L1 MOS

System Software Maintenance
User Guide

olivetti

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

PREFACE

This manual is a guide to be used as an aid to problem determination. It fulfills two functions: firstly it aids the user in resolving errors in the software for himself; secondly, it informs the user of the information which he should send to Olivetti software departement for interpretation if he is unable to resolve an error himself. A detailed hnozledge of the L1 MOS operatin system is assumed.

SUMMARY

- Chapter 1 is concerned with problem determination when there is no system debugger present.
- Chapter 2 deals with the operation of the ROM an RAM debuggers.
- Chapter 3 describes those utilities which act as a back-up to the system debugger.
- Chapter 4 describes in full the Shell maintenance commands.
- Appendix A describes the causes of the errors returned by the operating system primitives, and wherever possible, explains what the user can do to resolve these errors.
- Appendix B deals with the error messages returned by the Shell maintenance commands.
- Appendix C describes the upper bounds certified for some Shell maintenance commands.

REFERENCES

Read first...

SHELL Commands - Reference Manual Code 4002770 Q (vol.3)

For further information, read...

System Software Generation and Installation Reference Manual Code 4002160 B (vol. 7A)

MTS - Configuration Guide Code 4003450 L (vol. 7A)

Terminal Emulators - Configuration Guide Code 400093 U (vol. 7A)

L1 Hardware Architecture Code 4102210 Z (vol. 8)

MOS - Basic Architectural Concepts Code 4002710 J (vol. 8)

MOS - Structure and Functioning Code 4002420 M (vol. 8)

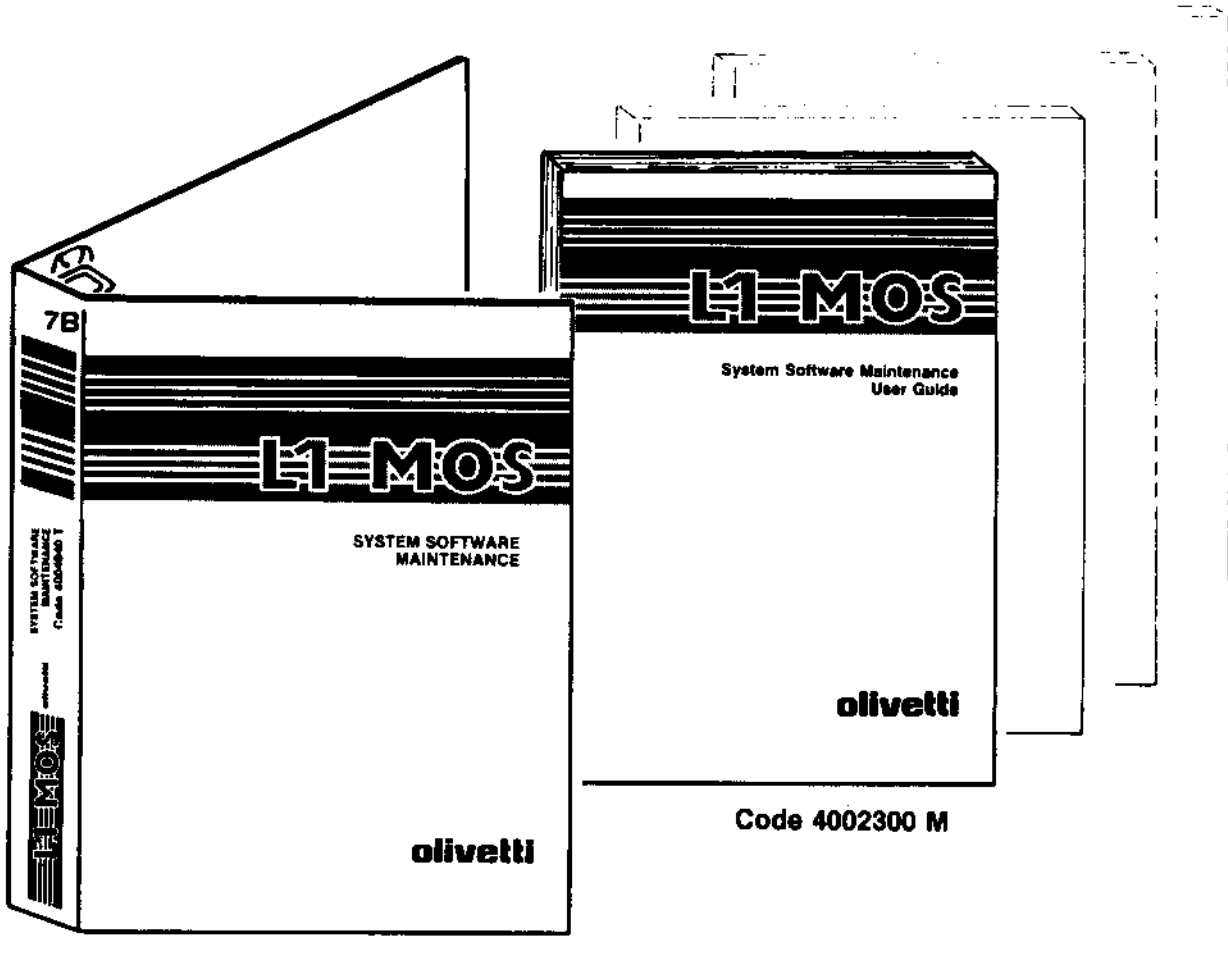
MOS - System Areas - Reference Manual Code 4002410 L (vol. 8)

MOS - Programmer Guide Code 4002570 L (vol.6 G)

FIRST EDITION MARCH 1984 - Release 3.0

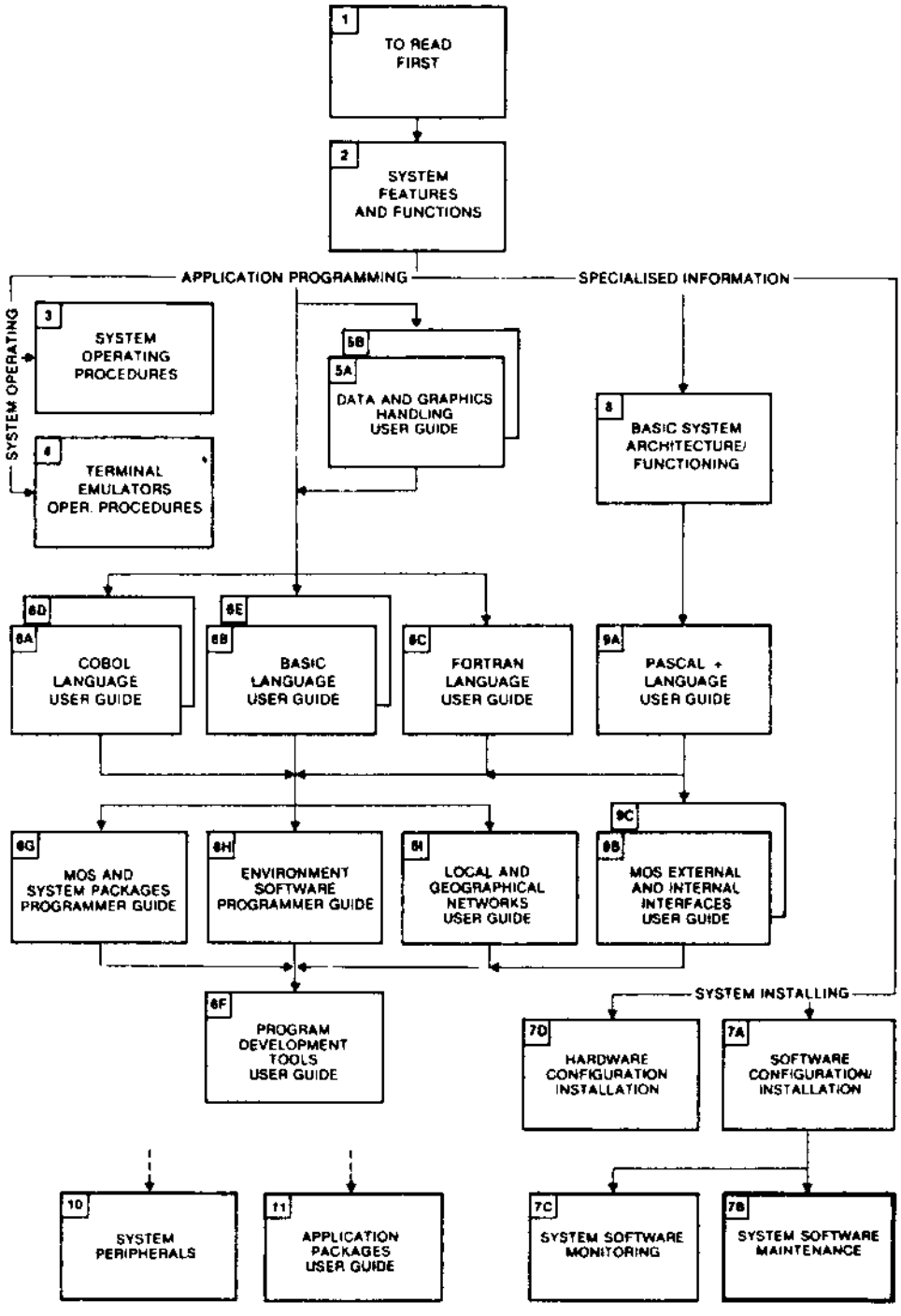
SECOND EDITION September 1984 - Release 4.0

UPDATES: January 1985 - Release 4.1
October 1985 - Release 5.0 also valid for 5.1
May 1987 - Release 5.2



Code 4002300 M

Code 4004940 T


1
TO READ FIRST
2
SYSTEM FEATURES AND FUNCTIONS
APPLICATION PROGRAMMING
SPECIALISED INFORMATION
SYSTEM OPERATING
3
SYSTEM OPERATING PROCEDURES
4
TERMINAL EMULATORS OPER. PROCEDURES
5B
5A
DATA AND GRAPHICS HANDLING USER GUIDE
8
BASIC SYSTEM ARCHITECTURE/ FUNCTIONING
6D
6A
COBOL LANGUAGE USER GUIDE
6E
6B
BASIC LANGUAGE USER GUIDE
6C
FORTRAN LANGUAGE USER GUIDE
9A
PASCAL + LANGUAGE USER GUIDE
9C
9B
MOS EXTERNAL AND INTERNAL INTERFACES USER GUIDE
6G
MOS AND SYSTEM PACKAGES PROGRAMMER GUIDE
6H
ENVIRONMENT SOFTWARE PROGRAMMER GUIDE
6I
LOCAL AND GEOGRAPHICAL NETWORKS USER GUIDE
6F
PROGRAM DEVELOPMENT TOOLS USER GUIDE
SYSTEM INSTALLING
7D
HARDWARE CONFIGURATION INSTALLATION
7A
SOFTWARE CONFIGURATION/ INSTALLATION
10
SYSTEM PERIPHERALS
11
APPLICATION PACKAGES USER GUIDE
7C
SYSTEM SOFTWARE MONITORING
7B
SYSTEM SOFTWARE MAINTENANCE

CONTENTS

2. SYSTEM DEBUGGING

If the operating system fails and there is a system debugger available on the system, the user may send to Olivetti for interpretation all the information as requested in Chapter 1 plus the information derived from the use of the debugger. With MOS operating systems, two system debuggers are available; they both use virtually the same set of commands and provide the same facilities, but one debugger resides in ROM whereas the other is held in RAM. The debuggers are mutually exclusive, and they are activated differently, by the methods described below. The information output by the debuggers includes the register contents, the program status area contents, the contents of specified blocks of memory, and Memory Management Unit segment register contents.

This chapter contains all the information which the user will need to know to be able to use the debuggers effectively.

Further information about many of the concepts dealt with in this chapter may be found in the manual which describes the working of the processor.

WHICH DEBUGGER IS ACTIVATED?

If there is a ROM debugger board present, the only debugger which may be activated is the ROM debugger,in the manner described below. However, the RAM debugger is an l-module that may or may not be loaded at IPL time depending on the configuration file $CON and whether or not there is a ROM debugger board (deb8000). The RAM debugger file name in the IPL starting device must be $DEB, in the SYS volume.
At IPL time, if there is no ROM debugger board, and if the RAM debugger is included in the system configuration file $CON, and the l-module $DEB is included under the volume SYS, the IPL program loads the $DEB l-module into memory, which means that only the RAM debugger may be used.

DEBUGGING TERMINAL

ROM DEBUGGER

At power-on reset time, if the operating system has a ROM debugger present, one terminal is selected as the debugging terminal; this is a terminal whose only function is to run the ROM debugger. By convention, this is the last terminal in relation to its position on the rack of terminal boards. The ROM debugger board contains an instruction to use this terminal. If there is no ROM debugger present, the terminal will be available for normal use. If the user wishes subsequently to select a device other than this default device, commands are available for this purpose. The user should refer to the section of this chapter entitled "Debugger I/O Device Commands".

RAM DEBUGGER

The RAM debugger does not require one terminal to be exclusively dedicated to its operation. However, it may only be activated from one preselected terminal, this terminal reverting to normal use when the RAM debugger is not in operation. The RAM debugger, as a system 1-module, is activated with a configuration record. This contains two parameters, which are:

- The number, in hexadecimal format, of the work station from which the RAM debugger must be activated during the life of the operating system.
  This number "n" may have the following values:

```
0000 :  1st KDC
0001 :  2nd KDC
0002 :  3rd KDC
.... :  .......
```

  where:

  n is the work station number (see Note).

- The positional code in the keyboard table of the PION of the key which must be pressed in order to run the RAM debugger. This may be any key, alphanumeric or function.

USE OF THE KEYBOARD WHILE DEBUGGING

The user must be aware when activating the system debugger that, while the RAM debugger permits the normal use of the alphanumeric keys, when running the ROM debugger on the dedicated debugging terminal, for terminals with a full keyboard of alphanumeric and function keys, the alphanumeric keys are disabled (except for the control key) and the function keys have values other than those indicated.

For each of the standard keyboards, business and scientific, there is given below the layout of the function keys, and a table showing the values which these keys have when the terminal is the debugging terminal. A key not included in a table has the same value whether the terminal is the debugging terminal or not.

Note

The terminal can only be of KDC type or an industry standard one (VT100-like, M10, etc.) connected to the RS 232 of the TWIN or CPU board.

ACTIVATION OF THE SYSTEM DEBUGGER

ROM DEBUGGER

Whenever the power is restored to the operating system (using the reset button), the operator is able to choose when the debugger is to be activated by typing either

d

or

g

Typing 'd' indicates that the debugger is to run before the initialisation procedure has been completed by the ROM loader. This is necessary when the user requires to inspect memory locations of the crashed system. Typing 'g' indicates that the debugger is to run after the ROM loader has completed the initialisation procedure. This is the default sequence of events and as long as this sequence of events is required, there is no need to alter the situation. Note that it is not possible to switch between the two environments after the choice has been made until the next power-on reset.

