Internal Report
DESY F1-90-01
January 1990

How to Get Started on a VAX (Another Primer)

by

K. Gather

"Die Verantwortung für den Inhalt dieses
Internen Berichtes liegt ausschließlich beim Verfasser"

```
*********************************
*                               *
*  How to get started on a VAX  *
*                               *
*      ( another Primer)        *
*                               *
*        by Karl Gather         *
*                               *
*        - DESY -    F1         *
*                               *
*        Version 3.0            *
*********************************
```

Abstract:

This document introduces the reader to some features of the VAX/VMS system. The intent is to limit the items to the extent needed to have a comfortable environment for the following users:

- the casual user using command procedures and mail
- the high level language programmer (FORTRAN, C,...) who is usually not interested in system details or internals

We do assume that some sort of computer has already been used by the reader,so questions of memory, disk, terminal etc. are not addressed. Whenever the reader finds a disagreement between this document and the reality, please give your comments to the author.

How to use this document:

It is recommended to reserve one or two hours of time just to read the document (depending on experience with interactive systems three hours may be needed). After this initial reading it is recommended to really try the commands mentioned while scanning through the document a second time.

Introduction:

The following topics are addressed (reason included):

1) The File-System (directory structure)
   reason: Since all information on the computer is stored in or retrieved from files, the major features of files (names, directory-structure) have to be known to the user to efficiently structure their work.

2) The Command Language (DCL) and HELP
   reason: Mandatory for the use of a computer is the way commands can be given to it, it is not just a list of commands, it is a concept (hopefully). The HELP structure and the availability of HELP within the commands is essential for the casual programmer.

3) Command Procedures
   reason: The first thing the user wants to accomplish is how to get rid of the ever repeating work (commands).

4) Definition of Symbols
   reason: The second thing the users want is to set up special symbols meaning more to them than the standard commands or to have their special short-cuts for the commands that would become ever longer with increasing skill.

5) Logical Names
   reason: To become more and more independant from the actual hardware, the users want to set up a logical "world" for their development, so having adopted the logical world to a new machine, everything should run again.

6) Utilities
   reason: Some things are essential prerequisits for the use of a computer system (e.g. Editor, Mail,...). Here we discuss EDT and LSE to some extent, MAIL is discussed to help the user set it up in the most appropriate way.

7) The Programming Cycle
   reason: A brief outline on how to make programs and how to maintain them on a VAX/VMS system is given as a help in getting started. The example will be in FORTRAN, so I apologize to all Cf's (meaning C-language fans). The hints given apply however to all languages.

1) The File_System (directory structure)

1.1 File naming convention

The naming convention for files under VAX/VMS consists of three items:
- the file_name (up to 39 characters)
- the file_extension (up to 39 characters)
- the file_version_number (1 - 99999)

the name is seperated by a fullstop or period from the extension, a semicolon seperates the version number from the rest, so in general a file is referred to with :

```
file_name.file_extension;file_version_number
```

Examples:

```
  Event_Display.FOR;10
  RUN_6990_Summary.Plots;1
  Standard_Definitions.com;110
```

Some extensions are known to the system as special to allow ease in using them (i.e. default extensions, if they are not used the the non default extension must be declared explicitely) :

FOR	normally used for FORTRAN source files
C	normally used for C source files
OBJ	normally used for object code (output of the compiler)
EXE	normally used for files containing executable code
LIS	normally used for files containing a line printer output
MAI	normally used for Mail files
COM	normally used for files containing command procedures
DIR	normally used for directory files (folders)

Most services support the so called "wild-card", e.g. the name *.FOR;* would address all files with FOR as extension in the current directory. If you want just to wildcard a character, the sign % is to be used. A few legal wild cards are displayed below when used in a directory command (see chapter 2):

```
  $ DIR *.FOR;*                   displays file information for all
                                  files with extension FOR
  $ DIR DESY_*.accounts;*         displays file information for all
                                  files starting with DESY_ and having
                                  the extension accounts

  $ DIR DESY_F%%_members.doc      displays file information for all
                                  files matching the name when the two
                                  characters at the places of the % are
                                  arbitrary.
```

1.2 Directory Structure

As already mentioned in 1.1 the name LETTER.DIR indicates a folder or directory. It is a file created and maintained by file manipulation commands and its nature is indicated by the extension DIR. This concept of maintaining a directory as a file easily allows for a tree structure for the directory , which should be familar to PC users.

Each tree starts with a top directory, normally this will be the directory you find yourself when you login to the VAX. However it should be noted, that a subdirectory is the top-directory of all lower levels, so the concept does not change. A tree is referred to by

```
  [tree_name...]
```

where the three dots are significant. All lower levels existing are affected by a command using the syntax [name...]. Note that there is a maximum of eight levels available. If you ommit tree_name, the current directory is assumed. There is one directory, which is different, it is the Root-directory of a disk, which contains a lot of system and storage medium related files and its own directory-file. This does not apply to virtual disks, so if you do not find index-files etc. on your root-directory, you may be on a logically defined disk. So don't be dissapointed if you do not find the special files mentioned.

1.3 Protection of Files

Depending on definitions you may have seen already strange letters filling up your screen, e.g.:

```
LOGIN.COM     3/4   (S:RWED,O:RWED,G:RE,W:RE)
```

The information within the parentheses is showing the protection of a file, this is the most obvious and most effective way to rule access to your files and it is completely up to you to do what seems to be appropriate. The following classes for the access are distinguished by the filing system:

S = System
O = Owner (e.g. you)
G = Group (depending on the local organisation you may be allocated to a special group, e.g. [ONLINE])
W = World (anybody not in S,O,G is in W, especially any access made via network !)

The protection shown above is the one normally recommended and set as the default one (at least on machines the author is responsible for). Here the meaning of that protection mask in ordinary language:

System may Read from, Write to , Execute or Delete this file
The Owner may Read from, Write to , Execute or Delete this file
Everybody within the group may Read from or Execute this file
Anybody may Read from or Execute this file

So the meaning of the letters RWED is as follows:

R = read access
W = write access
E = execute access
D = delete access

The protection given to a directory rules the possibilities to work in that structure, e.g. if the directory file "task.dir" does have the protection (S:RE,O:RWE,G:RE,W) the following rules apply:

The System may read or execute files being in that directory,
the Owner may read or execute files being in that directory or may create new files,
the group members may read or execute files being in that directory,
everybody else will learn nothing at all about files in that directory.

2) The Command Language (DCL) and HELP

The Digital Command Language (DCL) is used to direct the VAX/VMS system to do the service requested. DCL commands can be issued, whenever the prompt of the process is there (if you do not modify it yourself, your prompt will be a dollar ($), telling you VMS is ready to accept your directives. DCL commands are structured in the following way:

```
$ command/qualifier  operand/qualifier   <Return>
```

There may be many qualifiers separated by slashes and one or two operands, but only one command. The VAX HELP will tell you what qualifiers are valid for a certain command and what the default values are, i.e. what will be assumed by the system if you do not mention the qualifier at all. The command can be issued whenever the $ is displayed, just by typing it and finishing by hitting the <Return>-key. The <Return>-key always finishes a command input. As long as you have not pressed it you may correct your command, i.e. Line-Editing is possible. A few hints that may ease you in getting familar with the Line-Editing:

<CTRL>H brings you to the beginning of the current line
<CTRL>E brings you to the end of the line
<CTRL>A switches between insert and overstrike mode

<CTRL>H means that you have to press the <CTRL>-key and the "H"-key at the same time. We should briefly mention two more such couples you will use quite a bit:

<CTRL>Z normally finishes the current running mode, e.g. finishes the mail, returns to command mode in editor, ends a utility,..

