USING VAX/VMS ON ECSVAX

By
Peter Reid

November 1984

The DEC VAX 11/780 is one of the principal computing resources in the Department of Computer Science. The machine is situated in the Department's machine halls and supports around 30 simultaneous users running under the manufacturer's operating system "VAX/VMS" (Virtual Memory System).

This book is intended as a reference manual for those users with some previous experience of an interactive time sharing system (most probably EMAS). It provides introductory information on the most common commands and features of the system, and pointers to more detailed sources.

Commands marked with a † have been added locally and are not generally available on other VAX/VMS systems.

This document relects version 3 of the VMS operating system.

Table of Contents

1. Introduction

Users can access VAX/VMS service through terminals connected directly to the computer, through the University's network "Ednet" through the local RCO network or by typing VAX on a departmental APM (Advanced Personal Machine) workstations. Directly connected terminals offer both a greater number of facilities to the user and faster and more reliable communications.

1.1 Logging On and Logging Off

To log on press <RETURN> on a direct terminal (For access over the network see Section 8.1) and then enter your Username and Password.

If the login is successful the system will type out an identification message, that day's alert message and then a $ sign which is the prompt which says it is ready to accept input.

Logging out is achieved by typing **LOGOUT** or **LO**.

1.2 The HELP Command

This provides extensive information on commands in the system. Information is structured as a tree. At top level is a list of topics. The next level gives information on a topic and a list of subtopics. Occasionally another level exists. At each level the user is prompted for a topic. The following special characters can be typed:

<RETURN>	HELP jumps up one level
?	Reprints a list of current topics
*	Prints all information on all subtopics
Ctrl-Z	Exits HELP at any level

Typing HELP/PAGE allows you to scroll through the listed output one screenful at a time, using <RETURN> to advance through the information.

The following examples may help:

HELP
: Gives a list of topics, and prompts for a topic.

HELP PRINT
: Gives first level information on the PRINT command, a list of subtopics, and prompts 'PRINT Subtopic?', to which /COPIES could be typed, for example.

HELP PRINT /COPIES
: Gives information on the /COPIES Qualifier.

HELP PRINT Qualifiers
: Gives information on *all* Qualifiers to PRINT

HELP PRINT...
: Lists *all* levels of information on PRINT

HELP P*
: Lists first level information on *all* commands starting with 'P'. The '*' is used as a wildcard character, specifying any string.

HELP/OUTPUT=P.LIS PRINT...
: Lists all information on PRINT into the file P.LIS. If no filename is given the default is HELP.LIS.

In particular you should seek help on "SET", "SHOW" and "SPECIFY" as they contain much to help the fisrt time user of VMS.

1.3 Inhibiting Messages to the Terminal

In some situations you may wish to prevent messages from the system or from other users from appearing on the screen. Typing SET TERMINAL/NOBROADCAST will make your terminal 'deaf' (Note, Ctrl-T has no effect when deaf). The command SET TERMINAL/BROADCAST enables messages again, but any sent while deaf are discarded. On terminals connected directly to plotters and printers, where messages can affect plots and listings, you are made deaf automatically when logging on.

1.4 Login Command File

Each time you log in the commands in a file called LOGIN.COM in your home directory are executed. See 10.1.2 for more details.

1.5 Mail

The VMS mail system is used extensively for communications with other users. Typing MAIL will invoke this system full information can be found in 7.3.2 on its use. The built in help system to mail should be adequate for initial use however.

1.6 Editors

The VMS system has many editors. Those recommended for new users are Vecce and IE. Among the other editors available are EMACS and EDT. Chapter 4 discusses.

1.7 VAX Special Characters

The following Control Codes have special meanings on Vax (although certain ones may not work on remote terminals):

Ctrl-Y
: Halts execution of commands and program images, and cancels any type-ahead (unless SET NOCONTROL=Y is typed; see Section 11.2.3).

Ctrl-C
: Generally has the same effect as Ctrl-Y, but may be processed within programs.

Ctrl-S Halts output to the terminal screen.

Ctrl-Q Restarts the display of output halted by Ctrl-S.

Ctrl-Z End of File (EOF) - Signals the end of input for data entered at the terminal (eg: terminates a message to be sent by the MAIL command).

Ctrl-T Displays process statistics (CPU time, Page Faults etc) for the currently executing program image (unless SET NOCONTROL=T is typed; see Section 11.2.3). Note: Ctrl-T does not work if the terminal is 'deaf' (see Section 1.3).

Ctrl-O Halts output to the screen and discards further output until Ctrl-O is typed again or the next input is requested.

Ctrl-U Discards everything on the command line, but not any type ahead.

Ctrl-X Same as Ctrl-U, but also discards any type ahead.

2. Files

Permanent information on VMS systems is stored in files. File names are based on the following template:

File Specification: Device:[Directory]Filename.Type;Version
Example: DRA1:[ARTHUR]PROG.PAS;3

Certain defaults are used (see below) so that a file need not be given its (fairly lengthy) full name in every case.

Device — The device on which files are stored, such as a disk or magnetic tape. Device names are terminated with a colon, eg. DRA1:(physical disk drive), msa0:(magnetic tape drive), CS_PG:(concealed device used to store postgraduate student's files, see 10.2.1). Other device whick may be encountered are: NL: (acts like a black hole for data) and TT: (the default terminal a user is logged on to)

Directory — This is a file containing information on the files belonging to a user, and usually has the same name as the Username. Thus if ARTHUR has files on DRA1:, his directory is DRA1:[ARTHUR]. This is his default directory and disk at login. This elemant may also include subdirectory information. For example [ARTHUR.PROGS]

Filename — A 0 to 9 alphanumeric name given to a file by its owner.

Type — A 0 to 3 alphanumeric name, describing the type of data in the file. Certain defaults are supposed in some situations, and at these times the file type can be omitted. For example, a Pascal program would have the default extension '.PAS'. A list of default file types is given at the end of this section.

Version Number — A number from 1 to 32767. Version Numbers are incremented on creation of new versions of the same file. For most commands it is not neccessary to type a version number expicitly. In this case the latest version, i.e. the one with the highest version number, is used. Version Numbers can be limited automatically by the command SET FILE/VERSION_LIMIT= or the command SET DIRECTORY/VERSION_LIMIT= (the default of 3 is usually adequate).

Example:

DRA1:[ARTHUR]Prog.Pas;3 refers to the third version of ARTHUR's Pascal program 'Prog', found on the disk DRA1: in directory ARTHUR.

2.1 Default File Types

COM	Command File (Executed by @ or submitted as a batch job)
DAT	Data file
DIR	Directory file
OBJ	Object file created by a compiler or assembler
OLB	Library of object modules
EXE	Executable program image
PAS	Pascal source file
IMP	Imp source file (or data file for Imp program)
LIS	Compiler or LAYOUT listing. (Default for PRINT and TYPE)
TXT	General text file
MAI	Mail file
LAY	Source file for the LAYOUT text formatting program
MSS	Source file for the SCRIBE text formatting program
TEX	Source file for the TeX text formatting program
DIS	List of Usernames used by the MAIL command

2.2 Changing Default Device and Directory

The default device and directory are used whenever these parts of the file are omitted. At login, these are set to the user's disk and top level directory. These defaults can be altered, however. Two methods exist on Vax: SET DEFAULT (standard VMS) and BECOME †, which offers greater flexibility than the VMS standard, but is somewhat. An important difference between the commands is that BECOME will fault the user if say, the Directory specified does not exist on the given disk. SET DEFAULT, however, will indicate no error, and a fault will only occur later when the user tries to access files in the (non-existant) Directory.

Example: Equivalent **SET DEFAULT** and **BECOME** commands

```
$ SET DEFAULT [JIM]                $ BECOME JIM
Directory now defaults to [JIM]. The disk default is unaltered.

$ SET DEFAULT DRA0:[BOB]           $ BECOME DRA0:BOB
Directory and disk changed

$ SET DEFAULT SYS$LOGIN            $ BECOME
Initial default directory and disk restored
```

One useful facility available with BECOME (but not with SET DEFAULT) is the ability to type a username in parenthesis, for example: BECOME (ARTHUR) will set default to User ARTHUR's home directory whichever disk it is on.

The command SHOW DEFAULT (or equivalently BECOME ?) will print the current default device and directory.

2.3 Subdirectories

As well as ordinary files, directories may contain subdirectories. Subdirectories enable the user to group files in a tree structure. A subdirectory is itself a file(which has the file type ".DIR", created using the CREATE/DIRECTORY command (see Section 5). Subdirectories can themselves contain subdirectories.

Examples:

The directory SUB.DIR;1 exists in directory DRA1:[ARTHUR]. If the file PROG.LIS;1 is in this subdirectory, its full name is DRA1:[ARTHUR.SUB]PROG.LIS;1.

The following are the relevant SET DEFAULT and BECOME commands for subdirectories:

```
$ SET DEFAULT DRA0:[BOB.SUB]       $ BECOME DRA0:BOB.SUB
Default directory set to DRA0:[BOB.SUB]

$ SET DEFAULT [.SUB]               $ BECOME .SUB
If we were in directory ARTHUR, the default now becomes ARTHUR.SUB

$ SET DEFAULT [-]                                  $ BECOME ^
If we were in ARTHUR.SUB, this command moves us up to ARTHUR

$ SET DEFAULT [-.SUB2]             $ BECOME ^.SUB2
If we were in ARTHUR.SUB, the default becomes ARTHUR.SUB2
```

2.4 File Protection

Access to files can be restricted for the following four categories of user: **SYSTEM** (the system manager), **OWNER** (you), **GROUP** (users in the same group as you, ie CS2, STAFF etc) and **WORLD** (everyone else).

