FILE.DAT ! to print the file
or
\$! This is a full line comment.
- Online help is available.
 - A convenient subset of the DCL dictionary.
\$HELP ! The system will list the topics for which
! online help is available.

DCL Session

Starting a DCL Session

Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2

Username: bae

Password:

Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2 on node ELLIS

Last interactive login on Sunday, 14-MAR-1999 21:41:24.40

Last non-interactive login on Friday, 19-FEB-1999 23:12:34.99

\$

\$

\$ logout/full

BAE logged out at 26-MAR-1999 09:29:08.21

Accounting information:

Buffered I/O count:	53	Peak working set size:	1856
Direct I/O count:	31	Peak virtual size:	168016
Page faults:	244	Mounted volumes:	0
Charged CPU time:	0 00:00:00.36	Elapsed time:	0

\$ LOGOUT

Without the /FULL qualifier, DCL does not display accounting information.

Command Line Format

\$ VERB/QUALIFIERS(optional) PARAMETERS(optional)/QUALIFIER(optional)

- VERB: Specifies action
- PARAMETERS: Object of the command
- QUALIFIER: Modifies the default action of the verb (adverb). Can be used as a parameter qualifier or a command qualifier.

Types of Commands

There are three types of commands:

- **Standalone commands**

Consist of verbs used without parameters that produce results from the system by themselves.

\$ DIRECTORY

\$ PHONE

\$ PURGE

\$ MAIL

- **Commands requiring options**

All need a second part to be a complete command.

\$ SET PASSWORD

\$ SET TERMINAL

\$ SET DEFAULT

\$ SHOW DEFAULT

\$ SHOW PROCESS

\$ SHOW SYSTEM

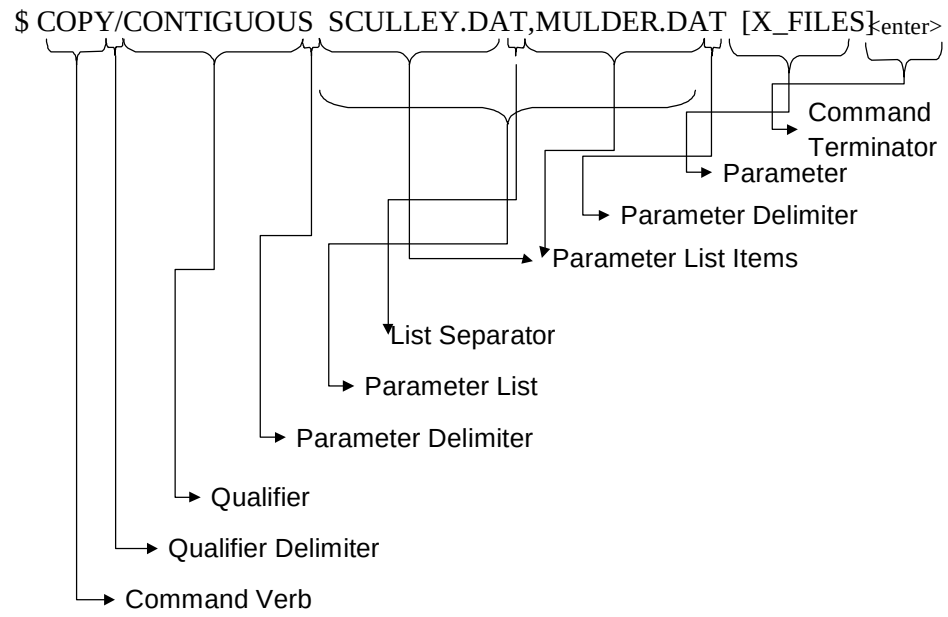
- **Commands requiring parameters**

\$ PRINT TEST.DAT

\$ COPY FRED.DAT MINE.DAT

\$ DELETE ABC.EFG;2

Command String Elements



DCL Informational Commands

\$ show users

OpenVMS User Processes at 26-MAR-1999 21:47:18.03

Total number of users = 2, number of processes = 5

Username	Interactive	Subprocess	Batch
BAE	1		
SYSTEM	3	1	

\$ show users/full

OpenVMS User Processes at 26-MAR-1999 21:47:25.69

Total number of users = 2, number of processes = 5

Username	Process Name	PID	Terminal
BAE	BAE	00000123	TNA4: (Host: 192.206.126.197 Port: 1067)
SYSTEM	DTSESSION	00000120	
SYSTEM	DTWM	00000128	(subprocess of 00000120)
SYSTEM	_FTA1:	0000012C	FTA1:
SYSTEM	_FTA2:	0000012D	FTA2:

\$

\$ show time

26-MAR-1999 21:48:13

\$ show status

Status on 26-MAR-1999 21:48:57.51 Elapsed CPU : 0 00:00:07.38

Buff. I/O : 1698 Cur. ws. : 2000 Open files : 0

Dir. I/O : 303 Phys. Mem. : 1072 Page Faults : 2372

\$

DCL Command Line Parsing

DCL parses a command by first determining the verb that has been entered. The verb is the first portion of the command line, delimited by white space.

Verb Processing

The verb may be:

- User defined (symbol)
- Known to DCL through its command tables (SYS\$SHARE:DCLTABLES.EXE)

The verb may require a parameter to complete its syntax. For example:

`$ SHOW TIME`

`$ SHOW STATUS`

The verb may be implemented by:

- A user program, explicitly or implicitly, activated by the \$RUN command.
- DCL internally
`$SHOW TIME`, `$SHOW STATUS`, `$SHOW TRANSLATION`, `$SPAWN`
- A DCL-defined image (program), located, by default, in the SYS\$SYSTEM: directory.
`$TYPE`, `$DIRECTORY`, `$EDIT`

Changing Verb Behavior

The behavior of a command may be changed by the use of parameters and/or qualifiers.

- It is the responsibility of the system/user program to obtain the parameters from DCL and to process them.
- A command can accept a maximum of eight parameters from the command line.
- Parameters can be comma-delimited lists (with no embedded white space).
- Qualifiers are designated by a leading slash (/).

DCL and File-Oriented Parameters

The most common form of parameter passed to a DCL command is a file specification. File specifications will be covered in more detail later. It is important to understand the basics of file specifications, prior to working with DCL.

- Files have names and types.
 - The name and type are delimited by a period (.).
 - The period delimiter is always part of the file name, even if you do not specify a type.
- OpenVMS supports multiple versions of files.

The version number follows the file type and is preceded by a semicolon (;).
- Files are cataloged in directories.

The directory name precedes the file name and is embedded between square brackets ([]).
- Directories are located on devices.

A colon (:) separates the device name and directory name.

File Specification Defaults

- Any piece of the device and directory name missing is taken from the default directory string.
- Many commands (but not all) have default input file types, for example:
 - \$ TYPE defaults to .LIS
 - \$ CC defaults to .C
 - \$ LINK defaults to .OBJ
 - \$ RUN defaults to .EXE
- Most commands default to the most recent version of the file.

User-Defined File Names

The user/system can define the file name (using logical names).

DCL File Name Parsing Example

The following command attempts to display a nonexistent file:

```
$ type x
```

```
%TYPE-W-SEARCHFAIL, error searching for ELLIS$DKA200:[BAE]X.LIS;
```

```
-RMS-E-FNF, file not found
```

```
$
```

- The file type is assumed to be .LIS. This assumption is made by the TYPE command.
- The semicolon, with no trailing version number, indicates the most recent version of the file.



Note

The error message indicates that TYPE is attempting to locate the file on the disk ELLIS\$DKA200: and in the directory [BAE].

These file specification components come from the default directory string, displayed by \$SHOW DEFAULT, as seen below:

```
$ show default
```

```
    ELLIS$DKA200:[BAE]
```

```
$
```

Qualifiers

There are three types of qualifiers:

- **Command qualifiers**

- Like adverbs, they modify the verb.
- Regardless of its position, the qualifier has the same effect.
\$ PRINT/HOLD FILE1.LIS
\$ PRINT FILE1.LIS/HOLD
\$ DELETE/CONFIRM TEST.TMP

- **Positional qualifiers**

- Modify the parameter before the qualifier.
- Have different meanings depending on where they appear on the command line.
\$ PRINT/COPIES=3 FILE1.DAT, FILE2.DAT
\$ PRINT TEST.DAT/COPIES=2, MUMBLE.DAT
\$ PRINT/COPIES=2 ALPHA.DAT, BETA.DAT/COPIES=3, AA.DAT

- **Parameter qualifiers**

Can only be used with certain types of parameters /SAVE_SET in the BACKUP command.

HELP

The HELP command invokes the OpenVMS HELP facility to display information about an OpenVMS command or topic. In response to the “Topic?” prompt, you can:

- Type the name of the command or topic for help.
- Type INSTRUCTIONS for more detailed instructions on how to use Help.
- Type HINTS if you are not sure of the name of the command or topic for which you need help.
- Type a question mark (?) to redisplay the most recently requested text.
- Press the Return key one or more times to exit from Help.
- Abbreviate any topic name, although ambiguous abbreviations result in all matches being displayed.

Getting Started with HELP

\$ HELP HINTS

HINTS

Type the name of one of the categories listed below to obtain a list of related commands and topics. To obtain detailed information on a topic, press the RETURN key until you reach the "Topic?" prompt and then type the name of the topic.

Topics that appear in all upper case are DCL commands.

Additional information available:

Batch_and_print_jobs	Command_procedures	Contacting_people
Creating_processes	Developing_programs	
Executing_programs	Files_and_directories	Logical_names

HINTS Subtopic? CONTACTING_PEOPLE

Contacting_people

Communicating with other people using the system.

MAIL	Sends/reads messages to/from other users.
PHONE	Permits users to communicate by typing messages to one another's terminal screens.
REPLY	Displays a message on one or more terminal screens.
REQUEST	Displays a message on the operator's console.

SHOW USERS Lists the interactive users on the system.

Type ? to display the list of HINTS.

Type CONTACTING_PEOPLE to redisplay this particular hint.

Press the RETURN key before typing a command or topic name.

HINTS Subtopic?

Topic?

\$

Summary of Basic DCL Commands

Command	Description
\$SHOW PROCESS	Views process parameters
\$SHOW USERS	Views who is logged on
\$SHOW SYSTEM	Views information on all processes
\$SHOW TERMINAL	Views terminal characteristics
\$SET TERMINAL/WIDTH=132	Sets terminal to largest width (default width is 80)
\$SET TERMINAL/INQUIRE	Asks terminal for type and sets characteristics appropriately
\$SET TERMINAL/NOBROADCAST	Shuts off messages from phone, mail, etc.
\$CREATE FILE.DAT	Creates a file
This is the contents of a test file. <Ctrl/Z>	Puts an end of file marker in the file and closes the file
\$TYPE FILE.DAT	Views contents of file on the terminal screen
\$PRINT FILE.DAT	Requests printing of the file
\$COPY DISK\$A:[OBRIEN.NEXT]FILE.DAT DISK\$B:[DOBIE.GILLIS]FILE.DAT	Makes a copy of a file
\$DELETE FILE.DAT;1	Deletes the file
\$PURGE	Deletes all but the highest version of all files in your directory
\$PURGE/KEEP=3	Keeps the highest three versions of all files in your directory
\$RENAME FILE.DAT HISTORY.DAT	Renames a file. FILE.DAT is now called HISTORY.DAT and no new file is created
\$RUN PGMA.EXE	Executes a program
\$SPAWN	Creates a subprocess

SPAWN Command

The SPAWN command creates a subprocess, running in a DCL environment.

SPAWN Scenario

```
$RUN MAJOR_LEAGUE_PROGRAM.EXE
```

```
.  
.   
.   
.    !oops! forgot to prepare a file that will  
.          be opened at the end of the program.  
                                <Ctrl/Y>
```

```
$SPAWN    !creates a subprocess.
```

```
Process OBRIEN_1 Spawned.
```

```
$RUN MINI_UPDATE.EXE    !do the stuff you forgot.
```

```
.  
.   
.   

```

```
$LOGOUT !DON'T PANIC
```

```
! You are logging out of the subprocess.
```

```
$CONTINUE    !continues the running of the  
    ! MAJOR_LEAGUE_PROGRAM.
```

Command Line Editing

Command line editing allows you to correct, change, and reexecute a previously entered DCL command.

- **Moving the cursor**

<Ctrl/H> moves the cursor to the left end of the command line.

<Ctrl/E> moves the cursor to the right end of the command line.

<LEFT ARROW> or <Ctrl/D> moves the cursor one character to the left.

<RIGHT ARROW> or <Ctrl/F> moves the cursor one character to the right.

- **Deleting data**

<DELETE KEY> or <BACKSPACE> eliminates the character at left of cursor.

<LINE FEED> eliminates the word at the left of the cursor.

<Ctrl/U> erases the line to the left of the cursor.

<Ctrl/X> discards the current command line regardless of the cursor position.

- **Adding data**

<Ctrl/A> toggles between overstrike and insert modes (same as \$ SET TERMINAL/INSERT and \$ SET TERMINAL/OVERSTRIKE)

- **Recalling command lines**

<Ctrl/B> recalls the most recently entered command line.

<UP ARROW> can recall up to the last 20 (VAX) or 254 (Alpha) commands.

<DOWN ARROW> recalls in the opposite direction than the up arrow.

\$ RECALL/ALL displays a numbered list of the last 20 (VAX) or 254 (Alpha) command lines issued.

\$ RECALL 4 recalls the fourth most recently entered command line.

\$ RECALL COPY recalls the most recent copy command issued.

\$ RECALL/ERASE empties the contents of the RECALL buffer.

Control Keys

- **Terminating or interrupting an operation**

<Ctrl/Y> interrupts the execution of the current image and returns to the \$ prompt

<Ctrl/C> returns to the utility prompt in some utilities; otherwise, it is the same as <Ctrl/Y>.

<Ctrl/Z> exits from some utilities (\$MAIL, \$PHONE). Places an end-of-file marker in current position in file operations (\$CREATE).

<Ctrl/T> displays status and timings of the running image. It must be enabled with \$SET CONTROL=T.

- **Controlling screen display**

<Ctrl/R> retypes the current line. It is used mostly at hard-copy terminals.

<Ctrl/W> refreshes the screen. It is useful for clearing off messages that might be sent to you during an editing session.

<Ctrl/S> suspends output and execution of an image.

<Ctrl/Q> restarts output and image execution.

<Ctrl/O> suspends output but image continues executing. Another <Ctrl/O> restarts output.

<Ctrl/L> page feed on hard-copy terminals. It clears your portion of screens in Phone utility.

Terminal Characteristics

- To view terminal settings:

```
$ SHOW TERMINAL
```

```
Terminal: _TWA2:      Device_Type: VT300_Series  Owner: OBRIEN_TWA2
Username: OBRIEN
Input:   9600      LFfill:  0      Width:  80      Parity: None
Output: 9600      CRfill:  0      Page:   24
Terminal Characteristics:
Interactive      Echo              Type_ahead      No Escape
Hostsync        TTsync            Lowercase       Tab
Wrap            Scope             No Remote       Eightbit
Broadcast       No Readsycn       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 No Disconnect     No Psthru       No Syspassword
SIXEL Graphics No Soft Characters Printer port     Numeric Keypad
ANSI_CRT        Regis             No Block_mode   Advanced_video
Edit_mode       DEC_CRT           DEC_CRT2        DEC_CRT3
$
```

- To change terminal settings:

```
$SET TERMINAL
```

```
$SET TERMINAL/INQUIRE
```

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

/PAGE Qualifier

Most commands that display output include a /PAGE qualifier to display output one screen at a time. The /PAGE=SAVE qualifier value gives a UNIX command-like feel.

- Example:

```
$ TYPE/PAGE=SCROLL filename.dat      ! Displays one page
                                         ! at a time.
```

```
$ TYPE/PAGE=CLEAR_SCREEN filename.dat (default)
```

- \$ TYPE/PAGE=SAVE allows you to save up to five screens and scroll forward and back using Prev/Next screen keys, similar to UNIX **pg** command. Also, supports searches, using the <find> key, as well as other options.
- \$ TYPE/TAIL=*n* allows you to examine last *n* lines, similar to the UNIX **tail** command. This option is not available on all display commands.
- \$ TYPE[/HIGHLIGHT][/EXACT] /SEARCH=*string* may be used to highlight search strings in \$TYPE display.

Error Messages

Sample Error Message

\$cipy !You meant to type COPY.

%DCL-W-IWVERB,unrecognized command verb, check validity and spelling /CIPY/

- System Message Delimiter (%)
Indicates some sort of message (similar to the way the \$ prompt indicates DCL)
- Facility Code (DCL)
Indicates which facility within the operating system is reporting the error
- Severity Code (W)
Indicates how serious the error is (see next page)
- Message Identification Code (IWVERB)
 - Indicates the category of problem.
 - You can look up this mnemonic in the *System Messages and Error Recovery Procedures Manual* of the OpenVMS documentation set.
 - May also be used with \$ HELP/MESSAGE
- Message Text (“unrecognized command verb...”)
Readable text describing the probable cause of the error.
- DCL Being Helpful (/CIPY/)
The failing portion of the command line.

Severity Levels

Each error message has a corresponding status value. A portion of the status value (the low three bits) identifies the severity level of the error status.

Severity	Symbol	Value	Meaning	Example
Warning	W	0	Operation partly successful	R TYPE nonexistent file
Success	S	1	Everything went ok, usually not displayed	\$ SPAWN
Error	E	2	Operation probably failed	\$ PURGE TEMP.DAT;3
Informational	I	3	Notification of an event	\$ COPY/LOG
Fatal	F	4	Operation failed	\$ ANALYZE/CRASH with no access to dump file

HELP Message Utility

OpenVMS provides a utility that allows you to access online descriptions of system messages.

- The HELP/MESSAGE command operates through a DCL interface that accesses message descriptions in a text file.
- This text file is derived from the latest version of the OpenVMS system messages documentation.
- By default, HELP/MESSAGE provides information about how the last executed command has completed.

HELP/MESSAGE

- HELP/MESSAGE also allows you to access message descriptions by specifying a message identifier, words from the message text or a status code.

HELP/MESSAGE *message-id*

HELP/MESSAGE ACCVIO

HELP/MESSAGE Example

```
$
$ type junk
%TYPE-W-SEARCHFAIL, error searching for _DKA300:[ELLIS]JUNK.LIS;
-RMS-E-FNF, file not found
$ help/message
```

SEARCHFAIL, error searching for 'file-spec'

Facility: Shared by several facilities

Explanation: The specified file could not be found.

User Action: Take action based on the accompanying message.

```
$ help/message fnf
```

FNF, file not found

Facility: SYSGEN, System Generation Utility

Explanation: SYSGEN is unable to locate the driver image file. The default directory and disk device for driver images is SYS\$SYSTEM. This message is also produced during an AUTOCONFIGURE operation if a UNIBUS device does not have a corresponding driver in SYS\$SYSTEM.

User Action: Restore the image file to the system disk from a backup disk.

FNF, file not found

Facility: RMS, OpenVMS Record Management Services

Explanation: The specified file does not exist.

User Action: Check the file specification and verify that the device, directory, file name, and file type are all specified correctly. If a logical name is specified, verify the current equivalence assigned to the logical name. If the equivalence is correct, verify that the correct volume is mounted on the specified device and that the file was not inadvertently deleted.

```
$
```

DCL Commands by Function

- **Getting Started**
 - LOGIN PROCEDURE
 - LOGOUT
 - SET PASSWORD
- **General Information**
 - HELP
 - SHOW CLUSTER
 - SHOW DEVICES
 - SHOW NETWORK
 - SHOW PROCESS
 - SHOW SYSTEM
 - SHOW TERMINAL
 - SHOW TIME
- **Directories**
 - CREATE/DIRECTORY
 - DELETE
 - DIRECTORY
 - PURGE
- **File Protection**
 - COPY/PROTECTION
 - CREATE/DIRECTORY/PROT
 - DIRECTORY/PROTECTION
 - EDIT/ACL
 - SET ACL
 - SET FILE/ACL
 - SET PROTECTION
 - SET PROTECTION/DEFAULT
 - SHOW ACL
 - SHOW PROTECTION
- **File Specifications**
 - SET DEFAULT
 - SHOW DEFAULT

- **Compile**
 - BASIC
 - BLISS
 - COBOL
 - CC
 - FORTRAN
 - MACRO
 - PASCAL
- **File Manipulation**
 - APPEND
 - COPY
 - CREATE
 - DELETE
 - DIFFERENCES
 - PRINT
 - RENAME
 - SORT
 - TYPE
 - UNLOCK
- **Symbols**
 - =, ==
 - DELETE/SYMBOL
 - LEXICAL FUNCTIONS
 - SHOW SYMBOL
- **Command Procedures**
 - @ (COMMAND PROCEDURE)
 - CLOSE
 - EXIT
 - GOTO
 - IF...THEN
 - INQUIRE
 - OPEN
 - READ
 - STOP
 - WAIT
 - WRITE

- **Error Handling**
 - ON CONTROL_Y
 - ON...THEN
 - SET [NO]CONTROL_Y
 - SET [NO]ON
 - SET [NO]VERIFY
- **Logical Names**
 - ASSIGN
 - CREATE/NAME_TABLE
 - DEASSIGN
 - DEFINE
 - MOUNT
 - SHOW LOGICAL
 - SHOW TRANSLATION
- **Network**
 - SET HOST
 - SHOW NET
- **Communication**
 - MAIL
 - PHONE
 - REPLY
 - REQUEST
- **Process**
 - DEFINE/KEY
 - DELETE/KEY
 - RECALL
 - SET BROADCAST
 - SET KEY
 - SET PROCESS
 - SET PROCESS/PRIORITY
 - SET PROMPT
 - SHOW BROADCAST
 - SHOW KEY
 - SHOW PROCESS

- **Job Control**
 - DELETE/ENTRY
 - PRINT
 - SET QUEUE/ENTRY
 - SHOW PRINTER
 - SHOW QUEUE
 - STOP/ABORT
 - STOP/ENTRY
 - STOP/REQUEUE
 - SUBMIT
- **Devices**
 - DEALLOCATE
 - DISMOUNT
 - ALLOCATE
 - INITIALIZE
 - MOUNT
 - SET TERMINAL
 - SHOW DEVICES
- **Link**
 - LIBRARY
 - LINK

Learning Check

1. What are the three parts of a DCL command?

.....

.....

.....

2. What are the three types of DCL command?

.....

.....

.....