<CTRL>Y normally finishes the current running image in a "brutal" way, and returns to the command mode (it basically interrupts like an error condition). For those coming from a PDP system, it corresponds to the former <CTRL> C which also works on VAX the way it worked on PDP's.

We have assumed here that you already did login. Just in case you didn't, hit the Return key on your keyboard, the VAX will make some announcement and ask for name and password.
There is normally no restriction on how often you may login at the same time on the same machine, which may be a bit confusing the first time, but will be so extensively used as soon as you know it, that you never will want to miss it. Up to twenty commands can be recalled to the screen by hitting the up-arrow-key, with the down-arrow-key you can go back into the direction of the current command (e.g. if you go to far to the past). This will be very helpful if you have to type several commands changing only slightly like in the following example taken from the section 2.1:

```
$ DIR disk$online:[CDAQ.DOCS]*.DOC
$ DIR disk$online:[CDAQ.RC.DOCS]*.DOC
```

To learn what commands still are in the command buffer of your session issue the command

```
$ RECAll/ALL
```

and the 20 commands issued most recently are displayed on your screen with a number in front of each command. If you want to use the 10th command to issue it again (may be slightly modified) you get it with:

```
$ RECAll 10
```

In order to recall the most recent command with say "run" in it you may say

```
$ RECAll RUN
```

and your current command line will contain the last command starting with "RUN" (e.g. $ RUN Event_Display). This saves you recalling all commands and then recalling according to the number.

Here we discuss a few of the many commands, those you need to get started on the VAX, i.e.:

```
$ HELP
$ DIRect
$ LOgout
$ COPy      $ REName
$ DELete    $ PUrge
$ CREATE    $ REName
$ SET       $ SHow
$ DEFine    $ DEASsign
$ SPAwn
$ EDIT      $ FORT      $ LINK      $ RUN       $ SUBmit    $ MAIL      $ LIBRary
$ PRInt
$ MONitor
```

The VAX is NOT case SeNsItIvE, as UNIX machines, so you may type as you like. Whenever a command is no longer ambiguous because of the characters typed, you may stop. The VAX will take it as such. This is shown in the above notation by switching from capital to small letter whenever the command is already unique. The first thing you will recognize is that whenever you forget something, but it is already clear what you want, the VAX DCL interpreter will ask for the rest. The second thing is that you will find "online" HELP in almost all places, so you finally will forget about manuals. Here a very brief intro is presented, it should allow you to get started, and then try on your own with the HELP etc. to find the rest of the goodies. So just type the command

```
$ HELP
```

to see what we will not discuss here. Having typed HELP you will see kind of a menue of all commands and the VAX-HELP will ask for further input to give more advice, so select one of the options that you think you might know something of and play a bit with the VAX-HELP.

2.1 DIR

```
$ DIR
```

without operands and qualifiers displayes the files in the current or default directory. If you want to see another directory, type the following (for example):

```
$ DIR ZEUS02::disk$online:[USER.subject.whatever]*.*;*
```

This displays all files on node ZEUS02 resident on the disk called disk$online, found in the third level of a directory tree starting with [USER] at the top. The " ;* " demands that all versions are displayed. You can remove wildcards as you please to shorten the output onto your screen, e.g.

```
$ DIR *.for
```

would display the latest version of all Fortran source files in the current directory. On ZEUS02 we have defined the command DIR to be actually the following:

```
$ DIR is  $ DIR/Size/Date=mod/prot
```

this displays next to the filename the size in blocks, the date of last modifcation and the protection mask (see 1.3). If it is not defined this way on your system you may want to do so in you Login procedure, see 3). Any definition of symbols can be done the following way:

```
$ DIR :== " DIR/SIZE/DATE=MOD/PROT "
```

If you want to save the output of the command, in most cases there is a qualifier to ask the system to do so, for the directory command this is done with

```
$ DIR/out=TEMP.lis *.doc
```

this will put the output normally going to the screen onto the file TEMP.LIS.

2.2 LOgout

closes the session you currently are in. If you went to a subprocess by saying SPAWN (see 2.8), the LOgout command will return to the main process. Whenever you finish work, do not forget to logout, so nobody can continue your session and do something undesirable with your files etc.

2.3 COPy and REName

The command

```
$ COPy  name1.ext name2.ext
```

will make a copy of file name1.ext to a file name2.ext within the current directory; without having specified the version it will only affect the highest or last version. Normally (when getting started) you may want to copy something from a different directory or even device into your current directory to play with it, for example:

```
$ COPy SYS$SYSDEVICE:[partner.commands]joke.com *
```

will make a copy of the file joke.com which resides in the specified directory into the directory you currently are in, and the copy will be your own file you can modify with EDIT.... Because of the wildcard the copy will have the same name "joke.com".

In order to give a new name to your file you do not have to copy it, a simple REName command will do it:

```
$ REName hunde.txt   junk.waste
```

will change the name of the file "hunde.txt" into the name "junk.waste". Because of the file system structure discussed in chapter 1 it is easy to "move" one file from a specific place into another one, provided protection mask and access rights allow for it:

```
$ REN  Water.Bottle  disk$online:[CDAQ.Goodies]Whiskey.Bottle
```

will put the file "Water.Bottle" from the current directory into the directory [CDAQ.Goodies] as Whiskey.Bottle without actually performing a copy but just changing the directory files accordingly. This is a very effective way of "moving around" files without duplicating them and/or doing a lot of I/O. Note that this will only work if the source directory and the destination are on the same physical disk and if the access to the directories is allowed in the way necessary for the operation, i.e. the process has to have (RWD) access to those Note, that it is up to the user to make sure that the content of a file has som connection to the name.

2.4 DELete and PUrge

Having told you how to duplicate information by doing a COPY it is fair to the SYSTEM-manager to tell you how to delete files from the directory and how to clean your area.

```
$ DELete *.*;*
```

is obviously a very dangerous command, it will erase all files and all versions in the current directory unless you have protected the files against deletion (see 1.3).

```
$ DELete name.ext;3
```

will delete exactly the version 3 of the file "name.ext",

```
$ DELete name.ext;
```

will just delete the last version of the file "name.ext". I have made a redefinition of the delete command to protect myself against erraneous deletion in the following (see also 4):

```
$ DEL*ete :== " Delete/confirm "
$ KILL    :== " Delete/log "
```

/confirm demands confirmation for each file to be deleted, /log displays at least what is going down the drain. The use of the "*" in the definition shown above specifies the characters mandatory for the command to become unique, so this allows to type DEL, DELE, DELET or DELETE and always doing the same.

More frequently used is the PURGE command, which is essential to get rid of the many versions of a file you may accumulate in the course of your development work. With for example,

```
$ PURge *.FOR/KEEP=2
```

you delete all but the last two versions of all Fortran source programs in the current directory. The protection of the files and the ownership may however not allow that, depending on where you are. If you omit the /KEEP qualifier, all but the last version are deleted.

