University of Edinburgh

Department of Computer Science

An Introduction to VAX/VMS

by

Natalie Royal

Internal Report CSR-93-81

James Clerk Maxwell Building,
The King's Buildings,
Mayfield Road,
Edinburgh,
EH9 3JZ.

September, 1981

CONTENTS

AN INTRODUCTION TO VAX/VMS

by

Natalie Royal

September 1981

The Digital Equipment Corporation's VAX-11/780 forms the principal computing resource 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 "VMS" (Virtual Memory System).

This guide gives a general introduction to the system and describes some of the more common commands and features.

Note: Commands marked with an asterisk (*) have been added locally and are not generally available on other VAX systems.

1. INTRODUCTION

1.1. Meeting The Terminal

VAX terminals can be found in Block O/P and in the dark room off the North Machine Hall. First of all, check that the terminal is switched on. The on/off switch can be situated in some very obscure places e.g. below the screen (Perkin-Elmer), below the keyboard (Tektronix), behind the terminal (Dacoll) and at the right-hand side of the terminal (Visual 200).

The terminal keyboard is used to communicate with the VAX system. The first key to become familiar with is the RETURN key (henceforth known as <CR>). This must be pressed at the end of every complete command line entered so that the command interpreter can start to process the line. There are other special function keys which should be noted, such as the DEL (or RUBOUT) key which deletes the last character entered on the line. The current line can be discarded by pressing the control (CTRL) key and the U key simultaneously. This is symbolised by CTRL+U. The control key has many other uses, which are summarised below:

CTRL+Y - Cancels an entire command and all those typed ahead. Can also be used to interrupt the system while it is executing a command.

CTRL+S - Temporarily halts further display of output at the terminal. Useful when the output exceeds the number of lines the terminal can display at one time to give the user time to read it.

CTRL+Q - Restarts the display of output which was stopped by CTRL+S

CTRL+R - Performs a carriage return and displays the current line, leaving the print position at the end of the line to continue typing input. (This is useful for terminals which cannot erase deleted characters, and instead print characters 'backwards' to show they have been deleted, leaving the line unreadable!)

CTR L+Z - Used to signal the end of input for data entered at the terminal, for example to terminate a message to be sent by the MAIL system.

1.2. Meeting The System

Before anyone can use the system they must identify themselves by entering their username and password. These are assigned to each user by the system manager. It is very important, for the protection of your data, programs and documents stored in the system, that you never reveal your password to anyone.

To get the attention of the system, press <CR>. There will be a prompt for a username and then for a password, each of which must be terminated by pressing <CR>. When a password is entered it is not displayed on the screen. An example of a log-in sequence might be:

```
Username: JIM  <CR>
Password:      <CR>
       Welcome to VAX/VMS Version V2.3
$
```

The "$" is a prompt for a new command and indicates that you have logged onto the system correctly. If an error is made in typing either the user name or password, then access can not be gained to the system. The only remedy is to try again.

Commands must be given to VAX to make it do useful work. This is done using the Digital Command Language (DCL). DCL commands are words from the English language which are suggestive of the function to be performed. The command format is described in Section 4. Commands are used to run programs and manipulate documents which are stored in files. Files are explained in Section 3.

One useful command to note here is the HELP command, which can be used to obtain information about any other command. For example:

```
$ HELP
```

will provide a list of all the commands available on the system. Note that the '$' is not typed by the user as this is the prompt from the system. For more information on a particular command, type:

```
$ HELP command
```

For example:

```
$ HELP PRINT
```

will provide a synopsis of what the PRINT command does and how to use it.

When the session with the terminal is completed, the user must log-off to make the terminal available to other users (and to protect personal files from accidental damage). Log-off by typing:

```
$ LOGOUT        or        $ LO
```

and the system will respond with a message similar to the following:

```
JIM logged out at 31-JUL-1981    11:05:45.40
```

2. FILES

2.1 How To Identify Files

A file is a collection of logically related data located on a medium, such as a disk, tape or floppy disk. To access files that already exist, or to give names to files created with system commands, the user must first know how to identify files.

The formal, unique specification of a single file is of the form:

device:[directory]filename.type;version

Although at first this may seem to be very complicated, certain defaults are used (see below) so that a file need not be given its full name in every case.

Device

This is the actual physical device on which the file is stored. Each user of the system is allocated an amount of storage space on a disk. This disk is the user's default disk and has the device name DRA0:, DRA1:, DRA2:, or DRA3: (note that device names are terminated with a colon). When referencing personal files or those belonging to others which reside on the same disk , the device name can be omitted and the system will use the default device name.

Directory

Since a disk can contain files belonging to many different users, each disk has a set of files called directories. A directory holds a list of the names of files on that disk belonging to a particular user. The name given to a directory is usually the same as the user's username. For example, the directory belonging to JIM on device DRA1: is called DRA1:[JIM]. If DRA1: was JIM's default device then this would be his default directory.

The default device and directory can be explicitly changed by using the command SET DEFAULT, described below. This is especially useful when using sub-directories (see Section 2.2). Note that the defaults are restored to their original values at the end of the terminal session. The command SHOW DEFAULT displays the current default device and directory.

If the file being referenced is located in the current default directory then the directory part of the file specification can be omitted.

Filename

The owner of the file gives it this name, consisting of a string of 1 to 9 alphanumeric characters, in order to distinguish it from others in the same directory. Usually the name is chosen so as to describe the contents. For example TEST1 may be the name given to a file containing data to be used for the first of a series of tests.

Type

The type of a file, sometimes called the extension, is an alphanumeric character string of maximum length 3 which must be preceded by a full-stop. It describes more specifically the kind of data in the file. For example a file of type .IMP contains the source statements of an IMP-77 program. Although any type can be given, the system recognises several default types used for special purposes. Some of these are given in the Appendix.

The type part of a file specification may be omitted when that file type is taken as default for the particular command involved. (See commands discussed later for specific examples.)

Version Number

This number, ranging from 1 to 32767, differentiates between versions of a file. The version number can usually be omitted from a file specification as the system assumes certain default values which are determined as follows:

1. When a new file is created the system assigns to it the version number 1.
2. For an input file, that is one to be acted upon by a command, the highest version number (most recent copy of the file) is used.
3. Whenever an existing file is updated in any way, or additional versions are created, the version number assigned to the new copy is the highest existing version number incremented by 1.

Example

DRA2:[FRED]NEWPROG.PAS;3

is the full specification of the third version of the file called NEWPROG which is a program of PASCAL source statements and can be found in FRED's directory on disk DRA2. In accordance with the defaults described above, the following command:

PASCAL NEWPROG

would cause the compilation of the program in the above file provided the current default directory was DRA2:[FRED].

SET DEFAULT- Changes the default device and/or directory name for the current process. The new default is applied to all subsequent file specifications that do not explicitly give a device or directory name. This command can be given as many times as required during a terminal session.

format: SET DEFAULT directory

examples: $ SET DEFAULT [JIM]
The directory name JIM is now assumed to be the default directory for subsequent file searches but the default disk device remains the same.
$ SET DEFAULT DRA0:[TIM]
Both defaults are changed.