The following types of access can be specified: READ, WRITE, DELETE and EXECUTE (the right to execute programs) Directories can be protected in the same way as other files, but the access types above refer to the ability to read, write, delete or *create* (rather than execute) files in the directory.

SET PROTECTION — This command establishes the protection to be applied to a file or group of files (see Section 3.3.1 for wild cards in file specifications), *or* sets the default protection for files created during a terminal session. Note that the various user groups can also be identified by first letter only (ie **S**, **O**, **G** or **W**). Type HELP SET PROTECTION for more details.

Examples:

```
$ SET PROTECTION=(SYSTEM:RE,OWN:RWED,GROUP:R,WORLD)/DEFAULT
```

This sets the default protection to be applied to all files created in this terminal session (this could be put in your LOGIN file - see Section 11.1). Note that WORLD have been denied all access to the files.

```
$ SET PROTECTION=(G,W) *.IMP;*
```

Here users in groups GROUP and WORLD have been denied access to all .IMP files by use of the wildcard '*' character (see Section 3.3.1). Access for the other groups is *unchanged*.

3. The Digital Command Language - DCL

DCL is so called because it it typed with the fingers.

A DCL Command has format: **COMMAND [Qualifiers] [Parameters]**, where the square brackets denote optional parts.

3.1 Parameters

These define what the command acts on, often a file or list of files. Parameters are separated by spaces, and can be single entities or lists of entities separated by commas.

```
$ COPY JIM.DAT BRIAN.DAT    ! Copies JIM.DAT to BRIAN.DAT
```

3.2 Qualifiers

These modify the actions of a command. Qualifiers begin with a '/', and can be entered either after the command, where they apply to all parameters, or after entities in parameter lists, where they apply to these entities alone, and override any command qualifier. Values associated with qualifiers are separated by a ':' or '='.

```
$ PRINT/COPIES=2  TOM.LIS, DICK.DAT/COPIES:3, HARRY.PAS
```

The global qualifier /COPIES=2 applies to all files in the (single) parameter except DICK.DAT, whose local qualifier overrides this.

3.3 Entering DCL Commands

There are a number of ways in which DCL will aid you in typing in commands.

Usually, if required parameters are not specified, the system prompts for them (this is a useful way of finding out the required order of parameters):

```
$ COPY
$_From:  FILE1.PAS, FILE2.PAS
$_To:    FILE3.PAS
```

For many commands, in which more than one file is indicated it is not neccessary to type the full file name on each occurrence. Sensible temporary defaults are automatically supplied by the system for the ommitted parts. Thus the following lines are equivalent:

```
$ DELETE MYFILE.LIS;1,MYFILE.LIS;2,MYFILE.PAS;1
$ DELETE MYFILE.LIS;1, ;2, .PAS;1
$ COPY THISFILE.PAS THATFILE  !.PAS extension assumed
```

3.3.1 Wild Cards

The wild card character '*' can be used in many cases in place of part of a file specification. It stands for any valid string. (The character '%' is similar, but stands for only one character). The name file type and version components are treated independently. Thus:


```
$ DELETE *.LIS;*   ! Delete all listing files
$ PRINT *.PAS      ! Print the latest version of all pascal files
$ DIR *OG*.*;*     ! Lists files with the string OG in their name
$ FILES P%.IMP     ! Lists all Px.IMP, where x is one character
```

When a wild card is used in an output file spec, the system uses the corresponding part of the input file spec. e.g.

```
$ COPY PROG.PAS;* NEWPROG.*.*
```

3.3.2 Truncating Commands

Commands, Qualifiers and Parameters need not be typed in full. In general, enough characters must be typed to make the command unique. Thus, H can be typed for HELP , but SET and SHOW must be entered as SE and SH to be distinguished. SHOW SYSTEM/PROCESSES can be truncated to SHO SYS/PROC. An exception to this rule is the command RUN, which can be entered as R (because it is so frequently used).

3.3.3 Comments

! in DCL is the comment symbol. Everything following this on a line is ignored. For example

```
$ DELETE *.*.* !This is not a DCL command to use lightly
```

3.3.4 Long Commands

Long commands can be carried on to another line by typing '-' at the end of the line, thus:

```
$ COPY FILE1.PAS, -
$_  FILE2.PAS FILE3.PAS    ! copies FILE1 and FILE2 to FILE3
```

3.4 Command Procedures

Where you wish to execute a number of DCL commands in sequence it is possible to put these into a file. This is called a command procedure. One of the most important ones is your login file. Each time you login a command procedure called LOGIN.COM in your home directory is executed. A short example is given in 10.1.2. Each line of DCL in the prcedure MUST start with DCL's $ prompt. Input to programs should NOT be preceded by this prompt. A very simple example follows:

```
$ SET DEFAULT CS_CS3:[EDDIE.PROGS]
$ RUN THISPROG
this data;
that data;
$
```

It is possible to write entire programs in DCL as it has a parameter passing mechanism and (primitive) control structures, they are however about 100 times slower than conventional compiled programs so you should not regard them as a replacement, they are however useful in some circumstances. More information can be found in chapter 11 on this.

4. Text Editors

Two editors are considered: VECCE (or V), the screen-oriented version of ECCE, (the Edinburgh Compatible Context Editor) which can be used on any terminal, and IE, which can, at present, only be used on terminals directly connected to VAX. Other editors available on the system, but not reccommended for general use are EMACS, EDT, TECO, SOS and BED. Vecce is the most powerful editor available (although some would give TECO this position). It is available on a number of different computer system, both within and outside Edinburgh. IE is the most user friendly editor. It is available also on the departmental APM systems.

4.1 VECCE †

VECCE is a video version of the Edinburgh Compatible Context Editor which is available on a number of computer systems as well as the Vax. It includes facilities to build up compound commands from basic commands using sequencing, iteration and condition-testing. Such compound commands may be defined as the meaning of single keys, so that the editing repertoire can be tailored by the user according to preference and application.

The editor can be used from a range of video terminals, including Hazeltine Esprit, VT100 and Visual 200. To avoid assuming the availability of particular control keys on every terminal, all of the commands are ultimately defined in terms of printing characters. So the editor can be used just by typing in command lines and terminating them by RETURN.

Where control keys are available, they can be defined to stand for arbitrary commands, simple or compound. A default set of control key definitions are built-in to the Editor, but a user can arrange to have a personal set brought in from a pre-definition file whenever the Editor is called.

One control key, typically ENTER, is used to cause the editor to switch between command mode, in which text is interpreted as commands, and data-entry mode, in which text goes into the file. A double operation of the key in data-entry mode switches between overwrite-mode and insert-mode. Insert-mode is not available on all implementations.

The editor is available on Vax/VMS as VECCE (or V). Example calling sequences are:

```
$ V TEXTFILE        !Edit TEXTFILE to its next version number
$ V OLD/NEW         !Edit OLD and put output in NEW
$ V NL:/PROG34      !Create new file PROG34
```

The first editing command when creating a new file would normally be "G*" (Get lines)

COMMAND CHECKLIST - EDITING COMMANDS

A	Adjust line length
B	Break line in two at pointer position
C	Case-change with right-shift
C-	Case-change with left-shift
D/*text*/	Delete first occurrence of *text*
D-/*text*/	Delete prior occurrence of *text*
E	Erase character to right of pointer
E-	Erase character to left of pointer
F/*text*/	Find first/next occurrence of *text*
F-/*text*/	Find prior occurrence of *text*
G/*text*/	Get (insert) *text* as complete line above current line
G-	Get back deleted line
I/*text*/	Insert *text* to left of pointer
I-	Insert back deleted character
J	Join next line to current
K	Kill (delete) current line
K-	Kill (delete) previous line
L	Left-shift one character position
<	Cursor Left
M	Move forward one line
}	Cursor down
M-	Move Back one line
{)	Cursor up
N	Next — locate next word
N-	Next Back — locate previous word
O/*text*/	Overwrite existing text with *text*
O-	Overwrite back character (undo)
P	Print line on terminal
Q	Query Spelling
R	Right-shift one character position
>	Cursor Right
S/*text*/	Substitute *text* for text last found
T/*text*/	Traverse first occurrence of *text*
U/*text*/	Uncover first/next occurrence of *text*
V/*text*/	Verify presence of *text* at pointer position
n	repeat last command *n* times

@i(n)	Align to column *n*
+*n*	Increment number by *n*
$	Switch Inputs
^	Set Marker
^1,..,^6	Define Macro letter X,Y,Z,x,y,z
=	Revert to Marker
|	Toggle Destructive mode

COMMAND CHECKLIST - SPECIAL COMMANDS

%A	Abandon edit without updating file
%C	Close edit normally
%)D	re-write Display
%D*n*	set minimum Display (MINWIN) to *n* and re-write
%E	alter Environment options
%G *file*	Get commands from file
%H	obtain Help information
%K	Key definition
%L*n*	set Line width (WIDTH) to *n*
%M*n*	set left Margin (MARGIN) to *n*
%P *file*	Put key definitions to file
%Q	Query key definitions
%S *file*	define Secondary input file
%W*n*	Wipe out record of *n* deleted lines

SPECIAL SYMBOLS

( )	command grouping parentheses
,	separator for alternatives
$0\)	suffix to invert failure condition
?	suffix to cancel failure condition
"	ditto text indicator
!	direct-entry text indicator

4.2 IE †

Format:

$ IE File	Create *or* Update FILE
$ IE Old New	Edit OLD to NEW

IE is one of the most user friendly editors supported on VAX. Type HELP IE for full documentation on IE. Also see ALERT:IE.TXT for any information on new versions of the editor.