2.5 CREate and REName

The Create command is only discussed in terms of creating directories, for all other flavours just do "$ HELP CREATE" and educate yourself. The command

```
$ CREATE/dir [.sources]
```

will create a subdirectory to the directory you are currently in. You have to be allowed by the protection mask of the higher directory with write access, otherwise you will not be able to create a subdirectory. Having created a directory, you are by no means bound to keep that name, whenever you want you may rename it, e.g. the command

```
$ Rename sources.dir  code.dir
```

will change the directory name from SOURCES to CODE. This rename will work on any file to which you have write access. So you will not rename my file if I have protected my file appropriately. Try as a starter just the following commands:

```
$ create/dir [.test]
$ create/dir [.log_files]
$ dir
```

and you will find the directory files in your current directory. It is of great advantage to structure your area in a reasonable directory tree structure (see 6.2 and 7.1).

2.6 SET and SHOW

2.6.0 The many SET commands

From the title of this section you may already have guessed, that we will not be able to discuss all "SET" commands in this primer. You can "SET" almost everything on the VAX, please use the VAX-HELP to learn the rest.

2.6.1 SET DEFault

This command allows you to select which directory is to be the current/default directory. You typically will "go" in this way to the directory most of the files you want to work on are located. For example

```
$ SET DEF [.MAIL]
```

will put you in your mail directory, assuming that you have it and that you were in your top-directory when issuing the command. This tells you two things:

- whenever the directory expression is not complete, the command refers to the current directory for making the command complete,

- whenever something is not mentioned (e.g. the disk) it is assumed to stay the same.

In case you want to go to a directory on a different device, you type for example

```
$ SET DEF dua0:[x.y]
```

If you now say

```
$ SET DEF dual:
```

VMS will put you to the same directory [X.Y] on dual. If that doesn't exist you will see the appropriate error message. You now cannot see any file because you are in a not existing directory, so you have to set yourself to a legal and existing one first (e.g. your top-directory,..). If you want to go up one level of directory, you may say

```
$ SET DEF [-]
```

which is very nice since it is short. Another nice use of the "-" is for the case, you want to go from one subdirectory to another one on the same level in the same tree (natural if the work is grouped reasonably):

```
$ SET DEF [-.another_subdirectory]
```

2.6.2 SET PROTection

As we have discussed already in section 1.3, protection is the easy way to rule access to your files and directories. Obviously you want to have a default protection for the standard files and a "high" protection for "special" files (e.g. mail).

```
$ SET PROT=(S:RWED,O:RWED,G:RE,W:RE)/default
```

will be the protection of all files you generate after having issued this command. If you now say

```
$ SET PROT file-name
```

the file "file-name" will get the default protection. Just create a file to see your current default protection and decide whether you like it. Note however, that protection masks are inherited, i.e. if you edit a file, the new generation will have the same mask for the protection as the previous generation. If you want to protect a file from being purged or deleted (even by yourself), you may want to say

```
$ SET PROT=(S:RWE,O:RWE) file-name
```

assuming that the protection was as the above mentioned default one. If you don't mention G or W, this part of the protection mask is not affected. In order to completely remove access for a species, you can issue

```
$ SET PROT=(W) file-name
```

which will not allow any access for users being not in the system or group category. So the command

```
$ SET PROT=(S,O:RWE,G,W) mail.mai
```

is appropriate for the personal mail. A system manager still would be able to look into that file, however he has to obey the laws about keeping the data confidential and not doing it just to satisfy his curiosity.

2.6.3 SET TERMinal

Standard terminals (VT320, VT340,..) and Emulators on PC's have a lot of parameters that can be set according to the need of the user. A few commands of that sort are discussed here.

```
$ SET TERM/echo
$ SET TERM/noecho
```

switches the terminal display mode to the different values, so you either see what you type (echo) or you don't. When you type your password, obviously your terminal is set to noecho. If you detect that you cannot do line editing of commands, try the following command:

```
$ SET/TERM/line_edit
```

To change the display width and length, two comands are useful:

```
$ SET TERM/WID=132
$ SET TERM/PAGE=40
```

will tell the VAX it should display 40 lines and 132 collumns on your display. This doesn't make sense if your terminal is not capable of doing so, but can be very nice for line printer output viewing or editing large documents. Again feel free to find more information by typing

```
$ HELP SET TERM
```

and looking up all the other possibilities.

2.6.4 SET PROC/NAME

Whenever you are starting to work under different assumptions, and eventually at the same time, you might want to give the two sessions different names. Since the username is the same for the session, there is a way to modify your processname the way you like it. The command doing this is e.g.

```
$ SET PROC/NAME="IDEFIX"
```

In the above displayed example the process name is changed to IDEFIX and hence the user could call another session OBELIX and by that distinguish between the two processes. VMS will give the first session of user GATHER the process name GATHER, the second one will get the name of the terminal port used (e.g. TXA3 So it is useful to decide oneself on the process name.

2.6.5 SET PROMPT

You may want to change the $ sign, also called the prompt. This can be rather helpful, if you want to remind yourself about your current situation, e.g. yo may want to remind yourself which directory you are in. Below, in chapter 3 o command procedures, you will learn an example how to do that in an automatic way if you wish. Here we just show the command modifying the "$" prompt to tr prompt "xyz>" and the resulting display:

```
$ SET PROMPT="xyz>"
xyz>
```

This is extremely useful, if you have different accounts for different purposes

2.6.6 SET HOST

Since you find normally lots of VAXes in one place or the one VAX is connected to the Wide Area Network with the DECNET protocol, you should know how to login from one VAX to another without leaving you session:

```
$ SET HOST name::
```

will bring you right to that VAX called name and you can login there if you have an account as well. This command has a very strong application even if you are not on DECNET with other nodes. The way you will want to use it immediately is

```
$ SET HOST 0::
```

Which command will login you onto the same node again. There are two major reasons why you may want to do so:

- You want to login onto the same VAX another time as a different user without leaving the previous session and without using a second terminal.

- You have done a lot of work and you do not know whether you current problem is caused by all the previous commands you do no longer remember. So it is very helpful just to login as yourself and try the "new" session which will not be burdened with history.

If your VAX is connected in some way to the public network (in Germany DATEXP), the following command allows you to connect directly to any service conforming to the X.25 standard:

```
$ SET HOST/X29  Number
```

Where Number is a long series of digits (e.g. 45400053029).

2.6.7 SET PASSword

Finally you should know that you may change your password at any time it seems convenient for you, i.e. noone else needs to know your password and noone else should know the password. A few exceptions could occur for special users, but these will not be discussed here. The sequence

```
        $ SET PASSword
        Old password: NONTRIVIAL
        New password: Reallynontrivial
        Verification: Reallynontrivial
        $
```

will set your Password from the value NONTRIVIAL to the value REALLYNONTRIVIAL. Your System Manager may have defined constraints on the minimum length and on the words allowed. I advise you to use a strange easy to remember nonsense combination, e.g. "blueisyellow" or "beatoftels"....