RAM DEBUGGER

The RAM debugger is activated from the preselected terminal by pressing the key indicated in the configuration record: the positional code of the default key is the hexadecimal constant 42 (see Fig. 2-1). The current contents of the video are immediately saved in a private area of the RAM debugger and the current contents of the registers are then displayed in the same format as for the 'R' command (q.v). The user can now use the RAM debugger. When the debug session finishes the user can retrieve the contents of the video saved at the beginning of the debugging session, by pressing the key "g" (GO command), and continue the running the application.

DEBUGGER COMMANDS

COMMAND PARAMETERS

The debugger command parameters are numbers which are keyed in or displayed in hexadecimal ASCII format.

Three types of parameter are defined, as follows:

- Byte: a number constituted by one or two hexadecimal characters (1 byte = 8 bits = 2 hexadecimal characters).

- Word expression: a word expression may be entered as one or more words, separated by either '+' or '-'. Each word may be one or more hexadecimal characters, with the last four characters significant. No

spaces or other delimiters are allowed within the expression. For example:

word=1000-23+16-300

(1 word = 2 bytes normally = 16 bits = 4 hexadecimal characters).

- Address: an address is specified as

offset

or optionally as an offset, in which case the segment is defaulted to the last segment entered. The segment is specified as a byte and the offset as a word expression.

All command lines must be terminated with a carriage return. In the command syntax definition optional parameters are enclosed in brackets []. If these parameters are omitted, default values will be inserted automatically.

COMMAND LIST

Below is given a list of the available commands, with a brief explanation for each.

- Display a block of memory or a single location ('D').
- Disassemble and display one or more instructions (RAM debugger only)('DD').
- Search a block of memory for a specified word ('FN').
- Displays the addresses of all last breakpoints executed ('H')
- Move blocks of memory ('M').
- Set or display a breakpoint ('B').
- Clear breakpoints ('BC').
- Set or display a stop (ROM debugger only) ('S').
- Display and permanently modify the registers of the Memory Management Unit segments (only for the RAM debugger) ('SD')
- Examine MMU segment registers ('SR')
- Modify the MMU segment registers ('SW').
- Clear stops (ROM debugger only) ('SC').
- Change from one CPU/MMU to another ('Z').
- Displays and modifies the last command entered ('?').

- Cause a single step operation to be performed ('N').
- Scans and displays system structures, codes and lists ('QD', 'QS', 'Q$', 'Q').
- Display registers ('R').
- Read from or write to a specified I/O port ('PR', 'PW').
- Restart a user program following a breakpoint or a stop ('J', 'G').
- Reselect the ROM debugger I/O device (ROM debugger only) ('X', 'C').
- Allows forced entry into the RAM debugger: NOT MASKABLE INTERRUPT Generator.

In this section the debugger commands are described in alphabetical order.

B : BREAKPOINT COMMAND

The 'B' command either sets a breakpoint within a program, or displays the currently active breakpoint. If 'B' is typed without any parameters, the current breakpoints are displayed. If an address parameter is entered, a breakpoint is set at that address.

It is possible to specify a breakpoint counter; if such a counter is specified, processing will halt for the specified number of times the breakpoint is encountered, and the breakpoint conditions, if any, are satisfied. One or two break conditions may be specified according to the contents of the processor registers. It is also possible to compare the contents of a data address with a specified value (only for RAM debugger).

The syntax of this command requires that the break address be the first word of an instruction. When a 'Breakpoint' command is executed, the extended processing bit in the flag and control word (FCW) must be set to zero. If this is the case, the current instruction in memory is replaced with an extended processing instruction, (which informs the processor what to do at a breakpoint) and the original instruction is saved in a table together with the break address. When the processor attempts to execute the extended processing instruction, as there is apparently no Extended Processing Unit connected to the CPU, a software trap occurs, the registers are saved, and the instruction is restored. The contents of all the registers are displayed on the screen at this point. Control is then returned to the monitor.

If however, the extended processing bit in the FCW is set to 1, the extended processing instruction may be executed, and therefore the software trap is disabled, and the Breakpoint command cannot function.

The program counter is adjusted so that when the user requests execution of the program to recommence, (using the 'Go' command), the replaced instruction will be executed correctly. Following a break, all the processor registers are saved in memory, except for the refresh counter.

The 'B' command is the main system debugger command, and it will operate in normal, system, segmented, or non-segmented modes. Note that the extended processing bit in the flag and control word (FCW) must never be set, as this will disable the trap. Programs which use extended processing instructions may specify breakpoints using the 'BX'command. This command specifies that the break instruction for this location is to be a privileged instruction. Thus programs which enable the extended processing bit may be debugged in normal mode.

The syntax of the 'B' command varies according to whether the system is running under the ROM or the RAM debugger.

Format for ROM Debugger

B[X] [breakpoint-address] [breakpoint-counter] [register1 value]
[register2 value]

PR,PW : I/O PORT COMMANDS

These commands allow reading from or writing to a specified i/o port. Note that the commands use the processor byte input and output instructions.

The commands have the following syntax:

```
PR port-address

PW port-address byte-value
```

where :

port-address is the address of the required port as a word expression.

byte-value is the value that is to be written to the required port.

QD : SYSTEM STRUCTURE SCANNING COMMAND

This command causes a symbolic name to be associated to queues or lists system.
The syntax of this command is as follows:

```
QD symbolic-name
```

where:

symbolic-name is the symbolic name of the structure to be scanned.

The user is requested to provide the following information after command activation:

*** ITEM SIZE ***	The size of the elements in the structure .
*** START OFFSET ***	The offset between the start of the element and the structure pointer, e.g. the offset for the queue pointer.
*** NEXT OFFSET ***	The start offset in the element of the field that contains the pointer to the next element.
*** START INDEX ***	Where the index is to start in the structure. If the structure is a list, this must be indicated. if it is a queue the field must be left blank.
*** QUEUE POINTER ***	The the logical address to which the queue or list is allocated.

The following commands can be used when scanning the structure:

QS : this command displays the symbolic name associated to the structure.
The syntax for the command 'QS' is as follows:

```
QS
```

Q$: this command displays the first element in the specified structure.
The syntax of the command 'Q$' is as follows:

```
Q$ symbolic-name
```

dove:

symbolic-name is the symbolic name associated to the system structure.

The following message is displayed after the command is activated:

CURRENTLY EXAMINED CPU NAME: - INPUT NAME:

THis specifies the number of the current CPU, and ashs the number of the CPU to be used. This may be:

0 for I CPU
4 for II CPU
8 for III CPU

? : LAST COMMAND MODIFICATION AND HISTORY COMMAND

This command displays the last command entered so that it can be modified.
The syntax is as follows.

?

NOT MASKABLE INTERRUPT GENERATOR

The not maskable interrupt generator is a button which, when pressed, causes a NMI (not maskable interrupt) that is on to the RAM debugger.

It is used to try to access the debugger if the system is blocked (e.g. it loops in a Vectonsed Interrupt routine), and pressing the required key on the keyboard does not activate the debugger.

If the system is blocked on NMI it is impossible to activate the RAM debugger.

3. OTHER DEBUGGING TOOLS

This chapter describes the operator interface of a number of utilities which may be used to complement the facilities offered by the ROM and RAM debuggers. (These programs are described in Chapter 2 of this document). The following utilities are dealt with:

- The Total Memory Dump Interpreter (TMDUMP).
- The Read/Alter Memory Utility, which enables the contents of memory to be read and/or modified (READALTM).

Note that TMDUMP and READALTM both use the same types of commands as the RAM debugger. (See Chapter 2).

DISPLAYING AND MODIFYING THE MEMORY LOCATIONS (READALTM)

RDALTER is a program which displays the contents of a given segment of a family selected by the user, and allows individual memory locations to be altered.

This utility can also be used in a multiprocessor environment.

ACTIVATING READALTM

READALTM is activated in the Shell environment simply by typing in the program name:

```
READALTM
```

When activated, READALTM responds with its own prompt,

```
ENTER COMMAND [HELP]:
```

Typing HELP produces a list of the commands offered by READALTM and their syntax, plus two important notes:

- All values must be entered in hexadecimal.
- Unless these are changed by the user, the default family number is 0, and the default process number is the first process of the requested family.

COMMAND PARAMETERS

The parameters of the READALTM command are numbers that are typed and displayed in ASCII and hexadecimal format.

Two types of parameters are defined, as follows:

- Word : a word consisting of two bytes (1 word = 2 byte = 16 bit = 4 hexadecimal characters).

- Address: an address is specified as:

   offset

  Segment is specified as a byte, and offset as a word.

All the lines in the command must be ended by pressing carriage return. In the command syntax, the optional parameters are shown in square brackets []. If these parameters are omitted, the default values are inserted automatically.

COMMAND LIST

A list is given below of the commands that are available, with a brief description of each of them.

- Displays a specific memory block or changes the contents of the memory word ('DM').

- Searches for a specific word ('FN') in a specified memory block.

- Exits from READALTM ('Q').

- Displays the entry of the MMU table corresponding to the specified segment ('SR').

- Selects the current family ('SWF').

- Selects the current process ('SWP').

DM : DISPLAY MEMORY

This command displays the contents of memory locations as selected by the user, and allows the user to change the current location.

The user can choose either to display the whole of a segment, or only a specific number of bytes in it; the contents of the memory at the current location can also be modified, by typing a valid hexadecimal number. The next word is displayed by pressing the /CR/ key. The user can exit from modify mode by pressing 'Q' and then /CR/.

The syntax of the 'DM' command is as follows:

DM [X] <seg-no> start-offset [no-bytes]

---

where:

seg-no is the segment number requested in the format

<ss>

ss is segment number made up of two hexadecimal digits from 0 to 3F or 7F inclusive, depending whethere there are one or two MMUs.

The segment number is relative to the current family address space, or to the stack segment (see the commands SWF and SWP).

start-offset is a number made up of max. four hexadecimal digits that specifies the start location to be displayed.

no-bytes is a hexadecimal number which shows how many bytes to display. If it is not provided, the word at the specified offset can be modified.

For example:

If the user types in

```
DM <1> 234 10
```

the following is output:

```
<01>0234:  D300 0000 D700 0000 FE00 0000 FD00 0000
```

The 'X' option causes the ASCII representation to be displayed along side.

If the user types in

```
DMX <3> 1234 30
```

the information given is as follows:

<03>1234: 34CA 033D 0CA5 1919 34CA 033C 0CA5 2828 "4,,=,,,,4,,<,,(("

FN : FIND WORD

This command searches a block of memory for a specified word pattern. Note that all the memory addresses in the input space containing the specified word are displayed. The command has the following syntax.

FN <seg-no>start-offset no-bytes value

---

where:

seg-no , **start-offset** , and **no-bytes** are all as defined above.

value is a word expression defined as in the introduction to this document.

For example:

If the user types in

FN <3> 1234 30 9E08

READALTM outputs the memory address which contains the specified word, which in this case is

<03>125A

Q : QUIT

This command is used to exit from the READALTM environment, and return to the user's original prompt. The command has no parameters, and therefore it's syntax is

---

Q [UIT]

This command can be specified both in upper and lower case characters.

SR : READ SEGMENT

This command allows memory management unit (MMU) segments to be examined. The registers of the required MMU segment are read and displayed on the screen. MMUs are described fully in the manual "L1 - Hardware Architecture"

The syntax of the 'SR' command is as follows:

```
SR <seg-no>
```

---

where:

seg-no is as defined for the 'DM' command. For example:

In response to the user input,

```
SR <1>
```

the following is output:

```
BASE    LIM     ATTR
16FC     03       C0
```

where:

BASE , **LIM** , and **ATTR** are the values base, limit, and attribute fields of the MMU register "Segment Descriptor".

SWF : SET WORKING FAMILY

This command sets the current working family, as it allows the user to specify a family number. The numbers of the active segments and the processes of the specified family are displayed. Subsequent commands ('DM', 'FN', etc.) will refer to segments of the selected family.

The command has the following syntax:

```
SWF family-no
```

---

where:

family-no is a valid family number.

An example of this command follows.

In response to the user input:

```
SWF 1
```

the following information is given:

```
ACTIVE SEGMENTS OF THE FAMILY 01 ARE :   (3C, 3D)
PROCESSES OF THE FAMILY 01 ARE :         (00, 04)
```

SWP : SET WORKING PROCESS

This command allows the user to select a process within a specified family. Subsequent 'DM' and 'FN' commands will refer to the stack segments of such a process. The command has the following syntax:

```
SWP process-no
```

---

where:

process-no is a valid process number.

If the process that is selected does not belong to the current family, a message is displayed.

TOTAL MEMORY DUMP

When the operating system crashes, a total memory dump can be initiated by an operator, as follows:

1. Push the computer cabinet RESET button.
2. Check that the numeral "1" is displayed on the computer panel.
3. Check that the letter "d" is displayed on the computer panel.
4. Press the IPL switch within three seconds.
5. Check on which terminal keyboard the L1 and L2 lights are lit.
6. When step 5 has been carried out, press the "0" key on the numeric keyboard within 30 seconds.
7. Check that the message "memory dump procedure, wait please 0000 0000" is being displayed. ("0000 0000" means that no parity errors were found when the dump program checked the contents of the RAM).
8. Put an unformatted disk in the unit that is to be used.
9. Check that the message "select unit n" is being displayed, where "n" is a number from 1 to 4, and give type the number of the unit the disk is in.
10. Check that the message "dump running; disk 1" is being displayed.
11. Wait till the disk has been recorded.
12. If there is not enough room on the disk to contain all the data being dumped, put another disk in the unit and repeat steps 11 and 12 until all the data has been recorded. (Messages will be displayed as in step 10 for each disk).
13. When dumping finishes, the message "memory dump completed, press reset for system restart" is displayed.
14. If the system is to be restarted, press the RESET button.
15. To interpret the dumped data, see the next section.

INTERPRETING THE TOTAL MEMORY DUMP

When an operating system has crashed, and the memory image has been dumped to floppy disk, the floppy disk may be mounted on another system, and the interactive utility TMDUMP may be run to interpret the data obtained. The commands which may be input to TMDUMP are outlined below.

ACTIVATING TMDUMP

TMDUMP is activated in the Shell environment by typing in the command line:

```
TMDUMP input-device [printer-name]
```

where

input-device is the path name of the device on which the floppy disk containing the memory dump resides.

printer-name is the name of the printer on which the information obtained through the use of TMDUMP is to be printed.

If no printer name is given, then the information given by TMDUMP is only displayed.

The first message output by TMDUMP is:

```
DISK_BEGINNING  start-page

DISK_ENDING     end-page

ENTER NUMBER OF CONFIGURATED FAMILY (DECIMAL):

ENTER NUMBER OF CONFIGURATED PROCESS (DECIMAL):

ENTER OFFSET OF SEGMENT TABLE:
```

where:

start-page and **end-page** are the limits in pages of the physical memory that has been dumped (one page is 256 bytes).

In reply to the next two prompts the user should write the number of families and the number of processes configured, respectively, in decimals as indicated.