2.2 Sub-Directories

Normally, the system manager provides each user with only one directory in which to maintain files. However VAX has a facility for a hierarchical structure of directories so that the files can be organised into suitable groups. This is achieved by using sub-directories which are created using the command CREATE/DIRECTORY (see Section 4).

A sub-directory is itself a file in the main directory, and has the file type .DIR, for example:

DRA1:[FRED]SUB.DIR;1

A file within a sub-directory is manipulated in the same way as those in the main directory, except that the directory name part of its specification consists of two components separated by a full stop. The first component is the main directory and the second is the sub-directory's name.

e.g. [FRED.SUB]

The full-stop thus has two uses in file specifications:

(i) To separate the file name and file type.
(ii) As here, to separate a directory name and the name of a sub-directory it contains.

Note that what might appear to be the more logical form of sub-directory identification, namely [[FRED]SUB], is NOT correct.

A sub-directory may itself contain sub-directory files

e.g. [FRED.SUB1]SUB2.DIR

which in turn may contain files which will have the directory name of

[FRED.SUB1.SUB2]

The number of these levels can not exceed 8 but there is no maximum on the number of such hierarchical structures you can create and access beginning with your own directories.

Sub-directories aid file management because programs and data files on different exercises can be grouped together in their own sub-directory. This prevents the main directory from getting too long, making it difficult to scan. It is usually convenient when accessing files in a sub-directory to change the default directory name to that sub-directory. For example:

```
$ SET DEFAULT [FRED.SUB]
```

The sub-directory [FRED.SUB] is now taken as the default for all subsequent file references in which the directory name is not explicitly specified.

2.3 Protection

Individual files are protected by means of a protection code which indicates who is allowed access for what purposes. The following four categories of user are defined:

SYSTEM - The System Manager.

OWNER - Self explanatory.

GROUP - Users who are in the same user "group" as the owner of the file. Each user is allocated to a particular group (e.g. CS2, STAFF, etc.) by the system manager when he/she is accredited to the system.

WORLD - All users who do not fall into one of the above categories.

Each of these categories of user can be allowed or denied one of the following types of access:

READ - the right to examine, print or copy a file.

WRITE - the right to modify a file.

EXECUTE - the right to execute files that contain executable images.

DELETE - the right to delete a file.

Since a directory is itself a file, it can be protected in the same way. In this case, the above types of access are interpreted respectively as the right to read files from the directory, write files to the directory, create files in the directory, and delete files from the directory.

The command used to specify these protection codes is:

SET PROTECTION - Establishes the protection to be applied to a particular file, or a group of files, or establishes the default protection for all files subsequently created during the terminal session.

format: SET PROTECTION [=code] [filespec]

example: $ SET PROTECTION=(GROUP:RWED, WORLD:R)/DEFAULT
This command sets the default protection, which is then applied to all files subsequently created, to allow users in the same group unlimited access, and all other users read access only. The default protection for system and owner are not changed.

$ SET PROTECTION=WORLD MYFILE.IMP
Here, the users in the world category are denied all access to the file MYFILE.IMP, and the access allowed to other users is left unchanged. The type of the file must be specified. As the version number is omitted above, the protection applies to only the highest existing version of the file. Type HELP SET PROTECTION for more details.

3. USING THE COMMAND LANGUAGE

3.1 Command Format

The general format of a DCL command is:

command-name [qualifiers] [parameter-1] ... [parameter-n]

Here, and in subsequent command descriptions, the square brackets [and] surround optional values. For example:

$ COPY [qualifiers]

indicates that the user does not need to supply any qualifiers to issue a valid COPY command.

Qualifiers

A qualifier modifies the action of a command, that is it provides the system with additional information on how to execute the command. A qualifier always begins with a slash (/), and its position in the command line depends on its type. There are two types - a command qualifier and a file qualifier. A command qualifier is placed immediately after the command, and affects each of the command parameters which follow. A file qualifier, on the other hand, follows a file specification in the parameter list and affects only that file.

If there is a value associated with a qualifier, then the two are separated by an equal sign (=) or a colon (:) as in the example below. This shows the use of the command qualifier /COPIES=n for the PRINT command, which, in this case, causes 2 copies of both MYFILE.DAT and PROG.PAS to be printed.

$ PRINT/COPIES=2 MYFILE.DAT,PROG.PAS

Although there are qualifiers associated with most commands, they are optional as each command has a default action. For example, the default action of the PRINT command is to print 1 copy of each specified file.

Parameters

Command parameters define what is to be acted upon by the command. In many cases they are specifications of files to be manipulated. Several DCL commands relating to files allow a parameter consisting of a single input file specification to be replaced by a list of the same. Each parameter, whether or not it actually consists of a list, is separated from the command name and each subsequent parameter by at least one space. For example the following command:

$ COPY FILE1.PAS,FILE2.PAS FILE3.PAS

has 2 parameters - a list of 2 input files and an output file. The effect of this particular command is to produce a new version of the file FILE3.PAS consisting of the FILE1.PAS followed by FILE2.PAS. The PRINT command in the example in the above section has 1 parameter - a list of 2 input files.

3.2 Entering A Command

A command can be entered whenever the "$" prompt appears at the terminal. This indicates that the system is ready to accept a command. If the execution of a command is taking a long time, then subsequent commands may be entered before the "$" reappears, but will not be displayed on the screen until the execution of the previous command has been completed. (Note, however, that the unsuccessful completion of a command may affect the result of any following related commands.)

A long command can be carried onto the next line by using a hyphen (-), which indicates this continuation and can be used at any point within the command. The system then prompts the next line with '$_' For example:

```
$ COPY FILE1.PAS,-
$_     FILE2.PAS FILE3.PAS
```

In general, if a command is entered without specifying the required parameters, the system prompts for the additional data it requires, as shown below:

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

Temporary Defaults

The system uses temporary defaults for the execution of commands which contain a list of file specifications as input. These defaults are applied to the device name, directory name, file name and file type of file specifications. Thus the command:

```
$ DELETE MYFILE.LIS;1, MYFILE.LIS;2, MYFILE.PAS;1
```

could be written as:

```
$ DELETE MYFILE.LIS;1, ;2, .PAS;1
```

Another example is as follows:

```
$ PRINT DRA1:[JIM]TEST1.DAT, -
$_ TEST2, -
$_ [FRED]SUMMARY.TST, -
$_ DRA2:FINAL
```

would cause the following files to be printed:

```
dra1:[jim]test1.dat
dra1:[jim]test2.dat
dra1:[fred]summary.tst
dra2:[fred]final.tst
```

Wild Cards

Many DCL commands accept a 'wild card' (*) in place of one or more of the directory, file name, file type or file version fields of a file specification. When a wild card replaces a field in an input file specification, the command is applied to all files whose identifications satisfy the fields that are specified. For example:

```
$ DELETE *.LIS;*
```

will delete all listing files, and

```
$ PRINT MYFILE.*
```

will print the latest version of each type of the file 'MYFILE'. When a wild card replaces a field in an output file specification, the system uses the corresponding field in the input file specification to fill in that field of the output file.