2.6.8 The many SHOW commands

Almost everything you can SET you also can show, e.g.

```
$ SHOW DEFault
$ SHOW PROCess
$ SHOW TERM
```

In addition you can inform yourself about a lot of things you cannot SET, because it is either the System-Managers task to set it for everybody or it is the result of other operations. Here are a few examples you may want to know as a starter, again feel free to look up the possibilities in the VAX-HELP as soor as you please.

$ SHOW TIme	shows the system clocks time and date
$ SHOW USers	displays the users currently logged in
$ SHOW SYS	displays information about all processes on the system you are logged in to
$ SHOW QUOTA	tells you, how many blocks of disk space you have used and how much is still available
$ SHOW NET	will tell you the network available to the node you currently are on
$ SHOW Queue/all	will display all queues defined on your VMS-system
$ SHOW Queue/all/full	will display further detailes (e.g. protection) of queues
$ SHOW Queue x	displays your jobs in the queue x
$ SHOW DEVice	displays all devices available to the system
$ SHOW DEV DUA	displays all disks of the type DUA
$ SHOW LOG xyz	displays whether a logical name for xyz has been defined and if has been defined, what it is
$ SHOW SYMBol xyz	displays the value of the symbol xyz, if it has been defined

In "SHOW LOG" and "SHOW SYMBol" commands wildcards are allowed, e.g.

$ SHOW LOG DISK*	displays all logical names (see chapter 5) starting with "DISK"
$ SHOW SYMB *	displays all symbols defined for that session

Please type "HELP SHOW" to get an idea of the other things you can show on a VAX.

2.7. DEFine and DEASSign

The DEFINE-command sets up logical names. The standard user may want to use it in order to make their programs more flexible/portable. Here we just give the syntax:

```
$ DEFine/process   TAPE  $1$MTA0:
```

will define for the process issueing this command a logical name TAPE. Whenever TAPE is used, the more messy string will be taken for it. The application for this will become obvious in chapter 5. If you want to get rid of the above issued definition, you may type

```
$ DEASSign/proc  TAPE
```

The normal user will not be allowed to issue definitions for the overall system, hence only the qualifier /process is mentioned here.

2.8 SPAwn

The command

```
$ SPAWN
```

will create a subprocess with the identical setup of your current process. One of the most obvious applications is the following:

You are in LSE and suddenly recognize that certain commands are required in order to reasonably continue your edit session. You hit the DO-key (or <CTRL>Z) and issue the command SPAWN. Now you find yourself in DCL with an ordinary $ prompt. You issue the commands you want and then you logout of the subprocess by issuing an ordinary $ LO and back you are in the command line of LSE.

2.9 Some more useful commands

```
$ EDIT     $ FORT     $ LINK     $ LIBRary  $ SUBmit   $ MAIL
```

2.9.1 EDIT

```
$ EDIT filename
```

envokes the standard editor on that system (e.g. EDT) to edit a file with the name "filename". If the file is already existing, the editor will read the last version (you may specify an older version) and create a new one. For a few more details see chapter 6.1. By typing

```
$ EDIT/READ filename
```

you specify that you cannot write a new version of that file just by finishing with "EXIT" rather that "QUIT". You may want to do that in case you read someone's file and you want to be sure that you don't modify it by accident. Another essential qualifier when envoking the editor is

```
$ EDIT/RECOVer filename
```

which tells the editor it should take the journal file and redo the editing journaled there. This is a feature that stems from the time VAXes used to crash somewhat more often than nowadays. The journal file is kept automatically when you edit, so if you or the machine just interrupt the session, you can go back to the situation you left by the above command. In case you wonder, just do a lot of fancy editing for curiosity and cancel it by <CTRL>Y. Now do type the appropriate "edit/recover" command and watch the VAX doing the editing once more. Normally you come within a few commands right to the place you left.
 Please feel free to fool around with the editor before continuing, i.e work through chapter 6.1 if you like and come back.

2.9.2 FORTran

```
$ FORT filename
```

filename envokes the fortran compiler for the source file "filename". The standard way of envoking is (e.g.)

```
$ FORT/LIST TEST
```

This will envoke the standard Fortran compiler, assuming a source file TEST.FOR and it will generate file TEST.OBJ and a listing TEST.LIS. This gives a feeling how the "standard extensions" mentioned in chapter 1 become useful. You also could have done the following:

```
$ FORT/LIS=DOGS.CATS  Prime_Numbers.Privat
```

which is very unusual but possible; this will create a listing DOCS.CATS and an object file Prime_Numbers.OBJ.

2.9.3 LINKage editor

```
$ LINK TEST
```

will generate an executable file TEST.EXE provided there is a file TEST.OBJ on the current directory, that does not refer to any other service than the ones provided by VAX/VMS. If you want to link several modules and also a library the following command is a good example:

```
$ LINK/EXE=TESTER Tester_main,module_1,..,module_n,Library_1/lib
```

This command generates an executable file TESTER.EXE by binding the follwoing files to one module:

```
Tester_main.obj
module_1.obj
.
.
.
```

module_n.obj and the library Library_1.olb. The /LIB qualifier tells the linkage editor that a library is to be expected.

2.9.4 The Librarian

The VAX supports the following types of libraries:

- OBJ code libraries	extension .OLB
- text libraries	extension .TLB
- HELP libraries	extensionn .HLB

Libraries are very helpful in making program development easy. It allows the user to keep the source modules in different files for the ease of editing and to put the object code into a library for the ease of linking. To create a library, just type

```
$ LIBR/create
```

and answer the questions appropriately.

The following commands will cover 90% of your needs if not more (see also chapter 3 example 3.3).

```
$ LIBR/I xyz  library      insert the new member xyz into the library, the
                           assumption is that xyz.OBJ is in the current
                           directory and you are dealing with a OLB-library.
$ LIBR/R xyz  library      replace the existing member xyz by the new xyz
$ LIBR/D xyz  library      delete  member xyz from the library
$ LIBR/E xyz  library      extract the member xyz out of the the library
$ LIBR/List        library list all members of the library
$ LIBR/List/Full   library list all members of the library with date
```

2.9.5 Run or Submit a job

With the command

```
$ RUN program
```

you will execute a file program.exe from your current directory. Obviously you also can say

```
$ RUN [colleague.utils]HP
```

to call a program from somewhere else for execution (provided your colleague allowes the execution by the protection mask). Instead of running a program from a terminal you may want to put it into a BATCH queue, especially if you expect the program to run a while.

The command

```
$ SUBmit RUNFILE
```

will execute a command file RUNFILE.COM, that could just contain one line, namely " $ RUN [colleague.utils]HP " (see also chapter 3) into the default queue of the system. For some further details check the VAX HELP for SUBMIT. It should be noted here that the output of the program (if the program doesn' explicitely redirect) will be put into a file

```
[your_login_directory]RUNFILE.log
```

if you don't specify differently (see example 3.4) in chapter 3.
The following switches are recommended to be used in the submit command:

```
$ SUBmit/noprint/keep Runfile.com
```

which obviously can be made rather userfriendly by the symbol assignment as follows:

```
$ SUB*mit :=="submit/noprint/keep"
```

2.9.6 How to envoke mail on a VAX

```
$ MAIL
```

is the command do mail (sending and receiving). The author recommends to say instead

```
$ MAIL/EDIT
```

which connects you to the editor of your choice (see 6.2) which makes life more comfortable. If you have created the file you want to send already, you may type

```
$ MAIL filename.ext
```

and then just answer the questions of the mail utility. If you want to learn more about mail right now, please read chapter 6.2 and then continue.