Text typed at the keyboard inserts text at the cursor position, and RETURN and DEL perform the same functions as outside the editor. The following are the IE commands specific to the Visual 200 keyboard (Note: CVT-CPY indicates pressing the CONVERT FUNCTION and CPY keys at the same time):

Arrows	move around file in direction of arrow key
CPY	close the edit
CVT-CPY	abort the edit
HOME	move to top of file
EF	move to end of file
CVT-EF	delete to end of file
EL	move to end of line
CVT-EL	delete to end of line
DL	delete line and put on deletion stack
CVT-DL	restore deleted lines and remove from deletion stack
CVT-HOME	(CL) Clone Line - duplicate last deleted line and restore
IL	insert line
CVT-IL	restore all deletions
ST	forwards text search
CVT-ST	repeat last search (including backwards search - see below)
CT	change text
CVT-CT	repeat last change
DC	delete char
EP	move to end of page
PRT	scroll to top of next page
CVT-EP	move to top of page
CVT-PRT	scroll back to last page
CP	mark an insertion point in any buffer (see next command)
CVT-DC	delete line and insert into buffer marked by CP
-	enter function mode (select from a list of functions)
ENTER	bind a command or "macro" to a key
LF	jump to line
TAB	move cursor one tab stop (default 3 spaces)
Shft-TAB	back two tab stop
Ctrl-@	jump to column
Ctrl-^	backwards text search
Ctrl-B	bracket matching (if on (, <, {, or [, jumps to matching bracket)
Ctrl-C	clear broadcast window (if a message is received) or abort search
Ctrl-N	repeat command a number of times
Ctrl-P	add control char (ie the char after Ctrl-P is added to the text)
Ctrl-R	refresh screen
Ctrl-T	duplicate current line into buffer marked by CP
Ctrl-U	undelete character (a packet of characters)
Ctrl-W	delete word
Ctrl-X	delete word
Ctrl-Y	abort edit (requires confirmation)
Esc Esc	Hit Escape twice to reset IE completely

IE Function Keys

F0	Move to word at left
F1	Move to word at right
F2	Up to start of line above
F3	Down to start of line below

F4 Change char to lower case and move right
F5 Change char to upper case and move right
F6 Change char to opposite case and move right
F7 Swap two chars to left of cursor
F8 Set text marker string
F9 Jump to text marker string
F10 Select a text buffer to edit (Ctrl-S does the same thing)
F11 Load a file into a buffer (Ctrl-G does the same thing)
F12 Output a buffer to a file (Ctrl-O does the same thing)
F13 Include a buffer inside the current one (Ctrl-] does the same)

IE has a set of default modes which the user can specify. Type (keypad) '-' and 'S' within IE for a list of available modes and their defaults. At command level, type HELP IE USER_PROFILE for information on creation and use of User Profile files.

IE can also be used on Visual 55, VT100 and VT220 terminals which each have a different keybinding, which can be found using the help command.

5. File Handling Commands

Full information can be got on all of these commands by typing HELP <Command> or from the 'VAX/VMS Command Language User's Guide'. In most cases only a few of the available qualifiers are given here for any command.

APPEND Adds the contents of one or more input files to the end of an output file.

/PROTECTION=code
Defines the protection to be applied to the output file

```
Example: $ APPEND BOTTOM.IMP TOP     ! add file BOTTOM to the end of TOP
         $ APPEND B1.IMP,B2.IMP TOP  ! add files B1 and B2 to TOP
```

ARCHIVE † Allows users to send files to the ERCC filestore, removing them from their directories. The /FETCH qualifier allows these same files to be restored to the directory. The IDX command lists the files currently held on archive for the user. ARCHIVE should be used to remove files you wish to keep, but will not be using.

/FETCH Specifies that a file is to be fetched from archive.

/KEEP Does not delete the file after archiving it.

/QUEUE Lists the files currently queued for archiving.

```
Example: $ ARCHIVE *.IMP             ! Sends all IMP files to archive
         $ ARCHIVE/FETCH OLD.IMP     ! Get OLD.IMP from archive
```

CLEAN † Invokes a program to help maintain your filespace. The name of each file specified is displayed, and a set of single character options are available to act on each file. Type 'H' within CLEAN for a list of options.

```
Example: $ CLEAN          ! Clean all files in the current directory
         $ CLEAN *.LIS;*  ! Clean listing files in the current directory
```

COPY Used to copy one or more files to a single file.

/PROTECTION=code
Defines the protection to be applied to the output file (default is current protection if omitted).

/[NO]REPLACE Requests the replacing of an existing file, rather than the creation of a new version (which is the default - NOREPLACE)

```
Example: $ COPY F1.IMP,F2.IMP F3    ! Concatenates F1 and F2 into F3.IMP
```

CREATE Used to create a file or subdirectory

/PROTECTION=code
Defines the protection to be given to the file or subdirectory

/DIRECTORY Indicates that a subdirectory is to be created

```
Example: $ CREATE DATA.TXT          ! Creates DATA.TXT (or a new version).
                                    ! Type in the file, followed by Ctrl-Z
         $ CREATE/DIR [FRED.SUB]    ! Creates [FRED]SUB.DIR;1
```

DELETE Enables you to delete files from your directory and from the printer and batch queues (the qualifier /SYMBOL, indicating SYMBOL deletion is dealt with in Section 10.3.3)

Deletion of Files on Disk

/[NO]CONFIRM DELETE will request confirmation for each file spec requested before it deletes it (default NOCONFIRM).

/[NO]LOG The filespec of each file deleted is displayed (default NOLOG).

```
Example: $ DELETE F.LIS;3 .EXE;2    ! F.LIS;3 and F.EXE;2 are deleted
         $ DELETE/LOG  PROG.*;*     ! All PROG files are logged and deleted
```

Deletion of Files Queued for Printing or Batch Execution

The DELETE command has a different format when deleting entries in a printer or batch queue.

/ENTRY=n Specifies the queued file to be deleted from the specified queue. The entry number can be obtained using the SHOW QUEUE command.

```
Example: $ DELETE/ENTRY=210 SYS$PRINT          ! Delete Job 210 from the
                                               ! printer in Block 0/P
         $ DELETE/ENTRY=(311,313) SYS$BATCH    ! Delete 311,313 from
                                                 the batch queue.
```

DIRECTORY Displays a list of files or information about a file or group of files.

/FULL Lists all information on the files, including protection, size, owner, date. (Separately got by /PROT, /SIZE, /OWNER, /DATE).

/OUTPUT=File Requests that output be sent to the given file, rather than the terminal.

```
Example: $ DIRECTORY                ! File list of current default
         $ DIR/FULL [JIM.SUB]*.PAS  ! List of all Pascal files in JIM.SUB
```

FILES † Similar to DIRECTORY, but offers greater flexibility.

/ALL Lists files in the current directory and ALL its subdirectories.

/SIZE, /PROT, /DATE, /OWNER
Lists size (in blocks), protection, creation date, owner of files

/OUTPUT=File Requests that output be sent to the given file, rather than the terminal.

```
Example: $ FILES/SIZE .EXE                ! Lists size of all EXE files.
         $ FILES/ALL DRA1:[FRED]*.COM      ! Search all FRED's directories
                                           ! for COM files
```

PRINT Queue one or more files to be printed on the Block O/P printer. The default filetype is .LIS.

/COPIES Specifies the number of copies - the default is one.

/[NO]NOTIFY Indicate whether a message is to be returned when the file has been printed (default NONOTIFY)

```
Example: $ PRINT/COPIES=2 F.LIS;*     ! Print 2 copies of all files F.LIS
```

PROTECT † Applies the protection status for the lowest version of a file to all higher versions, which do not under standard VMS inherit their protection from their predecessor, but from the current default.

```
Example: $ PROTECT *.IMP              ! Apply protection to all IMP files
```

PURGE Deletes all but the most recent version of files

/KEEP=n Specifies the maximum number of versions to retain (default 1)

/[NO]LOG Controls whether PURGE displays the filespec of files as it deletes them (default NOLOG)

```
Example: $ PURGE PROG.*               ! Purges all files called PROG
         $ PURGE DRA1:[FRED...]       ! Purges all Fred's directories and
                                      ! subdirectories (a useful command)
```

QUOTA † Indicates the total number of blocks used and maximum allowed for your username. If you exceed this maximum you are given one day to delete or archive sufficient files to get back under this level. If after this time you still have not done so, you will be unable to create any new files until sufficient have been removed.

RENAME Change the directory, name, type and version number of files.

/[NO]NEW_VERSION

Controls whether the output file, if it already exists, should have a new version number (default NONEW_VERSION)

```
Example:  $ RENAME A.LIS B            ! A.LIS changed to B.LIS
          $ RENAME *.TXT;* *.OLD;*    ! All .TXT files changed to .OLD.
                                      ! File names and versions unchanged
          $ RENAME *.IMP;* DRA1:[FRED.SUB]*.*;*
                                      ! All IMP files in current directory go
                                      ! to FRED.SUB (but keep their names).
```

TYPE — Display the contents of a text file or files (default .LIS).

```
Example:  $ TYPE *.IMP                ! Lists all the IMP files to the screen
```


6. Other Useful Commands

CONTINUE — Continues the execution of a program image or command procedure which was halted by Ctrl-Y or Ctrl-C (Useful if this was done by accident).