These values can be obtained by the $CON file of the machine on which the dump has been carried out.

The user should then specify the offset of the segment table. This value is stored in a specific location in memory that may vary according to release. For release 5.2 the location is as follows:

<0B>0012 for monoprocessors
<0B>0016 for multiprocessors

If this value is not entered, the following prompt is displayed:

ENTER COMMAND [HELP]:

and the only commands which function are HELP, DM, FN and SR. If any other command is entered, the message

SEGMENT TABLE NOT AVAILABLE

is output.

When all the initial data has been input, TMDUMP outputs the prompt,

ENTER COMMAND (HELP):

Any TMDUMP command may be entered in response to this prompt. If the user types HELP in reply to the prompt, a list of the commands available for input to TMDUMP is displayed, together with information on their syntax.

COMMAND PARAMETERS

The TMDUMP command parameters are numbers which are typed or displayed in hexadecimal ASCII format.

Two types of parameters are defined as follows:

- Word: a word consists of two bytes (1 word = 2 byte = 16 bit = 4 hexadecimal characters).

- Address: an address is specified as

  offset

  Segment is specified as a byte, and offset as a word.

All command lines must be terminated by pressing carriage return /CR/. In the command syntax definition optional parameters are enclosed in square brackets [].

Note that if the memory image of the crashed operating system has been dumped to more than one floppy disk, and an address is input as command parameter which is outside the bounds of the currently loaded floppy disk, the following message is output:

INCORRECT FLOPPY name number

CORRECT FLOPPY name number

TO CONTINUE HIT - C - OR HIT - Q - TO EXIT

If /C/ is pressed, the following instruction is displayed:

INSERT THE CORRECT FLOPPY AND HIT /CR/

The user should then replace the floppy disk with another one.

If /Q/ is pressed, the program exits from this state, and TMDUMP can accept another command.

COMMAND LIST

A list of the commands that are available is given below, with a brief description or each of them.

- Displays the base, limit and attribute fields of the segments of the family specified or all the segments in the table ('DF').
- Displays a specified memory block ('DM').
- Displays the base, limitand attribute fields of the stack for the requested process ('DP').
- Search a block of memory for a specified word ('FN').
- Print out data from the crashed operating system ('PR').
- Exit from TMDUMP ('Q').
- Displays programming of a segment or the whole of the MMU table ('SR').
- Set the current working family ('SWF').
- Set the current working process ('SWP').

DF : DISPLAY SEGMENTS TABLE

This command displays information about the segments relating to a particular entry in the segments table. If no entry is specified, the entire segment table is displayed (or printed).

This command has the following syntax:

```
DF[P] [family-no]
```

---

where:

family-no is a valid family number.

If 'P' is typed, then the output from this command is directed to the printer which was specified in the activation command for TMDUMP, provided that such a device was specified.

An example of the use of this command is given below.

If the user types in:

```
DF 01
```

TMDUMP responds with:

```
FAMILY: 01
SEG     BASE    LIMIT   ATTRIB
[3C]    0AE0      41      81
[3D]    0B22      0A      C0
```

where:

3C and 3D are the segments in family 01.

BASE , **LIMIT** and **ATTRIB** are the values of the base, limit, and attribute fields of the MMU register "Segment descriptor".

If the family number is is not specified, the whole segment table is printed in the following format:

```
FAMILY: 01

    SEG     BASE    LIMIT   ATTRIB.
    <00>    0000      FF       C0
    <01>    17FC      FF       C0
    <02>    12F2      BE       C0
     |       |        |        |

FAMILY : 02

    SEG     BASE    LIMIT   ATTRIB.
     |       |        |        |
```

DM : DISPLAY MEMORY

This command displays on the screen the contents of memory locations as selected by the user.

By typing an address after the command name, the user may choose to see the contents of a whole segment, or the contents of a number of bytes within the segment. The command has the following syntax.

DM [X] [P] <seg-no> start-offset no-bytes

---

where:

seg-no is the number of the segment required in the format

<ss>

where ss is a two-digit hexadecimal number in the range from 0 to 3F or 7F, depending on whether there are one or two MMUs.

Note that the segment number is relative to the current family address space, or a stack segment (see the SWF and SWP commands).

start-offset is a four digit hexadecimal number that specifies the start location from which to begin display.

no-bytes is the number of bytes to be displayed.

For example:

If the user types in

```
DM <1> 234 10
```

the following is output:

```
<01>0234:  D300 0000 D700 0000 FE00 0000 FD00 0000
```

The 'X' option causes the ASCII representation to be displayed along side, thus:

If the user types in

```
DMX <3> 1234 30
```

the information given is as follows:

```
<03>1234:   34CA 033D 0CA5 1919 34CA 033C 0CA5 2828 "4,,=,,,,4,,<,,(("
<03>1244:   E808 34CA 033D 0CA5 1919 34CA 033C 0CA5 ",,4,,=,,,,4,,<,,"
<03>1254:   5050 010F 0012 9E08 030F 0012 0DE5 0016 "PP,,,,,,,,,,,,,,"
```

Note that if the screen is full, the user can select the output of further information by typing in the subcommand 'MORE'.

DP : DISPLAY PROCESS

This command displays information about the segments relating to a particular process. If no process is specified, the stacks of all the processes are displayed or printed.

This command has the following syntax:

```
DP[P] [process-no]
```

---

where:

process-no is a valid process number.

If 'P' is typed, then the output from this command is directed to the printer which was specified in the activation command for TMDUMP, provided that such a device was specified.

An example of the use of this command is given below.

If the user types in:

```
DP 0B
```

TMDUMP responds with:

```
PROCESS: 0B

SEG     BASE    LIMIT   ATTRIB

[3F]    0C49      E8      20
```

where:

3F is the segment in process 0B.

BASE , **LIMIT** and **ATTRIB** are the values of the base, limit, and attribute fields of the MMU register "Segment descriptor".

Alternatively, if a process selected is not active, the message:

```
WARNING : THIS PROCESS IS NOT ACTIVE
```

is displayed.

FN : FIND WORD

This command searches a block of memory for a specified word pattern. Note that all the memory addresses in the input space containing the specified word are displayed. The command has the following syntax.

```
FN <seg-no> start-offset no-bytes value
```

where:

seg-no , **start-offset** , and **no-bytes** are all as defined above.

value is a word expression defined as in the introduction to this document.

For example:

If the user types in

```
FN <3> 1234 30 9E08
```

TMDUMP outputs the memory address which contains the specified word,

which in this case is

```
<03>125A
```

(See the 'DM' command).

P : PRINT DUMP

This command is used to print out areas of the crashed operating system, namely:

- the MMU registers in the same format as that of the 'SR' command
- the segments table in the same format as that of the 'DF' command
- all the system data segments and all the user data segments in the same format as that of the 'DM' command.

This command has no parameters, and thus has the following syntax:

```
PR
```

Q : QUIT

This command is used to exit from the TMDUMP environment, and return to the user's original prompt. The command has no parameters, and therefore it's syntax is

```
Q
```

SR : READ SEGMENT

This command allows memory management unit (MMU) segments to be examined. The registers of the required MMU segment are read and displayed on the screen. MMUs are described fully in the manual "L1 - Hardware Architetture"

The 'SR' command has the following syntax:

```
SR [ P ] <seg-no>
```

where:

seg-no is defined as for the 'DM' command. For example:

If 'P' is typed, then the output from this command is directed to the printer which was specified in the activation command for TMDUMP, provided that such a device was specified.

An example of how this command is used follows. If

```
SR <8>
```

is type, the following is displayed:

```
BASE     LIMIT     ATTRIB

16FC      03         C0
```

where:

BASE, LIMIT e ATTRIB are the value of the base, limit and attribute fields of the MMU register "Segment Descriptor".

SWF : SET WORKING FAMILY

This command sets the current working family, as it allows the user to specify a family number. The numbers of the active segments and the processes of the specified family are displayed. Subsequent commands ('DM', 'FN', etc.) will refer to segments of the selected family.

The command has the following syntax:

```
SWF [family-no]
```

---

where:

family-no is a valid family number.

SWP: SET WORKING PROCESS

This command enables the user to selecte a process in a specified family. The next 'DM' and 'FN' commands will be for the stack segments of that process. The syntax of the command is as follows:

```
SWP [process-no]
```

where:

process-no is a valid process number. If no process is specified, the first entry in the Process table is assumed by default.

ENTER FILE NAME:

Prompts you to enter the name of an l-module in the current working directory.

name/CR/	displays details of the message field of the specified l-module.
/CR/	displays the total size of all the l-modules specified. Control is then returned to Shell.

MESSAGE: operator-date-configuration

Gives the message field of the specified l-module.

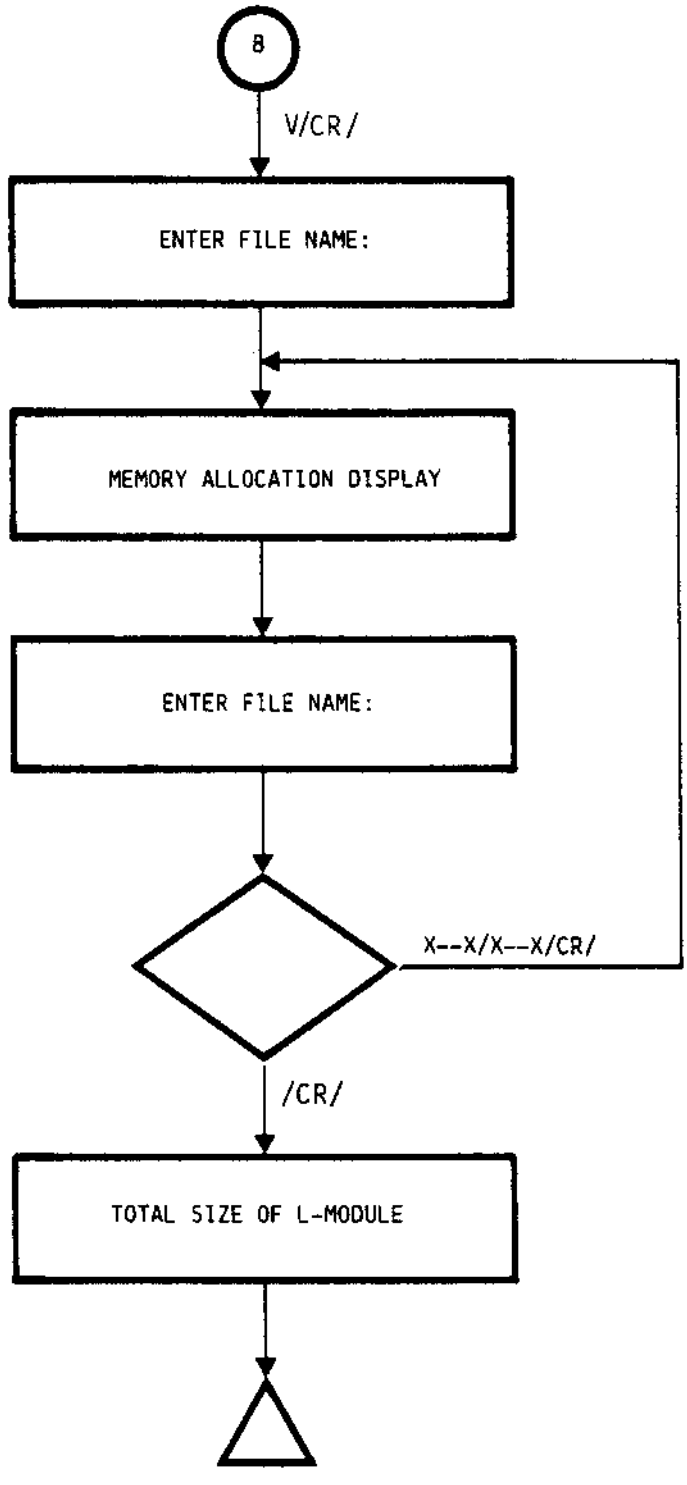
operator	is the abbreviated name of the programmer who linked the l-module, or another identifying code chosen by the programmer
date	is the date when the l-module was linked.
configuration	is a character string indicating the contents of the l-module. This information is compulsory only for the $CHM, and $IPL modules, where it shows the peripheral driver programs present. The character codes used in the strings for these modules are described in MOS - System Software Generation and Installing for the $IPL l-module and in "CSS Generation and Configuration User Guide" for the $CMM l-module. In other cases the string may be used to record any relevant information.

---

filename CODE & DATA SIZE : k..k KBYTES PLUS b..b BYTES

The specified file will occupy a total of k..k Kbytes plus b..b bytes of memory in all segments used.

DISPLAY THE MEMORY ALLOCATION
OF AN L-MODULE



If an error occurs, it is signalled in the appropriate section, and DISKCHECK may terminate prematurely. Errors are reported as follows. Blocks, PDDs and extents may each be reported as missing. Missing PDDs or extents limit the amount of disk space which may be handled by the system, but do not significantly lower the quality of system performance. Missing blocks are divided into two categories:

- Set but not allocated.

- Allocated but not set.

Blocks set but not allocated are inaccessible by the system, and thus the volume will seem smaller than it really is. The effects on the system are not serious. On the other hand, blocks allocated but not set indicate that valid data is stored on disk in locations flagged as being free. This means that data may be overwritten when more space is required on the volume. To remedy this problem, these blocks should be flagged as busy in the bit map. The certified limit (max number of extents) that can be handled by the utility is indicated in "Appendix C".

ERROR MESSAGES

"Appendix B" contains all the error messages that may be diplayed if errors occur during execution of the DISKCHECK utility.

```
VOLUME TO DUMP: IPL/SYS

SECTION 1:  FREE PDD LIST CHECKING

SECTION 2:  FREE EXTENT LIST CHECKING

SECTION 3:  PDD EXTENT CHECKING

SECTION 4:  BLOCK ALLOCATION CHECKING

SECTION 5:  FILE STRUCTURE CHECKING

SECTION 6:  MISSING PDDS CHECKING

SECTION 7:  MISSING EXTS CHECKING

SECTION 8   SUMMARY

VOLUME : IPL/SYS
        ACTIVE DATASET ENTRIES
DATASETS . . . . . 23
DIRECTORIES. . . . . .1
NESTED LOGICALVOLUMES. . . .0
        PDD RECORDS
ACTIVE. . . . . .24
FREE. . . . . .4
MISSING. . . . . .0
        EXTENT RECORDS
ACTIVE. . . . . .0
FREE. . . . . .56
MISSING. . . . . .0
        BLOCK ALLOCATION
FREE . . . . . 289
ACTIVE. . . . . 1176
MISSING . . . . . 0
DISKCHECK FINISHED  IPL 3/3
```

Fig. 3-1 DISKCHECK Output - Summary

DISKEDIT is an interactive program that allows you to modify the internal data structures of a disk volume. The following structures can be examined and modified:

- Bit map
- Extent tables
- PDD records
- Root directory
- Volume descriptor.