3. DCL begins parsing a command by determining the _____ that has been entered.

4. Describe the parts of the following file specification:

① ② ③ ④ ⑤

ELLIS\$DKA200:[HOME]FOO.LIS;2

①

②

③

④

⑤

Objectives

To produce data files and programs in an effective fashion you need to use some form of text editor. This module introduces three standard editors, and describes their features and differences. The EVE editor is the default editor for OpenVMS. The EDT editor is available on all OpenVMS systems. The CDE editor is available on Common Desktop workstation environments.

After completing this module, you will be able to:

- List the three basic editors available with OpenVMS.
- List at least 10 basic EVE commands.
- Identify and use the three EDT modes.
- Use the EVE and EDT Help utilities.
- Use the EDT keypad keys.

Topics

- EDT Overview
- EVE Overview
- EVE
 - Basic Commands
 - Keypad
 - Help
 - Recovery
 - Defining Keys
- EDT
 - Help
 - Line Mode
 - Keypad Mode
 - Recovery
 - Initialization

Editing Options

Three basic editors are available with OpenVMS.

- **EVE**

EVE is the current default editor. It is extensible and can easily define keys to handle commands.

- **EDT**

EDT is an older editor with a popular keypad interface. It is the only one of these three editors that supports a line mode for hard copy editing.

- **CDE Text Editor**

A mouse-driven PC-style editor, the CDE text editor is only available under the Common Desktop Interface (CDE). It is also available in the same format with Tru64 UNIX. This editor is fairly straightforward if you have used a PC and, therefore, will not be discussed further in this course.

EVE

Overview

- Extensible OpenVMS editor
- Used to create or modify files
- Supports multiple windows

The size and number of windows depend on the type of terminal you are using.
- Insert and overstrike mode
- Less system load than EDT
- Easy to learn
- Automatic word wrap
- Customizable using the TPU programming language
- Allows you to work on multiple files simultaneously

The EVE Editor

To use the EVE editor, enter:

```
$ EDIT/TPU file.dat
```

EVE Screen Layout

```
$ ed my_file.dat
[End of file]
```

Buffer: MY_FILE.DAT | Write | Insert | Forward

```
Editing new file. Could not find: ELLIS$DKA200:[BAE]MY_FILE.DAT;
```

EVE Help

Line Mode Help

- Press the DO key (VT200), <PF4> (VT100), or the KP - (keypad minus on a PC keyboard) to get the **COMMAND:** prompt.
- Type **HELP**.
A list of topics will be displayed from which you can receive help.
- Press the Return key to exit from Help.

Screen Mode Help

- Press <PF2> (VT100), <HELP> key (VT200), or the KP / (keypad slash on a PC) while in keypad mode.
- Press the key with which you need help.
- Press the Return key to exit.

Predefined Keypads

To use a predefined keypad, press the appropriate DO key and enter one of the following commands:

```
SET KEYPAD VT100
SET KEYPAD EDT
SET KEYPAD WPS
SET KEYPAD NUMERIC
```

Basic EVE Commands

Command	Function
TOP	Moves cursor to beginning of current buffer
BOTTOM	Moves cursor to end of current buffer
ERASE LINE	Deletes current line, from current cursor position
ERASE WORD	Deletes current word, from current cursor position
SELECT	Selects text for copy or cut
COPY	Copies selected text into buffer
CUT	Same as COPY, except deletes selected text
PASTE	Pastes buffer from cursor position
END OF LINE	Moves cursor to end of line, same as <Ctrl/E>
START OF LINE	Moves cursor to start of line, same as <Ctrl/H>
BOX SELECT	Selects a "box" of text
BOX CUT/COPY	Cuts/copies a box of text to buffer
BOX PASTE	Pastes a box of text
FIND	Searches for text
FIND NEXT	Searches for next occurrence of text
REPLACE	Replaces text, prompts to change and continues
GLOBAL REPLACE	Globally replaces text without prompting
SET GOLD	Defines gold key
DEFINE KEY	Defines command as key/key sequence
LEARN	"Learns" keystrokes and maps to key sequence
SAVE EXTENDED EVE <i>file</i>	Saves key definitions in <i>file</i> . tpu\$section section file; may use with \$ EDIT/TPU/SECTION= <i>file</i> or by defining TPU\$SECTION logical name
DCL	Executes a DCL command, and brings the output into a special buffer called DCL
SPAWN	Creates a subprocess
EXIT	Writes and exits
QUIT	Exits without saving changes
SAVE FILE AS <i>f</i>	Writes buffer to alternate file name
WRITE	Writes the contents of the current buffer out to a file
CENTER	Centers text on a line

Sample EVE VT100 Keypad

The following display is made after:

1. Entering the SET KEYPAD VT100 command.
2. Pressing the key corresponding to PF2 (KP / on a PC).

To get help on	HELP	Do					
commands, type							
a command name	_____	_____		_____	_____	_____	_____
or ? and press	_____	_____		_____	_____	_____	_____
RETURN.	Find	Insert	Remove	Find	HELP	Change	Do
		here				directi	
To list all key	_____	_____	_____	_____	_____	_____	_____
definitions,	Select	Previou	Next	Select	Remove	Insert	Move
type Keys and		screen	screen			here	by line
press RETURN,	_____	_____	_____	_____	_____	_____	_____
or press GOLD-		Move up			Move up		Erase
HELP.							word
	_____	_____	_____	_____	_____	_____	_____
To show a key	Move	Move	Move	Move	Move	Move	
definition, use	left	down	right	left	down	right	Change
the SHOW KEY	_____	_____	_____	_____	_____	_____	mode
command.					Next screen	Previou	
						screen	

Buffer: HELP

Press the key that you want help on (RETURN to exit help)

Sample EVE EDT Keypad

The following display is made after:

1. Entering the SET KEYPAD EDT command
2. Pressing the key corresponding to PF2 (KP / on a PC)

GOLD key functions are shown in reverse.

To get help on	HELP Do	
commands, type	KeyDefs	
a command name		
or ? and press		
RETURN.	Find Ins Her Remove	Gold HELP FndNxt Del L
	Wil Fin Restore Sto Tex	key KeyDefs Find Res Lin
To list all key		
definitions,	Select Pre Scr Nex Scr	MovByPa Sect Append Del W
type Keys and	Reset Pre Win Nex Win	Do Fill EDT Rep Res wor
press RETURN,		
or press GOLD-	Move up	Forward Reverse Remove Del C
HELP.	Top	Bottom Top Ins Her Res Cha
To show a key	Mov Lef Mov Dow Mov Rig	Word EOL Char
definition, use	StaOfLi Bottom EndOfLi	ChngCas Del EOL SpecIns Return
the SHOW KEY		Subs
command.		EDT Line Select
	Use the DO key to enter	Open Line Reset
	advanced commands	

Buffer: HELP

Press the key that you want help on (RETURN to exit help):

Startup Keypads

Keypads can be enabled automatically when you start EVE using the logical name `EVE$KEYPAD`. The logical name should be set at DCL level prior to activating EVE. This activity is normally performed in your login command procedure (`LOGIN.COM`).

To automatically enable the VT100 keypad, enter:

```
$ DEFINE EVE$KEYPAD VT100
```

To automatically enable the EDT keypad, enter:

```
$ DEFINE EVE$KEYPAD EDT
```

Exiting EVE Editor

End the EVE editing session in any of the following ways:

- Press `Ctrl/Z` to exit the editing session and save the modifications.
- Press the DO key (VT200) or the `<PF4>` key (VT100) to get the command: prompt.
- Enter `QUIT` to end the session, which does **not save** modifications.
- Enter `EXIT` to end the session and **save** modifications.

EVE File Recovery

■ **Journaling**

When an editing session is exited abnormally (Ctrl/Y or system interruption), you can recover your modifications.

■ **Journal file**

- Contains the keystrokes and editing commands of your current editing session.
- File name is same as input file name.
- File type is `.TPU$JOURNAL`.
- It is created at beginning of editing session.
- It is deleted automatically at the end of a session that is ended normally.
`$ EDIT/TPU WORK.COM` creates `WORK.TPU$JOURNAL`
- If the edit session is exited abnormally, it leaves `WORK.TPU$JOURNAL`.
- To recover changes, enter:
`$ EDIT/TPU WORK.COM/RECOVER`

EDT

Overview

- Used to create or modify files
- Has more system overhead than EVE
- Can work on multiple files simultaneously
- Three modes are available:
 - Line mode
 - Keypad mode
 - Nokeypad mode

Line Mode

- Works with individual lines in file
- Used for hard copy terminals
- Has asterisk prompt (*)
- Default mode for EDT editor

Keypad Mode

- Allows you to work with the whole file on the screen
- Requires a video terminal
- To start keypad mode, enter the CHANGE command at line mode prompt (*)
- See EDT keypad layout described under EVE

Nokeypad Mode

Most commonly used for DEFINE KEY commands

The EDT Editor

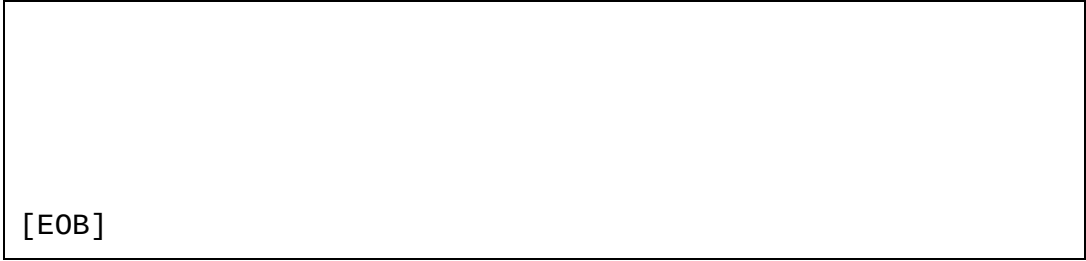
To use the EDT editor, enter:

```
$ ed/edt my_file.dat
```

```
1      7spawn
```

```
*
```

EDT Keypad Mode Screen Layout



[EOB]

EDT Help

To get line mode help, enter HELP at the line mode prompt (*).

```
$ EDIT file.dat  
1      This is the first line of the file  
*HELP
```

You can get help on a topic by typing:

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

A topic can have one of the following forms:

- An alphanumeric string (for example, a command name, option, and so forth)
- 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.

Screen Mode Help

- Press <PF2> or <HELP> key while in keypad mode.
- Press the key with which you need help.
- To exit, press the space bar.

EDT Line Mode Commands

Command	Function
CHANGE	Changes from line mode to keypad mode
COPY	Duplicates line(s) of text
DELETE	Removes line(s) of text
EXIT	Saves all changes to the file, and ends the editing session
HELP	Invokes EDT line mode help
INSERT	Inserts text into the buffer
MOVE	Moves line(s) of text from one location to another
QUIT	Does not save changes to the file, and ends the editing session
REPLACE	Deletes line(s) of text and allows user to add new text
RESEQUENCE	Assigns new line numbers in increments of 1
SUBSTITUTE	Replaces one line of text with another

EDT Keypad Keys

- Use the keypad keys to manipulate data
 - Advance or Backup (change direction of the cursor)
- Move the cursor
 - One character at a time
 - One word at a time
 - One line at a time
 - To the end of the line
 - Or using the arrow keys
- Delete
 - The character the cursor is on
 - Up to the next word
 - To the end of the line
 - The character before the cursor
- Search for a string
 - In either forward or reverse direction
 - EDT will show you each occurrence of the string
- Cut and paste text
 - From one location to another within the file
 - From one file to another (using buffers)

Exiting the EDT Editor

To end the EDT editing session:

- From screen mode, press `Ctrl/Z` to go to line mode
- From line mode (the asterisk prompt):
 - Enter `EXIT` to save the changes in the file.
 - Enter `QUIT` and the changes are not saved.
- An alternate way to enter the line mode commands is:
 - From screen mode press `<PF1>` then keypad key 7.
 - At the EDT command prompt, enter either `EXIT` or `QUIT` (`PF1 7` allows you to enter a line mode command while still in screen mode).

EDT File Recovery

■ Journaling

When an editing session is abnormally exited (**Ctrl/Y** or system interruption) you can recover your modifications.

■ Journal file

- A journal file contains the keystrokes and editing commands of your current editing session.
- File name is same as input file name.
- File type is `.jou`.
- It is created at beginning of editing session.
- It is deleted automatically at the end of a session that is ended normally.

■ Example

\$ EDIT TEST.COM creates TEST.JOU

- If this edit was abnormally exited, it would leave **TEST.JOU**.
- To recover changes, enter:
\$ EDIT TEST.COM/RECOVER

EDT Initialization File (EDTINI.EDT)

- Special file used to set up your EDT editing environment
- EDT editor looks for this file in the directory in which you are editing, and executes it.
- To make the EDT editor look for this file in a different directory, enter:
`$ EDIT/COMMAND=[otherdirectory]EDTINI.EDT file.dat`
EDTINI.EDT contains line mode EDT editor commands
- Example of an EDTINI.EDT file:
SET QUIET
SET WRAP 60
SET TAB 5
DEFINE KEY GOLD F AS "EXT EXIT."
DEFINE KEY GOLD Q AS "EXT QUIT."
SET MODE CHANGE

Learning Check

1. Match the following editors with the descriptions provided.
 - a. EVE Easiest to learn
 - b. EDT Default editor
 - c. CDE Keypad interface
..... Supports line-mode editing
..... Extensible and customizable
..... Popular, older editor
..... Mouse-driven
..... Also available with Tru64 UNIX

2. You can recover lost edits to an abnormally exited file using:
.....
.....

3. EVE is customizable using the:
.....
.....

4. Customize the EDT environment with:
.....
.....

Objectives

The OpenVMS Mail utility allows you to send messages to users on your system or on any other computer that is connected to your system. It allows you to read, file, forward, delete, print, and reply to messages that other users send you.

The OpenVMS Phone utility allows you to “talk” to other users on your system or on any other OpenVMS system in your network. The utility simulates the functions and features of a telephone.

This module details the use of each utility. After completing this module, you will be able to:

- Use the Mail utility to read, send, and delete a message.
- Get online help from the Mail utility.
- Use the Phone utility to send an interactive message.

Topics

- Mail Utility
- Phone Utility

Mail Utility

The Mail utility allows you to send and receive messages from other users on local or remote nodes whether or not they are logged in.

To invoke Mail, enter:

```
$ Mail
```

```
MAIL>
```

Mail allows you to:

- Read messages
- Send messages
- Delete messages
- Get online help
- File messages

Mail stores messages in file named MAIL.MAI located in your default/mail directory at login. Large messages are stored in files named MAIL\$xxxxxx.MAI.

To set a Mail subdirectory and move .MAI files to it, enter:

```
MAIL > Set mail_directory
```

Sending Messages

- To send a message, enter:
\$ mail

```
MAIL> send
T0: fred
SUBJ: secret message
```

```
ENTER YOUR MESSAGE BELOW.  PRESS CTRL/Z
WHEN COMPLETED OR CTRL/C TO QUIT:
```

```
.
.
.
text of message
```

```
.
.
.

ctrl/z
ctrl/c
```

- To send a message to another node, enter:
\$ mail

```
MAIL> send
T0: nodex::system
$ mail <ret>
```

```
MAIL> send          ! Across internet
T0: "'bellis@tesser.com'"
```

Using an Editor in Mail

- To use Mail and invoke a text editor, enter:
`$ mail`

`MAIL> send/edit`
 - By default, Mail invokes the EVE editor.
 - You are now in a standard edit session.
- To send the message, enter:
`Ctrl/Z`

`or`
`EXIT`
- To cancel the send operation, enter:
`QUIT`

Sending a File

- To send a file, enter:

```
$ mail
```

```
MAIL> send memo.txt
```

```
TO: obrien
```

```
SUBJECT: secret memo
```

or

```
$ mail/subject="secretmemo" memo.txt obrien
```

- To send to a distribution list:
 - Create a file containing user names (distribution list).
 - Send the message to the names in the file that contains the user names.

For example:

```
!My distribution list file
```

```
!Mydis.dis
```

```
WILLIAMS
```

```
HAMEED
```

```
SMITH
```

```
EVMS::ELLIS
```

```
$ mail
```

```
MAIL> send
```

```
TO: @mydis.dis
```

```
SUBJECT: meeting dates
```

Reading Messages

By default, Mail creates folders that hold new, old, and discarded messages.

- New messages are automatically placed in a folder called NEWMAIL.
- Old messages are held in a folder called MAIL.
- Deleted messages are stored in a folder called WASTEBASKET. The wastebasket is emptied when you exit the utility.

Reading New Messages

- You are notified when you have new mail.

```
$ mail
```

```
You have 1 new message
```

```
MAIL>
```

- Your current folder is automatically set to NEWMAIL.
- To read, simply press <RETURN>

```
#1      24-AUG-1989  21:15:30  NEWMAIL
```

```
FROM: ELLIS
```

```
TO: OBRIEN
```

```
SUBJ: COMPANY PICNIC
```

```
.
```

```
text of message
```

```
.
```

```
.
```

Reading Old Messages

After you have read a new message, it is automatically moved to the **MAIL** folder. To look at these messages, enter:

```
$ mail
```

```
MAIL> select mail
```

```
MAIL> directory
```

```
#    FROM      DATE    SUBJECT      MAIL
```

```
MAIL> read n
```

Deleting Mail

```
$ mail
```

```
MAIL> delete 4-6,11,15,22-35
```

Summary of Mail Commands

MAIL> COPY foldername [filename]

Copies a message to another folder. If the specified folder does not exist, Mail creates it.

MAIL> DIRECTORY

/FOLDER

Lists all folders in mail file.

/SUBJECT_SUBSTRING=text

Selects messages containing “text” in subject field of message.

MAIL> EXTRACT filespec

Places a copy of the current message in a sequential file.

MAIL> FILE foldername

Moves the current message to the specified folder.

MAIL> HELP

Enables you to obtain information about mail.

MAIL> PRINT

Prints current message.

/QUEUE=queue name

/NOW

Phone Utility

The Phone utility is an interactive communication facility that allows you to call another user who is currently logged on.

To invoke the utility:

```
$ PHONE [username]
```

Phone Screen

```
$ phone system
                                OpenVMS Phone Facility                26-MAR-1999
%
Ringing SYSTEM...              (Press any key to cancel call and continue.)
-----
                                ELLIS::BAE
-----
```

Phone Example

	OpenVMS Phone Facility	26-
	MAR-1999	
%		

	ELLIS: :BAE	
Can you help me?		

	ELLIS: :SYSTEM	
Sure, after you have checked the manual.		

Phone Command Summary

Command	Function
%HELP	Information about using this utility is displayed.
%	Entering the % sign allows you to enter commands.
%ANSWER (\$PHONE ANSWER)	Answers the phone when you receive a call.
%DIRECTORY or %DIRECTORY node::	Lists users you may call.
%HANGUP	Hangs up phone but still remain in utility.
%REJECT	Rejects a call from another user. The other user is informed that you have rejected them.
%EXIT (Ctrl/Z)	Exits the utility.
MOVE	Moves lines of text from one location to another.

Learning Check

1. The Mail utility allows you to send and receive messages from:
 - a. Users logged in to a local node.
 - b. Users logged in to local or remote nodes.
 - c. Users on local or remote nodes whether they are logged in or not.
 - d. All of the above.
2. Name the folders where Mail stores messages:
New messages:
Old messages:.....
Deleted messages:.....
3. In what important way is the Phone utility different from the Mail utility?
.....
.....

Objectives

This module introduces the directory structure on the OpenVMS system. It explains how to organize your files in subdirectories to facilitate easy access and retrieval.

After completing this module, you will be able to:

- Create subdirectories.
- Set default to a directory.
- Move files to a subdirectory.
- Delete subdirectories.

Topics

- File Specifications
- Directory Organization
- Creating Directories
- Deleting Directories
- Wildcards
- File Protection
- ACL Protection

Complete File Specification

`NODE::DEVICE:[DIRECTORY]FILENAME.TYPE;VERSION`

`BRUDEN::DUA2:[ELLIS]PROGRAM3.C;4`

NODE

The system name (1–6 characters). A networked environment has many systems.

DEVICE

The device where your files can be found (4–15 characters).

DIRECTORY

A designation for a subset of your file space, main level, or subdirectory (1–39 characters).

Or, a designation for a specific portion of the device.

FILENAME

The name of a specific file (0–39 characters; ODS-5 allows 236 characters for file names, including name and type).

TYPE

An extension indicating the type of file (0–39 characters; ODS-5 allows 236 characters for file names, including name and type).

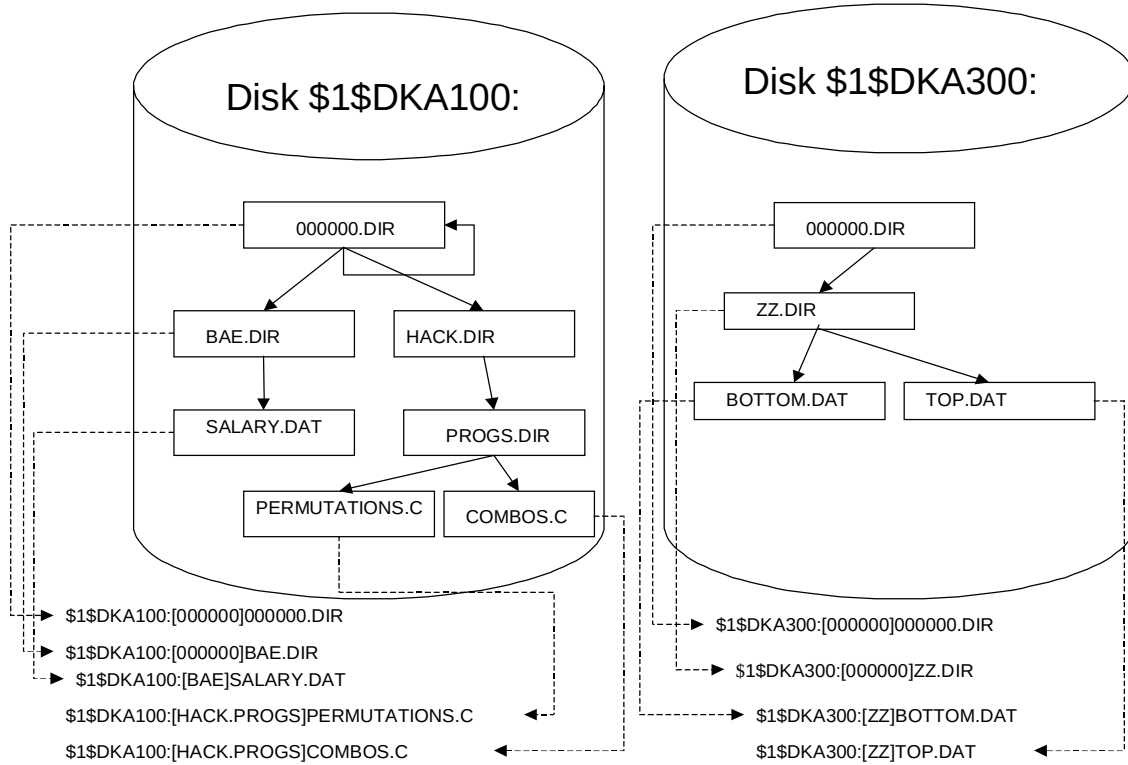
VERSION

A number used to differentiate files with the same name and type (1–32767).