Truncating Commands

When entering commands or qualifiers, the full command or keyword name need not always be entered. The rule to follow is that enough characters must be typed so as to make the command or keyword unique. For example, the HELP command is the only command that begins with the letter 'H' so has a minimum truncation of 1 letter. The commands SET and SHOW however can not be truncated to less than 2 letters. An exception to the rule is the command RUN which can be abbreviated to 'R'.

Not that this rule does not apply to user-defined commands (see Section 7 for Foreign Commands).

3.3 System Responses

The system responds to some commands by giving information about what it has done. For example, after a print command the system displays the job identification number it assigned to the print job:

```
$ PRINT MYFILE.LIS
Job 210 entered on queue TTF7
```

Not all commands display informative messages - in fact the successful completion of a command is most commonly indicated by a "$" prompt for another command. Non-successful completion is always indicated by an error message or messages.

Errors

If a mistake is made in entering a command the system will respond with an error message and the faulty line, with the error enclosed by \...\ . It will then prompt for another command as if none had been entered. For example:

```
$ CODY OLDFILE.IMP NEWFILE.IMP
%DCL-W-IVVERB,  unrecognised command
\CODY\ OLDFILE.IMP NEWFILE.IMP
$
```

The three-part code preceding the descriptive part of the message indicates that the message is from DCL, the command interpreter; that it is a warning (W) message; and that the mnemonic for this particular message is IVVERB.

Error messages can also be received during the execution of a command if the requested function can not be performed. For example, if a PRINT command is typed correctly, but the specified file does not exist, the PRINT command gives the following information:

```
$ PRINT NOFILE.DAT
%PRINT-W-OPENIN, error opening DRA1:[JIM]NOFILE.DAT
as input
-RMS-E-FNF, file not found
```

The first message is from the PRINT command which says it can not open the file. The second message indicates the reason - that the file cannot be found. The facility name in this message, RMS, is the VAX/VMS file system. For information on how to reduce the amount of error message that is printed out, type HELP SET MESSAGE.

4. FILE MANAGEMENT

4.1 Editing A File

The edit commands described below can be used to create new files or to update existing files. The default file types for file-specifications in each of these commands is .IMP.

E * - invokes the VAX implementation of ECCE, the Edinburgh Compatible Context Editor. (see Reference 4)

format: E filespec — updates an existing file or creates a new file if it does not already exist.
E old new — edits an existing file to a new file.

The default file type for the 'E' command can be changed using the command EXT.

example:
```
$ E MYPROG
```
Updates the file MYPROG.IMP. The file will be created if it does already exist.

```
$ EXT .PAS
$ E TESTPROG TEST1
```
Edits the file TESTPROG.PAS to TEST1.PAS.

V * - invokes the screen-oriented version of ECCE.

format: V filespec — updates an existing file.
V NL: new — creates a new file.
V old new — edits an existing file to a new file.

S * - invokes the screen-editor 'S' for Visual 200 terminals.

format: S filespec — updates an existing file.
S NL: new — creates a new file.
S old new — edits an existing file to a new file.

S accepts single key-stroke commands as follows:

On the main keyboard:

letters, digits, etc all replace the character under the cursor.

DEL — Deletes the character to the left of the cursor (even newline).

RETURN — Moves to the left margin or inserts newline if inserting.

LF — (line feed) Repeats last text search command.

TAB — Moves to next tab stop, SHIFT+TAB to last tab stop (or tab on words).

BS — Retrieve last line deleted and insert before current line.

On the small keypad to the top right:

ARROWS — Move the cursor in the direction indicated; moving from one line to another normally also moves to the first (non-space) character on the line.

HOME — Toggle whether the cursor should follow the indentation when moving between lines.

CP+CONVERT FUNCTION — Abandon the edit without altering the original file.

On the keypad to the right (no CONVERT FUNCTION is necessary):

CT — Clears all tabs.

ST — Sets a tab stop at the current cursor position.

IL — Inserts one or more lines (until next cursor move etc key hit).

IC — Inserts characters (without overwriting) until next cursor move etc.

DL — Deletes the line containing the cursor.

DC — Deletes the character under the cursor.

EL — Erases the rest of the current line.

EP — Move to next page.

EF — Move back to previous page.

PRT — Repaints the screen should it be suspiciously out of step.

\- — Rewind to top of file (- on keypad only).

0 — Move to end of file (0 on keypad only).

CPY — Closes the edit.

BS — Insert the last deleted line before the current line.

ENTER — Accept a command line of the form:

@number	move to line number.
@-number	move back number lines.
text	move to the next occurence of text (which may not start +-@#).
+text	move to the next occurence of text.
-text	move to the last occurence of text.
+	move to the next occurence of the same text as last time.
-	move to the last occurence of the same text as last time.
nothing	ignore (the ENTER was a mistake).
#W	cause TAB key to be interpreted as skip to start of next word.
#	cause TAB key to revert to skipping to next tab stop.

4.2 File Handling Commands

APPEND - This command adds the contents of one or more specified input files, to the end of a specified output file.

format: APPEND inputfile,... outputfile

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

examples: $ APPEND BOTTOM.IMP TOP
TOP.IMP now contains the same as it did before, but with the contents of BOTTOM.IMP joined onto the end.

$ APPEND BOTTOM.IMP;* TOP
This command joins all versions of BOTTOM.IMP onto the contents of TOP.IMP, starting with the most recent version. If more than one input file is specified, these must be separated with commas (,) or plus signs (+).

CLEAN * - This command invokes a program which allows you to 'clean up' your files. The name of each file you specify is displayed on the screen in turn, to which you can reply with special one-letter commands. These commands allow you to perform many different operations on each file, such as renaming it, deleting it, or changing the protection. Type 'h' for information about these commands.

format: CLEAN [filespec]

examples: $ CLEAN
Each file in the current default directory is 'cleaned'.

$ CLEAN [JIM]
Each file in Jim's directory is 'cleaned'.

$ CLEAN *.LIS;*
Each listing file in the current default directory is 'cleaned'.

COPY - This command is used to copy one file to another file, or to copy several files into a single file.

format: COPY inputfile,... outputfile

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

/REPLACE
/NOREPLACE
Requests that if a file already exists with the same file specification as the output file, the existing file is to be deleted. The COPY command allocates new space for the output file. The default is NOREPLACE.

By default, the COPY command creates a new version of a file if the file already exists, incrementing the version number.

examples: $ COPY DRA1:[FRED]FILE.IMP MYFILE
Copies FILE.IMP from the directory FRED to MYFILE.IMP in the user's directory.

$ COPY [FRED]FILE1.IMP,FILE2 NEWFILE
Copies two files from directory FRED to a single file in the users directory. The file NEWFILE is a concatenation of the other files. The comma may be replaced by a plus sign (+) with the same result.