The program offers a main menu of options and a series of submenus. These are shown in the following pages.

When a data structure is to be modified, the current values are displayed and new values are requested one at a time.

DISKEDIT vol-name ["W]

---

where:

vol-name is the complete pathname of a volume or device.

W is an option by means of which the user can select whether to modify the internal structure of the data in the volume.
If no option is specified, the contents of the volume may only be displayed.

When DISKEDIT is activated, a message is displayed showing the type of activity chosen by the user.

- If the user has chosen only to display the contents of the volume

  WARNING : READ ONLY

- If the user has chosen both to display and modify the contents of the volume

  WARNING : READ AND WRITE MODE

In both cases, the following message is displayed next:

```
DISKEDIT PROGRAM ENTERED. TYPE ? FOR HELP
```

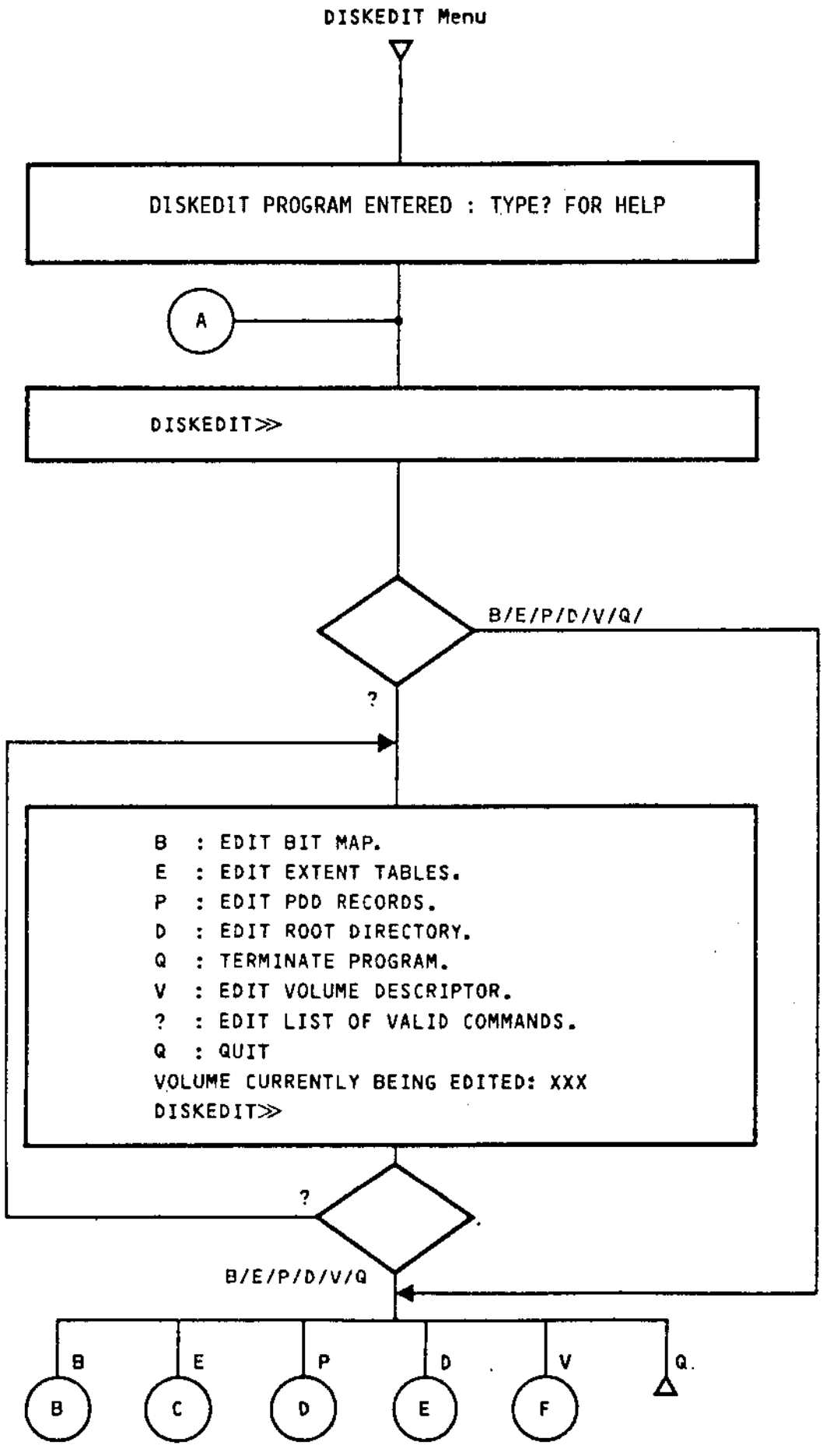
and then the utility prompt

```
DISKEDIT >>
```

A description of the all the menus in the utility follows.

ERROR MESSAGES

Appendix B contains all the error messages that may be displayed if errors occur during execution of the DISKEDIT utility.


DISKEDIT Menu
DISKEDIT PROGRAM ENTERED : TYPE? FOR HELP
A
DISKEDIT≫
B/E/P/D/V/Q/
?
B : EDIT BIT MAP.
E : EDIT EXTENT TABLES.
P : EDIT PDD RECORDS.
D : EDIT ROOT DIRECTORY.
Q : TERMINATE PROGRAM.
V : EDIT VOLUME DESCRIPTOR.
? : EDIT LIST OF VALID COMMANDS.
Q : QUIT
VOLUME CURRENTLY BEING EDITED: XXX
DISKEDIT≫
?
B/E/P/D/V/Q
B
E
P
D
V
Q.
B
C
D
E
F

Message Description

VDESC >>

Prompts you to select an edit volume descriptor option. The contents of the volume descriptor are described in the manual "MOS System Areas, Reference Manual".

? /CR/	Displays the edit volume descriptor option menu.
R,W /CR/	Selects an edit volume descriptor option.
Q /CR/	Returns you to the "DISKEDIT" prompt.

HEXED

The HEXED command displays and/or modifies the contents of a file of a l-module in hexadecimal format and ASCII, or in EBCDIC.

The user can edit the contents of the file/l-module by entering characters next to the cursor.

The syntax for this command is as follows:

HEXED filename [" E]

---

where:

filename is the name of the file or l-modulo to display and/or edit. It can be a complete pathname, or the name of a file/l-module in the current work directory.

E is the option for the display to be in EBCDIC. If this is not specified, the default is ASCII.
The option can be specified both in upper and lower case characters.

Characteristics of the HEXED Command

There are two modes for operating with the HEXED command:

- FILE mode: environment in which the files are processed; default mode
- PROGRAM mode: environment in which the l-modules are processed; this mode must be selected when required

When HEXED is activated, the operaing mode is FILE and the utility displayed a screen page like the one below.
The first page displayed corresponds to the first page of the file.

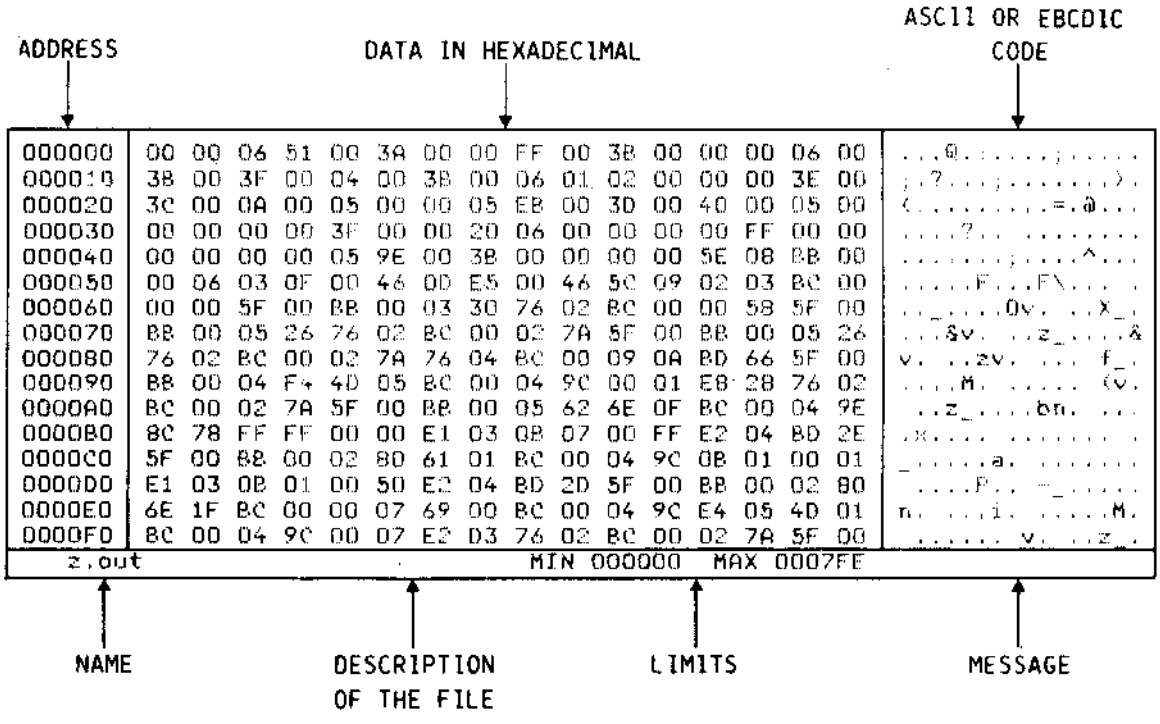


Fig. 4-15/A Screen Page associated to Displaying a File

where:

ADDRESS shows the address in hexadecimals of the first byte of each of the lines of 16 byte in the data field.

DATA IN HEXADECIMALS shows a page of 256-byte of hexadecimal data. The data are presented as 16 lines of 16-byte. If there are insufficient bytes to fill the page, the rest of the page is left blank.

ASCII shows the ASCII representation of the data field. If a byte represents a character that cannot be printed, a dot "." is displayed. When the option "E" or "e"is used, the characters in this field are displayed in EBCDIC format.

DESCRIPTION OF THE FILE shows the name of the file (NAME) that is being edited, and the first and last addresses in the file (LIMITS).

MESSAGE is used to display a warning message for the user.

When PROGRAM mode is selected the page described below is displayed. In PROGRAM mode the first page displayed corresponds to the 1-module entry-point.

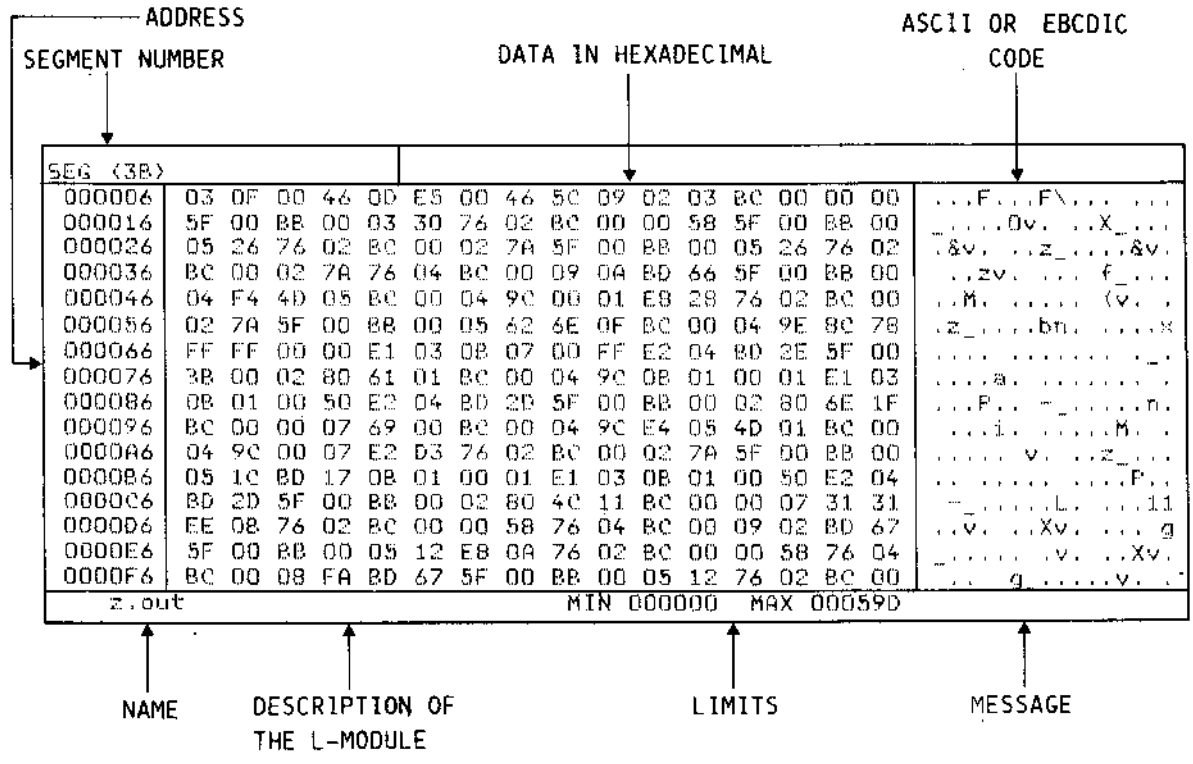


Fig. 4-15/3 Screen Page Corresponding to display of an l-module

where:

ADDRESS, DATA IN HEXADECIMALS, ASCII, MESSAGE have the same meaning as above.

DESCRIPTION OF THE L-MODULE shows the name of the l-module (NAME) that is being edited, and the first and last addresses (LIMITS) of the l-module.

SEG gives the segment in which this module of the l-module is allocated.

LIST OF COMMANDS

There are two basic types of command:

- immediate commands: that can be activated directly from the "data in hexadecimals" or "ASCII" windows in the screen.

- non-immediate commands: that must be activated by positioning the cursor on the specified screen window

All the utility commands, except for the command "display an l-module header" that is specific to the PROGRAM environment, are common to the both the HEXED operating modes.

A list of the commands is given below for each of the two types, with a brief description.

List of Immediate Commands

- Displays the list of commands ('?').

- Moves the cursor within the screen window, from which the non-immediate commands can be activated (':').

- Displays the previous ('B') / forward ('F') page.

- Move the cursor ('J', 'K', 'H', 'L', '/CR/').

- Displayed the first ('g') / last ('G') page in the file/l-module.

- Displays information about the file/l-module (' **CTRL + G** ').

- Searches for the next ('n') / previous ('N') occurrence of a string.

- Exits from HEXED ('Q').

- Replaces one ('r') or more ('R') bytes.

List of Non-Immediate Commands

- Searches for an ASCII/Hexadecimal string ('/', '?', '%', '$').

- Displays the header of an l-module ('H').

- Skips to a specific address ('J').

- Presets the utility for specific operations ('S').

IMMEDIATE COMMANDS

All the "immediate commands" available in HEXED are given below:

B : DISPLAYING THE PREVIOUS PAGE

This command displays the page before the current one, of the current file or l-module.

If the current page is the first one, it is displayed again.