Directories

- **Conceptually**
A file cabinet with drawers and file folders
- **Actually**
 - A file with a .DIR extension and version number of 1
 - File header has a bit set, marking the file as a directory
 - Created contiguously, must remain contiguous
 - Contains names of your files and information with which to locate your files
- **Directories**
Impose an organization on your files, yielding a level of security for your files and a potential decrease in access time
- **Subdirectories**
 - Your files can be further organized into subdirectories for even finer organization.
 - Can catalog other subdirectories as well as regular files.
 - Can have a maximum of 255 levels in the hierarchy.
 - Files are typically grouped by type or function.
- **Levels**
 - Master File Directory (MFD)
 - ◆ Catalogs UFDs
 - ◆ Beginning of file lookup path
 - User File Directory (UFD)
Where users' files are
 - Sub File Directory (SFD)
Any directory cataloged in a directory other than the **MFD**

Directory Organization and File Specifications



Creating Directories

To create a subdirectory under [OBRIEN], enter:

```
$SET DEFAULT [OBRIEN]
```

Make sure your default directory is at the UFD.

```
$CREATE/DIRECTORY [OBRIEN.BILLS]
```

This command creates a file called `BILLS.DIR` in [OBRIEN].

`BILLS.DIR` tracks the files placed in this subdirectory.

```
$SET DEFAULT [OBRIEN.BILLS]
```

Determine if there is anything in it.

```
$DIRECTORY
```

The system responds with `-- NO FILES FOUND --`

```
$CREATE TEST.DAT
```

This command creates a file in the directory (could use an editor also).

```
$CREATE/DIR [OBRIEN.BILLS.ELECTRIC]
```

This command creates a subdirectory under `BILLS` to be used for a tracking of the electric bill.

Directory Command

- Lists all or some of the files in a directory.
- Has many qualifiers with which to customize your request.

Qualifiers

`$DIRECTORY/DATE`

Tells when the file was created

`/PAGE=SAVE`

Causes command output to be displayed one page at a time

`/SEARCH="string"`

Highlights lines containing matching strings (must be used with `/PAGE` qualifier)

`/SIZE`

Includes size (blocks) with file names

`/PROTECTION`

Includes protection information

`/FULL`

Gives full information about each file

`/COLUMNS=n`

Displays *n* columns on a brief display

Many other qualifiers are available.

Deleting Subdirectories

To delete a directory, the directory must be empty and delete protection must be set on it.

Follow these steps:

```
$SET DEF [OBRIEN.BILLS]
```

```
$DELETE *.*;*           !Wildcards discussed shortly.
```

```
$SET DEFAULT [OBRIEN]
```

```
$SET PROTECTION=0:RWED BILLS.DIR
```

```
$DELETE BILLS.DIR;
```

Wildcards

Wildcard characters are special characters that can be used to:

- Abbreviate a file specification.
- Specify multiple files, with common characters in their file names.
- Match a specific count of characters in file specifications.

General-Purpose Wildcards

- Asterisk (*) matches zero or more characters of any type.
- Percent (%) matches exactly one character of any type.

Examples

`PAYROLL.COB;*`

Matches all versions of `PAYROLL.COB`.

`*.*;1`

Matches all files with version number of 1.

`PAYROLL%.COB;*`

Matches all files named `PAYROLL` followed by exactly one more character, of type `.COB`.

`PAY*.*;*`

All files whose first three letters are `PAY`.

Directory Wildcard Characters

- Minus sign (-) indicates moving up the directory hierarchy by one level.
- Ellipsis (...) searches all subdirectories beneath the current.
- Empty brackets ([]) can be used to designate the current directory.

Examples

```
$ show default
  ELLIS$DKA200:[BAE]
$ set def [.vism]
$ show default
  ELLIS$DKA200:[BAE.VISM]
$ set default [-]
$ show default
  ELLIS$DKA200:[BAE]
$ set def [.a.b]
$ dir
Directory ELLIS$DKA200:[BAE.A.B]
```

```
C.DIR;1          STALL.COM;1
```

```
Total of 2 files.
$ set def [--.vism]
$ show default
  ELLIS$DKA200:[BAE.VISM]
$
```

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

Displays the contents of all subdirectories from the current default directory down to lowest level subdirectory.

File Protection

In a multi-user operating system like OpenVMS, many users have access to the same disks and potentially the same files. OpenVMS maintains file protections to:

- Prevent unauthorized copying or deletion of files
- Restrict access to your files in general
- Allow the deletion of directories

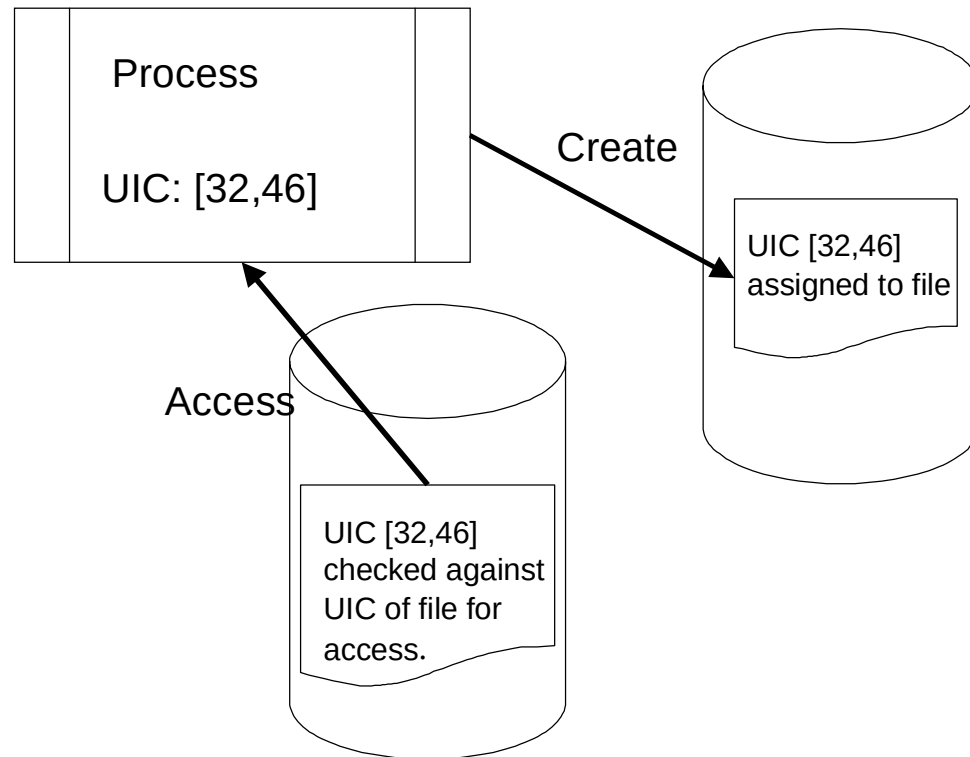
Protections are maintained on a variety of objects. Protection checking for files is generally hierarchical. Levels of protection include:

- Disk volume (volume or device protection)
- Directory (directory protection)
- File (file protection)

Protection can be:

- User Identification Code based (UIC)
- Access Control List based (ACL)

Processes, Objects, and Protections



When an object such as a file is created, the UIC of the process is assigned as owner of the object.

When an object is accessed, the process's UIC is compared to that of the object's owner to determine which protection category the process falls under.

Determining Your UIC

The UIC of a process is made up of an octal group number and an octal member number.

Numeric UICs are mapped to user identifiers through the Rightslist database. Because of this mapping, users seldom see their UICs. You can use the following command to determine the numeric UIC of your process or of another user.



Note

Do not worry about the details of the command. See Extending Command Procedures for more details.

Example

```
$ sh proc
```

```
27-MAR-1999 22:39:18.75  User: BAE          Process ID:  00000123
                          Node: ELLIS       Process name: "BAE"
```

```
Terminal:                TNA4: (Host: 192.206.126.197  Port: 1087)
```

```
User Identifier:         [BRUDEN,BAE]
```

```
Base priority:           4
```

```
Default file spec:      ELLIS$DKA200:[BAE.VISM]
```

```
Number of Kthreads: 1
```

```
Devices allocated:      ELLIS$TNA4:
```

```
$ write sys$output f$fao("!!U",f$identifier("BAE","NAME_TO_NUMBER"))❶
[42,1]
```

```
$ write sys$output f$fao("!!U",f$identifier("MJE","NAME_TO_NUMBER"))❶
[42,2]
```

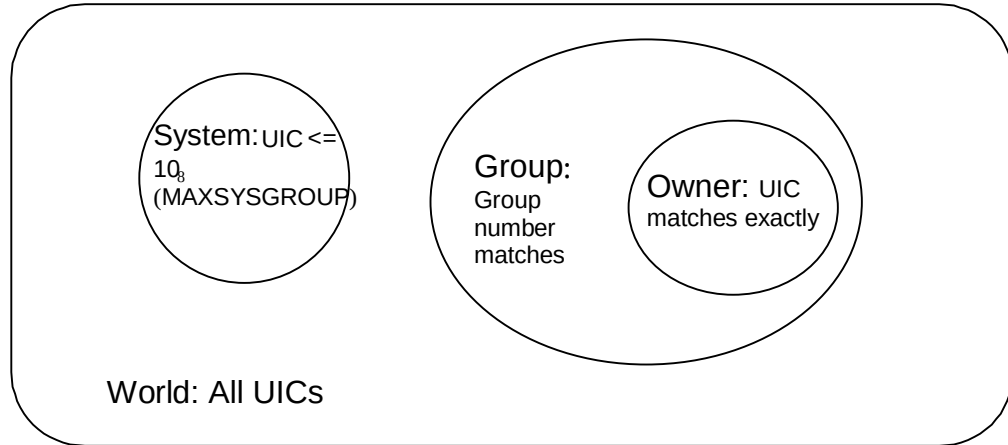
```
$ write sys$output f$fao("!!U",f$identifier("SYSTEM","NAME_TO_NUMBER"))❷
[1,4]
```

```
$
```

❶ Users BAE and MJE are in the same UIC group.

❷ User SYSTEM is in the SYSTEM category, since the group number is <10₈.

Protection Categories



Categories of Users

SYSTEM

Group UIC number must be less than or equal to 10 (octal) or SYSPRV privilege or same group and GRPPRV privilege.

OWNER

The process's UIC group and member numbers match those of the file owner's UIC.

GROUP

The requesting process's UIC group number must match the file owner's UIC group number.

WORLD

Everybody

Categories of File Access

OpenVMS supports four forms of access that may be allowed or denied in a given category. These categories include:

READ

Can TYPE, COPY, PRINT, BACKUP, etc. the file.

WRITE

Can APPEND, WRITE, etc. the file.

EXECUTE

Can RUN or @ the file.

DELETE

Can DELETE the file.

- WRITE access implies READ access.
- READ access implies EXECUTE.
- On a directory file, EXECUTE access with no READ access implies that a user can access a file by name, but may not use wildcard access to peruse the directory.

Examples

From MJE account:

```
$ dir [bae.vism]*.c
%DIRECT-E-OPENIN, error opening ELLIS$DKA200:[BAE.VISM]*.C;* as input
-RMS-E-PRV, insufficient privilege or file protection violation
$ dir/nohead/notrail [bae.vism]mutt.c
ELLIS$DKA200:[BAE.VISM]MUTT.C;4
ELLIS$DKA200:[BAE.VISM]MUTT.C;3
$ write sys$output f$user()
[BRUDEN,MJE]
$
```

From BAE account:

```
$ dir/prot/owner/NOHEADER/NOTRAIL [bae]vism.dir
ELLIS$DKA200:[BAE]VISM.DIR;1
[BRUDEN,BAE] (RWE,RWE,E, )
$
```

Setting File Protection

To view a file's protection:

```
$DIRECTORY/PROTECTION FILE.DAT
```

```
FILE.DAT      (RE,RWED,RE,E)
               |  |  |  |
               S  O  G  W
```

```
$DIRECTORY/OWNER FILE.DAT
```

```
FILE.DAT [FRED]      !FILE OWNER'S UIC.
                      !FRED maps to a numeric
                      !UIC through the
                      !rightslist database
```

```
$SET SECURITY/PROTECTION=(G:RW,W) FILE.DAT
```

This changes the protection on FILE.DAT to allow GROUP READ and WRITE access and deny all access for WORLD.

When a file is created, it is given an owner and protection. The protection code is the default code at the file's creation.

```
$SHOW PROTECTION
```

- Displays the default protection code for new files.
- Changing the default has no effect on existing files.

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

- Changes the code for files created after this point in time.

Access Control Lists

Access control lists (ACLs) provide an optional layer of protection beyond UIC based protection. ACLs allow for a finer granularity of user categories.

- You can grant or deny access for specific users rather than groups of users.
- Access information is held in Access Control Lists.
- Identifier information is held in the Rights Database.
- Users can hold more than one identifier.
- Order of protection checking:
 - If no ACL for file, OpenVMS uses UIC protection.
 - If ACL disallows access, file is off limits unless you are the owner or a system user.
 - If user's identifiers do not match files ACL, OpenVMS uses UIC-based protection.

To allow a user to add or change ACL entries for a file:

```
$SHOW SECURITY TEST.DAT
$DIRECTORY/FULL TEST.DAT
$DIRECTORY/SECURITY TEST.DAT
$EDIT/ACL TEST.DAT
```

To delete an ACL:

```
$SET ACL/DELETE TEST.DAT
```

Access Control Lists Example

```
$ show security party.dat
```

```
ELLIS$DKA200:[BAE.VISM]PARTY.DAT;1 object of class FILE
  Owner: [BRUDEN,BAE]
  Protection: (System: RWED, Owner: RWED, Group: RE, World)
  Access Control List: <empty>
```

```
$
```

```
$ set security/acl=(identifier=obrien,access=none) party.dat
```

```
$
```

```
$ show security party.dat
```

```
ELLIS$DKA200:[BAE.VISM]PARTY.DAT;1 object of class FILE
  Owner: [BRUDEN,BAE]
  Protection: (System: RWED, Owner: RWED, Group: RE, World)
  Access Control List:
    (IDENTIFIER=[BRUDEN,OBRIEN],ACCESS=NONE)
```

```
$
```

```
$ set security/acl=(identifier=kjm,access=read) party.dat
```

```
$ show security party.dat
```

```
ELLIS$DKA200:[BAE.VISM]PARTY.DAT;1 object of class FILE
  Owner: [BRUDEN,BAE]
  Protection: (System: RWED, Owner: RWED, Group: RE, World)
  Access Control List:
    (IDENTIFIER=[KJM],ACCESS=READ)
    (IDENTIFIER=[BRUDEN,OBRIEN],ACCESS=NONE)
```

```
$
```

```
$ write sys$output f$fa0("!%U",F$IDENT("BAE","NAME_TO_NUMBER"))
```

```
[42,1]
```

```
$
```

From OBRIEN's account (denied access due to ACL)

```
$ write sys$output F$USER()  
[BRUDEN,OBRIEN]  
$ write sys$output f$fao("!%U",F$IDENT("OBRIEN","NAME_TO_NUMBER"))  
[42,3]  
$ type sys$sysdevice:[bae.vism]party.dat  
%TYPE-F-SYSERROR, system service error  
-SYSTEM-F-NOPRIV, insufficient privilege or object protection violation  
$
```

From KJM's account (denied access due to directory protection)

```
$ write sys$output F$USER()  
[KJM]  
$ write sys$output f$fao("!%U",F$IDENT("KJM","NAME_TO_NUMBER"))  
[52,10]  
$ type sys$sysdevice:[bae.vism]party.dat  
%TYPE-W-SEARCHFAIL, error searching for  
    SYS$SYSDEVICE:[BAE.VISM]PARTY.DAT;  
-RMS-E-PRV, insufficient privilege or file protection violation  
$
```

From BAE's account (enable WORLD EXECUTE access on VISM.DIR)

```
$ dir/prot [bae]vism.dir
```

```
Directory ELLIS$DKA200:[BAE]
```

```
VISM.DIR;1          (RWE,RWE,E,)
```

```
Total of 1 file.
```

```
$  
$ set secu/prot=(w:e) [bae]vism.dir  
$
```

From KJM's account (allowed access due to ACL)

```
$ write sys$output f$USER()  
[KJM]  
$ write sys$output f$fao("!%U",f$IDENT("KJM","NAME_TO_NUMBER"))  
[52,10]  
$ type sys$sysdevice:[bae.vism]party.dat  
The party is at Bruce's at 8:00PM  
$
```

From MJE's account (allowed access due to UIC)

```
$ write sys$output f$user()  
[BRUDEN,MJE]  
$ write sys$output f$fao("!%U",f$ident("mje","name_to_number"))  
[42,2]  
$ type sys$sysdevice:[bae.vism]party.dat  
The party is at Bruce's at 8:00PM  
$
```

Learning Check

1. A user's files are kept in the:
.....
2. If you wanted to know everything about a file, what command would you use?
.....
3. What two conditions must exist on a directory before you can delete it?
.....
.....
4. To specify multiple files with common characters in their file names, you would use a:
.....
5. Name three reasons why OpenVMS provides file protections.
.....
.....
.....
6. What are the four categories of OpenVMS user?
.....
.....
.....
.....
7. What are the four categories of file access?
.....
.....
.....
.....

Objectives

Much of the work done by computer users centers on files. To facilitate daily needs and activities in managing files, OpenVMS includes several useful tools.

This module describes a variety of methods for managing, maintaining, manipulating, and viewing files. After completing this module, you will be able to:

- List DCL commands for file management.
- Clean up directories.
- Search files.

Topics

- File Management Commands
- \$APPEND
- Record Types
- \$DIFFERENCES

File Management Commands

`$CREATE TEST.TXT`

Creates a new file without using an editor.

`$COPY FROM.HERE TO.HERE`

Makes a copy of a file.

`$DELETE DEFUNCT.NOW;4`

Deletes a file (must specify a version number on it).

`$RENAME MAUDE.LIN TIFFANY.WOW`

Changes all or part of a file name.

`$DELETE/CONFIRM *.OOPS;*`

Prompts for confirmation before deleting files.

`$DELETE/LOG BYE.BYE;*`

Displays file names as they are deleted.

`$PURGE`

Deletes all but the highest version of your files.

`$PURGE/KEEP=3`

Deletes all but the highest three versions

`$CREATE/DIRECTORY/VERSION_LIMIT=3 ...`

Automatically controls the number of versions of files in the new subdirectory (default 32767).

`$SET DIRECTORY/VERSION_LIMIT=3 ...`

Limits versions on new files in an existing directory.

`$SET FILE/VERSION_LIMIT=3 [OBRIEN].FUN;`

Changes version limits on existing files.

`$DUMP/HEADER/BLOCK=(COUNT:0) BILLS.DIR`

Displays file header information including version limits.

APPEND Command

The APPEND command allows the concatenation of two or more files into one output file.

- The output file must exist prior to the append operation (use **/NEW** otherwise).
- No new version of the output file is created by default.
- Input files are not affected by the append operation.

```
$APPEND SMALL.TXT BIG.TXT
```

Appends SMALL.TXT on to the end of BIG.TXT.

```
$APPEND *.DAT BIG.TXT
```

Concatenates the highest version of all DAT files in the directory to BIG.TXT.

```
$APPEND *.DAT;* BIG.TXT
```

Concatenates all versions of all DAT files in the directory to BIG.TXT.

```
$APPEND ARF.ARF CREATE_ME.PLEASE/NEW
```

Creates a new output file if one does not already exist.

```
$APPEND/NEW/LOG/READ_CHECK/WRITE_CHECK PUTTY.BALL BIG.TXT
```

Creates new version of BIG.TXT if necessary, informs you as it finishes processing a file, and reads each record from the input and output files twice to verify that they are correct.



Important

- Do not append files with different record types.
-

Record Types

- **Variable Length**
 - Varies in length with the number of bytes of information.
 - Actual length is stored in the front of the record.
 - Created by the EDT editor.
- **Fixed Length**
 - File would have been defined with fixed length records.
 - Size of record is constant even if the number of bytes in the record is not.
 - Created by programs.
- **Variable with Fixed Control**
 - Beginning portion is fixed length with remainder variable length.
 - Used by some editors and programs.
- **Stream**
 - Stream of bytes with terminator character or sequence.
 - Commonly generated by C programs.
- **\$DIRECTORY/FULL**

Tells what type of record is in a file.
- **\$CONVERT**
 - Utility to change record types in a file (among other things).
 - Discussed in the RMS course.

DUMP Utility

It may be useful in some circumstances to view the raw contents of a file. The DUMP utility allows you to dump a file in its raw form. This may be particularly useful for a programmer.