CREATE - This command is used to create a file or a sub-directory.

format: CREATE filespec

qualifiers /PROTECTION=code
Defines the protection to be given to the file or sub-directory.

/DIRECTORY
Indicates that a sub-directory is to be created.

examples: $ CREATE TESTDATA.DAT
The file TESTDATA.DAT is created - if it already exists then the version number is incremented by one, otherwise it is given the version number 1. The contents of the file can now be typed in from the terminal. Terminate this input with CTRL+Z.

$ CREATE/DIRECTORY [JIM.SUB]
The sub-directory [JIM.SUB] is created.

DELETE - This command is used to delete one or more files from the user's directory. All fields in the filename must be specified. If an * is used in any field, then all files that match the other fields will be deleted.

format: DELETE filespec,...

qualifiers /CONFIRM
/NOCONFIRM
Controls whether the DELETE command displays the file specification of each file before deleting, and requests you to confirm whether or not the file should actually be deleted. You should reply with a 'Y' if you want that file to be deleted. The default qualifier is /NOCONFIRM.

/LOG
/NOLOG
Controls whether the DELETE command displays the file specification of each file that it deletes. The default qualifier is /NOLOG.

examples: $ DELETE FILE.LIS;3, EXE;2
This command deletes the two files FILE.LIS;3 and FILE.EXE;2 from the current directory.

$ DELETE PROG.*;*
Deletes all types and versions of files with name PROG.

DIRECTORY - This command displays a list of files or information about a file or group of files.

format: DIRECTORY [filespec,...]

qualifiers /FULL
Lists all information about the specified files, including the protection on each file, the owner number and storage allocation.

examples: $ DIRECTORY
Displays a list of all files in the current default directory.

$ DIRECTORY/FULL [JIM.SUB]MYFILE.PAS
Displays all the information about each version of the file MYFILE.PAS in the sub-directory [JIM.SUB].

$ DIRECTORY *.IMP.*
Lists all the IMP files in the current default directory.

FILES * - This command prints out a list of the specified files. No information other than the file names is displayed.

format: FILES [filespec]

examples: $ FILES
The names of all the files in the current default directory are listed.

$ FILES DRA1:[FRED]
A list of FRED's files is printed at the terminal.

$ FILES [JIM]PROG
Causes a list of Jim's files with file name PROG to be displayed.

PRINT - Queues one or more text files for printing on the printer situated in Block O/P. This command uses the default file type of .LIS, if a file type for the first input file is not specified.

format: PRINT filespec,.....

qualifiers /HEADER
/NOHEADER
Controls whether the name of the file is printed at the top of each output page. The default qualifier is /NOHEADER.

/COPIES=n
Gives the number of copies to be printed. The default number is 1.

examples: $ PRINT FILE1,FILE2
This will cause FILE1.LIS and FILE2.LIS to be queued ready for printing. The comma may be replaced by a plus sign.

$ PRINT/COPIES=2 PROG.LIS;*
Prints two copies of each version of the file PROG.LIS

FS PRINT *- This command queues a text file to be printed on the filestore line printer, situated in the South Machine Halls. As this printer is faster than the one in block O/P, it should be used when printing out a long listing. The default file type is .LIS.

format: FS PRINT filespec

PURGE - This command is used to delete all but the most recent version of the specified files.

format: PURGE [filespec,.....]

qualifiers /KEEP=n
Specifies n as the maximum number of versions to retain. The default value of KEEP is 1.

/LOG
/NOLOG
Controls whether the purge command displays the file specification of files as it deletes them. The default qualifier is /NOLOG.

examples: $ PURGE
All but the most recent version of all files in the current default directory are deleted.

$ PURGE PROG.*
All but the most recent versions of all files called PROG are deleted.

RENAME - Used to change the name, type and version number of the specified file.

format: RENAME oldfilespec newfilespec

qualifiers /NEW_VERSION
/NONEW_VERSION
Controls whether the RENAME command automatically assigns a new version number to the output file, if a file with the same name and file type already exists. The default qualifier is /NONEW_VERSION.

examples: $ RENAME OLD.LIS NEW
This changes the file name of the latest version of OLD.LIS to NEW.LIS. If no file called NEW.LIS exists then the new file is given a version number of 1.

$ RENAME *.TXT;* *.OLD;*
All versions of all files with file types of TXT, to have file types of OLD. The file names and version numbers remain unchanged.

TIDY * - This command has the same effect as PURGE, plus all version numbers are returned to one.

TYPE - This command is used to display the contents of a text file on the current output device i.e. the terminal. If the file is longer than 24 lines, use CTRL+S and CTRL+Q to stop and start the display. The default file type is .LIS.

format: TYPE filespec

example: $ TYPE PROG
Displays the most recent version of PROG.LIS.

5. PROGRAM DEVELOPMENT AND EXECUTION

The source statements of a program are put into a file which is created using an editor.(See Section 4.) It is usual to give the file a name which describes the function performed by the program. Before a new version of a program can be executed it must go through two stages:

5.1 Compilation

This is the stage where the source program is checked by a compiler program for syntax and programming errors. If there are no errors in the program, it is translated into a binary form called object code. The translated code, that is the object module, is written into a file of type .OBJ. If there is a fault in the program then a message giving the type of fault and the line number on which it occured is displayed on the terminal. In this case no new .OBJ file is created.

For each high-level language there is a different compiler which can be invoked by its own command. For example:

IMP * -This command invokes the IMP-77 compiler. (See Reference 6). The default file type is .IMP.

format: IMP filespec

example: $ IMP IMPROG
The latest version of IMPROG.IMP, if error-free, is compiled and the object code which is produced is placed in a new version of IMPROG.OBJ.

PASCAL -This command invokes the PASCAL compiler. The default file type is .PAS.

format: PASCAL filespec

example: $ PASCAL PASPROG
The latest version of PASPROG.PAS, if error-free, is compiled and the object code which is produced is placed in a new version of PASPROG.OBJ.

Creating A Listing

The qualifier /LIST can be used with both of the above commands. Like other qualifiers it is optional, but when included it tells the system to create a compiler listing which it puts in a file with file type .LIS e.g.

```
$ IMP/LIST IMPROG
$ PASCAL/LIST PASPROG
```

produce new versions of IMPROG.LIS and PASPROG.LIS respectively. Compiler listings are helpful because they contain a copy of the program, with line numbers, and indicate any faults occurring in it.

See the HELP information for other qualifiers.

5.2 Linking

An object module is not, in itself, executable, as generally it contains references to other programs or routines. The linker combines these modules to produce an executable image, which is put into a file with the type .EXE. Files containing external procedures or other modules which are explicitly referenced by the object module, are included as parameters in the link command. There is no limit to the number of such files.

Linking IMP Programs

RLINK *
SLINK * -These commands are used to link compiled IMP programs with the appropriate IMP library procedures before they can be executed. SLINK rather than RLINK should be used when input or output streams other than the terminal are required.

format: RLINK filespec,....
SLINK filespec,....