The command "B" can be specified both in upper and lower case characters; its syntax is as follows:

```
B
```

F : DISPLAYING THE NEXT PAGE

Displays the next page to the current one, of the current file or l-module.

If the current page is the last one, the first one is displayed.

The command "F" can be specified both in upper and lower case characters; its syntax is as follows:

```
F
```

g : POSITIONING AT THE BEGINNING

Positions the cursor on the beginning of the current file or l-module, and displays the first page.

The command must be only be specified in upper case characters; its syntax is as follows:

```
g
```

G : POSITIONING AT THE END

Positions the cursor at the end of the current file or l-module, and displays the last page.

The command "G" must only be specified in upper case characters; its syntax is as follows:

G

CTRL + G : DISPLAYING INFORMATION

The command " **CTRL + G** " displays the name and the limits of the current file or l-module that is active at the bottom of the screen.

The command may be only specified in upper case characters; its syntax is as follows:

CTRL + G

H : MOVING THE CURSOR TO THE LEFT

The command "H" moves the cursor to the left of a byte. It can only be used in the current page.

The command can be specified both in upper and lower case characters; its syntax is as follows:

H

J : MOVING THE CURSOR DOWN

The command "J" moves the cursor down a line. It can only be used in the current page.

The command can be specified both in upper and lower case characters; its syntax is as follows:

```
J
```

K : MOVING THE CURSOR UP

Moves the cursor up a line. The command "K" can only be used in the current page.

The command can be written both in upper and lower case characters; its syntax is as follows:

```
K
```

L : MOVING THE CURSOR TO THE RIGHT

Moves the cursor to the right of a byte. The command "L" can only be used in the current page.

The command can be written in both upper and lower case characters; its syntax is as follows:

```
L
```

n : SEARING FOR THE NEXT OCCURRENCE OF A STRING

The command "n" searches in the current file or l-module, for the next occurrence of a data string.

If the string that is being searched for is not found, the following message is displayed in the the window at the bottom left of the screen:

```
NO PATTERN TO FIND
```

The command must be written in lower case characters; its syntax is as follows:

```
n
```

N : SEARCHING FOR THE LAST OCCURRENCE OF A STRING

The command "N" searches in the current file or l-module, for the last occurrence of a data string.

If the string being searched for is not found, the following message is displayed in the the window at the bottom left of the screen:

NO PATTERN TO FIND

The command must be written in upper case; the syntax is as follows:

N

Note The commands "n" and "N" are linked to the commands "/", "?", "%", "$" described below, as they search for the next occurrence of the string specified in any of these four commands.

Q : QUIT

The command "Q" is used to exit from the HEXED utility and return to the Shell environment. All changes made are recorded on the file or l-module.

The command can be specified both in upper and lower case characters; its syntax is as follows:

Q

---

r : REPLACING A BYTE

The command "r" replaces the byte indicated by the cursor, with a specified value.

The command must be written in lower case characters; its syntax is as follows:

---

r nn

---

where:

nn is the hexadecimal or ASCII value with which the byte indicated by the cursor is replaced. The value must be hexadecimal if the cursor is in the HEXADECIMAL CODE window, or ASCII if it is in the ASCII window.

R : REPLACING SEVERAL BYTES

The command "R" replaces several bytes, starting from the current position of the cursor, until /CR/ is pressed.

Whenever a byte is replaced, the cursor moves automatically right a byte. When it gets to the end of the current line it moves to the next one; when it gets to the end of the current page, the next page is displayed and the cursor moves straight to home.

the command must be written in upper case characters; its syntax is as follows:

```
R nn [ nn ... nn ] /CR/
```

---

where:

nn is defined as in the "r" command.

? : HELP

This command displays the list of all the commands available in the HEXED utility.

The syntax of the command is as follows:

```
?
```

: : SETTING-UP NON-IMMEDIATE COMMANDS

The command ":" moves the cursor from the current position to the window at the bottom left of the screen, so preparing the user to enter the "non-immediate" commands described below.

The syntax of the command is as follows:

NON-IMMEDIATE COMMANDS

These commands must be activated from the window at the bottom left of the screen, to which the user moves the cursor using the command ":" seen above.

H : DISPLAYING AN L-MODULE HEADER

The command "H" displays the header of the current l-module. It is only significant in PROGRAM mode.

The command can be specified either in upper or lower case characters; its syntax is as follows:

H

The SIT TABLE and the DATA_SECTION_HEADER are displayed an entry at a time as described below. If /CR/ is pressed the next entry is displayed, until the end of the list.

<SIT_ENTRY> ::= <SEGNUM> <SEGLENGTH> <SEGATTR> <SEGTYPE> <DATAPOINTER>

<DATA_POINTER> =
<SEC_COUNT> =
<SEG_NUM> =
<OFFSSET> =
<EOSBYTE> =

dove:

for SIT_TABLE the information specified is as follows:

- **SEGNUM** is the number of the segment described in the entry
- **SEGLENGTH** is the length of the segment, in pages of 256-bytes.
- **SEGATTR** are the attributes of the segment that has been created
- **SEGTYPE** is the type of the contents of the segment: data, code, stack, etc.
- **DATAPOINTER** is the offset in the load-file of the first Datasection that initialises the segment

for DATA_SECTION_HEADER the information specified is as follows:

- **DATA_POINTER** is the pointer to the next Datasection in this segment. If it is the last in the list, its value is NIL.

- **SEC_COUNT** is the dimension, in bytes, of the area containing the code and data of the segment

- **SEG_NUM** is the number of the segment

- **OFFSET** is the offset in the segment from which the area containing the code and data starts

- **EOSBYTE** is the end of the entry of the DATA_SECTION_HEADER.

For further details about the structure of an l-module, see "MOS - Programmer Guide" at the section "L-MODULE".

J : JUMP

This command jumps to a specified address.

The syntax and the way the command JUMP works is different in the FILE and PROGRAM modes.

FILE Mode

The syntax is as follows:

J address /CR/

---

where:

address is the address in the file to which to jump. It can be expressed in either decimals or hexadecimals, depending on the current base.

If the specified address is not valid, the message:

RETYPE ADDRESS

is displayed in the window at the bottom of the screen, and the cursor moves to the hexadecimal area of the screen. The utility is now ready to accept another command.

PROGRAM Mode

The syntax is as follows:

J [address] /CR/

---

where:

**** Is the segment number to which to jump.

address is the optional parameter which gives the offset in the segment to which to jump.

Both and address can be expressed in decimal and hexadecimal base, depending on the current base.

If only the segment number is given, the JUMP command displays the contents of the specified segment, starting from offset zero.

If the segment does not exist the following message appears:

RETYPE ADDRESS

in the window at the bottom right of the screen, and the cursor moves to the hexadecimal code area of the screen; the utility is ready to accept another command.

If the value specified for "address" is not withing the legal range, the following message is displayed in the window at the top right of the screen:

NO SUCH OFFSET SEE HEADERS

The cursor again moves to the hexadecimal area and the utility prepares to accept another command.

S : SET

Is a command made up of a set of options, each one of which performs a specific action.

The command cannot be used on its own, but always with one of its options. "S" can be specified both in upper and lower case characters; its syntax is as follows:

S { A C D F H P }

where:

A moves the cursor from the current position to the ASCII window of the screen. It can be specified both in upper and lower case characters.

C moves the cursor from the current position to the hexadecimal window of the screen. It can be specified both in upper and lower case characters.

D transform the address and limits of the current file/l-module being edited into decimal base.
After execution of this option, decimal values can be assigned to the parameters of the JUMP command.
The option can be written in upper and lower case characters.

F used to move from PROGRAM to FILE mode.
The screen assume the format described in the figure 4-15/A.
The option can only be written in upper case characters.

P used to move from FILE to PROGRAM mode, so allowing the user to display and edit the data-section of an l-module.
When is in this mode, the screen format changes to that described in the figure 4-15/B.
The option can be written in upper or lower case characters.
If a request is made to go into PROGRAM mode, on a file that is not an l-module, the utility displays the following message:

```
FILE "" MAYBE NOT A PROGRAM
```

and the cursor moves to the hexadecimal area.

H transforms the addresses and limits of the file/l-module that are being edited, into decimal base.
When this option has been executed, hexadecimal values can be assigned to the parameters of the JUMP command.
The option can be specified either in upper or lower case characters.

/ : SEARCHING AHEAD FOR AN ASCII STRING

Searches for the first occurrence of the specified ASCII string in the file or l-module, starting from the current cursor position, up to the end of the file or l-module.

The syntax is as follows:

```
/ ASCII-string
```

where:

ASCII-string is the ASCII string that is to be searched for.

If no occurrence is found of the specified string, the message "NOT FOUND!" is displayed in the bottom right window of the screen.

? : SEARCHING BACK FOR AN ASCII STRING

Searches in the file or l-module for the first occurrence of the specified ASCII string, starting from the current cursor position up to the beginning of the file or module.

The syntax is as follows:

```
? ASCII-string
```

where:

ASCII-string is the ASCII string that is to be searched for.

If no occurrence of the specified string is found, the message "NOT FOUND!" is displayed in the window at the bottom right of the screen.

% : SEARCHING AHEAD FOR A HEXADECIMAL STRING

Searches in the file or l-module for the first occurrent of the specified hexadecimal string, starting from the current cursor position, up to the end of the file or l-module.

The syntax is as follows:

```
% HEXCODE-string
```

where:

HEXCODE-string is the hexadecimal string that is to be searched for.

If no occurrence of the string is found, the message "NOT FOUND!" is displayed in the window at the bottom right of the screen.

$: SEARCHING BACK FOR A HEXADECIMAL STRING

Searches in a file or l-module for the first occurrence of the specified hexadecimal string, starting from the current cursor position, up to the start of the file or l-module.

The syntax is as follows:

```
$ HEXCODE-string
```

---

where:

HEXCODE-string is the hexadecimal string that is to be searched for.

If no occurrence of the string is found, the message "NOT FOUND!" is displayed at the bottom right of the screen.

Note

The commands "/, ?, %, $" can be used with the commands "n" and "N" that search for the next occurrence of the specified string.

The LISTMSG command displays the header messages of all the l-modules in the specified directory or volume. These messages are created at link time using the command MESSAGE (see "PASCAL+ Program Preparation and Execution"): they contain user-defined information.

The messages are displayed one at a time in an order corresponding to the order of the l-modules on disk. Each message is preceded by the name of the l-module it refers to. To display the next message in the sequence, you press the 'Y' key. The message display ends when all the l module messages for the specified directory or volume have been displayed, or when you press the 'N' key.

This command has the following syntax:

```
LISTMSG   { dirname  }
          { /IPL/SYS }
```

where :

dirname is the path name of the directory.

/IPL/SYS is the system volume.

Example:

```
MCL· LISTMSG /IPL/SYS

/SYS/$IPL

Cris-Tue May 10 13:09·03 CET 1984-FbRHiOtIgIwPlS


/SYS/$BST

Joe-Wed May 11 09:59:38 CET 1984-L


/SYS/$CHM

Fred- Thu May 12 11:12:34 CET 1984 Ch  Et  On, Po, Cn, Lm(s)  A(m)  B(g)
```

When LISTMSG is used on the system volume SYS (see above example) the messages listed are those in the operating system l-modules.

OS Messages

The operating system l-module messages have the following format:

"operator-date-configuration"

where:

operator is the name (abbreviated) of the programmer who linked the module, or some other identifying code chosen by the programmer.

date is the date when the l-module was linked.

configuration is a character string indicating the contents of the l-module. This information is mandatory only for the $CHM and $IPL modules, where it shows which of the peripheral driver programs are present. The character codes used in the string in this case are described in "CSS Generation and Configuration User Guide" and in "MOS System Software Function and Installation - User Guide" Otherwise the string can be used to record any relevant information.

INTRODUCTORY MENU

Introduces the user to the function of NOSE. The user may respond in two ways:

H/CR/ A brief description of the type of information output by NOSE is output to the screen, together with a list of the keys with which the user may control the operation of NOSE.

/CR/ The NOSE main menu is displayed.

NOSE MAIN MENU

Prompts you to select a feature of the operating system about which information is to be output.

L/CR/ The text

LEVEL 0 1 2

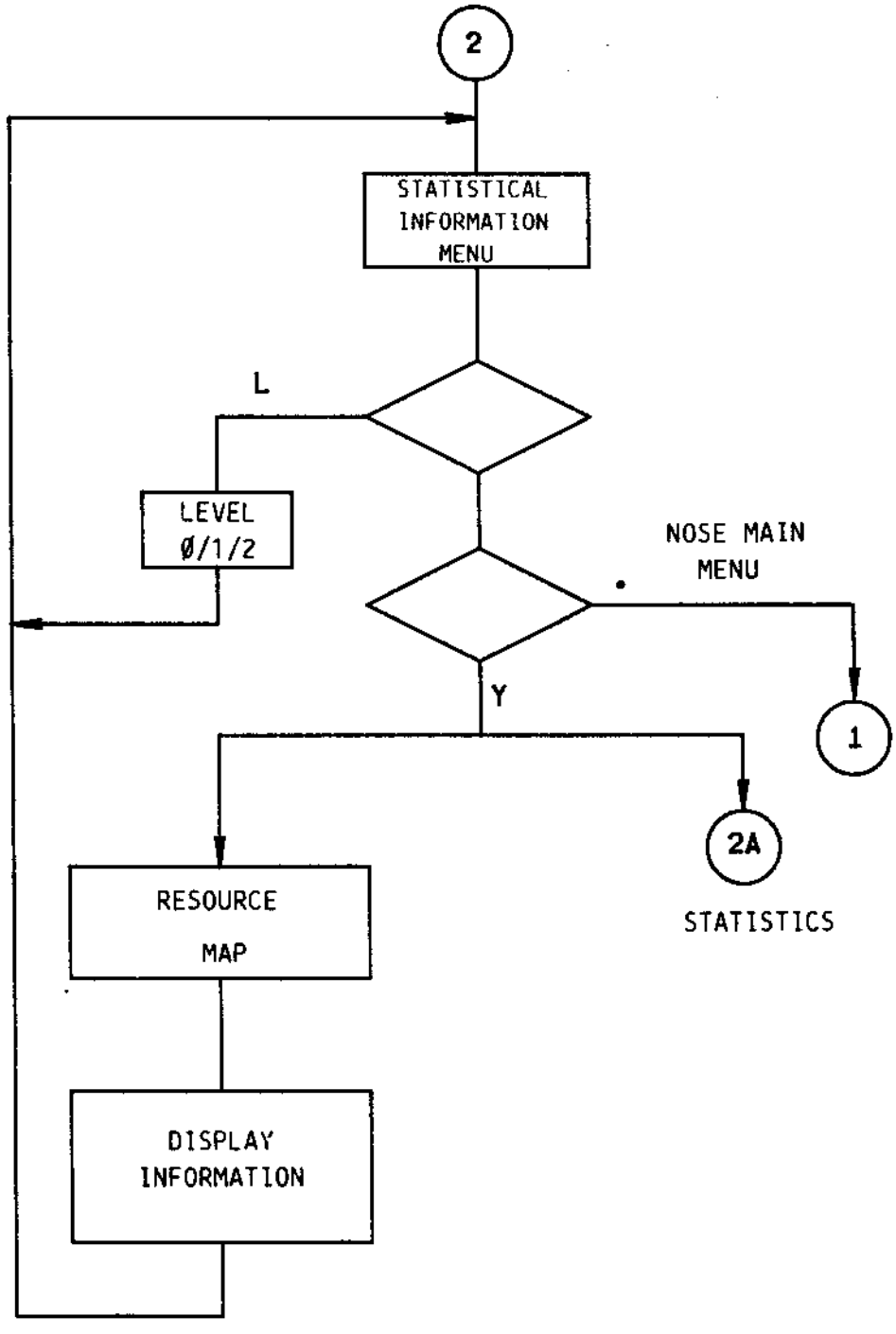
is output to the bottom left hand corner of the screen, with the cursor on 0, as the currently selected level. Pressing /CR/ moves the cursor to first 1, then 2, and pressing the /Y/ key selects the required level. The output level is set for the current execution of NOSE. However,it may be altered each time a menu is displayed.