Examples

Simply type the file.

```
$ type abc.dat
```

```
Line 1
```

```
Another line
```

Dump by record.

```
$ dump/record abc.dat
```

```
Dump of file ELLIS$DKA200:[BAE]ABC.DAT;1 on 28-MAR-1999 00:43:51.99
```

```
File ID (6532,32807,0) End of file block 1 / Allocated 3
```

```
Record number 1 (00000001), 6 (0006) bytes, RFA(0001,0000,0000)
```

```
3120 656E694C Line 1..... 000000
```

```
Record number 2 (00000002), 11 (000B) bytes, RFA(0001,0000,0008)
```

```
656E69 6C207265 746F6E41 Aoter line..... 000000
```

```
$
```

Generate a raw dump.

```
$ dump abc.dat
```

```
Dump of file ELLIS$DKA200:[BAE]ABC.DAT;1 on 28-MAR-1999 00:44:43.44
```

```
File ID (6532,32807,0)   End of file block 1 / Allocated 3
```

```
Virtual block number 1 (00000001), 512 (0200) bytes
```

```
7265746F 6E41000B 3120656E 694C0006 ..Line 1..Anoter 000000
00000000 00000000 FFFF0065 6E696C20  line..... 000010
00000000 00000000 00000000 00000000 ..... 000020
00000000 00000000 00000000 00000000 ..... 000030
00000000 00000000 00000000 00000000 ..... 000040
00000000 00000000 00000000 00000000 ..... 000050
00000000 00000000 00000000 00000000 ..... 000060
00000000 00000000 00000000 00000000 ..... 000070
00000000 00000000 00000000 00000000 ..... 000080
00000000 00000000 00000000 00000000 ..... 000090
00000000 00000000 00000000 00000000 ..... 0000A0
00000000 00000000 00000000 00000000 ..... 0000B0
00000000 00000000 00000000 00000000 ..... 0000C0
00000000 00000000 00000000 00000000 ..... 0000D0
...
00000000 00000000 00000000 00000000 ..... 0001A0
00000000 00000000 00000000 00000000 ..... 0001B0
00000000 00000000 00000000 00000000 ..... 0001C0
00000000 00000000 00000000 00000000 ..... 0001D0
00000000 00000000 00000000 00000000 ..... 0001E0
00000000 00000000 00000000 00000000 ..... 0001F0
$
```

DIFFERENCES Command

The DIFFERENCES command compares the contents of two disk files and creates a listing of the records that do not match.

Use DIFFERENCES to see which parts of the file were changed rather than trying to remember. This is useful if the previous version of a program works but the new one does not. For example:

```
$DIFFERENCES FIRST.COB;4 FIRST.COB;5
```

Files are compared on a record by record basis.

```
$DIFFERENCES/IGNORE=SPACING FIRST.COB;2 FIRST.COB;4
```

- Instructs DIFFERENCES to ignore spacing or blank lines.
- Writes to the terminal the record from each file that does not match as well as one matching line after that.

```
$DIFFERENCES/MERGED=3 file1 file2
```

Produces three matching records after the one that does not match.

```
$DIFFERENCES/OUTPUT=PHEW.TXT bigfile1 bigfile2
```

Places the output in the named file. Useful for comparison of long files with many differences.

SEARCH Command

The SEARCH command allows you to search files for lines that contain strings.

Format: `$ SEARCH filespec[,...] string[,...]`

For example:

```
$SEARCH HAYSTACK.DAT "NEEDLE"
```

Indicates all occurrences of the string “needle” in the file HAYSTACK.DAT.

- `/MATCH` allows you to match on AND, EQU, NOR, NAND, OR, and XOR criteria.
- `/STATISTICS` allows you to determine how many matching lines occur in the file.
- `/EXACT` does a case-sensitive search.

Examples

```
$ type x.dat
```

```
Bruce Ellis
```

```
Bruce Springsteen
```

```
Dock Ellis
```

```
$ search x.dat "bruce","ellis"
```

```
Bruce Ellis
```

```
Bruce Springsteen
```

```
Dock Ellis
```

```
$ search x.dat/match=and "bruce","ellis"
```

```
Bruce Ellis
```

```
$ search/stat/noout x.dat/match=and "bruce","ellis"
```

Files searched:	1	Buffered I/O count:	6
Records searched:	3	Direct I/O count:	1
Characters searched:	38	Page faults:	30
Records matched:	1	Elapsed CPU time:	0 00:00:00.03
Lines printed:	0	Elapsed time:	0 00:00:00.04

```
$
```

Learning Check

1. Match each command with its description.

- | | | |
|----------------------------------|-------|--|
| a. CREATE TEXT.TXT | | Deletes all but the highest three versions. |
| b. RENAME MAUDE.LIN TIFFANY.WOW | | Changes all or part of a file name. |
| c. DELETE DEFUNCT.NOW;4 | | Makes a new file without using an editor. |
| d. PURGE/KEEP=3 | | Deletes a file. |
| e. SET DIRECTORY/VERSION_LIMIT=3 | | Limits versions on new files in an existing directory. |

2. To make a copy of a file, you would use:

- a. DUMP
- b. COPY
- c. APPEND
- d. SEARCH

3. To view the raw contents of a file, you would use:

- a. DUMP
- b. COPY
- c. APPEND
- d. SEARCH

4. To find lines containing strings in a file, you would use:

- a. DUMP
- b. COPY
- c. APPEND
- d. SEARCH

5. Use the DIFFERENCES command to:

.....
.....

6. Use the APPEND command to:

.....
.....

Objectives

Logical names are names equated to an equivalence string. Equivalence strings are usually full or partial file specifications, another logical name, or any other character string.

Logical names serve two main functions:

- Shorthand and readability
- File and device independence

This module discusses the ways in which logical names are created, displayed, and used. After completing this module, you will be able to:

- List uses of logical names.
- Create a logical name.
- Describe four types of logical name tables.

Topics

- Creating Logical Names
- Use of Logical Names
- Process Logical Names
- Job Logical Names
- Group Logical Names
- System Logical Names
- Process Permanent Logical Names

Logical Names

A logical name is a string of characters defined as an equivalence string. A common usage for logical names is the representation of all or part of a file specification.

- The primary purpose of logical names is to support device and file name independence.
- Logical names can reduce typing and improve readability as well as reduce the likelihood of errors.
- Logical names can be used in programs and command procedures in place of actual file specifications.
- Logical names can make your specifications more descriptive.
- Full filespec:
BRUDEN : : \$1\$DKA200 : [OBRIEN] TEST . DAT ; 8

Logical Names Example

```
$ type dump_nums_bad.c
```

```
...
```

```
/* Open file. */
```



Note

The file name is hard coded into the program.

```
fp=fopen("dka200:[bae]cash5.dat","r");
if(!fp)
{
    perror("Open error");
    exit(EXIT_FAILURE);
}
```

```
...
```

```
return(EXIT_SUCCESS);
```

```
}
```

```
$ r dump_nums_bad
```

```
2- 19- 20- 27- 32
```

```
8- 15- 16- 27- 31
```

```
15- 24- 27- 28- 29
```

```
3- 10- 12- 21- 28
```

```
5- 7- 13- 21- 31
```

```
14- 17- 22- 26- 30
```

```
5- 7- 12- 17- 32
```

```
4- 6- 14- 18- 25
```

```
$ create/dir [.lost_money]
```

We move the file.

```
$ rename cash5.dat [.lost_money]cash5.dat
```

The program breaks and cannot be fixed without changing the program.

```
$ r dump_nums_bad
```

```
Open error: no such file or directory
```

```
$
```

Now we shorten the file name to a form that could be a logical name.

```
$ type dump_nums.c
...
/* Open file. */
    fp=fopen("cash5","r");
    if(!fp)
    {
        perror("Open error");
        exit(EXIT_FAILURE);
    }
    ...
    return(EXIT_SUCCESS);
}
```

Initially the program fails, until we define the logical name.

```
$ run dump_nums
Open error: no such file or directory
$ define cash5 dka200:[bae.lost_money]cash5.dat
```

After defining the logical name, the unchanged program works fine.

```
$ run dump_nums
2- 19- 20- 27- 32
8- 15- 16- 27- 31
15- 24- 27- 28- 29
3- 10- 12- 21- 28
5- 7- 13- 21- 31
14- 17- 22- 26- 30
5- 7- 12- 17- 32
4- 6- 14- 18- 25
$ show logical cash5
"CASH5" = "DKA200:[BAE.LOST_MONEY]CASH5.DAT" (LNM$PROCESS_TABLE)
$
```

We move the file.

```
$ rename [ .lost_money]cash5.dat []cash5.dat
```

Then we change the logical name to point to the new location.

```
$ define cash5 dka200:[bae]cash5.dat
```

```
%DCL-I-SUPERSEDE, previous value of CASH5 has been superseded
```

The unchanged program works fine.

```
$ run dump_nums
```

```
2- 19- 20- 27- 32
```

```
8- 15- 16- 27- 31
```

```
15- 24- 27- 28- 29
```

```
3- 10- 12- 21- 28
```

```
5- 7- 13- 21- 31
```

```
14- 17- 22- 26- 30
```

```
5- 7- 12- 17- 32
```

```
4- 6- 14- 18- 25
```

```
$
```

Logical Name Creation

Logical names can be created using either the of two DCL commands: DEFINE or ASSIGN.

```
$ DEFINE SHORT DUA2:[OBRIEN.EXTREME.MORE]
```

- You can now use the logical name SHORT anywhere a device and directory specification would appear.
- This command is functionally the same as:

```
$ ASSIGN DUA2:[OBRIEN.EXTREME.MORE] SHORT
```

For example:

```
$ SET DEFAULT SHORT
```

Changes the default directory to DUA2:[OBRIEN.EXTREME.MORE].

```
$ TYPE SHORT:NEWMAN.DAT
```

Displays the file DUA2:[OBRIEN.EXTREME.MORE]NEWMAN.DAT.

Logical names can be used to represent any of the following portions of a file specification:

- Device
- Directory (when the directory is on default disk)
- Device and directory
- File name (when file is on default device and directory)
- Full file specification

Logical names can be defined using other logical names. When using a logical name to represent a leading portion of a file specification, delimit the logical name from the remaining file specification using a colon (:).

Examples

```
$ define my_disk $1$DKA200:
```

```
$ define work_dir my_disk:[xyz.to_do]
```

```
$ type work_dir:lotsa_stuff.txt
```

Displays \$1\$dka200:[xyz.to_do]lotsa_stuff.txt

```
$ type my_disk:[xyz]z.dat
```

Displays \$1\$dka200:[xyz]z.dat

**Note**

Given the previous example, the following examples are **not** valid:

```
$ type work_dir:[more_work]awwww.dat
```

```
$ type work_dir:[.more_work]ohhhhh_nooo.dat
```

See rooted logical names for a solution to this problem.

Logical Name Tables

Logical names and their equivalence strings are stored in logical name tables. Logical name tables are stored in memory.

Process Logical Name Table

- Contains process private logical names.
- Is named LNM\$PROCESS_TABLE.
- Logical name table LNM\$PROCESS_DIRECTORY contains a logical name LNM\$PROCESS that is equated to LNM\$PROCESS_TABLE.
- Created when you log in.
- Deleted when you log out.
- This is normally the default table used for logical name creation.

Managing Process Logical Names

```
$ DEFINE ME $1$DKA200:[MY_DIR]PERSONAL.DATA
```

Creates process logical names

```
$ SHOW LOGICAL ME
```

Views a specific logical name

```
$ SHOW LOGICAL/PROCESS
```

Views all process logical names

```
$ DEASSIGN ME
```

Deletes a process logical name

Viewing Process Logical Names

```
$ show logical/process
```

```
(LNM$PROCESS_TABLE)
```

```
"SYS$COMMAND" = "_ELLIS$TNA3:"
```

```
"SYS$DISK" = "ELLIS$DKA200:"
```

```
"SYS$ERROR" = "_ELLIS$TNA3:"
```

```
"SYS$INPUT" = "_ELLIS$TNA3:"
```

```
"SYS$OUTPUT" [super] = "_ELLIS$TNA3:"
```

```
"SYS$OUTPUT" [exec] = "_ELLIS$TNA3:"
```

```
"TT" = "TNA3:"
```

```
$
```

Job Logical Name Tables

- The job logical name table contains logical names shared by the detached process and any subprocesses.
- The name of the job logical name table is LNM\$JOB_XXXXXXXX (where XXXXXXXX is an eight-digit hexadecimal number).
- The logical name table LNM\$PROCESS_DIRECTORY contains a logical name LNM\$JOB that is equated to LNM\$JOB_XXXXXXXX.
- Created when the first process in a job is created.
- Deleted when the last process in a job is deleted.

Managing Job Logical Names

```
$ DEFINE/JOB WORK_FILE $1$DKA200:[XYZ]WORK.DAT
```

Creates job logical names

```
$ SHOW LOGICAL/JOB WORK_FILE
```

Views a specific job logical name

```
$ SHOW LOGICAL/JOB
```

Views all job logical names

```
$ DEASSIGN/JOB WORK_FILE
```

Deletes job logical names

Viewing Job Logical Names

```
$ show logical/job
```

```
(LNM$JOB_80D19600)
```

```
"SYS$LOGIN" = "ELLIS$DKA200:[BAE]"
```

```
"SYS$LOGIN_DEVICE" = "ELLIS$DKA200:"
```

```
"SYS$SCRATCH" = "ELLIS$DKA200:[BAE]"
```

```
$
```

Group Logical Name Tables

- The group logical name table contains logical names that are available to users in your UIC group number.
- The name of your group logical name table is LNM\$GROUP_XXXXXX where XXXXXX = your group number.
- The logical name table LNM\$PROCESS_DIRECTORY contains a logical name LNM\$GROUP that is equated to LNM\$GROUP_XXXXXX.
- This name table is created when first name is added to the table (typically when system is bootstrapped).
- The table is deleted when the system shuts down.

Managing Group Logical Names

```
$ DEFINE/GROUP PROJECT $2$DQA2:[PROJ_FILES]
```

Creates group logical names

```
$ SHOW LOGICAL/GROUP PROJECT
```

Views a specific group logical name

```
$ SHOW LOGICAL/GROUP
```

Views all group logical names

```
$ DEASSIGN/GROUP PROJECT
```

Deletes group logical names

You need privilege to modify the contents of table. A group/project leader typically maintains the table. Any of following privileges allows you to modify the table:

- GRPNAM
- GRPPRV
- SYSPRV

Viewing Group Logical Names

View the protection for the table.

```
$ sh sec/class=logical_name_table lnm$group
```

```

LNMS$GROUP_000042 object of class LOGICAL_NAME_TABLE
  Owner: [BRUDEN,*]
  Protection: (System: RWCD, Owner: R, Group: R,
World)
  Access Control List: <empty>
```

With no privilege we can view the table but not change it.

```
$ def/group bruce ellis
%SYSTEM-F-NOPRIV, insufficient privilege or object
protection violation
```

With SYSPRV we can write to it.

```
$ set proc/priv=sysprv
$ def/group bruce ellis
$ set proc/priv=nosysprv
```

GRPPRV allows system access for resources owned by the same group as us, so we have write access to the table.

```
$ set proc/priv=grpprv
$ def/group albertellis
_Equ name: logical
$ set proc/priv=nogrprrv
GRPNAM allows write access.
$ set proc/priv=grpnam
$ def/group albert einstein
```

You can view the table with no privilege.

```
$ set proc/priv=nogrpnam
$ show logical/group

(LNM$GROUP_000042)

"ALBERT" = "EINSTEIN"
"ALBERTELLIS" = "LOGICAL "
"BRUCE" = "ELLIS"
$
```

System Logical Name Table

- The system logical name table contains logical names that are available to all users on the system.
- LNM\$SYSTEM_TABLE is the name of the system logical name table.
- The logical name table LNM\$SYSTEM_DIRECTORY contains a logical name LNM\$SYSTEM that is equated to LNM\$SYSTEM_TABLE and then LNM\$SYSCLUSTER (a clusterwide logical name table).
- The table is created when the system is booted.
- It is deleted when system shuts down.

Managing System Logical Names

```
$ DEFINE/SYSTEM ALL_FOR_ONE SYS$MANAGER:AND_1_FOR_ALL.DAT
```

Creates system logical names

```
$ SHOW LOGICAL/SYSTEM ALL_FOR_ONE
```

Views a specific system logical name

```
$ SHOW LOGICAL/SYSTEM
```

Views all system logical names

```
$ DEASSIGN/SYSTEM ALL_FOR_ONE
```

Deletes system logical names

Privilege is required to modify contents of table. Typically, this table is maintained by the system manager. The following privileges can be used to change the contents of the system logical name table:

SYSNAM

SYSPRV

Viewing the System Logical Names

```
$ show logical/system
```

```
(LNM$SYSTEM_TABLE)
```

```
"ACLEDTSR_TV" = "ACLEDTSR"
"ACP$BADBLOCK_MBX" = "MBA5:"
"AGEN$IMAGES" = "SYS$MANAGER:VMS$IMAGES_MASTER.DAT"
"ALPHA$LIBRARY" = "SYS$SYSROOT:[SYSLIB]"
"ALPHA$LOADABLE_IMAGES" = "SYS$SYSROOT:[SYS$LDR]"
"AUDSRV$CONTROL_MAILBOX" = "_MBA10:"
"BASRTL2_TV" = "BASRTL2_D53_TV"
"BASRTL_TV" = "BASRTL_D56_TV"
"BLAS1RTL_TV" = "BLAS1RTL_D53_TV"
"CDA$ACCESS_TV" = "CDA$ACCESS_TV_SUPPORT.EXE"
"CDDSHR_TV" = "CDDSHR"
"CDE$DETACHED_LOGICALS" = "DECW$DISPLAY, LANG"
"CDE$EXAMPLES" = "CDE$SYSTEM_DEFAULTS:[EXAMPLES]"
"CDE$LOG_PROCESSES" = "0"
"CDE$PATH" = "[]"
    = "CDE$USER_DEFAULTS:[BIN]"
    = "CDE$SYSTEM_DEFAULTS:[BIN]"
"CDE$SESSIONMAIN" = "mcr cde$system_defaults:[bin]dtsession"
"CDE$SPAWN_PROCESSES" = "0"
"CDE$SYSTEM_COMMON" = "ELLIS$DKA200:[SYS0.SYSCOMMON.CDE$DEFAULTS.SYSTEM.]"
"CDE$SYSTEM_DEFAULTS" = "CDE$SYSTEM_SPECIFIC:"
    = "CDE$SYSTEM_COMMON:"
"CDE$SYSTEM_SPECIFIC" = "ELLIS$DKA200:[SYS0.CDE$DEFAULTS.SYSTEM.]"
"CDE$USER_COMMON" = "ELLIS$DKA200:[SYS0.SYSCOMMON.CDE$DEFAULTS.USER.]"
"CDE$USER_DEFAULTS" = "CDE$USER_SPECIFIC:"
    = "CDE$USER_COMMON:"
```

```
...
"SDA$READ_DIR" = "SYS$SYSROOT:[SYS$LDR]"
    = "SYS$SYSROOT:[SYS$LIB]"
"SECSRV$MAILBOX" = "_MBA11:"
"SECURESHR_TV" = "SECURESHR_JACKET"
"SMBSRVSHR_TV" = "SMBSRVSHR"
"SMGSHR_TV" = "SMGSHR"
"SORTSHR_TV" = "SORTSHR"
"SPISHR_TV" = "SPISHR"
"STARTUP$PHASES" = "SYS$STARTUP:VMS$PHASES.DAT"
"STARTUP$STARTUP_LAYERED" = "SYS$STARTUP:VMS$LAYERED.DAT"
"STARTUP$STARTUP_LIST" = "STARTUP$STARTUP_VMS"
    = "STARTUP$STARTUP_LAYERED"
"STARTUP$STARTUP_VMS" = "SYS$STARTUP:VMS$VMS.DAT"
"SYS$ANNOUNCE" = "Welcome to OpenVMS (TM) Alpha Operating System, Version
V7.2    "
"SYS$COMMON" = "ELLIS$DKA200:[SYS0.SYSCOMMON.]"
"SYS$DECDTM_NODE_ADDRESS" = "1048"
"SYS$DECDTM_NODE_NAME" = "LOCAL:.ELLIS"
"SYS$DISK" = "ELLIS$DKA200:"
"SYS$ERRORLOG" = "SYS$SYSROOT:[SYSERR]"
"SYS$ETC" = "SYS$SYSROOT:[SYSHLP.UNSUPPORTED]"
"SYS$EXAMPLES" = "SYS$SYSROOT:[SYSHLP.EXAMPLES]"
"SYS$HELP" = "SYS$SYSROOT:[SYSHLP]"
"SYS$I18N_ICONV" = "SYS$SYSROOT:[SYS$I18N.ICONV.USER]"
    = "SYS$SYSROOT:[SYS$I18N.ICONV.SYSTEM]"
"SYS$I18N_LOCALE" = "SYS$SYSROOT:[SYS$I18N.LOCALES.USER]"
    = "SYS$SYSROOT:[SYS$I18N.LOCALES.SYSTEM]"
"SYS$INSTRUCTION" = "SYS$SYSROOT:[SYSCBI]"
"SYS$JOURNAL" = "SYS$COMMON:[SYSEXE]"
"SYS$LANGUAGE" = "ENGLISH"
```

```

"SYS$LIBRARY" = "SYS$SYSROOT:[SYSLIB]"
"SYS$LOADABLE_IMAGES" = "SYS$SYSROOT:[SYS$LDR]"
"SYS$LOCALTIME" = "SYS$SYSROOT:[SYS$ZONEINFO.SYSTEM.US]MOUNTAIN."
"SYS$MAINTENANCE" = "SYS$SYSROOT:[SYS$MAINT]"
"SYS$MANAGER" = "SYS$SYSROOT:[SYS$MGR]"
"SYS$MESSAGE" = "SYS$SYSROOT:[SYS$MSG]"
"SYS$NET_SERVICES_1" = "DECNET"
    = "TCP/IP"
"SYS$NET_SERVICES_10" = "mcr ncl show node 0 session control all counters"
    = "TCPIP SHOW COMMUNICATION"
"SYS$NET_SERVICES_11" = "@sys$startup:net$startup"
    = "@SYS$STARTUP:TCPIP$STARTUP.COM"
"SYS$NET_SERVICES_12" = "@sys$manager:net$shutdown"
    = "@SYS$STARTUP:TCPIP$SHUTDOWN.COM"
"SYS$NET_SERVICES_13" = " "
    = "TCPIP$PPPD_CALLOUT"
"SYS$NET_SERVICES_2" = "Digital Equipment Corporation"
    = "Digital Equipment Corporation"
"SYS$NET_SERVICES_3" = "ELLIS"
    = "ellis.zzzzzzzzzz"
"SYS$NET_SERVICES_4" = "1.24"
    = "192.206.126.180"
"SYS$NET_SERVICES_5" = "DNA V"
    = "TCP/IP"
"SYS$NET_SERVICES_6" = "net 0"
    = " "
"SYS$NET_SERVICES_7" = " "
    = " "
"SYS$NET_SERVICES_8" = "mcr ncl show node 0 all"
    = "TCPIP SHOW VERSION"

```

```
"SYS$NET_SERVICES_9" = "mcr ncl show node 0 session control port * all"
    = "TCPIP SHOW DEVICE_SOCKET"
"SYS$NLSPATH" = "SYS$SYSROOT:[SYS$I18N.MESSAGES.USER.%L]%N.CAT"
    = "SYS$SYSROOT:[SYS$I18N.MESSAGES.SYSTEM.%L]%N.CAT"
"SYS$NODE" = "ELLIS:."
"SYS$NODE_FULLNAME" = "LOCAL:.ELLIS:."
"SYS$POSIXRULES" = "SYS$SYSROOT:[SYS$ZONEINFO.SYSTEM.US]MOUNTAIN."
"SYS$PS_FONT_METRICS" = "SYS$SYSROOT:[SYSFONT.PS_FONT_METRICS.USER]"
    = "SYS$SYSROOT:[SYSFONT.PS_FONT_METRICS.SYSTEM]"
"SYS$SHARE" = "SYS$SYSROOT:[SYSLIB]"
"SYS$SPECIFIC" = "ELLIS$DKA200:[SYS0.]"
"SYS$STARTUP" = "SYS$SYSROOT:[SYS$STARTUP]"
    = "SYS$MANAGER"
"SYS$SYLOGIN" = "SYS$MANAGER:SYLOGIN"
"SYS$SYSDEVICE" = "ELLIS$DKA200:."
"SYS$SYSDISK" = "SYS$SYSROOT:."
"SYS$SYSROOT" = "ELLIS$DKA200:[SYS0.]"
    = "SYS$COMMON:."
"SYS$SYSTEM" = "SYS$SYSROOT:[SYSEXEC]"
"SYS$TEST" = "SYS$SYSROOT:[SYSTEST]"
"SYS$TIMEZONE_DAYLIGHT_SAVING" = "0"
"SYS$TIMEZONE_DIFFERENTIAL" = "-25200"
"SYS$TIMEZONE_NAME" = "MST"
"SYS$TIMEZONE_RULE" = "MST7MDT6,M4.1.0/2,M10.5.0/2"
"SYS$TOPSYS" = "SYS0"
"SYS$TZDIR" = "SYS$SYSROOT:[SYS$ZONEINFO.SYSTEM]"
"SYS$UPDATE" = "SYS$SYSROOT:[SYSUPD]"
"SYS$VMSDOC_TEXT" = "SYS$SYSROOT:[SYSHLP.VMSDOC]"
...

(LNM$SYSCLUSTER_TABLE)
$
```

Logical Name Translation

Logical names are translated automatically by OpenVMS. This is done when a program or a DCL command opens a file.