2.10 How to print

Usually the user will just say

```
$ PRInt file_name.ext
```

and the file file_name.ext will be printed on the standard printer queue. On most installations several queues are offered (e.g. Laser-printer, Line-Printer,...) and special commands will be defined for those. Here we just want to mention a useful qualifier to the print command which might be good to know depending on the installation:

```
$ PRInt/NOHEADer name.ext
```

will make sure, that no file information is printed on each page. This is the default, however it might have been changed on your installation. In case you want this line on each page, you can ensure that by

```
$ PRInt/HEADer name.ext
```

2.11 How to Monitor

The VAX offers a lot of monitoring tools to see the performance of the system and most of them are available to the average user as well and might help to understand certain performance problems without nerving the system manager in all cases:

```
$ MONitor PROCess
```

will display the processes currently known to the system and some of there current characteristics (page faults, cpu_time, priority,..).

```
$ MONitor MODes
```

will show you how the current system modes are occupied, e.g. heavy IO will load the interrupt stack and the Kernel Mode and slow down the overall performance.

```
$ MONitor CLUSter
```

will show you the overall occupancy of the cluster devices (CPU's, Memory, Disks, ..).

```
$ MONitor PROC/TOPCpu
```

will continuously display the fraction of CPU-time used by a particular process being one of the predominant users (only the eight largest users are shown).

3) Command procedures

DCL procedures are a way to first edit a sequence of commands you want to execute and then execute them as if issuing them directly. The way VAX/VMS supports this feature is described shortly and by no means to the greatest detail.

A command procedure usually is a file filename.COM containing a DCL-command in each line. The commands are written as if one would have to type them in, obviously the system response has to be anticipated.

The most famous command procedure is the one always executed when you login to the VAX, i.e. the LOGIN.COM in your top level directory. Example 3.1 shows a LOGIN.COM file I usually have. The comments inline ($! lines) explain briefly what is done and I hope that the reader finds it self explanatory. To fully understand the following examples you need however to know about logical names and symbols, so please read chapter 4 and 5 before going into details of the following examples. There are two ways to execute a command procedure:

a) interactively with the command

```
 $ @procedure.com
```

b) in batch mode with the command

```
 $ submit procedure.com
```

This example has "procedure.com" as filename for e change. When you login to the VAX, case a) is automatically done for you with your LOGIN.COM file. Normally a system wide login-procedure is executed before your login.com has a chance to do anything. This is a service and makes life easy for everybody, however it may cause a problem in the rare case that a definition made by the System login fools your LOGIN.COM. The easiest way to settle those questions is to have an empty LOGIN.COM and check the symbols and logical names existing then.

```
$ @sys$manager:advanced_definitions.com             ! execute another procedure
$ mail :=="mail/edit/self"                          ! redefine mail command
$ Inquire yesno " Do you want system stuff?"        ! ask for a decision
$ IF yesno.eqs."Y" then goto details                ! act depending on input
$ GOTO FINIS                                        ! unconditional jump
$ !********                                         ! comment line
$ details:                                          ! label
$ !********
$ show err                                      ! Show errors recorded by system
$ show sys                                      ! Show active processes
$ !********
$ FINIS:
$ !********
$ show time                                     ! show date and time of system
$ users                                         ! show interactive sessions
$ !
$ write sys$output " Do special definitions"    ! write a message to terminal
$ @[GATHER.UTI1S]KEY_definitions.com            ! execute another procedure
$ @[gather.utils]specials.com                   ! execute another procedure
$ !
$ !********
$ s1000:                                        ! another label
$ !********
$ I=0                                           ! initialize a counter
$ !********
$ S1100:                                        ! another label
$ !********
$ I = I + 1                                     ! increment a counter
$ ISYMB :='I'                                   ! make a string of that value
$ ON ERROR THEN GOTO S1100                      ! consider errors
$ PROCNAM :="KaGa''ISYMB'"                      ! compose a string from string
$ SET PROCESS/NAME='PROCNAM'                    ! give yourself a name
$ set prompt=" [7;1m''procnam'> [0m"            ! set a nice prompt as reminder
$ write sys$output " Do MODEL setup "           ! write a message to terminal
$ @MHI_Directory:mhi_login                      ! execute another procedure
$ write sys$output " Do ADAMO setup "           ! write a message to terminal
$ ADAMODEF                                      ! use a special command
$ exit                                          ! finish the procedure
```

Example 3.1: LOGIN.COM

IF the command procedure is supposed to write something to the terminal, this can be done by a line such as:

```
$ write sys$output " Hello, here I am "
```

Just type the line above and see the result on your terminal or write a short procedure like in example 3.1. Note another fact you may have detected by yourself, whenever an exclamation mark is on the command line, the rest is interpreted as comment. Hence the possiblity to comment INLINE.

Some nice features provided by the fact that parameters can be given to the command procedure become obvious in the following example, which moves to the higher level directory which is the parent directory of the one you currently are in and it executes a LOGIN.COM file eventually found there. A similar a bit more complicated command procedure exist to go down a level. Just try to think why it has to be more complicated.

```
$ SYMB1 = F$LOGICAL("SYS$DISK") + F$DIRECTORY()
$ SYMB2 = F$LOGICAL("SYS$LOGIN")
$ IF 'F$LOCATE(".",SYMB1)' .EQ. 'F$LENGTH(SYMB1)' THEN GOTO TOP
$ SET DEFAULT [-]
$ SHOW DEF
$ SYMB1 = F$LOGICAL("SYS$DISK") + F$DIRECTORY()
$ SYMB2 = F$LOGICAL("SYS$LOGIN")
$ IF 'F$LOCATE(".",SYMB1)' .ne. 'F$LENGTH(SYMB1)' THEN GOTO doit
$ IF SYMB1 .EQS. SYMB2 THEN GOTO doit
$ EXIT
$ !
$ doit:
$ @login
$ exit
$ !
$ reset:
$ set def 'symb2'
$ SHOW DEF
$ GOTO doit
$ exit
$ !
$ TOP:
$ IF SYMB1 .nes. SYMB2 THEN GOTO reset
$ WRITE SYS$OUTPUT "You are already at your top level directory."
$ EXIT
```