./CR/ Exits from NOSE and returns control to Shell.

Y/CR/ Selects a NOSE option, which is one of the following:

- STATISTICS
- MEMORY
- FAMILIES
- PROCESSES
- PROGRAMS
- CHANNELS

Statistical Information Menu



RESOURCE MAP

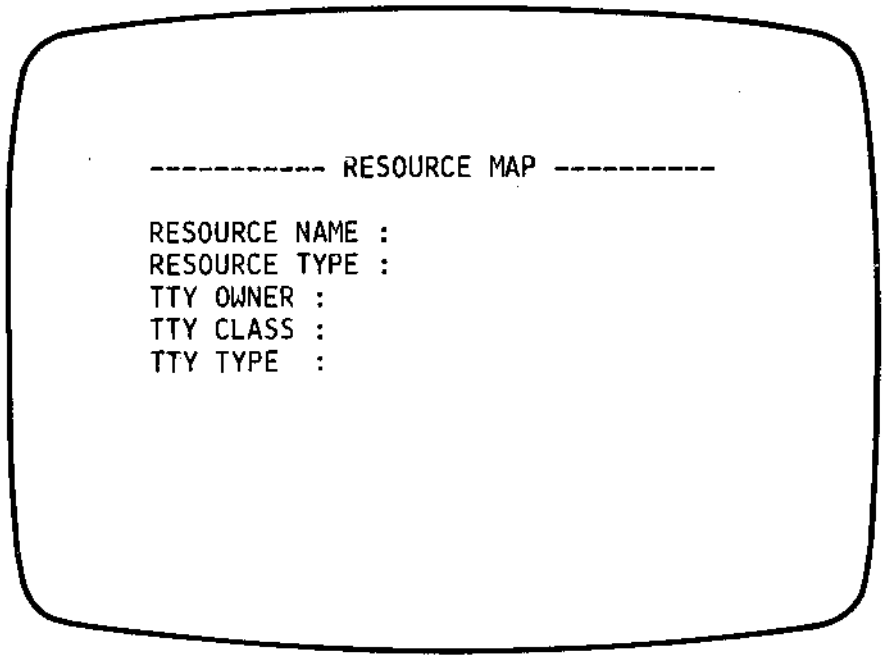
Gives information about system resources.
The information is more or less detailed depending on the level selected.

If LEVEL 0 is selected, a list is displayed of all the resources that, at IPL time, have been inserted in the resource map.
The name of each resource and its type is contained in this list.

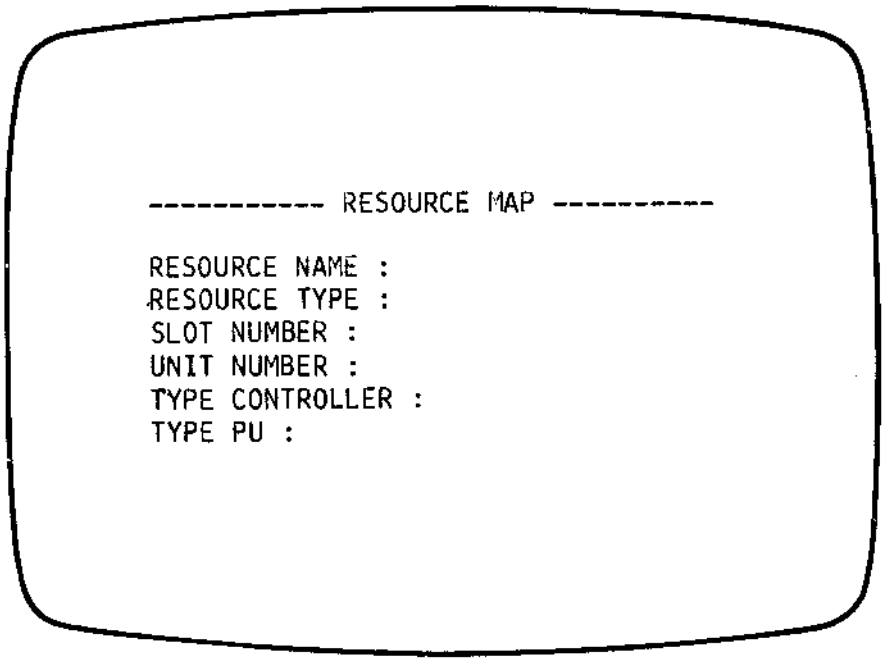
If level 1 or 2 is selected, one entry is displayed at a time, starting from the first one.
The next resource is displayed by pressing /CR/.

The type of information displayed by the utility for a Work Station Management or a magnetic unit are shown below.

Work Station Management



Magnetic Unit



dove:

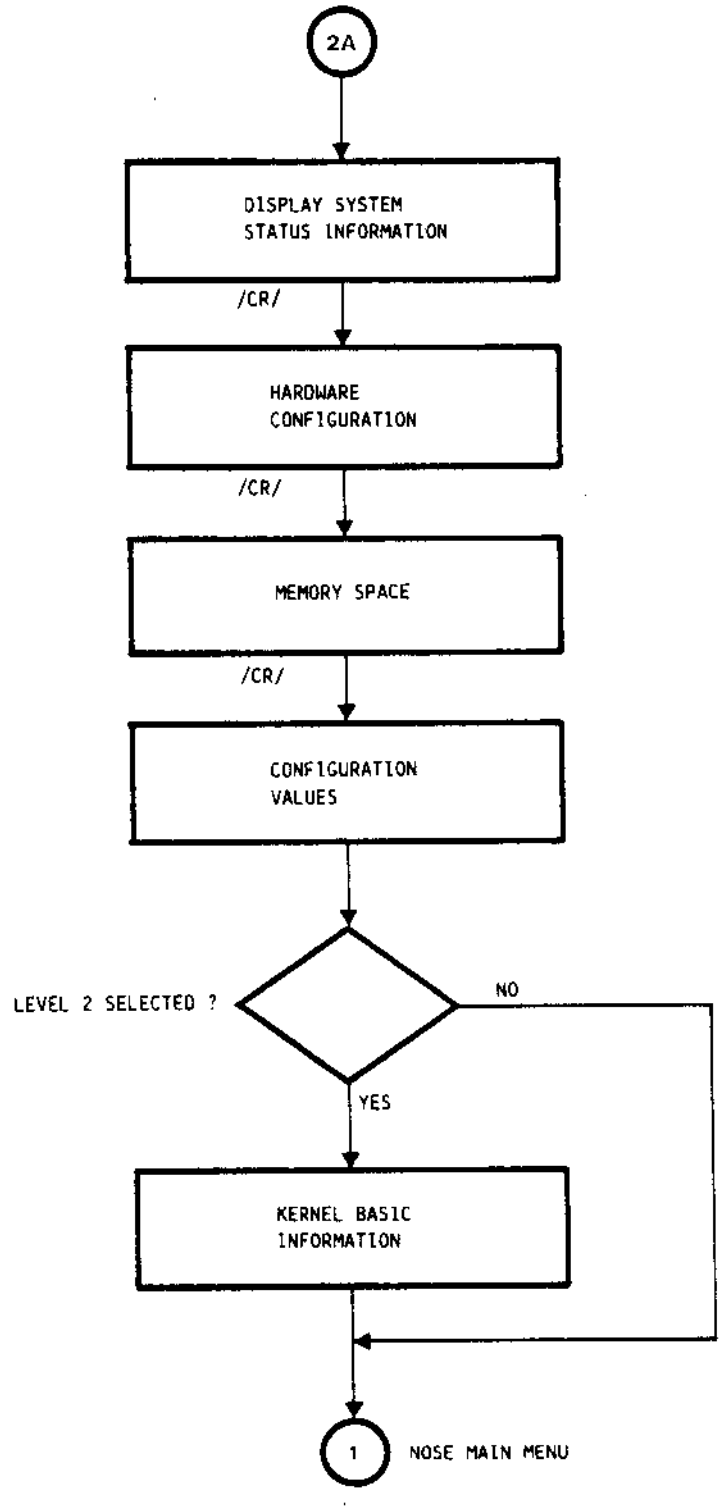
RESOURCE NAME is the name of the resource, and is described, for all resources, at the end of this chapter, at Table a).

RESOURCE TYPE is the type of the resource, and is described for all resources, at the end of this chapter, at Table a).

TTY OWNER, TTY CLASS, TTY TYPE are the physical characteristics of the resources in the Work Station Management, and are described at the end of this chapter at Table b).

SLOT NUMBER, UNIT NUMBER, TYPE CONTROLLER, TYPE PU are the physical characteristics of the magnetic resources, and are described at the end of this chapter, at Table c).

STATISTICS Menu



Message Description

CHANNEL NUMBER:

Prompts you to input the number of an active channel.

CHANNEL INFORMATION

FAMILY OWNER	Is the number of the family which created the channel.

Description of the Information given in the RESOURCE MAP

Table a)

RESOURCE NAME	RESOURCE TYPE	DESCRIZIONE
EXTERNAL_LINE		Communication Subsystem
		Floppy disk
FDRn (n= 1 ÷ 4)	FLOPPY DISK	8"
MFDn (n= 5 ÷ 8)	FLOPPY DISK	5"1/4, 320K bytes
MMDn (n= 1 ÷ 4)	FLOPPY DISK	5"1/4, 1 Mega byte
(where n is the driver number)		
		Hard Disk
ESDn (n= 1 ÷ 8)	HARD DISK	ESDI Interface
SASn "	"	SASI Interface
SMDn "	"	SMD Interface
HDCn "	"	ST 506 Interface
HDOn "	"	Integrated Hard Disk
(where n is the driver number)		
MTUn (n= 1 o 2)		Magnetic Tape Unit
(where n is the driver number)		
RSLtcsl	RS232	RS232/CL Line Controller
		Work Station Management
BDGtcsl	BADGE WS	Badge Unit
CADtcsl	CAD WS	Cash Adapter
MCRtcsl	MCR WS	Cheque Reader
PPDtcsl	PPD WS	PIN Pad
PRTtcsl	PRINTER	Printer
TTYx_tcsl	TTY	Screen/Keyboard
(where x is the TTY number)		

Note: **tcsl** assumes different meanings for the RS232 and for Work Station.

For **RS232**

- **t** it indicates the type of controller. It may be:
  - M = MUX and WSL1 (ELB 3683)
  - T = TWIN
  - R = CPU RS
- **c** indicates the channel identifier.
  a) In the case t = M, this value may be:
     A = RS232 A
     B = RS232 B
     T = MUX channel (transparent mode).
  b) In the case t = T, this value may be:
     A = RS232 A
     B = RS232 B
  c) In the case t = R, A is mandatory.
- **s** indicates the slot identity.
  - A: specifies the first controller within those of the same type.
  - B: specifies the second, and so on.
  - In the case t = R the value must be A.
- **l** indicates the line identity. The values are:
     0 if t = T o R
     a value in the range from 0 to 3 if t = M.

For
Work Station

- **t** indicates the type of controller. It may be:
  - M = MUX or WSL1 (ELB 3683)
  - E = ELB
  - T = twin

. R = RS CPU

- **c** indicates the channel identity.
  In the case of MUX or ELB, it can have one of the following values:

  . A = RS232 A

  . B = RS232 B

  . C = Badge Reader

  . D = PIN pad

  . E = TTY user (1st window)

  . F = TTY system (2nd window)

  . From G to L are reserved

  . T = MUX channel transparent mode

  In the case of RS CPU, this field can only be set to A.

- **s** indicates the sequential position of the controller within those of the same type.

  . A: means the first controller of a given type.

  . B: means the second controller of a given type, and so on.

  . In the case of RS CPU and line controllers, the value for "s" must be A.

- **l** indicates the line number of the controller. For all controllers except for MUX, this can only be 0. For MUX it can be in the range from 0 to 3.

Table b)

Describes the physical characteristics of the Work Station Management.

	TTY OWNER	TTY CLASS	TTY TYPE
TTY	This is made up of four bytes: . the first specifies the logical number of the work station to which the device belongs. . the remaining bytes are set to zero or to the logical number of the owner WS, if the device is shared.	0= non Olivetti 1= Olivetti	0= RS232 - no new line 1= RS232 - new line 2= current loop - no new line 4= current loop - new line
PRT		0= Olivetti Controll.mode 1= Free Running	0= RS232 1= current loop
BDG		0= Reader/ Writer	0
PPD		0	0
CAD		0= Normal 1= Teller	0
MCR		0= Reader 1= Reader/ Writer	0

Note: for RS232, the fields TTY OWNER, TTY CLASS and TTY TYPE are non significant.

Table c)

Describes the physical characteristics of Hard Disks, Floppy Disks, Magnetic Tape Units and Streaming Cartridge Tapes.

SLOT NUMBER	is an integer value specifying the number of the slot of the peripheral
UNIT NUMBER	is an integer value that must be interpreted as an array of 16 booleans, specifying the number of units in the system. The following rules must be followed in order to them: UNIT NUM[x]= 0 unit x is not available UNIT NUM[x]= 1 unit x is available
TYPE CONTROLLER	is an integer value specifying the code of the controller
TYPE PU	is an integer value specifying the type of controller

- Whether the offset at which the current page is located is associated, on the father page, with a key whose value is equal to the maximum key value of the current page.

For the last level of the tree, TREECHECK checks whether the record positions which are encountered represent valid positions within the data file, and whether all valid record positions are referenced.

TREECHECK's examination of the tree begins at the last level of the tree: the byte position of the first page of the last level of the tree is always zero. The examination of the tree stops after the root page has been examined, or after an error has been encountered, which makes it impossible to continue the examination of the tree.

Further checks are performed before the internal structure of the tree is checked. These checks are as follows:

- Whether the size of the data file is a multiple of the record length.

If the examination of the tree is successful, additional checks are made as follows.