- The logical name tables are searched for the first occurrence of a logical name.
- Search order of tables (based on logical name LNM\$FILE_DEV):
 - Process table
 - Job table
 - Group table
 - System table
- The translation is performed:
 - Up to 10 times recursively
 - Until no more equivalence names to be translated
 - Until equivalence name is a logical name that has the terminal attribute

Viewing LNM\$FILE_DEV

```
$ sh log/table=lnm$system_directory lnm$file_dev
  "LNM$FILE_DEV" = "LNM$PROCESS" (LNM$SYSTEM_DIRECTORY)
    = "LNM$JOB"
    = "LNM$GROUP"
    = "LNM$SYSTEM"
    = "DECW$LOGICAL_NAMES"
1  "LNM$SYSTEM" = "LNM$SYSTEM_TABLE" (LNM$SYSTEM_DIRECTORY)
    = "LNM$SYSCUSTER"
2  "LNM$SYSCUSTER" = "LNM$SYSCUSTER_TABLE"
(LNM$SYSTEM_DIRECTORY)
1  "DECW$LOGICAL_NAMES" [table] = "" (LNM$SYSTEM_DIRECTORY)
$
```

Process Permanent Logical Names

Process permanent logical names are created when you log in and describe:

- Files that can remain open for the life of your process.
- Useful directories.

Default Process Permanent Logical Names

By default, the following are created for you when you log in:

```
$ show logical/process/job
```

```
(LNM$PROCESS_TABLE)
```

```
"SYS$COMMAND" = "_ELLIS$TNA3:"  
"SYS$DISK" = "ELLIS$DKA200:"  
"SYS$ERROR" = "_ELLIS$TNA3:"  
"SYS$INPUT" = "_ELLIS$TNA3:"  
"SYS$OUTPUT" [super] = "_ELLIS$TNA3:"  
"SYS$OUTPUT" [exec] = "_ELLIS$TNA3:"  
"TT" = "TNA3:"
```

```
(LNM$JOB_80D19600)
```

```
"SYS$LOGIN" = "ELLIS$DKA200:[BAE]"  
"SYS$LOGIN_DEVICE" = "ELLIS$DKA200:"  
"SYS$SCRATCH" = "ELLIS$DKA200:[BAE]"  
$
```

Redefining Process Permanent Logical Names

- SYS\$OUTPUT is the default output device/file for your process.
\$ DEFINE SYS\$OUTPUT BIG.DAT
\$ DIRECTORY
.
.
.
\$ HELP
- All output is directed to BIG.DAT, until:
\$ DEASSIGN SYS\$OUTPUT
- To redirect output for the life of one image:
\$ DEFINE/USER_MODE SYS\$OUTPUT ONE_PROG.DAT

Other Logical Name Features

- **Logical name search lists**
Allows RMS to search more than one location for a file by logical name
- **User-defined logical name tables**
Allows for application/implementation specific shared logical name tables
- **Rooted logical names**
Allows for device and rooted directory specification to be defined as a logical name and used in the place of a device spec in a file specification

Learning Check

1. What is the purpose of logical names?

.....

.....

.....

2. Which two DCL commands can you use to create a logical name?

.....

.....

3. What is a logical name table?

.....

.....

.....

4. What are the four types of logical name tables?

.....

.....

.....

.....

Objectives

As you carry out computing tasks, you may need to store and manipulate data such as numbers and strings of characters. OpenVMS provides such a capability using symbols. Symbols are temporary. They exist only for the life of your computing session (process) or for the life of a command procedure's execution.

Symbols can also be used as command synonyms. They are used to shorten typing and to change the default action of DCL commands.

This module teaches you how to use symbols as temporary storage and command synonyms. After completing this module, you will be able to:

- Describe symbol usage.
- Describe the difference between local and global symbols.
- List when and where symbols should be used.

Topics

- Symbol Creation
- Symbol Use
- Local Symbol Table
- Global Symbol Table
- Wildcards
- Displaying Symbols
- Defining Keys

DCL Symbols

Symbols are names that represent character strings or numeric values.

- They can be used as DCL command synonyms.
- They are stored in one of two tables:
 - Local
 - Global
- They are often defined in a user's `login.com` command procedure.
- They are process private, only available to your process.
- DCL automatically looks for a symbol as the first set of characters on the command line (the command verb).

Creating Symbols

```
$ GO == "SET DEFAULT"
```

```
$ UP == "SET DEFAULT [-]"
```

```
$ DIR == "DIR/OWN/PROT"
```

```
$ DEL == "DELETE/CONFIRM"
```

Symbol Tables

- Local symbol table
 - Created with one equal sign
 - Visible to current command procedure and command procedures executed from within the current command procedure
 - Exits when the defining command level exits
- Global symbol table
 - Created with two equal signs
 - Available throughout your process

Viewing and Deleting Symbols

- To view one symbol (/LOCAL is the default):
\$ SHOW SYMBOL/LOCAL SUB
\$ SHOW SYMBOL/GLOBAL GO
- To view all symbols in a table (/LOCAL is the default):
\$ SHOW SYMBOL/LOCAL/ALL
\$ SHOW SYMBOL/GLOBAL/ALL
- To delete one symbol from table (/LOCAL is the default):
\$ DELETE/SYMBOL/LOCAL SUB
\$ DELETE/SYMBOL/GLOBAL GO
- To delete all symbols from table (/LOCAL is the default):
\$ DELETE/SYMBOL/LOCAL/ALL
\$ DELETE/SYMBOL/GLOBAL/ALL

Using Wildcards to Abbreviate

You can abbreviate symbol names using the asterisk (*) as the wildcard character, for example:

```
$ DIR*ECTORY == "DIR/OWNER/PROT/SIZE"
```

- Characters to the left of the asterisk are required.
- Characters to the right of the asterisk are optional.
- DIR, DIRE, DIREC, DIRECT, DIRECTO, DIRECTOR, DIRECTORY all cause symbol match to succeed.
- DIREX causes symbol match to fail.
- To delete an abbreviated symbol you should use the full symbol name without the embedded asterisk.

```
$ DELETE/SYMBOL DIRECTORY !(omit asterisk)
```

Symbol Substitution

Symbols can be used for DCL command synonyms or for variables. They are often used within command procedures.

Symbol substitution occurs when:

- DCL processes a command verb starting at the beginning of the command line, ending with the first white space character.
- A symbol is enclosed within apostrophes (').
- A symbol is assigned to another symbol.
- A symbol is used as an argument to a lexical function.
- A symbol is embedded in a quoted string and preceded by two apostrophes.
- A symbol is used as an argument to a DCL `WRITE` command.
- A symbol is used in a conditional expression (an `IF` statement).

Symbol Substitution Examples

```
$ param="time"
```



Note

No automatic substitution of symbols.

```
$ show param
```

```
%DCL-W-IVKEYW, unrecognized keyword - check validity and
spelling
```

```
\PARAM\
```

```
$
```

```
$ show 'param'
```

```
3-JUN-1997 00:33:52
```

```
$
```

```
$ x="abcdefg"
```

```
$ write sys$output x
```

```
abcdefg
```

```
$
```

Apostrophes are ignored within double quotes, unless there are two adjacent apostrophes.

```
$ write sys$output "x='x'"
```

```
x='x'
```

```
$ write sys$output "x=' 'x'"
```

```
x=abcdefg
```

```
$ write sys$output "x=",x
```

```
x=abcdefg
```

```
$
```

Automatic substitution on symbol assignment.

```
$ y=x
```

```
$ sh sym y
```

```
Y = "abcdefg"
```

```
$
```

Symbol Types

The symbols used up to this point are *string* symbols, as designated by the quotes around the equivalence string.

- Symbols can also be defined as integer symbols, for computational purposes.
- An integer symbol is defined with an equivalence string that:
 - Is not quoted
 - Is made up of integer digits
- An integer symbol assignment may be converted from hexadecimal or octal by using a leading %X for hexadecimal or a leading %O for octal.
- Symbols may be converted:
 - From string to integer using the F\$INTEGER(*symbol*) lexical function.
 - From integer to string using the F\$STRING(*symbol*) lexical function.

Integer Symbol Examples

```
$ x=42
$ sh sym x
  X = 42   Hex = 0000002A   Octal = 00000000052
$ x=%x1F
$ show symbol x
  X = 31   Hex = 0000001F   Octal = 00000000037
$ x=%o37
$ sh sym x
  X = 31   Hex = 0000001F   Octal = 00000000037
$
```

The following definitions are for string substitutions.

```
$ x="31"
$ y="7"
```

With string symbols + gives you concatenation.

```
$ z=x+y
$ sh sym z
  Z = "317"
```

We can force addition using the F\$INTEGER lexical function.

```
$ z=f$integer(x)+f$integer(y)
$ sh sym z
  Z = 38   Hex = 00000026   Octal = 00000000046
$ x=f$string(z)
$ sh sym z
  Z = 38   Hex = 00000026   Octal = 00000000046
$ sh sym x
  X = "38"
$
```

Symbols Compared to Logical Names

- Symbols are processed by DCL and should be used for command substitution and as variables.
- Logical names are processed by RMS and other OpenVMS system services and are automatically translated wherever a file name or a device name is used.

Function/Capability	Symbol	Logical Name
Processor	DCL	OpenVMS system services and RMS
Usage	Command synonyms and variables	File/device names and defining context for programs and processes
Translation	Automatic as the command verb and with specific commands, such as conditionals, \$WRITE, symbol assignment, etc. When used on the command line in other instances, you must force substitution with apostrophes (').	Automatic when used as a portion of or a complete file specification.
Accessibility	Current process only	Current process or job, group-wide, and systemwide
DCL	Required	Not required

Defining Keys at the DCL Prompt

Keys can be defined that are set equivalent to DCL commands or any character string.

- Syntax:
\$ DEFINE/KEY KEYNAME DCL-COMMAND
- Example:
\$ DEFINE/KEY PF1/NOTERMINATE "RECALL "
\$ DEFINE/KEY PF2/TERMINATE "/ALL"
- Then you can accomplish a \$RECALL/ALL command string by pressing the PF1 key followed by the PF2 key
- Deleting keys
\$ DELETE/KEY [keyname]

Define/Key Command Qualifiers

`/ALL`

Removes all key definitions in a specified or current default state.

`/[NO]ECHO`

Determines if the string is displayed on screen. (Commands execute unseen with `NOECHO.`) Default is `/ECHO`.

`/[NO]ERASE`

Determines whether the current line is erased before the key translation is inserted. Default is `/NOERASE`.

`/[NO]LOG`

Controls whether the system displays a message to indicate key definition was successful. Default is `/LOG`.

`/[NO]TERMINATE`

- Specifies whether the current string is to be processed immediately following the pressing of the key.
- That is, do you need a `<ret>`
- `/noterminate` allows you to press other keys before the equivalence string is processed. Default is `/NOTERMINATE`.

Learning Check

1. What is the purpose of symbols?

.....

.....

.....

2. What is the difference between local and global symbol tables?

.....

.....

.....

3. Symbols used for computational purposes are called:

.....

.....

4. You can define keys at the DCL prompt to be equivalent to:

.....

.....

Introduction to Command Procedures

Module 10

Objectives

A command procedure is a file that contains DCL commands and data lines used by DCL commands. DCL provides many features that allow you to write simple or very complex command procedures.

This module discusses techniques used to develop command procedures. Pipes are also introduced in this module. After completing this module, you will be able to:

- Describe the structure of command procedures.
- Create a personal LOGIN.COM command procedure.
- List various techniques for controlling I/O within a command procedure.

Topics

- Rules for Developing Command Procedures
- Introductory Examples
- LOGIN.COM
- Pipes

Command Procedures

Command procedures are files containing DCL commands.

- Command procedures are run using *@command_procedure_filename*.
- The @ character automatically appends a .COM file type on the end of the command procedure file name if no file type is specified.

Use command procedures to:

- Provide a simpler, shorter method for running complex commands.
- Execute a set of commonly used commands in order.

At login, a command procedure normally called LOGIN.COM is run automatically. LOGIN.COM can be used to perform any tasks you always want to occur at login, such as:

- Defining symbols
- Defining logical names
- Setting terminal characteristics

Other features:

- Input parameters can be used to generalize command procedures.
- Command procedures can be written in a more sophisticated fashion, supporting basic programming methods.
- Command procedures can be submitted to run a sequence of commands in batch mode.

WRITE Command

Command Procedure Output

A command procedure can display output to your screen, using:

```
$ WRITE SYS$OUTPUT Expression
```

where *Expression* can be a comma-delimited combination consisting of:

- Symbol names
- Character strings in quotation marks (" ")
- Literal numeric values
- A lexical function

Example

```
$ name="Ellis"  
$ write sys$output "Hello, ",name  
Hello, Ellis  
$
```

Command Procedure Usage Example

The following complex command was issued earlier in the course. We did not discuss the details of the command then and will not here. For now, note:

- The command syntax is long and easy to forget.
- In fact, we do not even know all of the details of the command at this point in the course.
- The user name “BAE” could be replaced by another user name to determine the given user’s UIC.

```
$ write sys$output f$fao("!%J",f$identifier("BAE","NAME_TO_NUMBER"))  
[42,1]
```

Take the command and put it into a command procedure.

```
$ create my_uic.com  
$ write sys$output  
    f$fao("!%J",f$identifier("BAE","NAME_TO_NUMBER"))
```

A Ctrl/Z was entered here.

```
Exit  
$  
$ type my_uic.com  
$ write sys$output  
    f$fao("!%J",f$identifier("BAE","NAME_TO_NUMBER"))  
$
```

Execute the command procedure

```
$ @my_uic  
[42,1]  
$
```

The previous command procedure is fine, but if you want to determine the UIC of another user, you need to write another command procedure. Instead of writing another command procedure, you could generalize the existing one using *positional parameters*, just as DCL commands do.

- DCL creates eight positional parameter symbols when a command procedure is executed. They are named **P1**, **P2**, **P3**, ... , **P8**.
- The symbols correspond to white space delimited parameters on the command line after the command procedure file name.
- If a positional parameter has not been passed, its symbol is initialized to the null string ("").
- Unless the parameter values are quoted, DCL converts the symbol values to uppercase.

Now generalize the command procedure, using the P1 parameter symbol in place of the user name.

```
$ create uic.com
$ write sys$output f$fa0("!%U",f$identifier(P1,"NAME_TO_NUMBER"))
```

A Ctrl/Z was entered here.

```
Exit
```

Now run the command procedure, using the P1 parameter to designate the user name.

```
$ @uic bae
[42,1]
$ @uic system
[1,4]
$ @uic mje
[42,2]
$ @uic kjm
[52,10]
$
```

Positional Parameter Symbol Example

The command procedure displays positional parameters passed to it from the command line.

```
$ type show_positional.com
$ SHOW SYMBOL P1
$ SHOW SYMBOL P2
$ SHOW SYMBOL P3
$ SHOW SYMBOL P4
$ SHOW SYMBOL P5
$ SHOW SYMBOL P6
$ SHOW SYMBOL P7
$ SHOW SYMBOL P8
```

Sample run with three parameters.

```
$ @show_positional Kilroy was here
P1 = "KILROY"
P2 = "WAS"
P3 = "HERE"
P4 = ""
P5 = ""
P6 = ""
P7 = ""
P8 = ""
```

Maintaining case sensitivity.

```
$ @show_positional "Bruce Ellis" "was" "here" "or" "there"
P1 = "Bruce Ellis"
P2 = "was"
P3 = "here"
P4 = "or"
P5 = "there"
P6 = ""
P7 = ""
P8 = ""
```

Too many parameters.

```
$ @show_positional 1 2 3 4 5 6 7 8 9
%DCL-W-DEFOVF, too many command procedure parameters - limit to
eight
$
```

Command Procedure Development

1. Design it.
 - Determine the tasks the procedure should perform.
 - Format of the output, if any.
2. Create it.
 - Use any editor.
 - Specify a .COM file type.
3. Test it.
 - `$ @COMMAND_PROCEDURE .`
 - `$ SET VERIFY` to debug.
4. Fix it.
 - Edit to correct coding errors.
5. Comment it !

Conventions and Rules

- Begin each command line, comment line, and label with a dollar sign (\$).
- Use exclamation mark (!) as a delimiter for comments.
- Use complete names for commands and qualifiers.
- A label is a string, ending with a colon (:), that can be used to change the flow of execution of a command procedure, through a \$GOTO command.
Labels mark branch locations within a procedure and should be placed on a line by themselves.
- An \$EXIT command terminates execution of the command procedure.
- Separate blocks of code with a dollar sign (\$) followed by an exclamation mark (!).
- SYS\$INPUT points to the command procedure for default program input in a command procedure.

Do not begin data lines with a dollar sign.

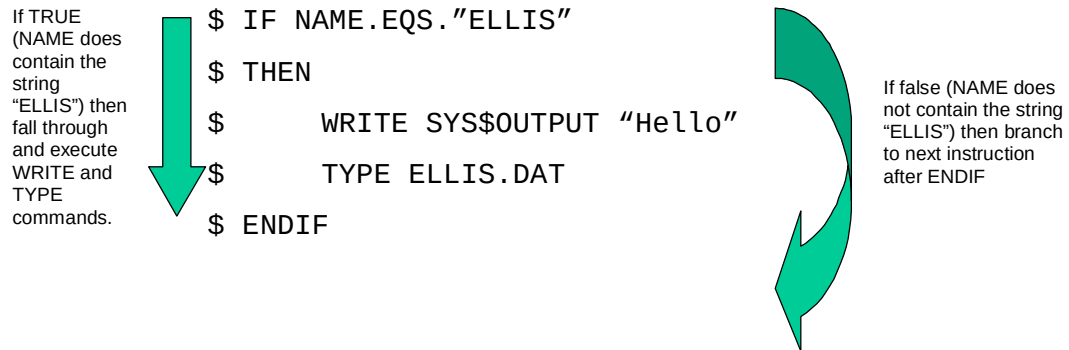
Example

```
$
$ type cleanup.com
$!!!!!!!
$! Command procedure to cleanup login directory
$!!!!!!!
$!
$ purge/keep=3 sys$login: ! Keep only the three most recent versions of
  files
$ delete sys$login:*.obj;*      ! Discard object files
$
$ @cleanup
$
```

Conditional Execution

When writing a LOGIN.COM command procedure, there may be some commands, like \$SET TERMINAL, that you only want to execute if you are logging in interactively. To determine whether you are running interactively you need to be able to understand:

- Basic conditional execution
- The lexical function F\$MODE ()



- The .EQS. syntax is a relational operator that compares the string/symbol on the right for equality with the string/symbol on the left. If they are equal, the expression evaluates as true.
- The .EQS. equality test is case-sensitive.
- The \$ IF statement accepts several relational operators, for now note that .NES. checks for two strings being not equal.

Determining Mode of Execution

F\$MODE()

Lexical functions are callable interfaces to DCL that return a symbol value.

- Lexical function can be used anywhere you might normally use a symbol in an expression.
- Lexical functions always have an F\$ prefix.
- Lexical functions accept parameters in a parenthesized suffix.
- The lexical function F\$MODE () returns one of the following strings, identifying your current mode of execution:
 - INTERACTIVE
 - BATCH
 - NETWORK
 - OTHER

Examples

```
$ write sys$output f$MODE()
INTERACTIVE
$
Common mistake
$ type bad_test.com
"Interactive" should be "INTERACTIVE"!
$ if F$MODE().EQS."Interactive"
$ then
$     write sys$output "Running interactively."
$     exit
$
$ endif
$ write sys$output "NOT Interactive"
$
$ @bad_test
NOT Interactive
$
```

LOGIN.COM

LOGIN.COM is executed automatically by OpenVMS when you log in.

Characteristics

- Must be named `login.com` (unless name is redefined by system manager in UAF).
- Must be in your `sys$login` directory.
- Typically contains commands to customize your environment.
 - Create symbols
 - Create logical names
 - Set terminal characteristics
 - Define keys

Considerations for LOGIN.COM

- Symbols defined in LOGIN.COM should be defined as global symbols so they remain valid after `login.com` finishes.
- Defining job logical names, where appropriate, enhances the time it takes to execute a \$SPAWN command.