Example 3.2: UP.COM

A more refined way to play with parameters and actually a quite useful tool is given in example 3.3. It is shown to give a hint on how to structure a command procedure with the IF-THEN-ELSE facilities available to DCL since VMS 5.0.

```
$ ! help maintain Libraries
$ ! current library is to be kept as logical name
$ ! in lib$current
$ ! P1 = command
$ ! P2 = module
$ !
$ wout :=="write sys$output "
$ if f$trnlnm("lib$current") .eqs. "" then goto cry
$ !
$ !***
$ CRY:
$ !***
$ curlib = f$trnlnm("lib$current")
$ GOTO WORK
$ !
$   wout " Please set first current library by the "
$   wout " logical name  [7,1m lib$current  [0m "
$   exit
$ !
$ !
$ !****
$ help:
$ !****
$ !
$ wout "  "
$ wout " +-------------------------------------------------------+ "
$ wout " |            [7;1m     LIB_Manipulation.com         [0m          | "
$ wout " +-------------------------------------------------------+ "
$ wout "  "
$ wout "  [7;1mp1  [0m= command     [7;1mp2  [0m= member affected/selected "
$ wout " without any parameter the current library is shown "
$ wout " The following commands are supported: "
$ !
$ wout " -------------------------------------------------------- "
$ wout "  [7;1m F [0m find member p2 in library lib$current "
$ wout "  "
$ wout "  [7;1m I [0m insert member p2 into library lib$current "
$ wout "  "
$ wout "  [7;1m R [0m replace member p2 in library lib$current "
$ wout "  "
$ wout "  [7;1m D [0m delete member p2 from library lib$current "
$ wout "  "
$ wout "  [7;1m E [0m extract member p2 from library lib$current "
$ wout "  "
$ wout "  [7;1m C [0m compress library lib$current "
$ wout "  "
$ wout "  [7;1m L [0m list library content (p2 optional) of lib$current  "
$ wout "  "
$ wout "  [7;1m H [0m this HELP "
$ exit
$ !
$ !****
$ WORK:
$ !****
$ if p2 .eqs. ""  then goto nolib
$ goto libido
$ !
$ !*****
$ nolib:
$ !*****
$   if p1 .eqs. "H" then goto help
$   if p1 .eqs. "L" then goto liste
$   if p1 .eqs. "C" then goto compress
$ goto cry2
$ !
$ !*****
$ Liste:
$ !*****
$     libr/list/full  lib$current
$     exit
$ !********
$ compress:
$ !********
$     libr/compress/log   lib$current
$     libr/list/full      lib$current
$     exit
$ !******
$ libido:
$ !******
$   if p1 .eqs. "F" then goto find
$   if p1 .eqs. "L" then goto listmem
$   if p1 .eqs. "I" then goto insert
$   if p1 .eqs. "R" then goto replace
$   if p1 .eqs. "D" then goto delete
$   if p1 .eqs. "E" then goto extract
$   goto cry2
$ !
$ !****
$ find:
$ !****
$     libr/list/full/only='p2  lib$current
$     exit
$ !*******
$ listmem:
$ !*******
$     libr/list/full/only='p2  lib$current
$     exit
$ !******
$ insert:
$ !******
$     libr/ins/log lib$current 'p2
$     exit
$ !*******
$ replace:
$ !*******
$     libr/repl/log lib$current 'p2
$     exit
$ !******
$ delete:
$ !******
$     libr/delete='p2/log lib$current
$     exit
$ !*******
$ extract:
$ !*******
$     libr/extract='p2/log lib$current
$     exit
$ !****
$ cry2:
$ !****
$     wout " no Name -> no action "
$     wout " see you later "
$     wout " Your current library is set to "
$     wout " -> "" ''curlib' "" <- "
$     exit
```

Example 3.3: LIB_MANIPULATION.COM

The following shows the way to deal with the fact, that recursive definitions are not allowed and hence certain things have to be redefined. It is a procedur envoked by the command SUBmit, which is itself defined as symbol.

```
$ if p1 .eqs. ""  then goto complain                          ! check on input
$ goto doit
$ complain:
$   write sys$output " I need a name for the command file to be submitted ! "
$   exit
$ doit:
$   sub*mit :==                                    ! because submit was redefined
$   place = F$trnlnm("dir$logs")+p1+".log"         ! define a log_file
$ set ver                                          ! set verification activ
$   submit/noprint/keep/log_file='place   'p1      ! do the submit
$ set nover                                        ! switch to non verification
$   sub*mit :==" @dir$util:submitter.com "         ! redefine submit again
$   exit                                           ! done
```

Example 3.4: SUBMIT.COM

4) Definition of Symbols

The normal users (after a first time of having learned the standard commands) will find a lot of "long" commands repeated quite often, so they feel a desire to give the computer a shortened command, even in view of the fact that they are already allowed to stop typing as soon as it is unique. Instead of

```
$ @dua0:[gather.commands]Standard_definitions
```

you may prefer just to say

```
$ STA
```

This you achieve by having the following line in your login.com file for instance:

```
$ STA*ndard :== " @dua0:[gather.commands]Standard_definitions "
```

which will allow the user to stop at any character after STA to execute the rather long command. Below I have listed the definitions made in our group for the ease of use just to give an impression. It may be worthwhile to look in greater detail into the section dealing with Directory-commands and purge and delete.

```
$ !
$ DIRF        :== "Direct/security/size=all/date=created"
$ DIRO        :== "Direct/owner/size=all/date=created"
$ DIR         :== "Direct/size=all/date=created/prot"
$ DIRs        :== "Direct/columns=4"
$ DIRDIR      :== "Direct/columns=4 *.DIR"
$ down        :== "@dir$util:down.com"
$ up          :== "@dir$util:down.com \"
$ wd          :== "@dir$util:wd.com"
$ !
$ set prot=(s:rwed,o:rwed,g:re,w:re)/def
$ save   :=="set prot=(s:rwe ,o:rwe ,g:re,w:re)"
$ !
$ del*ete        :=="delete/confirm "
$ Kill           :=="delete/log "
$ rew*ind        :=="set magtape/rewind "
```

Example 4.1   Set of Symbol Definitions found in SYLOGIN.COM

If you want to get rid of symbol definitions in order to avoid unforeseen misinterpretations, this can be done via

```
$ DELete/SYMBOL xyz
```

where the defintion of the symbol xyz is cancelled.

5) Logical Names

Logical names are a very powerful way to make programs and procedures more flexible or portable. A logical name is a convention of translation accepted either on

- system level or
- group level or
- process level

and can be understood as a sequence of translation directives (tables) for the system to interpret your commands and strings. The different levels are protected by priviledges so not everybody can modify the system wide translation table. Typically devices are named system wide by logical names rather than by their hardware name. This allowes to exchange the hardware without the necessity of all users to change everywhere their nomenclature. Two examples may convey the idea:

```
$ DEF/SYSTEM  DISK$ONLINE   $1$DUA6:
```

will allow all sources and all procedures to use DISK$ONLINE rather than the real disk name. So if the whole group is eventually moved to a disk $0$DUA4: only programs not using the logical name have be modified. Whenever a user wants a different translation he can do so for his process by e.g.

```
$ DEF/PROCess/trans=concealed  DISK$ONLINE   $1$DUA6:[MYDIR.]
```

The VMS system first checks the translation in the process table and then goes to group and system table. This example shows two more things:

- it is possible to define a directory as virtual device, i.e. the directory [MYDIR] becomes the root directory for the logical disk DISK$ONLINE.

- You can tell the system to conceal the translation fo the system, such allowing the software to react as if DISK$ONLINE: would be a hardware device.

One nice example for using logical names to have a more general utility is shown in the example 3.3 of chapter 3. Since the procedure uses the logical name "LIB$CURRENT", everybody by setting this logical name for his session appropriately can use the procedure for his specific library.

6) Utilities

From the many utilities here only the MAIL and the EDT or LSE editor are discussed, because these will cover 90% of your needs excluding a few special cases (e.g. system manager, field service,..).