The first file specification is the object module, that is the compiled program. Others, if present, refer to files containing external procedures or modules. The default file type is .OBJ

example: $ RLINK IMPROG
IMPROG.OBJ is linked to the IMP system library and a new version of the file IMPROG.EXE is created.

$ SLINK NEWPROG,EXROUT
The files NEWPROG.OBJ and EXROUT.OBJ are linked, and a new version of the file NEWPROG.EXE is created.

Linking Other Programs

LINK -This command is used to link programs written in languages other than IMP - e.g.PASCAL.

format: $ LINK filespec,...

The first file specification is the object module, that is the compiled program. Other file specs, if present, refer to files containing external procedures or modules. The default file type is .OBJ

example: $ LINK PASPROG
The file PASPROG.OBJ is linked to the system library and a new version of PASPROG.EXE is created.

$ LINK MYFILE, EXROUT
The files MYFILE.OBJ and EXROUT.OBJ are linked, and a new version of MYFILE.EXE is created.

Object Module Libraries

Object module libraries can be created to store procedures that are called frequently by many programs. For example, the compiled object modules named TIMER, CALC, and SWITCH can be put into a library called MYLIB as follows:

```
$ LIBRARY/CREATE  MYLIB  TIMER,CALC,SWITCH
```

The library is given the default file type of .OLB. Suppose an object module named TESTPROG.OBJ calls at least one of these routines. Then before it can be executed it must be linked using the command below, which causes the linker to search the library MYLIB.OLB automatically when it encounters any undefined external references:

```
$ LINK TESTPROG,MYLIB/LIBRARY
```

5.3 Executing A Program

When a program has successfully completed the above two stages, it is ready to run.

RUN -This command causes a program to be executed. The default file type is .EXE

format: RUN filespec

example: $ RUN MYPROG
The latest version of the image MYPROG.EXE is executed.

Input And Output Streams For IMP Programs

When running an IMP program which was linked using the command SLINK, there will be a prompt to provide file-specifications for the input and output streams used by the program. The default file type is .IMP. The response should have the following format:

```
input-filespec,....   output-filespec,....
```

The file specifications entered are interpreted as defining the input streams 1,2,.... and output streams 1,2,.... The default assignments to the 16 possible input and output streams are as follows:

	Input	Output
0	terminal	terminal (Cannot be re-assigned)
1	terminal	terminal
2-15	null stream	null stream

The terminal or null stream can also be assigned to a stream by entering TT or NL: respectively in place of a file specification. Initially, stream 1 is selected for input and output.

There are no prompts for streams when running programs linked by the commands RLINK or LINK.

A running program may be halted by typing CTRL+Y. The execution of an IMP program can also be suspended, by pressing CTRL+C. In this case, the prompt Int: will appear at the terminal. A reply of '?' will cause diagnostics to be displayed at the terminal, but the execution will continue. A reply of '.' will halt the execution altogether. If any other reply is given, it is stored in the external string CONSOLEINT which may be accessed from the program if required, and the execution will continue.

Run-Time Faults

If, for any reason, a program fails during execution, then the system will output diagnostics giving the reason for its failure. These diagnostics will consist of the number of the line at which the program failed along with the name of the procedure it was executing at the time. The diagnostics produced for an IMP program are somewhat better than those for PASCAL or FORTRAN, as the value of each variable at the time of failure is also given.

6. COMMUNICATING WITH OTHER USERS

VAX has facilities for inter-user communication. The commands for this are explained below:

NOTIFY * - Used to send a short message to a user who is currently logged-on

format: NOTIFY username " message

example: $ NOTIFY FRED"time for coffee?
FRED is in block o/p #1

The message "time for coffee?" will appear at FRED's terminal along with your name (therefore no anonymous messages!).

FIND * - Used to obtain information on the location of the specified user. If that user is not logged on then the system will respond with a message containing this fact.

format: FIND username

example: $ FIND FRED
Fred Smith (FRED) is in block o/p #1

This shows that FRED is currently logged on and is using terminal no. 1 in block O/P.

GONE * - This command may be used as an alternative to LOGOUT when logging off. It can provide extra information to any user using the FIND command to obtain your location.

format: GONE message

example: $ GONE FISHING
Fred Smith (FRED) going fishing at 11:28, Wednesday

User FRED has now logged off. If another uses the FIND command to find FRED, he will get FRED's message. For example:

$ FIND FRED
FRED went fishing at 11:28, Wednesday

USERS * - This command will cause a list of current users and the terminals they are using to be displayed at your terminal.

format: USERS

NAME * - This command is used to find the name of another user when only part of the name - for example the initials or first name - is known.

format: NAME text

A list of user's names for which a match is found is displayed on the screen.

6.1 The Mail System

Mail can be sent to other VAX users via the mail system, which is invoked by the command:

```
$ MAIL
```

The system prompts with:

```
MAIL>
```

and will then accept special commands that enable you to send mail to other users and to forward, delete, read, reply to, file and print mail you receive. Some of the mail commands are as follows:

SEND - Sends a file or a message to another user. The sender is prompted for the username(s) of the recipient(s), and the subject of the message. If no file is specified after the SEND command, the user is then prompted for the text of the message, which is terminated by CTRL+Z. Note that mail can be sent to users on the network - see Section 10.4 for details.

READ - Displays one page of a message.

DELETE- Deletes the message which is currently being read.

DIR - Displays a list of the messages which have not yet been deleted.

Type HELP for more information about these commands and others.

Any mail that is sent to you is put into the file MAIL.MAI in your directory, which is given protection which prevents you from deleting it, in case you do so by mistake. This file is deleted when there are no more messages left in it.

An alternative way to mail a file to another user is to use the MAIL command as follows:

```
$ MAIL filespec username
```

e.g. `$ MAIL NOTICE.IMP JIM`

The default extension for the file is taken to be .TXT.

7. TAILORING THE COMMAND LANGUAGE

7.1 Assignment Statements - Creating Synonyms For DCL Commands

An assignment statement defines a symbolic name, or 'symbol' for short, for a character string and in this way allows synonyms to be defined for commands. The format of an assignment statement is as follows:

```
symbol-name  := [=] string
```

The 'symbol-name' is an alphanumeric string of 1 to 255 characters, which must begin with a letter. If a single equal sign (=) is specified in the assignment, the symbol is put into the local symbol table for the current command level, and if a double equal sign is used it is put into the global symbol table. A global symbol is recognised in any command file and at the interactive level at the terminal - that is at any command level - whereas a local symbol is recognised only at the command level at which it was defined.

The 'string' part of the assignment specifies a character string value (or expression resulting in a character string value) to be equated to the symbol. The string must be enclosed in quotation marks (") if it contains any multiple blanks or tab characters, lowercase letters, an exclamation mark or quotation marks.

In the example below:

```
$ TIME := SHOW TIME
$ TIME
23-JUL-1981  9:50:12
```

The symbol TIME is put into the local symbol table with the string SHOW TIME as its value. The word TIME can then be entered as if it were a system command. The system substitutes the string SHOW TIME for the symbol TIME and then executes this command.