Sample LOGIN.COM

The following is a relatively sophisticated login command procedure. The lexical functions F\$PARSE and F\$GETJPI are used. See the “Extending Command Procedures” module for more details. This example also shows how to create a logout command procedure.

```
$ type login.com
$!!!!!!!!!!!!!!!!!!!!
$!      Sample login.com
$!!!!!!!!!!!!!!!!!!!!
$
$!Define Logical names
$ DEFINE/JOB EVE$KEYPAD EDT
$!Define Logical names
$ DEFINE/JOB EVE$KEYPAD EDT
$ DEFINE/JOB COMS      SYS$SYSDEVICE:[BAE.COM]
$ DEFINE/JOB SYS$SCRATCH SYS$SYSDEVICE:[BAE.SCRATCH]
```

The lexical function F\$PARSE returns the full file specification if the file name in the first parameter exists, the null string otherwise.

```
$ !If [.com] subdirectory does not exists create it
$ IF F$PARSE("SYS$DISK:[BAE.COM]").EQS.""
$ THEN
$     CREATE/DIRECTORY SYS$DISK:[BAE.COM]
$     WRITE SYS$OUTPUT "Created [BAE.COM]"
$ ENDIF
$ !If [.com] subdirectory does not exists create it
$ IF F$PARSE("SYS$DISK:[BAE.SCRATCH]").EQS.""
$ THEN
$     CREATE/DIRECTORY SYS$DISK:[BAE.SCRATCH]
$     WRITE SYS$OUTPUT "Created [BAE.SCRATCH]"
$ ENDIF
$
```

```
$ MORE=="TYPE/PAGE=SAVE SYS$INPUT"
$ CL*EAR=="TYPE/PAGE NL:"
$ LO*GOUT=="@COMS:LOGOUT"
$
$!Define keys
$ DEFINE/KEY PF1 "CLEAR/TERMINATE"
$
$!Set terminal characteristics
$ IF F$MODE().EQS."INTERACTIVE"
$ THEN
$     SET TERMINAL/INQUIRE/NONUMERIC
$ ENDIF
$
```

This command procedure is invoked whenever \$LOGOUT is entered, based on the symbol we defined in LOGIN.COM.

```
$ type logout.com
$!!!!!!!!!!!!!!
$! Sample logout command procedure
$!!!!!!!!!!!!!!
$! If we spawned a subprocess do not cleanup
```

If we are a subprocess, then we are deleting our copy of the LOGOUT symbol, not the parent's. This deletion is necessary to make sure that we do not attempt to run the command procedure recursively.

```
$ DELETE/SYMBOL/GLOBAL LOGOUT
```

If we are a subprocess, for instance created through a SPAWN command, we do not want to clean up. The F\$GETJPI lexical function, using the "OWNER" argument will return our creator's process id. If we are a detached process, i.e. the one that logged in, it returns a null string.

```
$ IF F$GETJPI("", "OWNER").NES. ""
$ THEN
$     LOGOUT
$ ENDIF
$ WRITE SYS$OUTPUT "Cleaning up"
```

This command changes the default directory of the process that invoked the command procedure. That is okay, since we are going away. If you want to save the previous directory, use F\$ENVIRONMENT("DEFAULT").

```
$ SET DEFAULT SYS$LOGIN
$ PURGE/KEEP=3 [...]
$ DELETE *.OBJ;*
$ LOGOUT
$
$
```

Test it out.

```
$ create/dir [.com]
$ copy logout.com [.com]
$
$ @login
Created [BAE.SCRATCH]
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$
$
```

The symbols have been defined.

```
$ show symbol/global/all
$RESTART == "FALSE"
$SEVERITY == "1"
$STATUS == "%X10000001"
CL*EAR == "TYPE/PAGE NL:"
LO*GOUT == "@COMS:LOGOUT"
MORE == "TYPE/PAGE=SAVE SYS$INPUT"
$
```

Test out logging out of a subprocess.

```
$ spawn
%DCL-S-SPAWNED, process BAE_1 spawned
%DCL-S-ATTACHED, terminal now attached to process BAE_1
$ lo
    Process BAE_1 logged out at 31-MAR-1999 21:23:20.40
%DCL-S-RETURNED, control returned to process BAE
```

Test out logging out of a detached process.

```
$ lo
Cleaning up
Session Closed (0)
Hit ENTER To Restart Session
```

PIPE Command

- The PIPE command was introduced in Version 7.1.
- The PIPE command provides several levels of UNIX shell functionality, including:
 - Support for the pipe (|) character as a method of sending SYS\$OUTPUT from one command as SYS\$INPUT to the next command.
 - Support for a “sub-shell” environment, which does not affect the current process environment.
 - The logical or (| |) and logical and (&&) characters.
 - Non-newline terminators (;).

PIPE Examples

```
$ more == "type/page=save sys$input"
```

```
$
```

Displays the output of show system one page at a time.

```
$ pipe sh sys|more
```

```
OpenVMS G7.1 on node ELLIS 3-JUN-1997 04:43:15.44 Uptime 0 04:26:25
```

Pid	Process Name	State	Pri	I/O	CPU	Page flts	Pages
00000041	SWAPPER	HIB	16	0	0 00:00:06.75	0	0
00000082	_FTA3:	HIB	5	5174	0 00:00:52.44	9075	55
00000044	LANACP	HIBO	13	--	swapped out --		27
00000046	IPCACP	HIB	10	10	0 00:00:00.03	32	20
00000047	ERRFMT	HIBO	8	--	swapped out --		21
00000048	OPCOM	HIBO	6	--	swapped out --		22
00000049	AUDIT_SERVER	HIB	8	90	0 00:00:00.39	497	23
0000004A	JOB_CONTROL	HIB	8	90	0 00:00:00.39	1143	18
0000004B	QUEUE_MANAGER	HIBO	9	--	swapped out --		32
0000004C	SECURITY_SERVER	HIB	10	53	0 00:00:29.20	12541	90
0000004D	DNS\$ADVER	LEFO	5	--	swapped out --		31
0000004E	LES\$ACP_V30	HIBO	8	--	swapped out --		32
0000004F	REMACP	HIBO	8	--	swapped out --		16
00000050	NET\$ACP	HIBO	4	--	swapped out --		24
00000051	NET\$EVD	HIBO	6	--	swapped out --		30
00000052	DTSS\$CLERK	LEFO	12	--	swapped out --		31
00000053	TP_SERVER	HIB	8	1088	0 00:00:08.78	8546	41
00000055	DECW\$SERVER_0	HIB	8	7022	0 00:01:34.00	14933	242
00000056	DTSESSION	LEFO	4	--	swapped out --		27
00000099	BAE_1	CUR	4	30	0 00:00:00.12	104	113 S
0000005A	DTWM	LEF	4	15260	0 00:01:04.87	47534	106 S

```
RETURN/SPACE=More, PREV/NEXT=Scroll, INS/REM=Pan, SELECT=80/132, Q=Quit
```

00000044	LANACP	HIBO	13	--	swapped out --		27
00000046	IPCACP	HIB	10	10	0 00:00:00.03	32	20
00000047	ERRFMT	HIBO	8	--	swapped out --		21
00000048	OPCOM	HIBO	6	--	swapped out --		22
00000049	AUDIT_SERVER	HIB	8	90	0 00:00:00.39	497	23
0000004A	JOB_CONTROL	HIB	8	90	0 00:00:00.39	1143	18
0000004B	QUEUE_MANAGER	HIBO	9	--	swapped out --		32
0000004C	SECURITY_SERVER	HIB	10	53	0 00:00:29.20	12541	90
0000004D	DNS\$ADVER	LEFO	5	--	swapped out --		31
0000004E	LESS\$ACP_V30	HIBO	8	--	swapped out --		32
0000004F	REMACP	HIBO	8	--	swapped out --		16
00000050	NET\$ACP	HIBO	4	--	swapped out --		24
00000051	NET\$EVD	HIBO	6	--	swapped out --		30
00000052	DTSS\$CLERK	LEFO	12	--	swapped out --		31
00000053	TP_SERVER	HIB	8	1088	0 00:00:08.78	8546	41
00000055	DECW\$SERVER_0	HIB	8	7022	0 00:01:34.00	14933	242
00000056	DTSESSION	LEFO	4	--	swapped out --		27
00000099	BAE_1	CUR	4	30	0 00:00:00.12	104	113 S
0000005A	DTWM	LEF	4	15260	0 00:01:04.87	47534	106 S
0000005B	BAE	LEF	4	1128	0 00:00:15.17	9174	36
0000009C	BAE_2	COM	4	8	0 00:00:00.13	99	99 S
0000005F	DECW\$TE_005F	LEF	5	296	0 00:00:54.79	28114	250
00000060	_FTA1:	LEF	5	12875	0 00:00:29.97	13695	41

End-of-File | PREV/NEXT=Scroll, INS/REM=Pan, SELECT=80/132, Q=Quit

\$

The || says do the command on the left and if it fails do the command on the right.

```
$ pipe type non_existent || write sys$output "File not found"
```

```
%TYPE-W-SEARCHFAIL, error searching for
```

```
DKA200:[BAE]NON_EXISTENT.LIS;
```

```
-RMS-E-FNF, file not found
```

```
File not found
```

```
$ sh def
```

```
DKA200:[BAE]
```

The parentheses say that the set default will only be in effect until the close parenthesis is encountered.

```
$ pipe (set def sys$system: ; dir sysuaf.dat) ; sh def
```

```
Directory SYS$COMMON:[SYSEXE]
```

```
SYSUAF.DAT;1
```

```
Total of 1 file.
```

```
DKA200:[BAE]
```

```
$
```

Learning Check

1. What is a command procedure?
.....
.....
.....
2. Name three uses for command procedures.
.....
.....
.....
3. What are the five steps for command procedure development?
.....
.....
.....
4. To customize your OpenVMS environment, you create a command procedure called:
.....
.....
5. What is the purpose of the PIPE command?
.....
.....
.....

Objectives

On occasion, you will want to (or be told to) move some files from the disk to a tape or floppy disk for storage or transport. At larger sites, these activities might be done by an operator. Even so, you should be at least conversant with the options for disk-to-tape activities.

This module discusses techniques and options for disk-to-tape activities. After completing this module, you will be able to:

- Initialize a tape.
- Copy files to and from tape.
- Use BACKUP to store and retrieve your directory hierarchy.

Topics

- Tape Initialization
- Using COPY to and from Tape
- MOUNT
- BACKUP
- Processing Floppies

File Copy from Disk to Tape

If you would like to move files to tape, you can use the COPY command. Tapes are usually protected and you normally need an operator/system manager to perform these operations.

- The COPY command does not replicate the directory tree structure.
- It copies all the files in all your directories into a single area. The COPY command creates an ANSI standard 7-bit ASCII tape.

Steps to Copy a Tape

1. Acquire the tape drive for exclusive access and enter:
\$SHOW DEVICE MT/FULL
\$ALLOCATE MTA0: TDRIVE
2. Initialize the tape (write an ANSI header and label to it).
\$INITIALIZE MTA0: DENNIE



Note

Do this only if the tape is new or you do not need the previous information on it.

3. Physically mount the tape on the drive, or request that the operator do it for you, by entering:
\$REQUEST/REPLY
\$MOUNT MTA0: DENNIE
4. Now write files to tape.
\$COPY *.* MTA0:

Further Tape Activities

1. To view the contents of the tape, enter:
`$DIRECTORY MTA0:`
You should see the names of the files that are stored on the tape.
5. To unload the tape and make the drive available for other MOUNT commands by your process, enter:
`$DISMOUNT MTA0:`
Dismounting the tape causes it to rewind.
6. After you are done with the tape drive, you can make it available to others by deallocating it as follows:
`$DEALLOCATE MTA0:`
`$DEASSIGN TDRIVE`
This deletes the logical name that was never used.

To restore the files later on, repeat the above steps, with the following exceptions:

- Do not initialize the tape!
- Change the COPY command to: `$COPY MTA0:*. *;* *. *;*`

User-Level Backup

The BACKUP utility is a tool normally used by the system manager to perform backup and restore operations on entire devices.

The BACKUP utility may be a valuable, general-purpose tool for users, because it replicates directory hierarchies.

Replicating a Directory Hierarchy

```
$ dir [bae_too]
%DIRECT-W-NOFILES, no files found
$ show default
  ELLIS$DKA200:[BAE]
$ dir [.com...]

Directory ELLIS$DKA200:[BAE.COM]

CALM.DIR;1          LOGOUT.COM;3

Total of 2 files.

Directory ELLIS$DKA200:[BAE.COM.CALM]

STORM.DAT;1

Total of 1 file.

Grand total of 2 directories, 3 files.
$
```

Replicate the [.COM] directory and its subdirectories under the directory [BAE_TOO].

```
$ backup [.com...]*.* [bae_too.com...]
```

Verify the copy.

```
$ dir [bae_too...]
```

```
Directory ELLIS$DKA200:[BAE_TOO]
```

```
COM.DIR;1
```

```
Total of 1 file.
```

```
Directory ELLIS$DKA200:[BAE_TOO.COM]
```

```
CALM.DIR;1          LOGOUT.COM;3
```

```
Total of 2 files.
```

```
Directory ELLIS$DKA200:[BAE_TOO.COM.CALM]
```

```
STORM.DAT;1
```

```
Total of 1 file.
```

```
Grand total of 3 directories, 4 files.
```

```
$
```

Backups to Tape

When writing to tape, the BACKUP utility replicates your directory tree structure in a container file called a save set.

- Only BACKUP can interpret a save set.
- Typically, you will use .BCK as the save set file type, although BACKUP does not support a default file type.
- BACKUP assumes that anything on tape is a SAVE_SET and anything on disk is not.
- If you want to write a save set to disk, you can use the /SAVE_SET qualifier.

The steps for writing a backup save set to tape are the same, with the following exceptions:

- Initialize the tape with a label that matches the first six characters of the backup save set, or use the /IGNORE=LABEL_PROCESSING qualifier on the BACKUP command.
- Identify that this is not an ANSI tape.
`$MOUNT/FOREIGN MTA0:`
- The BACKUP command replaces the COPY command.
`$BACKUP/LOG [...]*. *;* MTA0:AUGUST.BCK/SAVE_SET`
- To display the contents of the save set, enter:
`$BACKUP/LIST MTA0:AUGUST.BCK`
- To restore the entire directory, enter:
`$BACKUP/LOG MTA0:AUGUST.BCK/SAVE_SET [*...]`
- To restore a specific file, enter:
`$BACKUP/LOG-
/select=([.x_files]mulder.dat, [.x_files]scully.dat) -
MTA0:AUGUST.BCK/SAVE_SET [*...]`

Processing Floppy Disks

You may want to copy files to a floppy disk to move them between systems. Although you will probably not be privileged enough to directly access the floppy, you may be able to convince the system manager to set up an ACL that will allow you to access it.

The general steps for accessing a floppy are the same as a tape, with the following exceptions:

- A floppy is a disk and when it is initialized a Files-11 file structure is imposed on it.
- Directories can be directly copied to floppy.

Sample Floppy Copy

Find the floppy.

```
$ sh dev dka500/full
```

Disk ELLIS\$DKA500:, device type RX26, is online, file-oriented device,
shareable, available to cluster, error logging is enabled.

Error count	0	Operations completed	0
Owner process	""	Owner UIC	[SYSTEM]
Owner process ID	00000000	Dev Prot	S:RWPL,0:RWPL,G:R,W
Reference count	0	Default buffer size	512

A system manager or suitably privileged user would have to set up the security
ACL.

```
$ set proc/priv=all
```

```
$ set security/class=device/acl=(ident=bae,access=read+write+physical+logical) -  
_$ dka500
```

```
$
```

```
$ sh dev dka500/full
```

Disk ELLIS\$DKA500:, device type RX26, is online, file-oriented device,
shareable, available to cluster, error logging is enabled.

Error count	0	Operations completed	0
Owner process	""	Owner UIC	[SYSTEM]
Owner process ID	00000000	Dev Prot	S:RWPL,0:RWPL,G:R,W
Reference count	0	Default buffer size	512

Device access control list:

(IDENTIFIER=[BRUDEN,BAE],ACCESS=READ+WRITE+PHYSICAL+LOGICAL)

```
$
```

Go back to being nonprivileged.

```
$ set proc/priv=(noall,tmpmbx,netmbx)
$
```

Claim ownership of the floppy.

```
$ allocate dka500
%DCL-I-ALLOC, _ELLIS$DKA500: allocated
$ sh dev dka500
```

Device	Device	Error	Volume	Free	Trans	Mnt
Name	Status	Count	Label	Blocks	Count	Cnt
ELLIS\$DKA500:	OnLine alloc	0				

Initialize the floppy as “high density” with a label of x_files.

```
$ init/dens=hd dka500 x_files
$
```

Make the floppy available for processing.

```
$ mount dka500 x_files
```

```
%MOUNT-I-MOUNTED, X_FILES mounted on _ELLIS$DKA500:
```

```
$ show device/full dka500
```

Disk ELLIS\$DKA500:, device type RX26, is online, allocated, mounted, file-oriented device, shareable, available to cluster, error logging is enabled.

Error count	0	Operations completed	76
Owner process	"BAE"	Owner UIC	[SYSTEM]
Owner process ID	0000012A	Dev Prot	S:RWPL,O:RWPL,G:R,W
Reference count	2	Default buffer size	512
Total blocks	2880	Sectors per track	18
Total cylinders	80	Tracks per cylinder	2

Device access control list:

(IDENTIFIER=[BRUDEN,BAE],ACCESS=READ+WRITE+PHYSICAL+LOGICAL)

Volume label	"X_FILES"	Relative volume number	0
Cluster size	1	Transaction count	1
Free blocks	2849	Maximum files allowed	720
Extend quantity	5	Mount count	1
Mount status	Process	Cache name	"_ELLIS\$DKA200:XQPCACHE"
Extent cache size	64	Maximum blocks in extent cache	284
File ID cache size	64	Blocks currently in extent cache	0
Quota cache size	0	Maximum buffers in FCP cache	820
Volume owner UIC	[BRUDEN,BAE]	Vol Prot	S:RWCD,O:RWCD,G:RWCD,W:RWCD

Volume Status: ODS-2, subject to mount verification, file high-water marking, write-back caching enabled.

\$

Create a UFD.

```
$ create/directory dka500:[bae]
```

```
$
```

Copy over the [.com] directory.

```
$ backup [.com...] dka500:[bae.com...]
```

```
$ dir dka500:[bae...]
```

Directory DKA500:[BAE]

COM.DIR;1

Total of 1 file.

Directory DKA500:[BAE.COM]

CALM.DIR;1 LOGOUT.COM;3

Total of 2 files.

Directory DKA500:[BAE.COM.CALM]

STORM.DAT;1

Total of 1 file.

Grand total of 3 directories, 4 files.

```
$
```

```
$ dir [.vism_labs...]
```

```
Directory ELLIS$DKA200:[BAE.VISM_LABS]
```

BIG_CODE.EXE;2	BIG_CODE.EXE;1	BIG_CODE.MAR;2	BIG_CODE.MAR;1
BIG_CODE.OBJ;2	BIG_CODE.OBJ;1	BIG_DATA1.C;2	BIG_DATA1.C;1
BIG_DATA1.EXE;2	BIG_DATA1.EXE;1	BIG_DATA1.OBJ;2	BIG_DATA1.OBJ;1
BIG_DATA2.C;2	BIG_DATA2.C;1	BIG_DATA2.EXE;2	BIG_DATA2.EXE;1
BIG_DATA2.OBJ;2	BIG_DATA2.OBJ;1	GEN_LOW_PRI.COM;2	GEN_LOW_PRI.COM;1
IO_BLASTER.C;2	IO_BLASTER.C;1	IO_BLASTER.EXE;2	IO_BLASTER.EXE;1
IO_BLASTER.OBJ;2	IO_BLASTER.OBJ;1	LAB_SETUP.COM;4	LAB_SETUP.COM;3
LOCK2.C;2	LOCK2.C;1	LOCK2.EXE;2	LOCK2.EXE;1
LOCK2.OBJ;2	LOCK2.OBJ;1	LOCK_FOREVER.C;2	LOCK_FOREVER.C;1
LOCK_FOREVER.EXE;2	LOCK_FOREVER.EXE;1	LOCK_FOREVER.OBJ;2	LOCK_FOREVER.OBJ;1
LOOPER.C;2	LOOPER.C;1	LOOPER.EXE;4	LOOPER.EXE;3
LOOPER.OBJ;4	LOOPER.OBJ;3	MUTT.C;2	MUTT.C;1
MUTT.EXE;2	MUTT.EXE;1	MUTT.OBJ;2	MUTT.OBJ;1
SETUP_RWAST.COM;2	SETUP_RWAST.COM;1	STUB.C;2	STUB.C;1
STUB.OBJ;4	STUB.OBJ;3	SY\$OUTPUT.;1	TIMER.C;2
TIMER.C;1	TIMER.EXE;2	TIMER.EXE;1	TIMER.OBJ;2
TIMER.OBJ;1			

```
Total of 65 files.
```

```
$
```

Back up the [.vism_labs] directory to a save set, without the .OBJ and .EXE files.

```
$ backup/exclude=(*.obj,*.exe) [.vism_labs...]*.*; -
_$ dka500:[bae]vism_labs.bck/save
```

Examine the save set.

```
$ backup/list dka500:[bae]vism_labs.bck/save
Listing of save set(s)
```

```
Save set:          VISM_LABS.BCK
Written by:        BAE
UIC:              [000042,000001]
Date:             31-MAR-1999 23:03:12.93
Command:          BACKUP/EXCLUDE=(*.OBJ,*.EXE) [.VISM_LABS...]*.*; DKA500:[BAE]
VISM_LABS.BCK/SAVE
Operating system:  OpenVMS Alpha version V7.2
BACKUP version:    V7.2
CPU ID register:   80000000
Node name:         _ELLIS::
Written on:        _ELLIS$DKA500:
Block size:        32256
Group size:        10
Buffer count:      56
```

```
[BAE.VISM_LABS]BIG_CODE.MAR;2      1  18-MAR-1999 01:43
[BAE.VISM_LABS]BIG_DATA1.C;2       1  18-MAR-1999 01:44
[BAE.VISM_LABS]BIG_DATA2.C;2       1  18-MAR-1999 01:45
[BAE.VISM_LABS]GEN_LOW_PRI.COM;2    1  18-MAR-1999 01:42
[BAE.VISM_LABS]IO_BLASTER.C;2      4  18-MAR-1999 01:45
[BAE.VISM_LABS]LAB_SETUP.COM;4     15 18-MAR-1999 01:49
```

```

[BAE.VISM_LABS]LOCK2.C;2                2  18-MAR-1999 01:42
[BAE.VISM_LABS]LOCK_FOREVER.C;2         2  18-MAR-1999 01:42
[BAE.VISM_LABS]LOOPER.C;2               1  18-MAR-1999 01:42
[BAE.VISM_LABS]MUTT.C;2                 2  18-MAR-1999 01:47
[BAE.VISM_LABS]SETUP_RWAST.COM;2         1  18-MAR-1999 01:47
[BAE.VISM_LABS]STUB.C;2                 1  18-MAR-1999 01:44
[BAE.VISM_LABS]SYS$OUTPUT.;1             0  18-MAR-1999 01:33
[BAE.VISM_LABS]TIMER.C;2                2  18-MAR-1999 01:46

```

Total of 14 files, 34 blocks

End of save set

\$

Examine a file that was backed up.