NAMETT † — Prints a list of terminals connected to VAX. If a string parameter is given, only those terminal locations containing that string will be printed (see example below).

/FREE — Only those terminals currently unoccupied are listed.

```
Example: $ NAMETT/FREE "O/P"    ! Lists all free terminals in Block O/P
```

SEE † — Prints out *both* the symbolic *and* logical assignments to any name if they are defined.

SET — Changes characteristics associated with files and devices owned by the process. HELP SET gives a full list of parameters and actions. The following are a few examples to show what is available:

SET [NO]CONTROL	Determines whether Ctrl-Y and Ctrl-T are accepted
SET DEFAULT	Sets the default device and directory
SET PASSWORD	Changes your password
SET [NO]ON	Sets error-check mode in Command Procedures
SET PROTECTION	Sets file/default protecions
SET TERMINAL	Alters various terminal characteristics
SET [NO]VERIFY	Controls the printing of DCL commands in COM files

SHOW — This command displays information about the current status of the process, the system, or of devices in the system. HELP SHOW gives a list of parameters. The following are a few examples:

SHOW DEFAULT	Displays the default device and directory
SHOW LOGICAL	Displays all logical names (see SEE above)
SHOW PROCESS	Displays information about your process
SHOW QUEUE SNAIL	Lists jobs on the batch queue SNAIL
SHOW SYMBOL DESPOOL	Shows translation of the symbol DESPOOL
SHOW SYSTEM	Displays information on the entire system
SHOW TIME	Gives the current date and time

SUICIDE † — Deletes incarnations of your process which have 'hung' for some reason. Typing /ALL causes all processes other than the calling process to be deleted. If no qualifier is entered, the program prompts for a one character action for each process in turn. Type 'H' for a list of options. 'D' will cause a process to be deleted.

7. Communicating with other Users

7.1 NOTIFY †, FIND † and GONE †

These utilities are used to send single line messages and to keep track of where people are or where the intend going when logging off.

Examples:
```
$ NOTIFY FRED "Coming for a drink?"
Fred King (FRED)is in his room

$ GONE "to the pub" !log off leaving a message
Fred King (FRED) going to the pub on Monday 10th at 10.03pm

$ FIND FRED            ! Typed by another user
Fred King (FRED) went to the pub on Monday 10th at 10.03pm
```

7.2 WHOIS † and NAME †

Examples:
```
$ WHOIS GLC
George Cleland [002,007] (Our Beloved System Manager)

$ NAME King            ! Any fragment of a name will do
B. B. King (BBK)       ! All matches are printed
Fred King (FRED)
```

7.3 The MAIL Command

Used to send mail to users on Vax and elsewhere and receive mail from other users. Within MAIL, type HELP for information on available commands. A summary of useful commands is given here. Incoming messages are placed in the file MAIL.MAI in the user's home directory and the users is informed of the number of new messages at each login.

<RETURN>	List the next message in the directory, starting with any new ones
n	List message n (equivalent to READ n)
SEND	send a message to a user or list of users. If no file is specified the user enters the message, terminating it with Ctrl-Z. If the /EDIT qualifier is used, MAIL invokes a text editor to create the message. See Section 7.3.2 below for information on how to select an appropriate editor.
SEND *file*	Sends the text of *file* instead of prompting for the message text.
READ n	display page one of message n (if n is omitted, the next message is read).
READ File	Close current mail directory, and open FILE.MAI for reading.

READ MAIL	Special case of the above command which you can use if you are already reading mail when new mail arrives.
DELETE	Deletes the current message being read. This abbreviates to 'D'. Note that messages are not actually deleted until you type EXIT.
DIRECTORY	Display a list of messages in the mail file. DIR X opens and subsequently lists messages from X.MAI.
FILE	Append the current message to a file (eg: FILE WORK, appends to WORK.MAI). Useful for maintaining separate files of messages.
EXIT	Leaves MAIL, and any messages marked for deletion are removed.
QUIT	Aborts from MAIL - marked deletions are *not* carried out.

If you receive mail while within the MAIL program, type READ MAIL to accept it. Mail messages are sent to MAIL.MAI in your home directory by default. The file is set such that you cannot delete it (unless you alter the protection). The file is automatically deleted when empty.

The command

```
$ ENQUIRE BOB
```

will tell you how many messages user BOB has from you still in his mail file and the subjects of any unread ones.

7.3.1 User Lists and Foreign Machines

A list of users can be entered, one to a line in a file with extension .DIS and this file can be used when sending mail. See example below.

Mail can be sent to users at other sites by typing NODE::USER. Thus:

EMAS::EXMS01	User EXMS01 on the 2972
BUSH::ERCI35	User ERCI35 on the 2988
RCO::"J.Smith"	EMAS User J.Smith (RCO is the directory of names)
cstvax::glc	GLC on the theory VAX UNIX system

Example: the reply to a 'To:' prompt when sending mail could be:

```
To: PR, EMAS::ERCI35, JIM, RCO::"F.Bloggs", @USERS
```

This sends mail to Vax users PR and JIM, to ERCI35 on the EMAS, to F.Bloggs on EMAS, and to the list of users contained in USERS.DIS.

There are a number of user lists available in the directory ALERT: on ECSVAX. For example STAFF.DIS,CS2,3,4.DIS,PG.DIS,MSC.DIS etc.

7.3.2 Editing within MAIL

By setting the logical name MAIL$EDIT to one of the command files shown below you can use an editor to edit your message file within MAIL by typing SEND/EDIT (or just S/E) instead of SEND. The appropriate editor is invoked, and upon closing the edit the temporary file (MAIL.TMP) is mailed to the appropriate user (See Section 10.2 for logical names and ASSIGN).

The following assignment should be made:

```
$ ASSIGN CS_SYS:MAILIE MAIL$EDIT          to use IE
$ ASSIGN CS_SYS:MAILV  MAIL$EDIT          to use VECCE
```


8. The ERCC Network

8.1 Logging On to ECSVAX from the Network

VAX is available as the host ECSVAX on the ERCC network. When accessing VAX in this way, certain utilities such the editor IE may not be used. These require terminals connected directly to the system.

Utilities like Edwin and Vecce will then choose the correct defaults without further terminal dependent commands being issued.

ECSVAX can be called from almost every host within Edinburgh and most other academics sites in the UK will have facilities to do this also. As each systems network software user interface is different it is not possible to describe the method of access in a general way. We will restrict ourselves to the public network facilities provided in Edinburgh. Users of other facilities should consult a local expert.

When certain network terminals are used, such as the Hazeltine Esprit, some utilities must be informed of this fact. An example of this is the VECCE editor (see Section 4.1, but note that the keypad facility is NOT available over the network). The commands for certain terminals are:

```
$ SET TERMINAL/FT1        !For Visual 200s
$ SET TERMINAL/FT2        !For Perkin Elmer 550s
$ SET TERMINAL/FT3        !For Hazeltine Esprits
```

There are two kinds of devices used to provide public terminal access in Edinburgh: TCPs (Terminal Control Processor's) and JNT/PADs (so called because the Joint Network Team of the UK Research Councils commissioned construction of these Packet Assembler-Dissasemblers). Most public terminals are controlled by TCPs at the moment but over the next few years these will be replaced by PADs.

Use of TCPs

When the space bar of a TCP is struck the terminal will prompt for a host name. Typing ECSVAX at this point will normally get you through. If this does not work try XGATE and then "CALL ECSVAX" once you are logged onto XGATE. If this does not work seek expert advice.

When <ESC> is typed, the EMAS TCP prompts 'Int:'. Interrupts are interpreted as follows:

```
INT:Y     Same as Ctrl-Y on VAX. (Type Ctrl-Y to get 'End Of Input')
INT:A     Same as Ctrl-Y on VAX
INT:C     Same as Ctrl-C on VAX, ie: program interceptible interrupt
INT:.     Same as Ctrl-C and '.' (halts IMP program without diagnostics)
INT:?     Same as Ctrl-C and '?' (halts IMP program with diagnostics)
```

If using screen based facilities on a TCP (e.g. screen editors) it may be neccessary to go into graphics mode by typeing the sequence "CTRL-A G".

Use of JNT/PADs

Striking the return key (it might take several goes) will cause a prompt for a host to appear on the terminal. ECSVAX should always work with a PAD. HELP HOSTNAMES will give you a list of the other hosts the PAD knows about.

8.2 Network Access, File Transfers and Printing

CALL †

This command allows Vax users to access other computers on the ERCC network. Type CALL HELP for a list of available computers (Hosts).

Format: CALL <Host> [/NORISK] [/INPUT=<File>] [/LOG=<File>]

/NORISK — If you type Ctrl-Y whilst accessing another system you will be asked for confirmation before aborting the session.

/INPUT — Input is taken from the file given. At the end of the file input reverts to the terminal. See example below.

/LOG — A record of all I/O is kept in the given file.