The following assignment:

```
$ LIST := DIRECTORY/FULL
```

defines a synonym for the DIRECTORY command that automatically includes the /FULL qualifier. Then, if you issue the command line:

```
$ LIST MYFILE.DAT
```

the system substitutes the name DIRECTORY/FULL for the symbol LIST and executes the command string DIRECTORY/FULL MYFILE.DAT.

When you log out, any symbols assigned at 'terminal level' during that session will be lost.

Concatenating Symbols

If a symbol-name appears as the first word in a command string, the command interpreter automatically substitutes it with its string value. Symbols can also, however, be concatenated with other symbols or items on a command line, if they are enclosed in apostrophes (') to indicate to the system that it must perform symbol substitution. For example:

```
$ PQUALS := /COPIES=2/HEADER
$ PRINT REPORT.DAT'PQUALS'
```

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

```
PRINT REPORT.DAT/COPIES=2/HEADER
```

Foreign Commands

The assignment statement can also be used to define a 'foreign command' as follows:

```
$ symbol-name:= [=] $image-file-spec
```

After this assignment, whenever the symbol-name appears as the first item in a command, the command interpreter automatically executes the specified image. For example:

```
$ PROCESS := $DRA1:[JIM.PROJ]PROCESS
$ PROCESS
```

Here, the symbol-name PROCESS is defined as a foreign command - that is one that was previously not recognised by the command interpreter - and the image PROCESS.EXE is executed. The dollar sign ($) preceding the image-file-specification is necessary, as it implies a request to execute the image. A file type of .EXE and the highest version number are taken as default in the file specification, but a device, directory and file name must be given.

Variable data can be passed at execution time to IMP programs which are executed as foreign commands using the external string function CLIPARAM. In the example below:

```
$ PROCESS DATA TEST
```

the parameter string 'DATA TEST' which follows the command is fetched by the function CLIPARAM, and can be parsed and analysed within the program as required. Note that if there are any symbols (which will be enclosed in apostrophes) in the parameter string, substitution of these symbols occurs before the resulting string is passed to the program.

7.2 Command Files

A command file is a file that contains a sequence of DCL commands. By placing sets of frequently used commands in command files, you can construct command language 'programs' from DCL commands.

The default file type for a command file is .COM, and each command in it must begin with a dollar sign. Any data required by a command in the file - for example specifications for input and output files - immediately follows the command and does not begin with a dollar sign. In this way the data can be distinguished from the commands. A file named MYPROG.COM might look like the following:

```
$PASCAL MYPROG
$LINK MYPROG
$RUN MYPROG
```

These three commands can be executed, in the given order, by issuing the command:

```
@MYPROG
```

When this command is executed, the system locates the file MYPROG.COM, then reads each command line in the file and executes it.

One command file can invoke another with the @ command to give different command levels. The maximum number of command files that can be nested in this way is eight.

A command file can contain special DCL commands to conditionally control the execution of the file. For a detailed explanation of these commands, see Reference 1.

Generalising Command Files

The sample command file shown above is not very flexible, as it can be used to compile, link and run only the PASCAL program called MYPROG. Command files can be made much more general by using symbols, and letting the system substitute the symbol's value while it executes the file. For example, suppose that the command file DOPAS.COM contained the lines:

```
$PASCAL 'PROGRAM'
$LINK 'PROGRAM'
$RUN 'PROGRAM'
```

then this file could be used to compile, link and execute any PASCAL program if, before executing the file, an assignment statement is given to define a value for the symbol PROGRAM. For example:

```
$ PROGRAM :== MYPROG
$ @DOPAS
```

will have the same effect as the command file MYPROG.COM above. Note that a double equal sign is used, as the symbol PROGRAM is to be recognised at a different command level.

An alternate way to generalise a command file is to make use of special symbols that the system defines automatically when you execute the file. These symbols, called parameters, are named

P1,P2,...P8 and the values given to them are specified on the @ command line.
For example, the above file DOPAS.COM would look like this:

```
$PASCAL 'P1'
$LINK 'P1'
$RUN 'P1'
```

The value for the symbol P1 is defined when executing the command file, as follows:

```
$ @DOPAS  MYPROG
```

The system automatically equates the name MYPROG to the symbol P1, and the null string to each of the symbols P2,...P8.

Redefining System Commands

Command files and assignment statements can be used together to redefine and expand system commands. For example, you may wish to purge listing files on a regular basis to keep your disk file directory uncluttered. To do this, you could create a command file, say OUT.COM, that contains the following lines:

```
$PURGE *.LIS
$LOGOUT
```

You can use this command file in place of the LOGOUT command to end the terminal session, as follows:

```
$ @OUT
```

Then, the PURGE command is automatically executed before you are logged out of the system. Moreover, you could define a symbol named OUT that is equated to the following command string:

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

Then, when you type the command line:

```
$ OUT
```

the system substitutes the symbol OUT with the @OUT command string, and executes your command file.

The LOGIN.COM File

Each time you log in to the system, the command interpreter looks for a file called LOGIN.COM in your default directory, and if it exists then automatically executes it. This file can thus be used to execute any commands or sequences of commands that you would normally want to execute at the start of each terminal session. For example, a LOGIN.COM file could contain the following statements:

```
$TIME :== SHOW TIME
$LIST :== DIRECTORY/FULL
$OUT :== @OUT
$TEST :== SET DEFAULT [JIM.TESTFILES]
$PROG :== $DRA1:[JIM]PROG
```

Note that the above symbols are defined as global symbols, so they will not be deleted when the file finishes executing.

A LOGIN.COM file can also contain commands to set up terminal characteristics, assign logical names, run programs, execute command files or display message files.

You can update your LOGIN.COM file at any time to change, add or delete commands. After you first create the file, or after updating it, you can execute it with the @ command so that the commands it contains become effective.

7.3 Batch Job Processing

A lot of time can be wasted in sitting at the terminal waiting for a command or command file to finish executing, when they require a lot of processing time - for example the compilation of a large source program. Instead, these can be submitted for execution as batch jobs, which are put onto a queue and executed in due course, leaving the terminal free for you to continue interactive work.

A command file is submitted as a batch job using the following command:

SUBMIT -Enters one or more command files in the batch queue. The default file type is .COM.

format: SUBMIT filespec1, filespec2,...

The system displays a message indicating that the job was successfully queued for processing, and gives the job identification it has assigned to the job. All output from the command file is written to a file of type .LOG in your default directory, which is queued for printing then deleted when the batch job completes.

example:
```
$ SUBMIT TEST
Job 112 entered on queue SYS$BATCH
```

The file TEST.COM is entered on the batch job queue. When the job has been processed, the file TEST.LOG is queued for printing and then deleted.

A single command can be submitted as a batch job using the command DO. For example:

```
$ DO RUN TEST1
```

In this case, system messages are put into the file DOJOB.LOG in your default directory.

8. LOGICAL NAMES

Logical names are used to keep programs and command files independent of physical file specifications. They also provide a convenient shorthand way to specify files that are used frequently. (Note - Do not confuse logical names with symbolic names which normally relate to command definitions.)