\$ type [.vism_labs]stub.c

```
void    stub(int x)
```

```
{
```

```
}
```

\$

Oops! It is gone.

\$ delete [.vism_labs]stub.c;*

\$

\$ type [.vism_labs]stub.c

%TYPE-W-SEARCHFAIL, error searching for ELLIS\$DKA200:[BAE.VISM_LABS]STUB.C;

-RMS-E-FNF, file not found

\$

Recover the file.

```
$ backup/sele=stub.c dka500:[bae]vism_labs.bck/save [bae.vism_labs]
```

```
$ type [.vism_labs]stub.c
```

```
void stub(int x)
```

```
{
```

```
}
```

```
$
```

Make the floppy available.

```
$ dismount dka500
```

```
$ deal dka500
```

```
$ show device dka500
```

Device	Device	Error	Volume	Free	Trans	Mnt
Name	Status	Count	Label	Blocks	Count	Cnt
ELLIS\$DKA500:	OnLine	0				

```
$
```

Learning Check

1. When is it appropriate to initialize a tape?
.....
.....
2. The COPY command replicates the directory structure.
☐ True
☐ False
3. When moving files from disk to tape, the BACKUP command replicates the directory structure.
☐ True
☐ False
4. The BACKUP utility uses a container file called a:
.....

Queues: Batch and Print

Module 12

Objectives

Some OpenVMS operations are unwieldy to perform while you wait for a terminal prompt. To automate and ease the burden of time-consuming jobs, OpenVMS has queues: batch queues for execution requests and print queues for printing requests. Batch and print queues are implemented in OpenVMS as entries in a special file.

This module discusses queues and their uses. After completing this module, you will be able to:

- Submit requests to batch and print queues.
- Describe the commonly used qualifiers for batch and print requests.

Topics

- Queue Concepts
- Batch Processing
- SUBMIT
- PRINT

Queues

Queues are waiting lists that control access to printers or batch processors.

- Queues have characteristics (set up by the system manager).
- Jobs executing off a queue take on the characteristics assigned to that queue.
- A batch job runs off a batch queue. It executes the commands in a command procedure running as a detached (batch) process.

```
$SUBMIT UPDATE.COM
```

```
job 124 submitted to SYS$BATCH
```

- You can view queue activities through:

```
$SHOW QUEUE SYS$BATCH
```

or

```
$SHOW QUEUE SYS$PRINT
```

Batch Queues

`$SUBMIT/NOTIFY UPDATE.COM`

- Means submit this request as a batch job and let it take the characteristics of the batch queue. Some characteristics that could be different are priority, process permanent logical names, and working set constraints.
- By default, all output is directed to a file called `UPDATE.LOG` which is automatically printed and then deleted unless you use the `/KEEP` qualifier.

Commonly Used Qualifiers

`/NOTIFY`

 Informs you when the job is done if are logged on.

`/PARAMETERS=(P1,P2,...,P8)`

 Passes parameters to a job.

`/PRINTER`

 Specifies where to print the log file.

`/NOKEEP`

 Deletes the log file.

`/HOLD`

 Job will not become a run candidate until the job is released.

`/AFTER= 18:00`

 Job will not become a run candidate until after 6:00 p.m.

`/QUEUE=fastbatch`

 Sends to an alternate queue.

`$SET QUEUE/ENTRY= ... /RELEASE SYS$BATCH`

“Unholds” a job.

`$DELETE/ENTRY =... SYS$BATCH`

Cancels a job submission.

`$SET QUEUE/ENTRY= ... /REQUE FASTBATCH SYS$BATCH`

Changes queues.

`$SET QUEUE/ENTRY= ... /AFTER=20:00 SYS$BATCH`

Changes the starting time of a submitted job.



Note

A batch job logs into your account, executes your `LOGIN.COM` (`SYLOGIN` too), runs the job, and produces the log file which is normally printed and deleted. Use the `/NOPRINTER` qualifier to block automatic printing of the batch log file.

Sample Command Procedure Using Parameters

Command procedure accepts a file name to compile, link, and run and a privilege list parameter.

```
$ type build.com
$if P2.nes.""
$then
$      set process/priv='P2'
$endif
$cc 'P1'
$link 'P1'
$run 'P1'
```

Submit the job.

```
$ submit/param=(image_count,world)/noprint build
Job BUILD (queue LOW_RIDER, entry 20) started on LOW_RIDER
$ sh sys/batch
```

```
OpenVMS V7.2  on node ELLIS   1-APR-1999 00:18:44.83  Uptime  0 05:30:10
  Pid   Process Name      State Pri    I/O      CPU      Page flts  Pages
00000137 BATCH_20          LEF    5     77    0 00:00:00.60      337   207  B
$
```

View the log file.

```
$ type build.log
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
Process: SWAPPER          PID: 00000101  Image count:    0

Process: LANACP           PID: 00000106  Image count:    0

Process: FASTPATH_SERVER  PID: 00000108  Image count:    0

Process: IPCACP           PID: 00000109  Image count:    0

...
```

...

Process: DECW\$SERVER_0 PID: 0000011f Image count: 0

Process: DTSESSION PID: 00000120 Image count: 1

Process: BAE PID: 0000012a Image count: 110

Process: DTWM PID: 0000012d Image count: 1

Process: DECW\$TE_0130 PID: 00000130 Image count: 1

Process: _FTA1: PID: 00000132 Image count: 1

Process: _FTA2: PID: 00000133 Image count: 1

Process: _FTA3: PID: 00000134 Image count: 11

Process: BATCH_19 PID: 00000136 Image count: 4

BAE job terminated at 1-APR-1999 00:17:40.30

Accounting information:

Buffered I/O count:	274	Peak working set size:	15328
---------------------	-----	------------------------	-------

Direct I/O count:	566	Peak virtual size:	186848
-------------------	-----	--------------------	--------

Page faults:	1744	Mounted volumes:	0
--------------	------	------------------	---

Charged CPU time:	0 00:00:14.81	Elapsed time:	0 00:00:21.27
-------------------	---------------	---------------	---------------

\$

Process Permanent Logical Name Contrasts

Logical Name	Function	Interactive	@	Submit
SYSS\$INPUT	Default input	Your keyboard	.COM	.COM
SYSS\$OUTPUT	Default output	Your screen	Your screen	.LOG
SYSS\$ERROR	Default output for error messages	Your screen	Your screen	.LOG
SYSS\$COMMAND	Original location to read DCL commands	Your keyboard	Your keyboard	.COM
SYSS\$LOGIN	Default device and directory	From UAF	From UAF	From UAF

Print Queues

A print queue is a holding tank for files that will eventually be opened and printed by a special process called a print symbiont.

Qualifiers

`$PRINT/DEVICE=TTA5:`

Selects an alternate printer

`/QUEUE=OTHER_PRINT`

Selects an alternate queue

`/NOTIFY, /HOLD, /AFTER=`

Same as batch

`/FLAG_PAGE`

Produces header page for job

`/HEADER`

Prints one-liner with page number

`/DELETE`

Prints then deletes file

`/COPIES=3 A.A, B.B`

A A A B B B

`/JOB_COUNT=3 A.A, B.B`

A B A B A B

`$SHOW QUEUE/DEVICE/ALL/FULL`

Views print queues

Learning Check

1. What is a queue?

.....
.....

2. What command do you use to request a batch job?

.....
.....

3. What command do you use to request a print job?

.....
.....

4. What command allows you to see all queues for your process?

.....
.....

Extending Command Procedures

Appendix A

Objectives

A command procedure is a file that contains DCL commands and data lines used by DCL commands. DCL provides many features that allow the writing of simple or very complex command procedures.

This module discusses techniques used to develop command procedures. After completing this module, you will be able to:

- List various lexical functions and their uses.
- Describe techniques for reading and writing to a sequential file.
- Describe error handling options within command procedures.

Topics

- Reading from a File
- Writing to a File
- Lexical Functions
- Error Handling

Terminal I/O

Terminal I/O from within a command procedure can be used to:

- Display messages and command output on the terminal screen.
- Prompt you for input.
- Redirect terminal output to a file.
- Allow the use of an interactive utility from within a command procedure.

Process permanent logical names for I/O include:

`SYS$COMMAND`

`SYS$INPUT`

`SYS$OUTPUT`

`SYS$ERROR`

UNIX-style pipes may be used with the `PIPE` command.

Terminal Output and Input

The following techniques can be used to send output to the screen:

```
$ WRITE SYS$OUTPUT string
```

```
$ WRITE SYS$OUTPUT symbol
```

```
$ TYPE SYS$INPUT
```

```
.
```

```
.
```

```
.
```

```
TEXT
```

```
.
```

```
.
```

```
.
```

```
$
```

The following techniques can be used to accept user input from the keyboard or command line:

- Inquire example

```
$ INQUIRE symbol "prompt"
```

- Read example

```
$ READ/PROMPT=string SYS$COMMAND symbol
```

- Passing parameters

- Can pass up to eight parameters on command line.
- Passed as string symbols in the symbols: P1, P2, ...P8.
- In this example, P1=="54", P2=="78", P3=="96", and P4 through P8 all equal the null string ("").

```
$ @SUM 54 78 96
```

INQUIRE Example

```
$ TYPE SUM1.COM
```

```
$!
```

```
$! This command procedure prompts the
```

```
$! user for two numbers, sums
```

```
$! them, then outputs the result.
```

```
$!
```

```
$   INQUIRE NUMBER1 "ENTER FIRST NUMBER"
```

```
$!
```

```
$   INQUIRE NUMBER2 "ENTER SECOND NUMBER"
```

```
$!
```

```
$   SUM = F$INTEGER(NUMBER1) + F$INTEGER(NUMBER2)
```

```
$!
```

```
$   WRITE SYS$OUTPUT " THE TWO NUMBERS ENTERED WERE ",NUMBER1, " AND ", NUMBER2
```

```
$!
```

```
$   WRITE SYS$OUTPUT "THE TOTAL IS ",SUM
```

```
$!
```

```
$   EXIT
```

```
$ @sum1
```

```
ENTER FIRST NUMBER: 22
```

```
ENTER SECOND NUMBER: 109
```

```
THE TWO NUMBERS ENTERED WERE 22 AND 109
```

```
THE TOTAL IS 131
```

```
$
```

Example of Using Command Parameters

The following command procedure accepts input, using the parameter symbols P1 and P2.

```
$ TYPE SUM2.COM
$!
$! This program receives 2 numbers
$! passed in as parameters and
$! calculates the sum.
$!
$ SUM = F$INTEGER(P1) + F$INTEGER(P2)
$!
$ WRITE SYS$OUTPUT " THE NUMBERS PASSED IN ARE ",P1, " AND ", P2
$!
$ WRITE SYS$OUTPUT "THEIR TOTAL IS ",SUM
$!
$ EXIT

$ @SUM 81 97
  THE NUMBERS PASSED IN ARE 81 AND 97
  THEIR TOTAL IS 178
$
```

Example of Changing Process Permanent Logical Names

SYSS\$INPUT implicitly reads from a command procedure file. To redirect SYSS\$INPUT to the user keyboard for the next command, use a USER mode logical name.

```
$ EDIT TERM.COM
```

```
$!
```

```
$! This program edits a file, then
```

```
$! changes it's protection and verifies
```

```
$! it was changed.
```

```
$!
```

```
$  INQUIRE FILE "ENTER FILENAME"
```

```
$!This command sets it up so the next command reads from the keyboard  
instead
```

```
$!of the command procedure.
```

```
$  ASSIGN/USER_MODE SYSS$COMMAND SYSS$INPUT
```

```
$!
```

```
$  EDIT/edt 'FILE'
```

```
$!
```

```
$  SET PROTECTION=(S:RwE,O:RwED,G,W) 'FILE'
```

```
$!
```

```
$  DIRECTORY/PROTECTION 'FILE'
```

```
$!
```

```
$  WRITE SYSS$OUTPUT "The file edited was ''FILE''"
```

```
$!
```

```
$  EXIT
```

```
$ @term
ENTER FILENAME: xxx.dat
Input file does not exist
[EOB]
*i
          X marks the spot.
          ^Z
[EOB]
*exit
ELLIS$DKA200:[BAE]XXX.DAT;1 1 line

Directory ELLIS$DKA200:[BAE]

XXX.DAT;1          (RWE,RWED,,)

Total of 1 file.
The file edited was XXX.DAT
$
```

Reading from a File

DCL supports reading records from a file.

Steps to Read a File

1. Open the file.
`$OPEN/READ logical filename`
2. Begin the read loop.
`$READ/END_OF_FILE=CLOSE_IT logical symbol`
3. Process the data.
4. Go to the beginning of the loop if not at end of file (step 2).
5. End the loop with a label matching the `/END_OF_FILE` string and close the file.
`$CLOSE logical`

Example of Reading from a File

```
$ TYPE REMINDER.COM
$ !
$ !      THIS COMMAND PROCEDURE OPENS A FILE FROM WITHIN
$ !      THE COMMAND PROCEDURE, READS IT, AND WRITES THE
$ !      RECORDS TO THE SCREEN.
$ !
$ !
$ OPEN/READ WHAT MESSAGE.TXT
$ BEGIN:
$ READ/END = STOP WHAT RECORD
$ WRITE SYS$OUTPUT " "
$ WRITE SYS$OUTPUT RECORD
$ WRITE SYS$OUTPUT " "
$ GOTO BEGIN
$ STOP:
$ CLOSE WHAT
$ EXIT
$ TYPE MESSAGE.TXT
    CALL MARY
    PAY BILLS
    REVISE LABS FOR INTRO TO VMS
    BEGIN READING TOMMYKNOCKERS
    PUT COMMAND FILES ON LINE
```

\$ @REMINDER

CALL MARY

PAY BILLS

REVISE LABS FOR INTRO TO VMS

BEGIN READING TOMMYKNOCKERS

PUT COMMAND FILES ON LINE

\$

Sample Use of READ with a Scratch File

Error handling and lexical functions will be discussed later.

```
$ type frag.com
$!!!!!!!!!!!!!!
$! Sample command procedure to display a count of the
$! number of mapping pointers in a file, indicating level
$! of fragmentation in the file.
$!!!!!!!!!!!!!!
$!! Create a unique scratch file name, using the PID and IMAGECOUNT
$ tmp_file=" sys$scratch:_frag_."f$getjpi("", "PID")'f$getjpi("", "imagecount")'
$
$!! Dump the header only and count the number of lines containing
$!! the string "LBN:". These lines are mapping pointers. The output
$!! of the search statistics goes to the scratch file.
$ pipe dump/head/bl=c:0 'P1' | search/nooutput/stat sys$input "LBN:" >'tmp_file'
$
$!! If we encounter an error or control Y, close and delete the scratch file
$ on control_y then goto scratch_scratch
$ on error then goto scratch_scratch
$
```

This is where we read the scratch file.

```
$!! Open the scratch file and read and discard the first 4 lines
$ open/read stats 'tmp_file'
$ x=1
$ loop:
$     read stats junk
$     x=x+1
$     if x.le.4 then goto loop
$
$!! The fifth line contains the count of matching lines
$ read stats count
$
```

```
$!! We are done with the scratch file, close and delete it
$ close stats
$ delete 'tmp_file';
$
$!! Reset Control Y and Error processing
$ on control_y then exit
$ on error then goto exit
$
$!! Determine where the count string starts and how big it is
$ start = f$locate(":",count)+1
$ size = f$locate("E",count)-1 - start
$
$!! Get the count and get rid of extraneous whitespace
$ count= f$edit(f$extract(start,size ,count),"COLLAPSE")
$
$!! Display the count
$ write sys$output count
$ exit
$
$!! On error or control Y get rid of the scratch file
$ scratch_scratch:
$ close stats
$ delete 'tmp_file';
$
$
```

Sample runs of command procedure.

\$ @frag indexed.dat

43

\$

\$ @frag sys\$manager:accountng.dat

37

\$

\$ @frag frag.com

1

\$

\$ @frag

_File: SYS\$SYSTEM:PAGEFILE.SYS

60

\$

\$

Writing to a File

DCL supports writing to files through the WRITE command.

The following are methods for opening the file for write access:

- \$OPEN/WRITE logical-name filename
- \$OPEN/APPEND
- \$OPEN/SHARE
- \$OPEN/READ/WRITE

Steps to Write to a File

1. Begin the write loop.
2. Read the data to be written, until done then go to step 6, using:
 \$INQUIRE
 or
 \$READ/PROMPT=
3. Test the data.
4. Write the data to the file, using:
 \$ WRITE logical string_or_symbol
5. Go to the beginning of the loop.
6. End the loop and close the file.

Relational Operators

Relational operators are used with \$IF statements to perform conditional operations.

String Operators

Operator	Function
.EQS.	Tests if 2 character strings are equal.
.GES.	First string >= the second string?
.GTS.	First string > second string?
.LES.	First string <= second string?
.LTS.	First string < second string?
.NES.	Two strings not =?

Numeric Operators

Operator	Function
.EQ.	Two numbers equal?
.GE.	First number >= second?
.GT.	First number > second?
.LE.	First number <= second?
.LT.	First number < second?
.NES.	Two strings not =?

Logical Operators

Operator	Function
.NOT.	Tests for opposite of a condition.
.AND.	Both of two conditions true?
.OR.	Tests if any of a group of conditions have been met.

Controlling Code Flow

The IF command tests the value of an expression and causes different commands to execute if the expression is true vs. when it is false.

One-line conditional format

```
$ IF condition THEN command
```

Block conditional format

```
$ IF condition
$ THEN
$ ...
$ commands
$ ...
$ ENDIF
```

If then else format

```
$ IF condition
$ THEN
$      ...
$ commands
$      ...
$ ELSE
$      ...
$ commands
$      ...
$ ENDIF
```

Loops

Loops are used to repeat a sequence of steps a fixed number of times or until some condition terminates the loop.

Steps to Code a Loop

1. Begin the loop.
2. Change the terminator variable.
3. Test the terminator variable.
4. Perform commands in the body of the loop.
5. Return to beginning of loop.
6. End the loop.

Example Loop

```
$ COUNT = 1
$ LOOP:
$
$ !body of loop
$
$ COUNT = COUNT + 1
$ IF COUNT .LT. 11 THEN GOTO LOOP
```

Simple Loop

```
$ type countdown.com
$ X=10
$loop:
$  write sys$output x
$  if x.eq.3 then write sys$output "Ignition..."
$  wait ::1.0
$  x=x-1
$  if x.ge.1 then goto loop
$write sys$output "Liftoff!"
$ @countdown
10
9
8
7
6
5
4
3
Ignition...
2
1
Liftoff!
$
```

Sample Loop to Delete Directory Tree

```
$ TYPE DEL_DIR.COM
$! Simple example of deleting a directory tree
$! It will accept directory names of the form:
$! [UFD] or [.SFD]. Needs more error checking.
$! If no directory is specified exit
$ if P1 eqs. ""
$ then
$     write sys$error "Invalid format use: @del_ufd
directory
$ endif
$
$! Gotta have a "Are you sure" query
$ inquire/nopunc response "Are you sure you want to delete
''P1'? "
$ if .not.response then exit
$
$! Build a symbol of the form [dir...]
$ all_levels=p1-"]"+"...]"
$
$! Enable delete access for all files in tree
$ set file/protection=(w:rwed) 'all_levels'*..*;*
$!
$! Keep running if error, but do not display error messages
$! We will loop and repeatedly delete all files below the
directory
$! until it is empty. If the directory is non-empty we will
get the status
$! value %X109311F0 in $STATUS
$ set noon
$ set message/noident/noseverity/notext/nofacility
```

```
$ del_loop:
$      delete 'all_levels'*..*;*
$      if $status.eqs."%X109311F0" then goto del_loop
$ set mess/ident/severity/text/facility
$! If any other errors, terminate
$ if .not.$status then exit $status
$! Reenable error processing
$ set on
$
$! Determine the directory name
$ dir_name=p1-"["-"]"+"dir;1"
$
$! If leading "." get rid of it.
$ if f$locate(".",dir_name).eq.0 then dir_name=dir_name-"."
$
$! Delete the directory
$ set file/protection=(w:rwed) 'dir_name'
$ delete 'dir_name'
$
$ create/dir [.x.y.z.a.b.c.d.e.f.g.h.i.j.k.l]
$ @del_dir [.x]
Are you sure you want to delete [.X]? y
$ dir x.dir
%DIRECT-W-NOFILES, no files found
$
```

Subroutines

A non-structured subroutine can be entered using the `$ GOSUB` command. This subroutine shares the context of the caller including all local symbols.

- `GOSUB` transfers control to a labeled subroutine.
- `Return` terminates a subroutine.
- It is a local subroutine call meaning that it does not create a new command level.

GOSUB Format

```
$          .  
$          .  
$ GOSUB SUBRTN1  
$          .  
.  
.  
$ SUBRTN1:  !Label defining the beginning of the  
! subroutine.  
.  
.  
.  
$ RETURN           !Terminates the subroutine.
```

The **\$CALL** command can be used for a more structured approach to calling a subroutine.

- The **CALL** command transfers control to a labeled subroutine.
- Creates a new command level
- Can pass up to eight parameters to a subroutine using **CALL** command
- Can create up to 32 command levels with the **CALL** command
- **SUBROUTINE** and **ENDSUBROUTINE** commands define the beginning and end of a subroutine

CALL Format

```
$ .  
$ .  
$ CALL SUBRTN2  
.  
.  
.  
$ SUBRTN2: SUBROUTINE  
.  
.  
.  
$ EXIT  
$ ENDSUBROUTINE
```

Lexical Functions

Lexical functions are DCL subroutines called within command procedures or at the command level that return values.

- Return value is stored as a symbol
- General lexical format:
 F\$LEXICAL NAME(optional-Parameters)

Lexical functions can translate logical names, manipulate strings, return information about the system, the process, or devices.