Examples:

```
$ CALL EMAS/NORISK                ! protects against accidental Ctrl-Y
```

When "calling" EMAS the following control codes are important:

Ctrl-C is equivalent to typing ESC on a TCP
Ctrl-Y is aborts the session
Ctrl-Z is equivalent to typing 'End-of-File' on a TCP

In the next example the file PSSLOG.TXT could have a user's PSSE Username and Password. Thus by typing CP, a user can log on to PSSE. Input reverts to the terminal at the end of the file.

```
$ CP :== CALL PSSE/INPUT=PSSLOG.TXT
$ CP     ! Will log the user on to PSSE
```

FTP †

This command allows file transfers between Vax and other systems. The HELP command can be typed in response to the 'FTP>' prompt to get information on any of the commands below. All FTP commands can be distinguished by the first letter only.

DEVICE — selects the remote system (eg EMAS). Type HELP PLACES within FTP for a list of systems.

USER — selects Username and Password (eg: USER EXMS01 <pass>). If the Password is not entered it is prompted for, and not echoed. Note that for device EMAS it is the Background and not Foreground Pass that is required.

EXIT — exit from FTP

FETCH — bring a file from the selected computer to Vax. HELP FETCH for full format. The simplest format (and one most often used) is:
FETCH Vaxfile Remotefile

SEND — send a Vax file to the remote system. The simplest format is:
SEND Vaxfile Remotefile

QUEUE — displays queued FTP transfers (eg: QUEUE EMAS for EMAS queues only, QUEUE alone for all queues).

REMOVE n — remove request n from a queue.

If the logical names FTP$DEVICE and FTP$USER are previously assigned, FTP will use these by default, and the commands DEVICE and USER need not be entered. For example:

```
$ ASSIGN EMAS FTP$DEVICE       ! These entries in your LOGIN file?
$ ASSIGN "EDGAR <pass>" FTP$USER
$ FTP                           Assumes Device EMAS
FTP> SEND FILE                 ! No need to enter DEVICE and USER
```

Full Example:

```
$ FTP
FTP> DEVICE EMAS
FTP> USER EXMS01                  ! User is EXMS01 on EMAS
Password:                         ! NB Background Pass for EMAS
FTP> SEND PROG.IMP P2             ! Queue PROG.IMP for transfer
 Dra1:[EDGAR]PROG.IMP;1 will be sent to P2
FTP> Q EMAS                       ! Check transfer status
 EMAS QUEUE
 Ident User, File
   8   EXMS01, Send Dra1:[EDGAR]PROG.IMP;1 (active)
FTP> REMOVE 8                     ! Halts transfer
FTP> EXIT
$
```

NETPRINT †

This command allows the printing of Vax files on ERCC printers. The default printer is LP23, unless another is specified.

If the logical names USER_NAME and USER_DELIVERY are assigned values, NETPRINT will use these when setting up the delivery information on the file header (The default values are the Vax Username and "ERCC Front Door"). For example:

```
$ ASSIGN "EDGAR Smith" USER_NAME
$ ASSIGN "JCMB Level Three" USER_DELIVERY
```

Example:

```
$ NETPRINT X.LIS                  ! X.LIS queued for LP23 by default
$ NETPRINT/DEVICE=LP15 P1.PAS,P2.LIS
```

The FTP command can be used to follow the queuing status of the files. Typing QUEUE EMAS will show the desired queue, and the REMOVE command can be used to abort the transfer.

9. Program Development and Execution

A variety of languages are available on ECSVAX, such as Imp, Pascal, Modula-2, Fortran, Simula, Algol68, PS-Algol, C, Prolog and ML. The principal languages for student use are Imp and Pascal.

9.1 Compilation

Most compilers produce object modules with extension .OBJ, and optional listing files of type .LIS if the /LIST qualifier is given after the compiler command. If any errors occur during compilation no new .OBJ file is produced.

IMP † Invokes the IMP compiler. The default filetype is .IMP.

PASCAL Invokes the PASCAL compiler. The default filetype is .PAS.

Type HELP IMP and HELP PASCAL for information on other qualifiers.

Other compilers are invoked in a similar way. Note that the languages Prolog and ML are interpreted and the text is acted upon immediately by the appropriate system. No compilation stage is involved. See the help information for further assistance.

9.2 Linking

An object module is not itself executable, as generally it makes reference to other procedures. The linker combines object files with files containing these referenced external procedures, or other modules explicitly referenced by the object module, which are included as parameters of the command, to produce an executable image of type .EXE.

LINK Used for *all* language .OBJ files.

```
Example: $ LINK IMPPROG,MYLIB   ! Links IMPPROG.OBJ with the MYLIB library
         $ LINK PASPROG         ! Links PASPROG.OBJ with system routines
```

9.2.1 Object Module Libraries

These can store procedures frequently called by many programs.

```
Example: $ LIBRARY/CREATE LIB1 MOD1.OBJ,MOD2.OBJ,MOD3.OBJ
```

The compiled object modules MOD1, MOD2 and MOD3 are put into LIB1.OLB. If any of these routines are used by a program, the library must be linked as follows:

```
$ LINK PROGRAM,LIB1/LIBRARY
```

This includes only the modules referenced by PROGRAM in the file PROGRAM.EXE.

9.3 Program Execution

To run a program image, type **RUN** (abbreviates to R). The default file assumed is .EXE. The qualifier /OUTPUT=Filespec enables output to be redirected to the given file, rather than the terminal.

```
Example: $ RUN/OUTPUT=RESULTS.DAT PROG.EXE  ! Output to RESULTS.DAT
```

9.3.1 Input and Output Streams - General

In general the logical names SYS$INPUT and SYS$OUTPUT are set to be the default input and output streams for most programs. If they are redefined to be files, programs can read or write data using these files. For example:

```
$ ASSIGN IN.DAT  SYS$INPUT
$ ASSIGN OUT.LIS  SYS$OUTPUT
```

For Pascal programs, channel INPUT would be connected to IN.DAT, and OUTPUT to OUT.LIS, whereas for Imp programs, Input and Output Streams 0 would be connected to the appropriate files. Other languages would use the appropriate defaults.

The method above is general, but has the disadvantage that the logical names must be DEASSIGNed in order to reset them to their default values for terminal I/O after the program has terminated. The methods outlined in the next two sections may therefore be more appropriate.

9.3.2 Input and Output Streams - Pascal

The logical names PAS$INPUT and PAS$OUTPUT can be used to redirect data input and program output from their default values (from and to the terminal). The ASSIGN command is covered in detail in Section 10.2.

```
Example: $ ASSIGN INPUT.DAT PAS$INPUT     ! Input taken from file INPUT.DAT
```

9.3.3 Input and Output Streams - Imp

Suppose an Imp program PROG.EXE contains the following statements:

```
OPEN INPUT (1, "INFILE");   SELECT INPUT (1); READ (INVAR);
OPEN OUTPUT (1, "OUTFILE"); SELECT OUTPUT (1); WRITE (OUTVAR,0);
```

The following assignments could be made (see Section 10.2 for a full description of logical names and ASSIGN):

```
$ ASSIGN INSTUFF.DAT INFILE        ! Uses the default directory
$ ASSIGN OUTSTUFF.DAT OUTFILE      ! for both assignments.
$ RUN DATES                        ! Uses the above assignments
```

9.4 Batch Jobs

Programs or command procedures which take a long time to run can be submitted for batch execution, freeing the terminal for the user to continue interactive work. See Section 11.1 for more information on Command Procedures.

9.4.1 Batch Queues

On ECSVAX there are two principal queues onto which batch jobs are placed. These differ in the number of jobs they can run at one time, and in the maximum amount of CPU time that can be used by any one job. The queues are SYS$BATCH (the default) and SNAIL. The essential characteristics of each are as follows:

	Max No of Running Jobs	Base Priority	Default CPU Time Limit	Maximum CPU Time Limit
SYS$BATCH	6	3	5 minutes	30 minutes
SNAIL	2	2	2 hours	Infinite

Jobs queued on the SNAIL queue can run for longer than on SYS$BATCH, but have a lower priority and are executed less frequently. Jobs should initially be tested on queues with short CPU time limits in case of infinite loops. The system management should be made aware of jobs with a CPU requirement in excess of 2 hours, to avoid unnecessary termination of jobs.

9.4.2 Submission of Batch Jobs

SUBMIT — Specifies a command file to be queued for batch execution. The following qualifiers are used to override the various defaults applied by the command. Output from a batch job is usually sent to a file .LOG which has the same name as the command file in the user's home directory.

/QUEUE — Which queue the job should be placed on, default SYS$BATCH

/NAME — Sets a name for the job (defaults to the command file)

/KEEP — Prevents deletion of the .LOG file

/CPUTIME — Overrides the default CPU maximum for the queue. The string 'Infinite' will select the largest time available.

Example:

```
$ SUBMIT/QUEUE=SNAIL/NAME="simulate"/CPUTIME=1:30:00 X.COM
```

DO † — Specifies a single line command to be batch executed on SYS$BATCH. The output from the job goes to DOJOB.LOG in your default directory.

DOSNAIL † — The equivalent command for batch execution on the SNAIL queue.

```
Example: $ DO RUN TESTPROG
         $ DOSNAIL LAYOUT BIGFILE
```

SHOW QUEUE/BATCH will show the status of all batch queues.
SHOW QUEUE SNAIL shows the status of the SNAIL queue only.
SHOW SYSTEM/BATCH gives useful info on currently active batch jobs.