To achieve this independence, input and output files are referenced from the program according to the syntax requirements of the programming language. Then after the program is compiled and linked, but before running it, the connection is made between the logical names used in the program and the actual files and devices.

The ASSIGN command makes this connection: it establishes a correspondence between a logical name (the one used in the program) and an equivalence name (the actual file or device used). The program can then be run with different input and output files without having to re-code it, or having to type in the appropriate streams at run-time. Instead, an ASSIGN command is executed to change the current equivalence of the logical name. This method of independence is particularly useful when the same files are used 'most of the time' as input or output to a program. In this case, as logical names lose their value at the end of the terminal session, it is a good idea to keep the assign commands within the LOGIN.COM file.

ASSIGN - Equates a logical name to a physical device name or file specification, and places the pair of names in a logical name table which is maintained by the system.

format: ASSIGN equivalence-name[:] logical-name

The equivalence name can be one of the following: a device name, a device and directory name, a complete file specification or another logical name. It must be followed by a colon when it is either a device name on its own or a logical name which translates to a device name. The logical name is a 1 to 63 character string which is to be associated with the given device or file.

example: Suppose the IMP program DATES.EXE contained statements to read and write to files as follows:

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

The following commands show how the same program can be used with different files as input and output:

```
$ ASSIGN JAN.DAT INFILE
$ ASSIGN JAN.OUT OUTFILE
$ RUN DATES
$ ASSIGN FEB.DAT INFILE
$ ASSIGN FEB.OUT OUTFILE
$ RUN DATES
```

The system uses the default disk, directory and version numbers to complete the above file specifications.

8.1 Logical Names In Commands

A logical name can, as already mentioned, be used as shorthand for a device name, device and directory name or a complete file specification. Commands that read or write files, therefore, also accept logical names for file specifications. For example:

```
$ ASSIGN DRA2:[BERT.PROJECT.TESTS]TEST1.EXE  TEST
$ RUN TEST
```

When the system processes the above RUN command, it replaces the logical name TEST with its equivalence name and executes the program DRA2:[BERT.PROJECT.TESTS]TEST1. In the following example:

```
$ ASSIGN DRA0:  TEMP
$ ASSIGN TEMP:  FULL
$ ASSIGN FULL:[FRED]  FREDDY
$ PRINT FREDDY:PROG.IMP
```

the translation from logical name to equivalence name occurs three times in the PRINT command, and the file DRA0:[FRED]PROG.IMP is printed. Note the necessary use of colons - the reason for this is that the system always checks for a logical name to the left of a colon in a file specification.

8.2 Displaying And Deleting Logical Name Table Entries

The SHOW LOGICAL command displays current entries in the logical name table. For example, to display the logical name TEST the following command should be entered:

```
$ SHOW LOGICAL TEST
  TEST = DRA2:[BERT.PROJECT.TESTS]TEST1.EXE
```

All logical name equivalences that are created are automatically deleted at the end of the terminal session, but can be deleted before this by either assigning a different value to the logical name, or by using the DEASSIGN command:

DEASSIGN - Cancels a logical name assignment made by an ASSIGN command.

format: DEASSIGN logical-name[:]

example:
```
$ ASSIGN DRA1:  DISK
$ DEASSIGN DISK
```
The logical name DISK is deleted from the logical name table.

9. VAX AND THE FILESTORE

The filestore is used, as its name suggests, for storing files. VAX files can be sent to the filestore to provide a backup against losing valuable data in the case of failure. Having copies of files on the filestore also means that they can be edited even when no access can be gained to VAX, by logging on to a LEGOS or ISYS80 terminal. In this case, the updated filestore version can be copied back to the original file before the next editing session on VAX.

Entry to the filestore is gained by typing the command:

```
*                $ FS
```

The response will be a prompt for a filestore username (normally the same as the VAX username), followed by a prompt for a filestore password (need not be the same as the VAX password). If <CR> is pressed instead of typing a username and password, only a limited number of filestore facilities can be accessed. The filestore prompts for commands with:

```
Fs:
```

Files in a filestore directory other than your own can be accessed by prefixing the filename with the owner's username, for example JIM.BACKUP defines a file called BACKUP which belongs to user JIM. Note that the naming conventions for filestore files differ from those on VAX.

VAX files are sent to the filestore with the command SEND:

```
Fs: SEND MYPROG/BACKUP
```

The default file type for the VAX file is .IMP (or the current default file type if changed by the EXT command), and the default filestore file extension is null, so the command above sends the latest version of the VAX file MYPROG.IMP to the file BACKUP on the filestore, overwriting any existing version there.

```
Fs: SEND MYFILE.PAS
```

If no file name is specified, then the VAX filename is used (adjusted for the differing conventions for separating file type or extension), so the above command sends the file MYFILE.PAS to the filestore file MYFILE:PAS. Files are fetched from the filestore in a similar way:

```
Fs: FETCH BACKUP/MYPROG
```

copies the file BACKUP from the filestore to a new version of the VAX file MYPROG.IMP in the current default directory.

```
Fs: FETCH MYFILE:PAS
```

fetches the file MYPROG:PAS from the filestore to a new version of the file MYPROG.PAS.

The command FILES gives a list of files on the current filestore directory. To exit from the filestore and return to command level, type STOP.

10. VAX AND THE ERCC NETWORK

10.1 Logging On To VAX From The ERCC Network

Vax is available as a host on the ERCC network. To access the system, type "VAX" in response to the "HOST:" prompt. The next prompt should be "Username:" as for a normal Vax login. At the Vax end, the terminal should appear like any other terminal except that :

1. Broadcast messages do not break through when the user is being prompted.
2. The end of input symbol is CTRL+Y, not CTRL+Z.

Escapes

TCP interrupts (in response to ESC) are interpreted as follows :

INT:Y Equivalent to Vax CTRL+Y : command interpreter break-in. (do not confuse with CTRL+Y on the network terminal, which means end of input)

INT:A Equivalent to Vax CTRL+Y.

INT:C Equivalent to Vax CTRL+C : program interceptible interrupt. If no interception routine is set up, equivalent to CTRL+Y.

INT:O Equivalent to Vax CTRL+O : discard output until the next INT:O or the next input request.

INT:. Equivalent to Vax CTRL+C followed by a line of input containing just a ".". This is sensible only for IMP programs with the standard diagnostic system, when it will stop the program without diagnostics.

INT:? Equivalent to Vax CTRL+C followed by a line of input containing just a "?". This is sensible only for IMP programs with the standard diagnostic system, when it will print diagnostics.

Other Ignored.

10.2. Logging On To EMAS from VAX

The effect of typing the command "ERCC" when logged on to VAX is similar to hitting the space bar on an EMAS terminal to initialise the logging on sequence. The major difference is that VAX, rather than EMAS, terminal conventions apply. Three important differences should be noted :

(i) CTRL+C is equivalent to typing ESC on an EMAS terminal
(ii) CTRL+Y is equivalent to CTRL+A followed by CTRL+D
(iii) CTRL+Z is equivalent to typing EM (end of file) on an EMAS terminal.