It is also possible to test other values against the values contained in the information file for the index. Each indexed file has such a file associated with it. The structure of this file is given in "MOS - System Areas Reference Manual". The checks which may be made on the information file are:

- Whether the root page byte position obtained when examining the file is equal to the position in the information file.

- Whether the number of tree levels examined is equal to the value contained in the information file.

- Check the correspondency between the indices described in the info file and those actually present.
  This check is connected to the error "FILE INFO CORRUPTED".

The certified limit (max number of records) that can be handled by the utility is indicated in "Appendix C".

TREECHECK OPERATION

The TREECHECK command has the following syntax:

```
TREECHECK file-name [L] [M]
```

---

where:

file-name is the path name of the indexed file which TREECHECK is required to check.

L is an option which specifies more detailed output if there are errors in the index (see below).

M is an option which also gives more detailed output, but whenever the screen is full the following prompt appears:

```
MORE
```

and the user can read the next page of messages by pressing the space bar, or exit from the procedure by pressing q or Q.
Depending on the resulting data, TREECHECK outputs diagnostic information. The header shown below is output first:

```
TREECHECK PROGRAM RUNNING

ACTUAL CONDITION OF file-name
```

Then, if the file specified is a packed positional file, i.e. the record position is the key, only the text

```
PACKED POSITIONAL
```

is output. For keyed files, the output is as shown below.

```
FIELD       INDEX      KEY         KEY          PAGE        SPLIT
NR          TYPE       START       LENGTH       SIZE        STRATEGY

<field 1>   <type 1>   <start 1>   <length 1>   <size 1>    <strategy 1>
    .           .          .           .            .           .
    .           .          .           .            .           .
    .           .          .           .            .           .
<field n>   <type n>   <start n>   <length n>   <size n>    <strategy n>


FIELD NUMBER <number 1>

    INDEX OK
        .
        .
        .
FIELD NUMBER <number n>

    INDEX OK
```

where:

field is a number associated with the index

type is the type of index, and may be one of the following:

- PM, to denote a primary index
- SU, to denote a secondary index without duplicated keys
- SD, to denote a secondary index with duplicated keys.

start shows, in bytes, the offset of the key within the record.

length is the length of the key, in bytes.

size is the size of the page, in bytes.

strategy indicates how much information is to be transferred to a new page if a page becomes full, and a new page has to be created. It may take the following values:

- 50, to denote that half the information is to be transferred
- 90, to denote that ten percent of the information is to be transferred
- 100, to denote that no information is to be transferred, but a new page is to be created with only one key in it.

ERROR MESSAGES

If the execution is not successful, the information given to the user will differ according to the type of output selected. If short output was requested, instead of the text

INDEX OK

the text

INDEX NOT OK

is output where appropriate.

If, on the other hand, extended output was requested, instead of the text

INDEX NOT OK

one or more error messages are output, defining in what way the indices are incorrect. These error messages are descripted in "Appendix B".

It is advisable therefore to reactivate the utility using the "L" option in reply to the message "INDEX NOT OK" so that the user can decide which action it would better take (execute TREEMEND or DELINDEX and MKINDEX or to make a back-up copy), depending on the error messages returned.

It must be borne in mind that particular sictuations can occur, in which recovery of one index can influence recovery of another one. In this case the user must choose whether to perform a DELINDEX and MKINDEX on the index that may influence the consistency of the other indices, or perform TREEMEND on that index, and then DELINDEX and MKINDEX on the others. This error islinked to display of the error messages "NO REFERENCE TO POSITION <position>" and "POSITION OUT OF BOUNDS: <position> - OFFSET: <offset> - LEVEL:1"; see "Appendix B" in this manual.

TREEMEND is a Shell utility for keyed files, that may be used to rebuild a B*-tree index damaged in a system crash. TREEMEND can operate only if the last level of the B*-tree is able to be recovered. Therefore, TREEMEND checks the last level, and if there are errors in the last level of the index, such as keys in the wrong order, page links broken, or record positions unreferenced, TREEMEND outputs an error message and then terminates.

The utility TREECHECK (described in the previous section of this chapter) may be used first to determine whether or not an index is damaged, and if so, how serious the damage is.

Throughout this description, a knowledge of the structure and function of B*-tree indices is assumed. Information on this subject may be found in the manual "MOS - Structure and Functioning".

TREEMEND has the following syntax:

```
TREEMEND keyed-file-name [L] [M]
```

where:

keyed-file-name is the path name of the indexed file which TREECHECK is required to check.

L is an option that gives more detailed output if there are errors in the index.

M is an option that also gives more detailed output, but whenever the screen is full the following prompt appears:

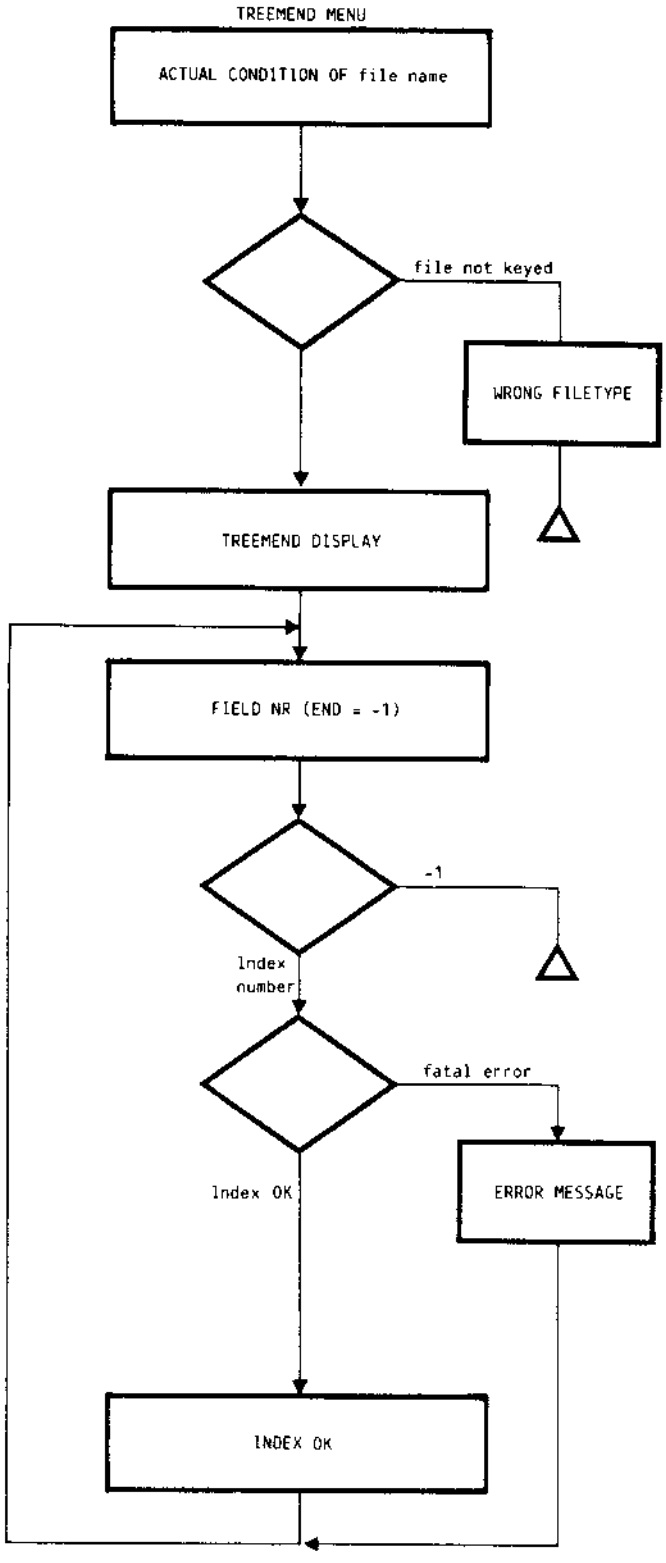
MORE

and the user can read the next page of messages by pressing the space bar or can exit from the procedure by pressing the key q or Q.

The operation of TREEMEND is shown overleaf.

The error messages displayed when execution of the utility does not finish properly an given in Appendix B.

The user should note that the utility does not finish properly either when a HARDWARE ERROR occurs.
This can to be numeried only by recovering the back-up copy of the file.


TREEMEND MENU
ACTUAL CONDITION OF file name
file not keyed
WRONG FILETYPE
TREEMEND DISPLAY
FIELD NR (END = -1)
-1
Index number
fatal error
ERROR MESSAGE
Index OK
INDEX OK

Message Description

TREEMEND DISPLAY

FIELD NR	INDEX TYPE	KEY START	KEY LENGTH	PAGE SIZE	SPLIT STRATEGY
<field 1>	<type 1>	<start 1>	<length 1>	<size 1>	<strategy 1>
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
<field n>	<type n>	<start n>	<length n>	<size n>	<strategy n>

FIELD NR — Is a number associated with the index.

TYPE — Is the type of index, and may be one of the following:

PM, to denote a primary index

SU, to denote a secondary index without duplicated keys

SD, to denote a secondary index with duplicated keys.

START — Shows, in bytes, the offset of the key within the record.

LENGTH — Is the length of the key, in bytes.

SIZE — Is the size of the page, in bytes.

STRATEGY — Indicates how much information is to be transferred to a new page if a page becomes full, and a new page has to be created. It may take the following values:

50, to denote that half the information is to be transferred

90, to denote that ten percent of the information is to be transferred

100, to denote that no information is to be transferred, but a new page is to be created.

FIELD NR (END = -1) — Prompts the user to input the number of an index to be checked and mended. If -1 is entered, TREEMEND terminates and control is returned to the Shell environment.

VOLGC

The Shell command VOLGC checks any volume specified by the user, and repairs damaged data structures if any are found. VOLGC performs the same initial checks as the Shell command DISKCHECK, but VOLGC corrects any inconsistencies which may be present. The user is informed of any damaged data structures encountered.

VOLGC checks the following:

- PDDs (permanent dataset descriptors), which describe file structure.
- Extents, which describe the allocation of file space on the volume.
- Blocks, which are the basic allocation units of file space on disk, and are generally 512 bytes long. (A block is set if it is marked busy, and allocated if it is referenced by a PDD).
- Directories, which are organised as an array of elements consisting of a name and an index.

The certified limit (max number of extents) that can be handled by the utility is indicated in "Appendix C".

VOLGC has the following syntax:

VOLGC volume path name [list-option] [F]

---

where:

list-option

where:

list-option is either L or M. If L is typed, all the information produced by VOLGC is output immediately; this is most useful if the output has been redirected to a printer. If M is typed, only the first 20 lines are output, and the user is then prompted as to whether or not the output is to continue after this and each subsequent 20 lines. If neither L nor M is typed, a summary of the VOLGC information is output.

F is an option that indicates that, if there is no space free on the disk, VOLGC removes all the files whose PDDs have been lost or are corrupted.

The utility sets the option "F automatically anyway if during its execution it does not find enough space on the disk. Note that to guarantee data integrity it is not advisable to kill the utility during its execution.

The checks carried out, and the action taken by VOLGC are as follows:

Section 1 FREE PDD LIST CHECKING

The list of free PDDs is a set of pointers from one free PDD to the next, held in the nextAvPdd field of the PDD record itself. This list is checked to see that each PDD marked free is indeed free, and that starting from PDD number 0, each PDD points to the next PDD until the last PDD is reached.

A header is printed, and, if there are no errors, the following message is output:

CORRECTLY ENDED AT RECORD NO. <number>

where:

number is the number of the last Pdd record in the list of free Pdds.

Section 2 FREE EXTENT LIST CHECKING

The list of free extent table entries is checked in the same way as the list of free PDDs above.

A header is printed and, if there are no errors, the following text is output:

CORRECTLY ENDED AT RECORD NO. <number>

where:

number is the number of the last free extent record in the free extent list.

Section 3 PDD EXTENT CHECKING

The links between extent table entries for extents belonging to the same file are checked. A list of the extents corresponding to the PDDs in use is printed, showing the length in blocks and initial block numbers of each extent.

The PDDs are checked for internal consistency. All inconsistencies are resolved, thus:

- If the attributes of a PDD are not those described by its identifier, then the attributes are amended to fit the description.

- If the length of a PDD is given as less than zero, or greater than the sum of the count of its extent table, then:

- If the file has been created but not written to, the length is assumed to be zero.

- If the file has had data written to it, the length is assumed to be equal to the sum of the count of its extent table.

If some blocks overlap one another, the error message

BLOCKS MULTIPLY ALLOCATED

and VOLGC frees that the corrupted PDD refers and inserts the PDD in the list of PDDs.

If there is a PDD which has an undefined file type, the message:

```
UNDEFINED CLASS : THIS INFORMATION IS UP TO YOU,
                  IF YOU LIKE USE DISKEDIT
NO SPACE ASSOCIATED.
```

is output. The user may then use the utility DISKEDIT, described in this chapter, to display the PDD record, and subsequently to edit the "field fileTp". If there is an inconsistency between the PDD and its associated descriptors, and if the PDD is not in the free list, it is freed.

Section 4 BLOCK ALLOCATION CHECKING

The bit map is scanned and a check is made that all blocks marked as in use (corresponding bit set) are covered by an extent table entry; also, conversely that no blocks marked as not in use (bit clear) are referenced in an extent table entry.

If a block is found allocated but not set, the bit map is set busy relative to this block.

If a block is found set but not allocated, the bit map is set free relative to this block.

A header is printed but nothing further is output unless there are errors.

Section 5 FILE STRUCTURE CHECKING

Starting from the root directory of the volume, the tree structure of files in the volume is scanned to check for consistency between the directories and the PDD table. If there are any inconsistencies in the directory entries, they are removed. A list of all objects in the volume is printed with their corresponding PDD numbers.

Section 6 MISSING PDD CHECKING

A missing PDD is one which is not referenced in a directory or in a free list. VOLGC saves any files referenced by a missing PDD in a LOST_FILES directory, and gives them a name LOSTF_pdd number.

If such a PDD references a directory, then VOLGC recovers this directory, and any files or directories under it, and places them in a LOST_FILES directory, with the names LOSTD_pdd number, or LOSTF_pdd number, as appropriate. If the missing PDD refers to a directory in which o or oo is missing, the utility VOLGC inserts it and truncates the directory.
If the LOST_FILES directory already exists, then a warning message is output.

If there is no space in the volume to create the LOST_FILE and the option "F has been specified, VOLGC removes all files referenced by corrupted or missing PDDs. If the option "F has not been specified, a message is displayed that advises use of the utility with the option "F (if the files are not to be recovered), or reactivation of the utility after the space required to create the LOST-FILE directory is freed.

A header is printed, but nothing further is output unless there are errors.

Section 7 MISSING EXTENTS CHECKING

If an extent is not referenced by a PDD, or it is not in a free list, then, it is given an entry in the extents free list. The extents are similarly checked for internal consistency.

A header is printed but nothing further is output unless there are errors.