6.1 The Editor

No matter whether you want to use the EDT or the LSE editor, you will hit the keypad syndrom, i.e. after one hour of moaning you will immediately start the " never want to miss it " phase. We do not discuss the TPU editor, which has a lot of power and underlies the LSE-editor, since this really is for experts.

Before going to the explanations of the editor itself it has to be made clear that editing a file on the VAX is just dealing with a sequence of characters, where the carriage return <CR> starts a new line; but as a character it can be deleted or inserted or copied just as any character can. Each line is called a record, a record can have up to 256 characters.

On the next page is displayed the "keypad" as defined for the EDT- and the LSE-editor. You may want to recognize that each key has two functions, one if you just press the key, one if you first press the "GOLD"key and then the key. Normally "GOLD" "KEY" reverses what "KEY" does.
The editor has two states:

- command mode (the cursor is on the lowest line and a prompt > is displayed, to tell you that commands may be issued)

- fullscreen mode (this is the mode you normally are in, i.e. you type text or manipulate pieces of text via the keypad).

The major functions used in fullscreen mode are (all keys mentioned in " " are the keypad keys)

FORWard to change mode of operation to the forward direction hit "4"

BACKWard to change mode of operation to the backward direction hit "5"

PAGE to move cursor by one page hit "8" in the direction selected

FORMFEED to move cursor to the next FormFeed in the direction selected hit "7"

DelLine to delete the line you currently are in from the position of the cursor to the end including the <CR> hit "PF4"

DelRest to delete the line you currently are in from the position of the cursor to the end but leaving the <CR> hit "GOLD" "2"

DelWord to delete the word next after the cursor hit "-", if you are in the middle of a word, the rest until the end is removed

DelChar to delete a character to the left use the <DEL> key,
to delete the character at the cursor position hit the keypad ","

UnDel to cancel the last deletion, hit the "GOLD" key and the key used for the deletion you want to undo, e.g. "GOLD" "PF4" will undo the last line delete

CUT move cursor to beginning of string you want to cut, hit "." then move the cursor to the first position after the end of the string you want to cut and hit "6"

PASTE move cursor to position you want to insert the string you removed by the CUT-operation and push "GOLD" and then "6". The string will be inserted before the cursor position.

SEARCH to search for a string, hit "GOLD" "PF3" and the editor will prompt you in the command line for the string. Finish the input by "ENTER". Hitting "PF3" consecutively will do a repetative search. The search will work in the direction chosen by the keys "4" or "5".

Move by Beginning of Line hit "0" and cursor positions itself at the beginning of the line up or down depending on current direction mode

Move by End of Line hit "2" and cursor positions itself at the end of the line and moves up or down depending on current direction mode

Move by Word hit "1" and cursor moves word by word in the direction of the current mode

Other positioning can be done by using the cursors. More powerful operations are offered by other commands possible in the command mode. Please use the online HELP to find the commands to REPLace or SUBSTitute a string etc. Last but not least you have to know, how to finish editing. Hit the DO-key or the <CTRL>Z and then say "EXIT" if you want to save the work done, say QUIT if you want to forget what you did. EXIT will generate a new version of the file you started with and it's version number is increased by 1.

↑ UP	↓ DOWN	← LEFT	→ RIGHT

PF1 GOLD	PF2 HELP	PF3 FNDNXT FIND	PF4 DEL L UND L
7 PAGE COMMAND	8 SECT FILL	9 APPEND REPLACE	- DEL W UND W
4 ADVANCE BOTTOM	5 BACKUP TOP	6 CUT PASTE	, DEL C UND C
1 WORD CHNGCase	2 E O L DEL EOL	3 CHAR SPECins	ENTER SUBS
0 LINE OPEN LINE		. SELECT RESET	

Fig.1 : The Keypad as assigned in EDT editor

6.2 MAIL

The Mail utility allows one to send or receive mail on the VAX. It is advised to say MAIL/EDIT so you come to the editor of your choice rather than using the reduced Mail editor. Mail is ready for service when you see the prompt

MAIL>

A short sketch through the commands of interest to the general user is given:

MAIL> SEND	=> inititiate a send, mail asks for dest. dest: node_name::User_name
MAIL> DIR	=> shows the current mails available to be read, i.e. all mails contained in the selected folder
MAIL> DIR/FOLDER	=> displays all folders defined within your mail file
MAIL> Read #	=> read message number # from the current folder
MAIL> DELete #	=> delete message number #, omitting the # directs mail to delete the message currently being read
MAIL> MOVE subject	=> moves the mail currently selected into folder SUBJECT
MAIL> SEL NEWMAIL	=> selects the messages in folder NEWMAIL
MAIL> PRINT	=> will print the current mail when you leav the MAIL-utility
MAIL> EXTRACT TEMP.WORK	=> extracts the mail currently read into the file TEMP.WORK in your current directory
MAIL> EXTR/Nohead TEMP.WORK	=> extracts the mail currently read into the file TEMP.WORK in your current directory and removes the mail header. This is very convenient for extracting code etc.
MAIL> FORWard	=> forwards the mail currently read to an address the user may specify interactivel as inquired by the mail facility
MAIL> Exit	=> leaves mail

All mail is maintained within folders, without doing anything the folders

NEWMAIL	for new delivered mail
MAIL	for old mail
WASTEBASKET	for deleted mail

are defined. It is very convenient to sort ones mail in folders related to major mail subjects (e.g. NEWS, DESIGN, ORDERS, DOCS,...). Having selected a folder you only see the mail in that folder. Whenever you remove the last mail from a folder, the folder will vanish. Whenever you want to store the first mail into a folder, VMS-MAIL will ask you whether you want to create it or not. The author recommends to do the following SET commands within mail:

MAIL> SET mail_dir [.MAIL]	creates a subdirectory that will contain all mail files, so keeps your top directory clean.
MAIL> SET SELF_COPY REPLY,SEND	will send you a copy of whatever you send or reply
MAIL> SET EDITOR LSE	will set your mail editor to be LSE

Some settings are possible that are strongly depending on the system used, here we mention those more often occurring in one or the other way:

MAIL> SET FORWard node::user_id	will forward all messages to the address node_id::user_id, very helpful if you travel a lot.
MAIL> SET QUEUE queue_name	specifies the default printer queue in case you say print inside MAIL. Thi is obviously useful, if several printe are available, Line Printer for lots o paper and Laserwriters for graphs and mail.

Normally the additional features will be found with increasing skill and thus they are not subject to this primer. On the next page is displayed how the standard MAIL definition for the keypad is, i.e. you can do a lot just by one keystroke.