10. Tailoring DCL - I

10.1 Introduction

DCL allows the user great scope in the modification of the command interface. Command names can be changed and new commands added. Files of commands, with facilities similar to a high level language can be written to carry out complex tasks. Users who wish a simple introduction to DCL should read Sections 10.1, 10.2, 10.3.1, 10.3.2, 10.3.3, and the first part of 11.1. Some simple examples are given here.

```
$ B*ecome  :== 'Become          ! User can now type B alone
$ Tim      :== Show Time        ! Type TIM to execute SHOW TIME
$ Prog     :== $U1:[BERT]P1     ! Type PROG to run P1.EXE
$ C        :== @Mycom           ! C executes the file MYCOM.COM
$ !
$ Assign  U1:[BERT.COM] Com:  ! Logical name to use
$                             ! command file directory.
```

10.1.1 Foreign Commands

You can create new commands within DCL by setting up symbol names to indicate the execution of program images. Then, by typing the symbol name, you can cause the system to execute the desired image. The format of such commands and an example are given here:

```
Format:  $ Symbol :== $ImageFile     ! The '$' indicates an EXE image
Example: $ PROC :== $DRA1:[BOB.PROJ]PROCESS
         $ PROC                      ! PROC causes PROCESS to run
```

When a symbol defined in this way is used a a command, it can be followed by parameters just like a built in command. The parameters are made available to the program for interpretation. For example in

```
$ PROC SOME INPUT
```

the paramter string "SOME INPUT" is made available to the program. Substitution of symbols (denoted by apostrophes) takes place before the string is passed to the program. Section 11.1 has more information on the use of symbols within command procedures.

10.1.2 Redefining System Commands

The following example shows how system commands can be redefined or expanded. Consider the two line command procedure OUT.COM:

```
$ PURGE DRA1:[BERT...]
$ GONE 'P1
```

which purges all files in all of Bert's directories, and then executes the GONE command, taking as parameter the symbol P1. The following definition (perhaps in Fred's LOGIN file):

```
$ LO :== @OUT
```

allows him to type:

```
$ LO "home to bed
```

A short specimen LOGIN.COM file is given here.

```
$ Set Prot=(S:RE,O:RWED,G,W)/Default
$ !
$ Assign U1:[BERT] Bertie
$ !
$ B*ecome  == "Become"                    ! Note, all GLOBALS (==)
$ Pals     == "Find Phineas,Franklin"
$ BQ       == "Show Queue Sys$Batch"
$ DB       == "Delete Sys$Batch/Entry="
$ Deaf     == "Set Terminal/NoBroadcast"
$ Hear     == "Set Term/Broad"
$ Hash     == "$Bertie:Grass"
$ Write Sys$Output "Hi there Bertie!"
```

10.1.3 Logical Names and Symbols

Symbols are defined using the == or = notation and are most commonly used to define new commands. Symbols are usually only interpreted by DCL.

Logical names are usually used to refer to system entities. For example in the login file above Bert can refer to his home directory by the logical name bertie: e.g.

```
$ dir bertie:    ! instead of $dir u0:[bert]
```

Logical name translation takes place in both DCL and system and user programs.

10.2 Logical Names

The ASSIGN command provides:

1. A way of equating logical names used within programs or command files to physical file specifications, and
2. A shorthand way of referring to frequently used devices, directories or files.

Format: $ ASSIGN *Equivalence_Name*[:] *Logical_Name*

'Equivalence_Name' is a device, directory, filespec or another logical name. The colon is required when the name equates to a device. *'Logical_name'* is a 1-63 character string.

Examples of the first use of ASSIGN can be found in Sections 9.3.1, 9.3.2 and 9.3.3. To show the second use to which ASSIGN can be put, a specimen list of assignments (which could be entered in a LOGIN.COM file) is given, followed by some commands which show the potential usefulness of these assignments.

```
$ ASSIGN    DRA0:                 D0
$ ASSIGN    D0:[BERT]             Me
$ ASSIGN    Me:                   Bertie:
$ ASSIGN    D0:[BERT.MISC]        Misc

$ Become Bertie                   ! Sets directory to DRA0:[BERT]
$ Files D0:[Jim]                  ! Lists files in directory DRA0:[JIM]
$ Rename Me:Test.Exe  Misc:
            ! Moves DRA0:[BERT]TEST.EXE to DRA0:[BERT.MISC]TEST.EXE
```


The command SHOW LOGICAL is used to display logical name equivalences:

```
SHOW LOGICAL        displays all logical names.
SHOW LOGICAL ME:    display logical equivalence for 'ME:'

  /PROCESS          display only process created names
  /SYSTEM           display only system defined names
  /GROUP            display only group defined names
```

Note: the DEFINE command provides an identical facility to that of **ASSIGN**, but has the format:

$ DEFINE *Logical_Name Equivalence_Name*[:]

```
DEASSIGN        cancels logical name assignments set up by your process
                (this is done automatically when you log out.

  /ALL          cancels all logical name assignments set up by your
                process only
```

Example:
```
$ ASSIGN DRA0: Mydisk
$ DEASSIGN Mydisk
```

10.2.1 Concealed Device Names

The concealed device convention allows partitioning of a large physical disk into a number of smalled units. A concealed device name equates a partial directory specification with a logical name. An example would be:

CS_CS2: equates to the incomplete specification __DRA1:[CS2.]

Thus: $ BECOME CS_CS2:WALLY sets default directory to DRA1:[CS2.WALLY]

This allows multiple logical devices e.g. CS_CS2,3,4 etc. to reside on the same physical disk, while concealing this fact from the user.

10.3 Symbols

A symbol name is a 1 to 255 character alphanumeric string having a value which is a string or integer. It can be equated to string or integer expressions. The expression type is determined by the operators and values used.

Type HELP SYMBOL_ASSIGN for online information on symbols.

In integer expressions, strings are converted to integer as follows: any string such as "123" becomes 123. Any string beginning with T, t, Y or y evaluates to 1 (True). Any other string evaluates to 0 (False), including "". In string expressions, integers are converted to strings thus: 123 -> "123".

Integers can be specified using hexadecimal (%X) or octal (%O) radix operators (The decimal operator (%D) is the default, and can be omitted):

ie: $ A = 33 or $ A = %X21 or $ A = %041

10.3.1 Symbol Assignment

Valuse are assigned to symbols by using the following commands:

```
   Symbol  = [=]   Integer or String Expression
                   (or String enclosed in " characters)
or Symbol  := [=]  String Expression
```

A string MUST be in quotes if it contains multiple blanks, lowercase characters, or ! or " characters.

Local symbols, created by '=' or ':=' are accessible only at the current or lower command levels. Global symbols, created by '==' or ':==' are usable at all levels. Local symbols can be used inside command procedures without being visible to higher levels (they are deleted when the command procedure exits). Global symbols can be defined in the LOGIN.COM file for use during a terminal session. Type SHOW SYMBOL X or SEE X to show the value assigned to symbol X.

Examples:

```
$ COUNT = 1                 ! Value 1 assigned locally to COUNT
$ PF == "PRINT/FEED"        ! Global assignment to PF
$ SHOW SYMBOL PF
PF = "PRINT/FEED"
```

10.3.2 Substring Replacement in Character String Symbol Values

This has format: SYMBOL [Offset,Size] :=[=] String

Examples:

```
$ A = "ABCDEF"
$ A[0,3] = "DEF"                ! A is now "DEFDEF"
$ B[4,3] = "GHI"                ! B is "    GHI" (four blanks at start)
$ LINE[0,80] = " "              ! LINE has 80 blanks in it
```

10.3.3 Symbol Deletion

Symbols can be deleted from symbol tables by 'DELETE/SYMBOL SymbolName'. The /LOCAL or /GLOBAL qualifiers indicate which table to delete from. The /ALL qualifier causes deletion of all symbols in the required table.

10.3.4 Symbol Substitution

Apostrophes (') indicate symbol substitution. Thus:

```
$ Name  = "MYFILE"          ! Note: Quotes needed when using '='
$ Type := .TST              ! Quotes not needed with ':='
$ Print 'Name'Type'         ! Prints MYFILE.TST.
```

Within strings, " ' 'SYMBOL' " indicates substitution. Thus:

```
$ File == "DRA1:[BERT]' 'Name' ' 'Type' "     ! DRA1:[BERT]MYFILE.TST
```

Apostrophes are not needed if a symbol is first in a command line. So:

```
$ Del == "DELETE/ENTRY="  ! Sets Del to the string given
$ Del 123                 ! No ' needed around DEL
```


In some circumstances, symbol substitution is carried out automatically without the need for apostrophes, eg in IF or WRITE statements (see Section 11.2.1), or in assignments using '=' or '=='. For example:

```
$ B  == BECOME              ! These two lines have the same effect
$ B == "'BECOME"            ! that is, symbol substitution occurs.

$ IF C .EQ. D THEN EXIT     ! C and D assumed to be symbols
$ TOTAL = COUNT + 1         ! No ' needed around COUNT
```

Symbol assignments can be of the form: $ B*ECOME == "'BECOME
" Any characters after the '*' may be omitted. Thus, B, BE, BEC, etc would all have the same effect. Note that commands declared in this way override native DCL commands.

10.3.5 The & Substitution Operator

The & differs from the ' operator in that substitution does not take place until the command line is parsed, as opposed to when it is first encountered.