After normal disconnection or CTRL+Y, the terminal returns to VAX command level.

10.3 Transferring Files Between VAX And The ERCC Network

Files can be transferred between VAX and devices or mainframes on the ERCC network by using the Network File Transfer Utility (NETFS). To invoke NETFS, simply enter the following command:

* $ NETFS

and the prompt

>>

is given in response. Special commands can now be issued to transfer files, as described below.

First of all, a device or mainframe on the network must be selected, to and from which successive transfers will be made. This is achieved as follows:

>> DEVICE dev-name

where 'dev-name' is one of the following currently available devices:

2972:	2972 EMAS
2980:	2980 EMAS
FSTORE:	The ERCC filestore
TOWER:	LP41 - Lineprinter on Level 3, Appleton Tower
LP:	The 2972 lineprinter
LP<n>:	ERCC network lineprinters, sent via the 2972

Before a file can be transferred to or from a host, a username must be given to identify the file, using the START command, e.g.

```
>>START
User : ECSC98
Background pass :
```

The username and password can be entered on the same line, separated by a space, or in response to the prompts as above, in which case the password is not echoed. Note that there is a distinction between upper and lower case letters for the password, so ensure that you have typed it correctly.

Files can now be sent to the currently selected device or mainframe by issuing the command SEND. For example, assuming the above START command has already been entered:

>> SEND TEST.PAS TESTRUN

transfers the latest version of the file TEST.PAS (with default device and directory names completing the specification) to ECSC98.TESTRUN on the currently selected device. If the host filename is omitted, then the file takes the same name as the VAX file name. For example, the command:

>> SEND DRA1:[FRED]MYFILE.IMP

transfers the specified VAX file to the file ECSC98.MYFILE.

In the example below, the 2972 lineprinter is selected, and the file [JIM.PROJ]ESSAY.LIS is queued to be printed:

```
>> DEVICE LP
>> SEND [JIM.PROJ]ESSAY.LIS
```

Listings sent directly to the lineprinters will have the delivery information "ERCC front door" unless this is changed using the command DELIVER. For example:

```
>> DELIVER J.C.M.B. ROOM 1609
```

The owner's name on the listing can also be changed, as follows:

```
>> OWNER A.N. Other
```

Files belonging to the user specified in the START command can be fetched from the currently selected mainframe in a similar way. However, the VAX password must also be given. For example:

```
>> DEVICE 2980
>> START ERCM77 PASS
>> FETCH ERCCFILE VAXFILE.IMP
VAX password :
```

Here, the file ERCM77.ERCCFILE on the 2980 system is transferred to VAX, and is given the name VAXFILE.IMP, with the default version number, device and directory name completing the specification. If no VAX filename is given, then the file is named after the EMAS file (truncated to 9 characters) with the extension .IMP. For example the command:

```
>> FETCH TESTPROGRAM
```

would transfer the file ECSC77.TESTPROGRAM from 2980 EMAS to the VAX file TESTPROGR.IMP with the default values coming into force as before to complete the file specification.

File transfers do not execute immediately, and you will be sent mail by the NETFS utility when a file has been successfully transmitted or received. The command QUEUE displays a list of the files waiting to be transferred from VAX. Entries can be removed from this queue by using the command:

```
>> REMOVE identifier
```

where 'identifier' is the identification number given to the file, which is shown by the QUEUE command.

To communicate with a mainframe user, you can send a file or message using the TELL command as follows:

```
TELL ERCM77 [JIM.INFO]MESSAGE
TELL ECSC98,ECSC99
{message typed in at terminal, terminated by CTRL+Z}
```

If in doubt about any of the NETFS commands, you can type the command HELP. To exit from the NETFS program and return to VAX command level, type either EXIT, LOGOFF, QUIT or STOP.

10.4 Sending Mail To Users On The ERCC Network

Mail can be sent to users on the 2972 and 2980 systems using the MAIL program, which is described in Section 6.1. The only difference between this and sending mail to other VAX users is the way in which the recipients of the mail are specified. There are two possible ways of doing this:

(i) The EMAS username is prefixed by the host name as in the following example:

```
2972::ECSC78
2980::ECSC99
```

(ii) The name of the EMAS user, enclosed in double quotes, is prefixed by 'RCO', which is the directory containing a list of all EMAS users on both systems. For example:

```
RCO::"J.Smith"
RCO::"T.G.Brown"
```

The same message can be sent to both VAX users and EMAS users in the same way as it can be sent to more than one VAX user, that is by separating the recipients with commas. For example, the reply to the 'To:' prompt might look like the following:

```
To: 2972::ECSC78, NR, RCO::"J.Smith"
```

11. PRODUCING DOCUMENTATION

LAYOUT is a program which produces a paged, formatted document from a source file consisting of the document interspersed with simple commands. See Reference 5. for a complete description of these commands and how to use them.

To LAYOUT a source file on VAX, use the command:

```
$ LAYOUT filespec
```

The default file type is .LAY, and the output document is put into a .LIS file of the same name. For example:

```
$ LAYOUT ESSAY
```

lays the file ESSAY.LAY into ESSAY.LIS.

Up to three input files can be specified by + signs as follows:

```
$ LAYOUT DOC+APPENDIX
```

This lays DOC.LAY and APPENDIX.LAY into DOC.LIS.

If the command has the qualifier /DIABLO then a diablo file of the same name is produced, with the extension .DIA. This file can then be printed on the Diablo printer by issuing the command TODIABLO as follows:

```
$ LAYOUT DOC+APPENDIX/DIABLO
$ TODIABLO DOC
```

The first command above lays DOC.LAY and APPENDIX.LAY into DOC.LIS and produces the file DOC.DIA which, in the second command above, is sent to the Diablo to be printed.

APPENDIX: DEFAULT FILE TYPES

File Type	Contents
COM	Command file to be executed with the @ command, or to be submitted for batch execution with the SUBMIT command.
DAT	Input or output data file.
DIR	Directory file.
EXE	Executable program image.
FOR	FORTRAN program.
IMP	IMP-77 program, or data file for an IMP-77 program.
LAY	Source for LAYOUT, the document producing program.
LIS	Listing file created by a language compiler or assembler, or LAYOUT; default input file for PRINT and TYPE commands.
OBJ	Object file created by a language compiler or assembler.
PAS	PASCAL program.

REFERENCES

1. VAX/VMS Command Language User's Guide
   Digital Equipment Corporation

2. VAX/VMS Primer
   Digital Equipment Corporation

3. VAX-11 Software Handbook
   Digital Equipment Corporation

4. Edinburgh Compatible Context Editor (ECCE)
   (Dept. of Computer Science Internal Report)
   L.D.Smith, University of Edinburgh

5. LAYOUT
   (Dept. of Computer Science Internal Report)
   P.McLellan, University of Edinburgh

6. The IMP-77 Language
   (Dept. of Computer Science Internal Report)
   P.S.Robertson, University of Edinburgh