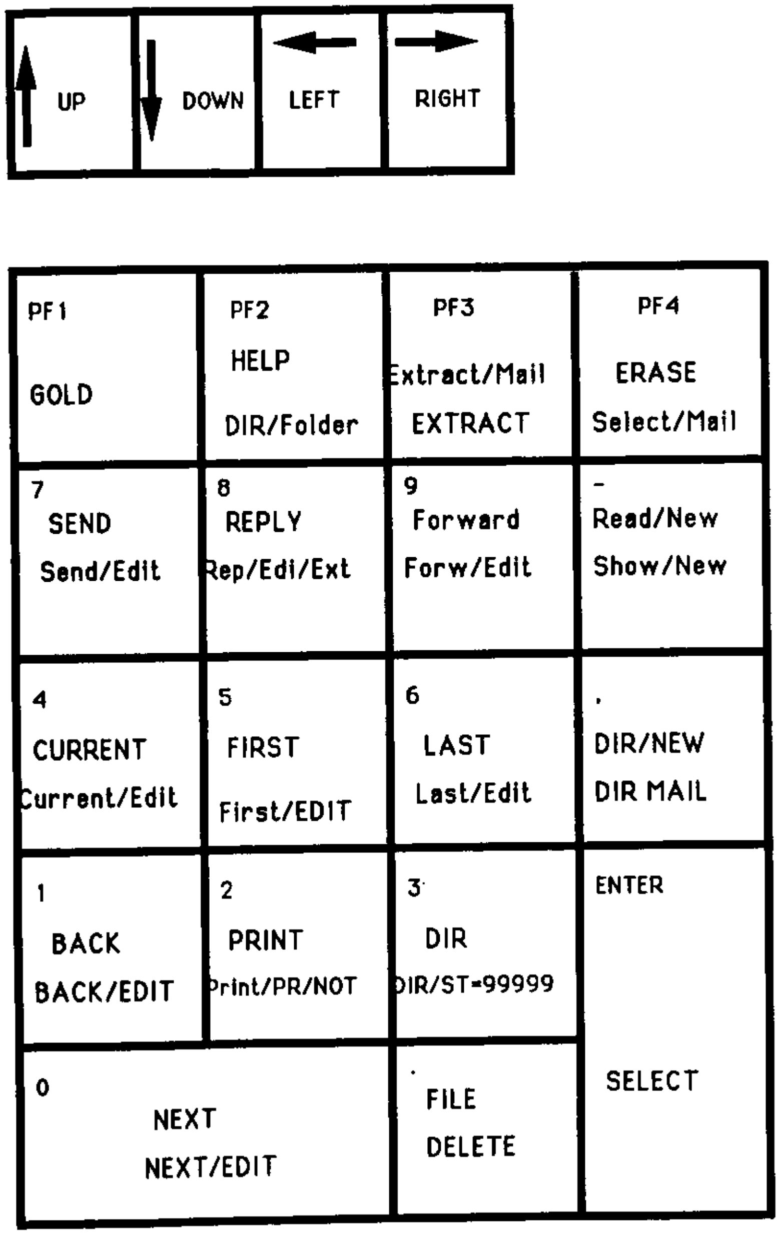

Fig.2 : The Keypad as assigned in the Mail utility

7) The Programming cycle

7.1 How to organize ones work without tools

Aim: Efficient Program Development

This section describes program development without referring to more advanced tools than the standard editor, compiler, linker and filing system.

You will definitely fined the most appropriate way to organize your work yourself. Here only a suggestion is given to allow you to start in a way, that is open for improvements and modifications according to your taste, delivering however right from the beginning some of the advantages offered by VAX/VMS.

If you develop different packages, it is obviously appropriate to put each package into its own subdirectory. Here we display a simple directory tree for a user, who receives mail and has some general documentation and who works on two subjects (Task_1 and Task_2):

```
node::disk:[user_name]
node::disk:[user_name.Documents]
node::disk:[user_name.MAIL]
node::disk:[user_name.Task_1]
node::disk:[user_name.Task_1.Commands]
node::disk:[user_name.Task_1.Data]
node::disk:[user_name.Task_1.Documentation]
node::disk:[user_name.Task_1.Execs]
node::disk:[user_name.Task_1.Includes]
node::disk:[user_name.Task_1.Library]
node::disk:[user_name.Task_1.Log_files]
node::disk:[user_name.Task_1.Save_Sets]
node::disk:[user_name.Task_1.Sources]
node::disk:[user_name.Task_2]
node::disk:[user_name.Task_2.Commands]
node::disk:[user_name.Task_1.Data]
node::disk:[user_name.Task_2.Documentation]
node::disk:[user_name.Task_2.Execs]
node::disk:[user_name.Task_2.Includes]
node::disk:[user_name.Task_2.Library]
node::disk:[user_name.Task_2.Partition_1.Sources]
node::disk:[user_name.Task_2.Partition_2.Sources]
node::disk:[user_name.Task_1.Save_Sets]
```

The advantage of sorting once work in such a way is obvious:

Whenever you are in a subdirectory, the amount of information to be delt with is limited to the subject. This is of advantage only if you have organized your work in a reasonable way, but that is an assumption for your work which should be trivially fulfilled. This treelike structure has been so unanimously accepted, that you find it on most of the widespread systems (UNIX, VAX/VMS, MS-DOS, Apple, Atari,...).

The Top directory (or folder) in the example mentioned above just contains the following files:

```
LOGIN.COM
Documents.dir
MAIL.dir
Task_1.dir
Task_2.dir
```

which allows a very easy judgement about the work and the organization of the work. In subdirectory of Task_1 we find the following folders:

```
Commands.dir          holding all command files for this task,
                      e.g. Link command files, setup files,...
```

Data.dir	containing all data files used or made by this task
Documentation.dir	holding all documentation for this task
Execs.dir	holding all executable images related to this task
Includes.dir	holding all Include files (MACROS on IBM) used in the sources for this task
Library.dir	holding all object modules for this task in a dedicated library
Log_files.dir	containing all log_files eventually created by this task
Save_Sets.dir	holding the distribution Save_sets made with BACKUP for this task
Sources.dir	holding all sources related to this task

It is essential for larger enterprises to separate commands from code, documentation, listings, object-code and linked programs.

Using the commands UP and DOWN mentioned earlier as command procedures to go from one directory to another and placing the appropriate file LOGIN.COM into each sub-directory you could set up your own logical environment appropriate for the subject.

Finally it should be mentioned that in the above example Task_2 is complex enough to suggest a further division of the source modules into Partition_1 and Partition_2 respectively. Having done the appropriate Directory structure, there are a few other rules, that might enhance your productivity considerably, if accepted and hence regarded in time:

a) DO not hesitate to make the file names expressive enough so they tell the reader right away what the purpose is, e.g.

```
[CDAQ.Event_Display.DBS]Detector_Geometry.dat

and not   [CDAQ.EVDSP]DTCTRGEO.DAT
```

b) PURge your files whenever a major step has been achieved, save previous key-versions with the command SAVE shown in example

c) MAINtain all object modules in a dedicated library

d) Provide a command file for the linking procedure

e) Establish all regularly occurring steps as commands, so you won't hesitate to do proper actions just because of the typing work

f) Use Names defined in include files rather than values hardwired in your program

e.g. having defined

```
PARAMETER ( RUN_NOT_Active = 10 )
```

in an included file allows to check in your module as follows

```
STATUS = ASK_RUN()
IF (Status .eq. RUN_NOT_Active ) then
 .
elseif ....
```

rather than

```
IF (Status .eq. 10 ) then....
```

obviously the first example is much easier intelligible.

g) Name the file exactly as the modul contained, whenever this is possible, e.g.

the file	Hunde.for
should contain e.g.	Subroutine Hunde(a,b,c)

7.2 How to setup ones environment using LSE,PCA,SCA,...

to be completed in next to next version