Example:

```
$ B :== F.LIS       ! NB: No quotes needed with :== here
$ A :== &B          ! No substitution here (Note: only one & used)
$ C :== 'B'         ! Substitution takes place here, ie C = "F.LIS"
$ B :== G.LIS       ! B is now "G.LIS"
$ TYPE 'A'          ! Substitution of A, ie types out G.LIS
$ TYPE 'C'          ! Types out F.LIS
```

The & is not allowed in character strings, or to request substitution inside strings. Symbols may not end in &, and may not be concatenated using &.

10.3.6 Steps in Symbol Substitution

Symbol substitution occurs in three phases:

1. Replace all symbols preceded by ' (or ' ' in strings within quotes)
2. Parse command, ie substitute first symbol on line if necessary. Also replace all symbols preceded by &.
3. Evaluate expression, ie replace symbols during execution of command.

Substitution via apostrophes is carried out iteratively until there are no apostrophes left, unless values are substituted within strings enclosed by quotes. The following exampleswill help illustrate this.

```
$ FILE = "'A'"          ! FILE='A' (quotes stop further substitution)
$ A = "TEST.IMP"        ! A given a value
$ TYPE 'FILE'           ! FILE substituted by 'A', then by TEST.IMP

$ SYMBOL = "Brian"      ! SYMBOL equated to the string "Brian"
$ A = "'SYMBOL'"        ! A := 'SYMBOL' (quotes stop substitution)
$ B = 'A'               ! B = 'A' = 'SYMBOL' = "Brian"
$ B = "' 'A"            ! B = 'A' = 'SYMBOL' (quotes stop substitution)

$ COMMAND = "TYPE X.LIS"
$ EXEC = "'COMMAND'"    ! EXEC equates to 'COMMAND'
```

```
$ EXEC                  ! The first symbol on line does not need an '
                        ! but it equates only to 'COMMAND'
$ 'EXEC'                ! Substitution of COMMAND
```

The & is useful when we wish substitution to occur from right to left:

```
$ COUNT = 1
$ DELETE &P'COUNT'      ! First replace COUNT by 1, then DELETE P1
```

Whereas using apostrophes only would cause an error:

```
$ DELETE 'P' 'COUNT'    ! Replaces P by "" and ends up with DELETE '1'
```

In some cases, substitution does not require apostrophes. For example:

```
$ FILENAME == "BERT.LIS"
$ IF 'FILENAME' .NES. "" THEN TYPE 'FILENAME'
```

would equate to:

```
$ IF BERT.LIS .NES. "" THEN TYPE BERT.LIS
```

This would fail, since the IF statement would look for a symbol called BERT.LIS. The following is the correct syntax for the statement:

```
$ IF FILENAME .NES. "" THEN TYPE 'FILENAME'.
```

10.3.7 Operators in Expressions

(Precedences in brackets: (1) = lowest)

```
Logical
.OR.    (1) .AND. (2) .NOT. (3)
Arithmetic Comparison
.EQ.    (4) .GE.  (4) .GT.  (4) .LE.   (4) .LT.   (4) .NE.   (4)
String Comparison
.EQS.   (4) .GES. (4) .GTS. (4) .LES.  (4) .LTS.  (4) .NES.  (4)
Arithmetic Operators
+ (5) - (5) + (unary:7) - (unary:7) * (6) / (integer:6)
String Operators
+ (concat:5) - (subtract:5)
```

String comparison is carried out in dictionary fashion.

Examples:

```
$ A = 3 .OR. 5                  ! A = 7
$ C = .NOT. 3                   ! C = -4 (Two's complement)
$ D = 3 + 4 .AND. 2 + 4         ! D = 6 (note precedences here)

$ F = "ABC" + "DEF"             ! F = "ABCDEF"
$ F = "LISTING.LIS"-"LIS"       ! F = "TING.LIS" (First occurrence only)

1 .GE. 2                        ! 0 (False)
"123" .EQ. 123                  ! 1 (True)
"MAYBE" .LTS. "maybe"           ! 1 (True) Lowercase > Uppercase
"YES"   .GTS. "YESS"            ! 0 (False)
```


11. Tailoring DCL - II

11.1 Command Procedures

A command procedure is a file containing a sequence of DCL commands, with default file extension .COM. Each command begins with a dollar sign. (See Section 11.1.2 for information on controlling Command Procedure I/O). In simplest form a file PROG.COM could look like this:

```
$ ! Comments can be entered using the ! symbol.
$ Pascal/List Prog              ! Comments can be on the
$ Link Prog, Sys:PasLib         ! same line as commands
$ Run Prog
```

It is executed by typing @PROG (the .COM is the default extension). The disk and directory should be typed if they are not the current default values. Thus the command could be @DRA0:[FRED]PROG. Command file execution using @ can be nested up to eight levels deep.

Lengthy command procedures can be submitted for batch execution using the SUBMIT command. See Section 9.4 for a full description of batch processing.

A special command procedure called LOGIN.COM in your base directory is executed by the system whenever you log on, or at the start of every batch job. This procedure can be used to execute commands commonly entered at login, or to initiate global symbol assignments (see Section 10.3.1) for the terminal session. A specimen LOGIN file is given in Section 10.1.2.

11.1.1 The SET VERIFY Command

This is a useful debugging aid when creating command procedures. It causes lines in a command procedure to be printed as they are executed. Thus a faulty command line can be more easily traced. SET NOVERIFY inhibits the printing of the command lines (the default). Verification can be altered during command file execution as follows:

```
$ @TEST.COM             ! set a command procedure running
(Type Ctrl-Y here)      ! stop the procedure with Ctrl-Y
^Y
$ SET VERIFY            ! set verification ON
$ CONTINUE              ! continue the command file
```

11.1.2 Controlling I/O in Command Procedures

The following logical names are defined at login (there are others - for a full list see the 'Vax/VMS Guide to using Command Procedures', or type SHOW LOGICAL/PROCESS SYS$*.

SYS$INPUT — the default command and data input stream. When a command procedure is executed, SYS$INPUT is assigned to the command file. It is restored to its previous value upon leaving the command file.

SYS$OUTPUT — the default output stream for commands and program images. Using /OUTPUT when executing command files causes SYS$OUTPUT to become the output file name for the duration of the command file.

Example: @PASTEST.COM/OUTPUT=RESULTS.DAT

SYS$COMMAND — the initial command input stream for the process (usually the terminal, or a command file for batch jobs). Thus at login, SYS$INPUT and SYS$COMMAND have the same assignment.

SYS$LOGIN — the user's home directory.

11.1.3 Redefining SYS$INPUT and SYS$OUTPUT

By default, SYS$INPUT is set to the level currently executing. Thus the following Command Procedure takes its data from the command file. The data is entered without a dollar sign, immediately after the appropriate command.

```
$ PASCAL PROG
$ LINK PROG
$ RUN PROG
10 20
30 35
```

Input can be supplied from the terminal by reassigning SYS$INPUT to the initial input stream SYS$COMMAND (See Section 10.2 for ASSIGN) Alternatively SYS$INPUT can be ASSIGNed to a file. Thus:

```
$ ASSIGN SYS$COMMAND: SYS$INPUT
$ RUN PROG1             ! Input now comes from the terminal.
$ RUN PROG2             !                          until...
$ DEASSIGN SYS$INPUT    ! The default stream is restored
$ RUN PROG3             ! Input now comes from the command file
```

Typing ASSIGN/USER_MODE will cause assignment for the execution of one image only. After that the previous assignment will be restored.

Similarly SYS$OUTPUT can be redefined as follows:

```
$ ASSIGN/USER_MODE RES1.DAT SYS$OUTPUT
$ RUN PROG1
```

Or it can be suppressed if output is not required at any stage, by assigning it to the null device NL:

```
$ ASSIGN/USER_MODE NL: SYS$OUTPUT
$ DIR [XXX!Directory results are thrown away
```

11.1.4 The INQUIRE Command

INQUIRE can be used within a command procedure to invite a response from the user and assign it as a value to a symbol. Thus:

```
$ ! This is file PASINQ.COM
$ INQUIRE Insym "Input File"          ! The string is the prompt
$ ASSIGN/USER_MODE 'Insym' Pas$Input
$ RUN PASTEST
```

Gives:

```
$ @PASINQ
Input File: stuff.dat       ! Note: INQUIRE adds ': ' to the prompt
```


If no prompt string is given to INQUIRE it uses the symbol name. It will add a ': ' to the string unless /NOPUNCTUATION is specified. If RETURN is typed when the prompt is shown, the symbol is assigned "".

11.1.5 Passing Parameters to Command Procedures

Eight local symbols (P1 to P8) are defined for use in passing parameters to command files. The command format is @FILE.COM Param1 Param2... Param8, and the list of parameters is assigned to P1, P2..P8. If values are not specified, "" is assumed.

Examples:

```
$ @TEST  IN.DAT OUT.DAT      ! P1 = "IN.DAT", P2 = "OUT.DAT", P3-8=""
$ @TEST  "" "Hello" 10       ! P1 = "", P2 = "Hello", P3 = 10, P4-8=""
```

In the first example above, the file @TEST could be as follows:

```
$ ASSIGN 'P1' Pas$Input
$ ASSIGN 'P2' Pas$Output
$ RUN PASTEST
```

It is not possible to write a command file of the type shown below, where the data to a program are entered as parameters:

```
$ RUN PROG.EXE              ! PROG takes its data from the file
'P1 'P2                     ! One might expect these to be
'P3                         ! translated but they aren't.
```

If the above file is called T.COM, typing @T 10 20 "Hello" does not result in the program PROG.EXE reading in 10, 20 and "Hello".