Lexical Function Summary

Lexical	Function
F\$CONTEXT(ctx-type,ctx-symbol, sele-item,sele-value, value-qualifier)	Sets up selection criteria for F\$PID.
F\$CSID(ctx-sym)	Gets next CSID.
F\$CVSI(start-bit,num-bits,string)	Extracts bits from string symbol.
F\$CVTIME([input_time] [,output_time_format] [,output_field])	Converts time to various format.
F\$CVUI	Extracts bits from string symbol.
F\$DEVICE([search_devnam],[devclass],[dev type], [stream-id])	Returns a list of devices on system.
F\$DIRECTORY()	Returns current directory.
F\$EDIT(string,edit-list)	Modifies string.
F\$ELEMENT(element-num,delim,string)	Extracts an element from a delimited string "array."
F\$ENVIRONMENT(item)	Returns information about process environment.
F\$EXTRACT(start,len,string)	Extracts a substring.
F\$FAO(control-str,[arg1,[arg2[,...]]])	Formats data into a string.
F\$FILE_ATTRIBUTES(filespec,item)	Returns file attributes on a file.
F\$GETDVI(devnam,item)	Returns device and volume information.
F\$GETJPI(pid,item)	Returns job and process information.
F\$GETQUI(function,[item],[object-id],[flags])	Returns batch and print queue information.
F\$GETSYI(item,node-name,csid)	Returns system information.
F\$IDENTIFIER(ident,conv-type)	Converts identifiers from name to numeric and numeric to name.
F\$INTEGER(expr)	Converts numeric expression to string.
F\$LENGTH(string)	Returns the length of a string.
F\$LOCATE(substr,str)	Locate a substring.
F\$MESSAGE(status-code[,msg-comp-list])	Translates error message value to string.
F\$MODE()	Returns process mode (INTERACTIVE, BATCH, NETWORK, or OTHER).
F\$PARSE(filespec [,default-spec] [,related-spec] [,field] [,parse-type])	Expands or extract a portion of a file specification.
F\$PID(ctx-sym)	Returns next PID in chain.
F\$PRIVILEGE(priv-states)	Returns TRUE/FALSE dependent on privilege settings.

Lexical Function Summary (Continued)

Lexical	Function
F\$PROCESS()	Returns process name.
F\$SEARCH(filespec[,stream-id])	Searches directory filespec and returns full filespec.
F\$SETPRV(priv-states)	Sets privileges and returns old privilege settings.
F\$STRING(expression)	Converts numeric expression to string.
F\$TIME()	Returns current date and time.
F\$TRNLNM(logical-name [,table] [,index] [,mode] [,case] [,item])	Translates a logical name.
F\$TYPE(symbol)	Returns INTEGER/STRING string dependent on type of symbol.
F\$USER()	Returns UIC in identifier name format.
F\$VERIFY([procedure-value] [,image-value])	Sets verification on/off and returns previous state.

Lexical Function Examples

F\$TIME()

```
$ TIME = F$TIME( )
$ SHOW SYMBOL TIME
25-AUG-1989 10:20:10.1
```

F\$USER()

```
$ UIC = F$USER( )
$ SHOW SYMBOL UIC
[FRED]
```

F\$PROCESS()

```
$IF F$PROCESS( ) .EQS. "OBRIEN"...

$ write sys$output f$process()
BAE
$ write sys$output f$getjpi("", "username")
BAE
$ spawn
%DCL-S-SPAWNED, process BAE_1 spawned
%DCL-S-ATTACHED, terminal now attached to process BAE_1
$ write sys$output f$process()
BAE_1
$ write sys$output f$getjpi("", "username")
BAE
$
```

F\$MODE()

```
$ IF F$MODE( ) .EQS. "NETWORK"...
```

F\$ENVIRONMENT()

```
$ old_dir=f$environment("default")
$ set def sys$system
$ set def 'old_dir'
$ sh sym old_dir
  OLD_DIR = "ELLIS$DKA200:[BAE]"
$
```

F\$DIRECTORY()

```
$ SAVE = F$DIRECTORY( )
$ SET DEF 'SAVE'
$ SHOW SYMBOL SAVE
SAVE = [OBRIEN]
```

F\$TRNLNM("logical")

```
$ TRANS = F$TRNLNM("SYS$OUTPUT")
$ Sh sym x
  X = "_ELLIS$TNA3:"
$
```

F\$STRING(integer)

```
$ A = 5
$ B = F$STRING(-2+A)
$ C = B + B
$ SHOW SYMBOL/ALL
C = "33"
B = "3"
A = 5
```

```
$ NAME = "DEBORAH"
$ LEN = F$LENGTH(NAME)          !LEN = 7
$ LOC = F$LOCATE("B",NAME)       !LOC = 2 (COUNTS FROM 0)
$ LOC2 F$LOCATE("Q",NAME)        !LOC2 = 7
$ EXT = F$EXTRACT(LOC,1,NAME)    !EXT = "B"
```

-OR-

```
$ EXT = F$EXTRACT(F$LOCATE("B",NAME),1,NAME)
```

```
$ INQUIRE FILE "Enter the filename (include version number)"
$
$
$ IF F$LOCATE(";",FILE) .EQ. F$LENGTH(FILE) THEN GOTO
NO_VERSION
$
$
```

Example of Using Lexicals

```

$ TYPE GREET.COM
$ !
$ ! This command procedure prints a greeting message which
$ ! varies depending on the time of day.
$ !
$ TIME = F$TIME( )
$ LEN = F$LENGTH(TIME)
$ LOC = F$LOCATE(":",TIME) - 2
$ HOUR = F$EXTRACT(LOC,2,TIME)
$ IF HOUR .EQS. "12" THEN GOTO LUNCH
$ IF (HOUR .GES. "13") .AND. (HOUR .LES. "18") THEN GOTO AFTERNOON
$ IF (HOUR .GTS. "18") .AND. (HOUR .LTS. "24") THEN GOTO EVENING
$ !
$MORNING:
$ WRITE SYS$OUTPUT "GOOD MORNING"
$ ! OTHER MORNING STUFF HERE.
$ EXIT
$ !
$AFTERNOON:
$ WRITE SYS$OUTPUT "GOOD AFTERNOON"
$ EXIT
$ !
$EVENING:
$ WRITE SYS$OUTPUT "GOOD EVENING"
$ EXIT
$ !
$LUNCH:
$ WRITE SYS$OUTPUT "LUNCH TIME!"
$ EXIT
$ @greet
GOOD MORNING
$ show time
13-FEB-1999 02:24:14
$

```

Using Lexicals to Determine Volume Label

```
$ Type mount_name_unknown.com
$! Command procedure to mount a volume /SYSTEM, when
$! the name is unknown
$ SET MESSAGE/NOTEXT/NOSEVERITY/NOIDENT/NOFACILITY
$ SET NOON
$ MOUNT/OVERRIDE=ID 'P1'
$ SET MESSAGE/TEXT/SEVERITY/IDENT/FACILITY
$ LABEL=F$GETDVI(P1, "VOLNAM")
$ DISMOUNT 'P1'
$ MOUNT/SYSTEM 'P1' 'LABEL'
$ sh dev dka300:
```

Device Name	Device Status	Error Count	Volume Label	Free Blocks	Trans Count	Mnt Cnt
ELLIS\$DKA300:	OnLine	0				

```
$ @mount_name_unknown ellis$dka300:
%MOUNT-F-NOSYSNAM, operation requires SYSNAM privilege
$ set proc/priv=sysnam
$ @mount_name_unknown ellis$dka300:
%MOUNT-I-MOUNTED, ELLIS_SYS2 mounted on _ELLIS$DKA300:
$
$ sh dev dka300:
```

Device Name	Device Status	Error Count	Volume Label	Free Blocks	Trans Count	Mnt Cnt
ELLIS\$DKA300:	Mounted	0	ELLIS_SYS2	184584	1	1

```
$
```

Using Lexicals to Set Prompt

```
$ typ sd.com
$!If no P1 parameter then go to login directory
$if P1.eqs.""
$then
$      new_def="sys$login"
$else
$      new_def=P1
$endif
$!
$! Change the default and the prompt
$set default 'new_def'
$! Get the current default device and directory string
$new_prompt=f$environment("default")
$! If it is too long truncate the prompt string
$if f$length(new_prompt).gt.30
$then
$      write sys$error "Default too long, truncating
prompt."
$      new_prompt=f$extract(0,27,new_prompt)+"...>"
$
$else
$      new_prompt=new_prompt+">"
$endif
$set prompt='new_prompt' " "
$
$ cd="@sys$login:sd"
$ cd
DKA200:[BAE]> create/dir [.sub_dude]
DKA200:[BAE]> cd [.sub_dude]
DKA200:[BAE.SUB_DUDE]>
```

Using Lexicals to Determine Day of Week

```
$ typ dow_christmas.com
$! Determine the year for christmas
$ if P1.eqs.""
$ then
$     year=f$cvtime("",,"YEAR")
$ else
$     year=P1
$endif
$ !Determine the day of week for christmas
$ Christmas_date = "25-DEC-"+year+" 0:0:0.0"
$ dow=f$cvtime(christmas_date,, "WEEKDAY")
$ !Blast out the output
$ ostr = f$fao("Christmas will fall on a !AS in the year
!AS.", -
               dow,year)
$ write sys$output ostr
$
$ @dow_christmas
Christmas will fall on a Thursday in the year 1997.
$
```

Using Lexicals to Obtain Device Information

```
$ type da2.com
$! Command procedure to display unmounted disks, using
$! F$DEVICE
$
$! read lines and extract the disk device names
$ write sys$output "List of unmounted disks:"
$ get_next_device:
$ disk_name = f$device("*$D*")
$ if disk_name.eqs. "" then goto thats_all_folks
$ if f$getdvi(disk_name,"MNT") .eqs. "FALSE"
$ then
$     write sys$output disk_name
$ endif
$ goto get_next_device
$
$! close the file and delete it
$ thats_all_folks:
$ @da2
List of unmounted disks:
_ELLIS$DKA0:
_ELLIS$DKA500:
$
```

Error Handling

OpenVMS terminates command procedures on error or severe (fatal) error.

- You can use the \$ON command to change the behavior on error conditions.
- Changing the OpenVMS default action on errors.

Setting	Warning	Error	Severe
DEFAULT	Continues	Exits	exits
\$ON WARNING THEN GOTO CLEAN	Clean	Clean	Clean
\$ON ERROR THEN GOTO CLEAN	Continues	Clean	Clean
\$ON SEVERE THEN GOTO CLEAN	Continues	Continues	Clean

- The \$ON command resets \$STATUS and \$SEVERITY to indicate success.
- \$ON format
\$ON severity-level THEN DCL command.
- \$ON examples:
\$ ON WARNING THEN EXIT
\$ ON SEVERE THEN GOTO BOTTOM
- \$SET NOON disables error checking.
- To reset the error checking use: \$SET ON.
- Upon exit from a command procedure, \$SET ON occurs automatically.

Learning Check

- 1. What command do you use to begin reading from a file?
 - a. DIRECTORY/READ
 - b. OPEN/READ
 - c. READ/DIR
 - d. EDIT/READ
- 2. Which command is **not** a method for opening a file for write access?
 - a. OPEN/WRITE
 - b. OPEN/APPEND
 - c. OPEN/EDIT
 - d. OPEN/SHARE
- 3. What are lexical functions?

.....

.....

.....
- 4. Match the following lexical functions with their descriptions.
 - a. F\$TIME Returns a process name.
 - b. F\$PROCESS Translates a logical name.
 - c. F\$ENVIRONMENT Returns UIC in identifier name format.
 - d. F\$USER Returns current date and time.
 - e. F\$TRNLNM Returns information about process environment.
- 5. What command do you use to enable error checking?
 - a. SET NOON
 - b. SET ON
 - c. SET WARNING
 - d. SET DEFAULT

Program Development Environment

Appendix B

Objectives

This module leads you through the development steps, including object libraries and linker. It also discusses the debugger and describes how it allows the programmer to observe and manipulate a program as it runs.

After completing this module, you will be able to:

- Use object libraries.
- Perform linker search.
- Conduct run-time program investigations.
- Run the debugger.

Topics

- Program Development Steps
- Object Libraries
- Introduction to the Linker
- The Debugger

Development Steps

The program development cycle under OpenVMS includes distinct development steps. In general, these steps are:

1. Create a text file of source code in your favorite programming language using your favorite editor.
2. Compile your source to produce a file containing the object code. If there are errors, go back to step 1.
3. Link your object module(s) to resolve external references flagged by the compiler and assign virtual addresses to your code and data. This step produces an executable file. If there are errors, go back to step 1.
4. Run your program. If an obvious output error occurs, then return to step 1. If a confusing run-time error occurs, you may need the help of the debugger.
5. Serious run-time program investigations can be conducted using the debugger.

For example:

```
$ EDIT SIMPLE.C
$ CC SIMPLE          ! Produces SIMPLE.OBJ
$ LINK SIMPLE ! Produces SIMBLE.EXE
$ RUN SIMPLE

$ EDIT COMPLEX.C
$ EDIT SUB1.C
$ EDIT SUB2.C
$ CC COMPLEX          ! Produces COMPLEX.OBJ
$ CC SUB1              ! Produces SUB1.OBJ
$ CC SUB2              ! Produces SUB2.OBJ
$ LINK COMPLEX,SUB1,SUB2 ! Produces COMPLEX.EXE
$ RUN COMPLEX
```

Sample Program Development Steps

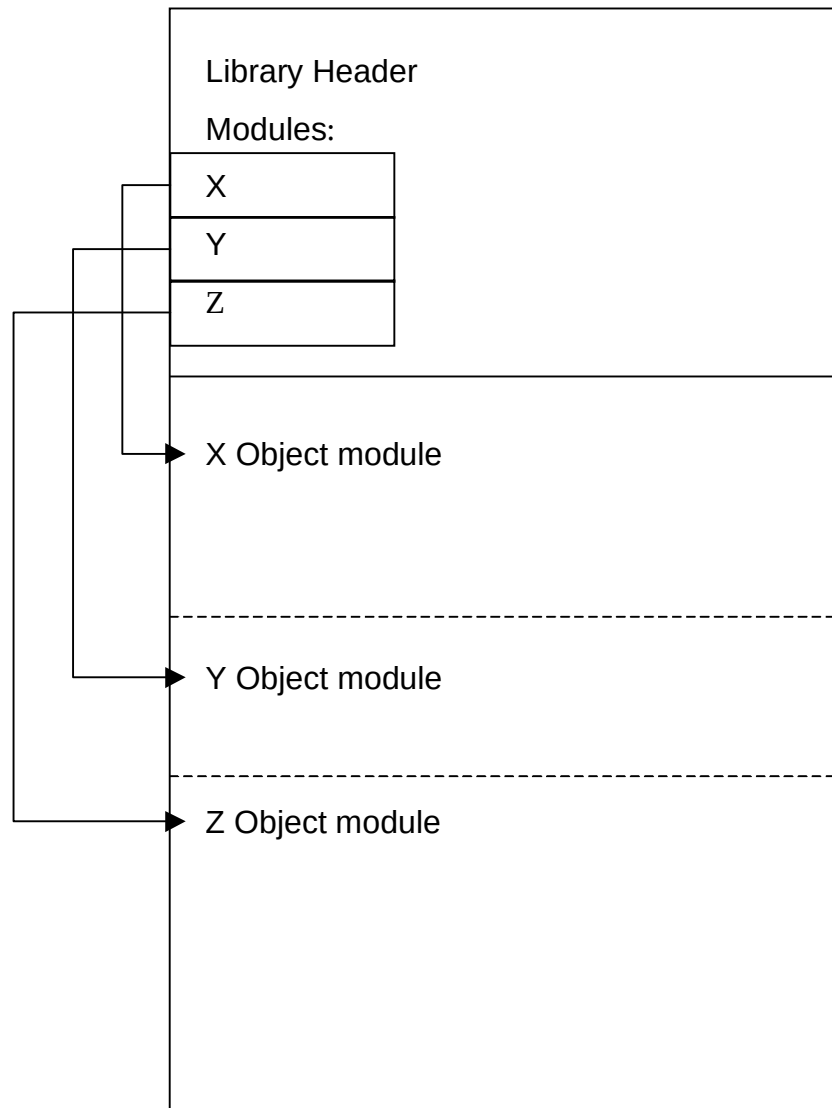
```
$ type hello.c
#include <stdio.h>
#include <stdlib.h>
int      main(void)
{
    printf("Goodbye cruel world, hello OpenVMS!\n");
    return(EXIT_SUCCESS);
}
$
$ cc hello
$ link hello
$ run hello
Goodbye cruel world, hello OpenVMS!
$
```

Object Libraries

Object libraries are files that contain compiled (.OBJ) modules of frequently used subroutines, subfunctions, math functions, and so forth

Object libraries:

- Contain modules written in any OpenVMS native mode language.
- Provide centralized storage and convenient access to the object modules.



`$LIBRARY/CREATE/OBJECT MATHEX.OLB FIND, FIGURE, TRANSLATE`

Creates an object library

`$LINK BUILD, MATHEX/LIBRARY`

Links against the library

`$LIBRARY/INSERT MATHEX.OLB OTHER, STILL_ANOTHER`

Adds an object module to an existing library

`$LIBRARY/REPLACE MATHEX FIND`

Replaces an existing module

`$LIBRARY/DELETE=CUBE MATHEX`

Marks a module for deletion

`$LIBRARY/COMPRESS MATHEX`

Reclaims space from modules marked for deletion

Automating Library Search Through Logical Names

You can automate the process of looking up modules in libraries by using logical names.

LNK\$LIBRARY

- Special series of logical names checked by the linker during its operation.
- Enables convenient sharing and LINK line simplification.

```
$DEFINE LNK$LIBRARY $1$DKA200:[OBRIEN.LIBS]MATHEX.OLB
```

- You can actually ASSIGN up to 1000 of these logical names.
- The linker translates logical names starting at LNK\$LIBRARY, going to LNK\$LIBRARY_1, then LNK\$LIBRARY_2, ..., until a translation fails. Any identified libraries are automatically searched by the linker.

LNK\$LIBRARY

LNK\$LIBRARY_1

LNK\$LIBRARY_2

...

LNK\$LIBRARY_999

Libraries and Object Modules

Remember, an object library is a file with a special organization that contains a copy of the object code.

- Therefore, you can delete the original .OBJ files.
- To extract a copy of a module from the library:
`$LIBRARY/EXTRACT=CUBE MATHEX/OUTPUT=FIGURE.OBJ`
- To determine the contents of a library, by module name:
`$LIBRARY/LIST MATHEX.OLB`

Library Example

```
$ type powerful.c
```

```
double square(double x)
```

```
{
```

```
    return(x*x);
```

```
}
```

```
double cube(double x)
```

```
{
```

```
    return(x*x*x);
```

```
}
```

```
$ cc powerful
```

```
$ library/create power powerful
```

```
$ library/list/names power
```

```
Directory of ALPHA OBJECT Library ELLIS$DKA200:[BAE]POWER.OLB;1 on 3-MAR-1999 2
1:45:38
```

```
Creation date: 3-MAR-1999 21:45:25      Creator: Librarian A09-22
```

```
Revision date: 3-MAR-1999 21:45:25      Library format: 3.0
```

```
Number of modules: 1                    Max. key length: 128
```

```
Other entries: 2                        Preallocated index blocks: 213
```

```
Recoverable deleted blocks: 0           Total index blocks used: 2
```

```
Max. Number history records: 20         Library history records: 0
```

```
Module POWERFUL
```



Note

The routine names are different than the module name.

```
CUBE
```

```
SQUARE
```

```
$
```

```
$ type powerful.h
double square(double x);
double cube(double x);
$ type test.c
#include <stdio.h>
#include <stdlib.h>
#include "powerful.h"
int main(void)
{
    double your_pleasure;
/* Get a number. */
    printf("Enter a value: ");
    scanf("%lf",&your_pleasure);
/* Display its square and cube. */
    printf("%lf**2 == %lf\n",your_pleasure,square(your_pleasure));
    printf("%lf**3 == %lf\n",your_pleasure,cube(your_pleasure));
    return(EXIT_SUCCESS);
}
$ cc test
$ link test,power/lib
$ run test
Enter a value: 2
2.000000**2 == 4.000000
2.000000**3 == 8.000000
$
```

Automate the use of the library.

```
$ define lnk$Library sys$login:power.olb
$ link test
$ r test
Enter a value: 3.11
3.110000**2 == 9.672100
3.110000**3 == 30.080231
$
```

Linker Search

The linker tries to resolve references flagged by the compiler by:

1. Looking for .OBJ or .OLB files from the LINK line.
2. Looking through special logical name libraries.
(LNK\$LIBRARY . . .)
3. Looking through the OpenVMS Common Run-Time Library.
(SYS\$LIBRARY:IMAGELIB.OLB)
4. Looking through the system object library.
(SYS\$LIBRARY:STARLET.OLB)
5. Looking for system service calls in the OpenVMS public vectors execlt.
(SYS\$SHARE:SYS\$PUBLIC_VECTORS.EXE)
6. If there are still unresolved references, you get a linker error.

The Debugger

The OpenVMS Symbolic Debugger allows you to observe and manipulate your program interactively.

- Works along the lines of most debuggers.
- Has GUI interface for workstations.

Steps

1. \$PASCAL/DEBUG BUILD
2. \$LINK/DEBUG BUILD
3. \$RUN/DEBUG BUILD

DBG> HELP

Accesses the excellent online help.

DBG> STEP

Executes one line of code.

DBG> STEP 5

Executes five lines of code.

DBG> GO

Executes until a break point or watch point is encountered or we complete execution.

DBG> SET BREAK %LINE 23

When it reaches the given line, the debugger suspends program execution, display its location, and gives you the DBG> prompt to allow you to examine and manipulate your variables.

DBG> SET TRACE %LINE 23

Same as break point except the program does not halt and give the DBG> prompt.

DBG> EXAMINE COUNTER

Shows the current value of a variable.

DBG> DEPOSIT COUNTER=50

Changes variable values.

DBG> SET MODE SCREEN

Displays source code and other windows on your screen as you debug.

DBG> EXIT or <CTRL Z>

Exits.

Learning Check

1. List the five steps of the program development cycle under OpenVMS.
.....
.....
.....
.....
.....
2. What are object libraries?
.....
.....
.....
3. What tool allows you to observe and manipulate your program interactively?
.....
.....