The 'solution' to this problem requires the creation of a temporary command file using the DCL I/O procedures covered in Section 11.2.4. An example command file is given here.

```
$ Open/Write OUT Temp.Com       ! Create TEMP.COM. ASSIGN to OUT
$ Write OUT "$ RUN PROG.EXE"    ! Write this line to TEMP.COM
$ Write OUT P1," ",P2           ! Write out values of P1 and P2
$ Write OUT P3                  ! Write out value of P3
$ Close OUT                     ! Close TEMP.COM
$ @Temp.Com                     ! Execute the file TEMP.COM
$ Delete temp.com               Tidy up when finished
```

If you type @T 10 20 "Hello", the file TEMP.COM will be created and executed, with the following three lines:

```
$ RUN PROG.EXE
10 20
Hello
```

11.1.6 Passing Parameters to Batch Jobs

The /PARAMETERS qualifier is used for SUBMIT thus (using the same parameter lists as in the previous section):

```
$ SUBMIT TEST/PARAMETERS=(IN.DAT,OUT.DAT)
$ SUBMIT TEST1,TEST2/PARAMETERS=("", "Hello",10)
```

In the last example, both TEST1 and TEST2 will use the parameters given.

11.2 Flow of Control and I/O in Command Procedures

11.2.1 Flow of Control

The flow of control in command procedures can be altered by the following commands (for more information type HELP <Command> or see the 'VAX/VMS Guide to Using Command Procedures'):

$ IF *Expression* THEN *Command*

Evaluates *Expression*. If it equates to true then *Command* is executed. The expression is true if it has an odd integer or numeric string value, or is a string beginning with "T", "t", "Y" or "y". Otherwise it is false.

Example:

```
$ IF A+B .EQ. 10 .OR. C THEN Command
$ IF A THEN Command
$ IF P1 .EQS. "" THEN Command

$ IF A .EQ. B THEN -                ! Note '-' continues command onto
$ IF C .EQ. D THEN -                ! new line.
$ RESULT = 1                        ! RESULT = 1 only if A=B and C=D
```

Remember, IF does not require apostrophes around symbols, unless iterative substitution is required (ie more than once).

$ GOTO *Label* Jumps to the line beginning '$ *Label:*'

Example:

```
$ Count = 1
$ Loop:
$    Count = Count + 1
$    If Count .LE. 10 Then Goto Loop
```

$ EXIT *n*

Exits the current command procedure and goes up one level, optionally setting the global symbol $STATUS to the integer *n*. (This can then be tested for in the outer level). If *n* is omitted $STATUS is unchanged.

$ STOP Exits to level 0 (the terminal, or halts a batch job).

11.2.2 Control of Errors in Command Procedures

The result of a command or program execution is one of 5 integer values:

0: *Warning*, 1: *Success*, 2: *Error*, 3: *Information*, 4: *Severe_Error* (Fatal)

This is placed in the system-defined globals $STATUS (also used by EXIT) and $SEVERITY (low-order three bits only). Thus, the success or failure of a command or program can be tested. Odd integers equate to true - thus the results *Success* and *Information* are True in an IF statement:

```
$ IF .NOT. $SEVERITY THEN GOTO ERRORMESSAGE
```

The ON command defines action to be taken when one of the even numbered results (*Warning*, *Error* or *Severe_Error*) occurs. Thus:

Format: $ ON *Level* THEN *Command*

Examples:

```
$ ON WARNING THEN CONTINUE                ! ie do nothing
$ ON SEVERE_ERROR THEN EXIT
```

If no ON is present when an error occurs, or a second error occurs without another ON command, the command procedure aborts. The command SET NOON disables error checking in command procedures (that is, $STATUS and $SEVERITY are still altered but not checked), allowing execution to continue even if errors occur, or the user to perform the error checks.

Examples:

```
$ SET NOON              ! disable checks in command procedure
$ RUN PROGA             ! If PROGA fails, the command file will not exit
$ RUN PROGB             ! PROGB runs even if PROGA causes an error
$ SET ON                ! Remember to reenable error checks

$ SET NOON
$ PASCAL 'P1            ! compile the file given as parameter
$                       ! Use the value of $STATUS as set by PASCAL to
$                       ! decide whether to link the OBJ file.
$ IF $STATUS THEN LINK 'P1
$                       ! Use the value of $STATUS as set by LINK (or
$                       ! by PASCAL if linking wasn't done) to decide
$                       ! whether to run the EXE file.
$ IF $STATUS THEN RUN 'P1
$ SET ON                eror checking enabled
```

11.2.3 Using SET CONTROL in Command Files

Ctrl-Y and Ctrl-C can be trapped within command procedures:

$ ON CONTROL_Y THEN GOTO *Label*

Ctrl-Y and Ctrl-T can be disabled or enabled:

$ SET [NO]CONTROL=(Y,T) ! No () needed if only one parameter

11.2.4 I/O in Command Procedures

The following commands are used to control I/O within command procedures. A list of examples is given at the end of the section.

OPEN Opens a file for reading or writing.

/READ	Read only (/READ and /WRITE are mutually exclusive)
/WRITE	Write only
/APPEND	Used with /WRITE to append records to an existing file
/ERROR=U	Label to jump to if an error occurs, eg: file not found

READ Read one record (in text files, one line) into a symbol

/ENDOFFILE	Gives a label to jump to when the end of file is reached

WRITE Write out one record

As in the case of the IF, the commands OPEN, READ and WRITE automatically perform symbol substitution (thus no apostrophes are needed).

Examples:

```
$ OPEN/READ/ERROR=NOFILE INFILE I.TXT   ! Equate INFILE to I.TXT
$ OPEN/WRITE/APPEND  OUTFILE  R.DAT     ! Append output to R.DAT

$ READ/ENDOFFILE=LABEL INFILE R         ! Read line from I.DAT into R
$ WRITE OUTFILE R                       ! Write (append) R into R.DAT

$ COUNT = 4
$ WRITE OUTFILE "Count=",COUNT,"."      ! Both lines write the string
$ WRITE OUTFILE "Count=' 'COUNT'."      ! "Count=4." into R.DAT
```

Writing to the terminal:

```
$ WRITE SYS$OUTPUT "Hello there, the time is ' 'f$time()' "
```

Reading from the terminal:

The following line will prompt 'Data:', read in a line of text, and equate it to the symbol TESTID:

```
$ READ SYS$COMMAND TESTID
```

11.3 Lexical Functions

Some Lexical Functions return process and system information; others are used to manipulate character strings. Type HELP Lexical for an in-depth description of all the functions (or see the 'Vax/VMS Guide to Using Command Procedures'). Only the simpler functions are given here.

The format of a lexical function is **F$NAME(*Parameter List*)**. The parentheses are required even when there are no parameters.

11.3.1 String Manipulation Functions

F$LENGTH (*String*)

Returns the length of *String*

```
$ Message :== "Hello"
$ SL = F$Length(Message)        ! SL = 5
```

F$LOCATE (*Substring,String*)

Locates *Substring* within *String*. Returns the length of *String* if *Substring* is not in *String* (useful for tests)

```
$ File :== "Login.Com;210"
$ NameLen = F$Locate(".",File)  ! NameLen = 5
```

F$EXTRACT (*Offset,Length,String*)

Returns a Substring of length *Length* starting at *Offset* in *String*.

```
$ Ext == F$Extract(6,3,File)    ! Ext = "Com" (using File above)
```

F$INTEGER (*Expression*)

Returns integer equivalent of *Expression*


```
$ A = "23"
$ I = F$Integer("-9"+A)         ! I = -923
```

F$STRING (*Expression*)

Converts integer *Expression* to string

```
$ A = 55
$ B = F$String(-2+A)            ! B = "53"
```

11.3.2 Other Useful Lexical Functions

F$TIME () Returns date and time string as "dd-mm-yyyy hh:mm:ss.cc"

F$MODE () Returns process execution mode - "INTERACTIVE", "BATCH" or "NETWORK". This is useful in creating command procedures which act differently when executing in batch mode.

```
$ IF F$Mode() .EQS. "BATCH" THEN GOTO BATSTUFF
```

F$PROCESS () Returns the current process name string (this is usually equal to the username, unless set by SET PROC/NAME="..." or a user is logged on twice).

F$DIRECTORY () Returns current default directory, including the directory brackets '[ ]'.

F$LOGICAL (*logical-name*)

Translates *logical-name* and returns the equivalent string (which may itself be a logical name). Note that the name must be included in quotes if it is not a symbol.

```
$ DEFINE/PROCESS TERMINAL 'F$Logical("SYS$COMMAND")'

$ if F$Logical("Infile") .eqs. "" Then Goto AssignFile
```

F$SEARCH (*filespec, [stream-id]*)

Returns as a string the full file specification matching the one given, which can have blank fields, wild cards, etc. *Stream-id* is any positive integer, and is used to maintain context for several concurrent searches.

```
$ Start:
$     Com = F$Search("SYS$SYSTEM:*.COM",1)
$     Dat = F$Search(".DAT",2)
$     Show Symbol Com
$     Show Symbol Dat
$     If (Com .eqs. "") .AND. (Dat .eqs. "") Then Exit
$  Goto Start
```

In this example, the SYS$SYSTEM directory is searched for .COM files, and the default directory for .DAT files. Each subsequent search finds the next file which matches the spec. (The null string is returned when there are no (more) matching files.