Summary The summarised information consists of totals of items in use (ACTIVE), or available (FREE), or inconsistent with some other item (MISSING), for the following groups of items:

- Objects in the volume (active datasets).
- PDDs in the PDD table (PDD records). (The PDDs are the first part of the table).
- Extent table entries (records) in the part of the PDD table reserved for extra extent tables. (The extra extent tables are the second part of the PDD table).
- Blocks in the volume's contiguous storage area on disk.

ERROR MESSAGES

The error messages displayed when execution of the utility does not finish properly are given in "Appendix B".

```
                ************************************************
                *                                              *
                *  OLIVETTI MOS  -  VOLGC  UTILITY   V4.0      *
                *                                              *
                ************************************************

VOLUME TO DUMP  IPL/SYS1

SECTION 1.  FREE PDD LIST CHECKING
        ***  ERROR IN FREE PDD LIST:STOPPED AT RECORD NO. 46

SECTION 2.  FREE EXTENT LIST CHECKING
            CORRECTLY ENDED AT RECORD NO. 107

SECTION 3.  PDD EXTENT CHECKING
===========================================
PDD N. 0
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 1 FOR 17 BLOCKS
===========================================
PDD N. 1
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 18 FOR 1 BLOCKS
===========================================
PDD N. 2
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 19 FOR 1 BLOCKS
===========================================
PDD N. 3
PDD_EXTENT_TBL
    NO SPACE ASSOCIATED.
===========================================
PDD N. 4
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 20 FOR 309 BLOCKS
    EXT_2 :FROM BLOCK 1312 FOR 116 BLOCKS
===========================================
PDD N. 5
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 510 FOR 10 BLOCKS
===========================================
PDD N. 6
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 520 FOR 33 BLOCKS
===========================================
PDD N. 7
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 553 FOR 29 BLOCKS
===========================================
PDD N. 8
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 582 FOR 28 BLOCKS
===========================================
PDD N. 9
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 610 FOR 45 BLOCKS
===========================================
PDD N. 10
PDD_EXTENT_TBL
    EXT_1 :FROM BLOCK 655 FOR 4 BLOCKS
===========================================
.
.
.
.
.
```

A. DESCRIPTION OF THE SYSTEM ERRORS

INTRODUCTION

This appendix contains a detailed description of the errors for the system primitives. These are T reply type (for the Pascal+ type see MOS, PMM and Driver Primitives, Reference Manual and CSS-COMMUNICATION SUBSYSTEM, Line Manager Interface, Programme Programmer Guide) for the PMM, file system, WSM driver and line interface manager primitives; these are the negative integer values for the port/Uport interface described in the manual MOS, Internal Primitives For Distributed System Environment, Reference Manual.

T reply errors are described in progressive numerical order according to the corresponding code. Next to each code there is a string that specifies the class of the error, corresponding to the system component that displays it (FS, PMM, WS, etc.), together with the string that identifies the error in the class, in the form "class : identifying string". The reply-codes of the port/Uport primitives are described in decreasing order, together with a string that identifies the type of error.

At the end of this appendix there is a table giving the errors by class, and in each class they are listed in alphabetic order by identification string. Another table lists the reply-codes for port/Uport in alphabetical order for descriptive string.

These errors can be generated whilst the system is working in any environment and a message is displayed, and possibly an error number, for each environment. A brief description is given of the cause of the error and, where possible, of the action that the user can take to correct it.

Notes on the T_reply Error Codes

The code associated to each error can be divided into two parts as follows:

xx yy

where:

xx is a decimal number that identifies the class of error, e.g.: 01 shows a system error, 02 shows a debugging error, etc.

yy is the decimal number that identifies the error in its class, according to the position of the error in the Pascal+ T_reply list.

T-REPLY SYSTEM ERROR CODES

01 00	SYSTEM : TERMINAL DOWN The required operation cannot be executed because the terminal is switched off.
01 01	SYSTEM : DIAGNOSTIC ERROR This is displayed by the driver OPENCR, OPENBG, OPENPR, OPENCA. A diagnostic error has been detected during initialisation of a peripheral driver or a controller board of the cheque reader/writer, the badge reader/writer, printer or cash adapter.
01 02	SYSTEM : RECEIVE ERROR Emitted by the READ primitive of the driver in the following circumstances: a reception error has occurred (parity, overload, etc.) on the line connecting the peripheral; incorrect characters have been received; a line error has occurred.
01 03	SYSTEM : HARDWARE ERROR The physical characteristics of the transmission line (such as parity, speed, number of bits, etc.) are incompatible with those of the driver. The user should contact Olivetti for assistance. This error is also generated for a particular hardware error on the cash adapter (see MOS, PMM and Driver Primitives, Reference Manual) or when an attempt has been made to access a magnetic unit which has been damaged;in this case another unit must be used.
01 04	SYSTEM : SYSTEM ERROR Emitted by file system primitives if there is inconsistency in the system data area.
01 05	SYSTEM : TABLE OVERFLOW Overflow in the system table: there is not enough space to create any more processes, channels, programs, families or files, or other indices for keyed files.

01 06	SYSTEM : OUT OF DISK SPACE Emitted by file system primitives when disk space requested, is not available. Obsolete files or directories can be deleted using the Shell command REMOVE.
01 07	SYSTEM : FATAL ERROR Emitted by PMM primitives when there are inconsistencies in the internal table of the system. It is usually emitted before a system crash. If it is emitted by SETTIME, an error in setting the time has been detected.
01 08	SYSTEM : READ-ONLY FAULT An attempt has been made to violate the read-only protection of a segment, by writing to it.
01 09	SYSTEM : KERNEL SEGMENT FAULT An attempt has been made to access a segment reserved for operating system. The user can only access the segments in the user area (from 32 to 63).
01 10	SYSTEM : SEGMENT LENGTH FAULT Attempts have been made to access over the actual length of the segment.
01 11	SYSTEM : INHIBIT SEGMENT FAULT An attempt have been made to access an invalid segment (inaccessible).
01 12	SYSTEM : EXECUTE FAULT Attempts at reading/writing a segment have been made, that contains the only executable code.
01 13	SYSTEM : INVALID INSTRUCTION An attempt has been made to execute an inexistent machine instruction.

01 14	SYSTEM : PRIVILEGED INSTRUCTION An attempt has been made to execute a privileged instruction in user mode.
01 15	SYSTEM : FATAL SEGMENT FAULT Two or more segment violations consisting of the following errors: 01 05, 01 07, 01 08, 01 09, 01 10, 01 11, 01 12, 01 13, 01 14.

Note: Usually the errors from 01 08 to 01 15 are not returned by the MOS external interface primitives, but are passed as parameters when the fault handling routine is executed (defined via SETHANDLER).

01 16	SYSTEM : NO PROCEDURE Shows that the process has called a procedure that is not available on the system.
01 17	SYSTEM : NO ROUTING Emitted by the file system primitive, in cluster configuration. In the routing table there is no route for accessing the file. This can be because the machine associed with the file is down or a line is down, or a system may not be connected, or an identifier has been corrupted.

01 18 SYSTEM : RESULTS DOUBTFUL

Emitted by the primitive PM.EXEC, by file system primitives in a cluster configuration and by line manager interface primitives.
Returned by the PM.EXEC, shows that a message has not been recognised. This code confirms that the part of the communication that is sent from the port to the line driver is working properly. An error can be caused by overloading the machine that exchanges messages or errors in the application program.
For file system primitives, the mechanism for transmitting messages has replied when the time-out has expired to a the operation has been executed or not. The user must try to send the message again.
Returned by a line manager interface request, shows the negative result of the remote procedure call associated to the request. A possible cause may be an insufficient or incorrect group of resources defined during configuration (see CSS - COMMUNICATION SUBSYSTEM, Line Manager Interface, Programmer Guide), that required CSS reconfiguration.

01 19	SYSTEM : OPERATION ABORTED Returned by the primitive PM.EXEC, when data exchange fails because of anomalies in communication between the port and the line driver. Emitted by the file system driver, in a cluster configuration, if there is an anomaly during remote accessing. The 'file server' may not exist, or the message is not coded properly. This replay may also be returned by the line manager interface READUNS, if the system crashes or an abnormal condition has caused the program to abort.
02 00	DEBUG : RECORD TOO LONG Emitted by the driver primitives READPR, READBG, READCA, READCR, if the user data buffer is too small, and by the line manager interface primitive WRITECU if the length of the data to send exceeds the space available in the buffer (LM buffer). For the latter, the possible actions are: change the program for sending shorter messages, reconfigure the LM buffers to make them longer.

02 01 DEBUG : INVALID OPERATION

This error can be emitted for various reasons: in relation to the primitives called.
By CALL when the program containing called procedure has not been loaded in the address space of the calling process, or when there are too many nested CALL calls.
By DESTROY when a family wants to destroy itself.
By INTERRUPT when an incorrect interrupt code has been given.
By JOIN when the specified process is not resident with the called process, or when the identifying parameter of the process which is to be received, identifies the current process.
By LOAD when the specified program has been loaded already.
By RESUME when a process wants to resume execution of the joint processes.
By SETSEGMENT and STATSEGMENT when a process has executed a HALT or or has not executed a START yet.
By SETTIME when the wrong date and time has been set.
By SLEEP when an invalid lapse of time has been set.
By SUSPEND when a family has tried to suspend itself.
By SWITCHPTABLE because the default procedure table cannot be modified.
By UNLOAD when the address space is not that of the calling process, and the family does not belong to the set of families that it controls, or when the process to be unloaded is still being executed.
By driver primitives when a call is inconsistent with the flow of operations.
By file system primitives when the operation is illegal because of access rights and file protection.
By line manager interface primitives, when an attempt has been made to emit a request which is not logically valid, or is incompatible with the specifications given when connecting to the remote application, or which is not allowed for the type of application.

02 02	DEBUG : INVALID IDENTIFIER An identifier given as a parameter is incorrect.
02 03	DEBUG : INVALID PARAMETERS An incorrect parameter has been given, or it points at an area which is not accessible, or in which it not possible to write.
02 04	DEBUG : ERROR IN THE STRING Emitted by the driver primitives CLOSEPR, WRITEPR, READCA, WRITECA and WRITEPP when a data string is incorrect.
02 05	DEBUG : SECURITY VIOLATION Emitted by the file system primitives when an operation is incompatible with the file protection.
02 06	DEBUG : NOT YET IMPLEMENTED Emitted by the file system primitives SETBOUNDSKEYED and SETBOUNDSPOS if an operation is being executed that is incompatible with the current implementation.
02 07	DEBUG : BAD SCOPE Emitted by FORK when the variable that represents the identifier of the process to create is not local for the calling process, and is therefore not allocated in its stack.
02 08	DEBUG : BAD STATE Emitted by some PMM primitives for various reasons. By LOAD when the specified family is not suspended. By RESET when the program to be reinisialised is not suspended. By SUSPEND, RESUME and SWITCHMTABLE when the process is in "halt" state (this is the state before a process is killed). By START when the process is already executing.

02 09	DEBUG : ILLEGAL INDEX Emitted by the PMM primitives CALL, START and WAITRESULT, when the parameters 'count' or 'startIndex' are incorrect: array index out of range, or index not in the segment table.
02 10	DEBUG : CHANNEL DESTROYED Returned by SUSPEND and WAITRESULT when the channel associated to the suspended family has been destroyed.
02 11	DEBUG : CHANNEL BUSY Returned by WAITRESULT when the communications channel between the families is being used by another process that is executing a WAITRESULT.
02 14	DEBUG : NO WRITERS Emitted by the primitive WAITRESULT when there are no process active that can write using the specified channel. This reply-code avoids the process waiting for ever for the WAITRESULT function, if, for example, the channel has not been opened.
03 00	PMM WARNING : MEMORY CONFLICT Emitted by the primitive LOAD when the address space where the code or the data should have been loaded is not available, as another process is occupying it. The user must either free the occupied segments using UNLOAD, or execute a new link operation on the program, specifying different segments.
03 01	PMM WARNING : NON-EXISTENT FILE Returned by the primitive LOAD when the file 'MAIN' has not been found when loading a Program Directory, or when the file identifier is incorrect. Returned by the primitive RESET when the file, from which the data must be read, has not been connected.

03 02	PMM WARNING : TYPE CONFLICT Emitted by the primitive LOAD when there is a conflict between types and the attributes of the segments specified in the link phase.
03 03	PMM WARNING : FORMAT ERROR Returned by the primitive LOAD if the file to be loaded is not in the correct l-module format.
03 04	PMM WARNING : ILLEGAL SEGMENT LENGTH Emitted by the primitives SETSEGMENT and LOAD if the length of the segments to modify or contained in the file to be loaded is illegal.
03 05	PMM WARNING : INVALID SEGMENT NUMBER Returned by the LOAD primitive when the file contains an invalid segment number (from 0 to 31 inclusive or greater than 61). Returned by SETSEGMENT and STATSEGMENT when a wrong segment number has been specified.
03 06	PMM WARNING : SOFTWARE INTERRUPT Emitted by some PMM primitives, which are aborted because of an external event
03 07	PMM WARNING : NO HALT A process is ended without performing a HALT.
03 08	PMM WARNING : PROCESS DESTROYED Emitted by the WAITRESULT primitive when the family being waited for has been destroyed.
03 09	PMM WARNING : PROCESS SUSPENDED Emitted by the WAITRESULT primitive when all the processes belonging to the family being waited for have been suspended.

03 10	PMM WARNING : BREAKPOINT Emitted by the WAITRESULT primitive when a process belonging to the family being waited for, has been interrupted.
03 11	PMM WARNING : NOT EMPTY An address space is not empty.
04 00	PMM ERROR : MEMORY OVERFLOW Emitted by the FORK, LOAD, SETSEGMENT, SPAWN, and START primitives, when a memory area is unsufficient for private segments, for loading a program, for increasing the dimensions of the segments, for creating a new family, for activating a program, respectively.
04 01	PMM ERROR : CONCURRENT ERROR Emitted by the FORK primitive when an attempt has been made to exceed the max. number of concurrent processes allowed (this value has been specified as a configuration parameter).
04 02	PMM ERROR : TOO LONG Emitted by the CALL primitive when the array of characters returned to the calling program exceeds the predefined limits and by the WAITRESULT primitive when the array containing the reply exceeds the dimensions of the array that must contain it.
05 00	FS WARNING : BYTE-STREAM OPERATION Emitted by the file system primitives OPENPOS and OPENKEYED. With OPENPOS, either no bit map has been found, or the information file is no accessible or has been altered, or the file is a byte-stream file. With OPENKEYED, either indices have not been found, or there is no information file, or the file is a byte-stream file. Only byte-stream operations can be carried out. Operations take place as if they had been opened by OPENBYTE. All READPOS, WRITEPOS, READKEYED, etc., will be ignored, but the READBYTE and WRITEBYTE will be accepted.

05 01	FS WARNING : POSITIONAL OPERATION Emitted by the file system primitive OPENKEYED. The index of a keyed file has been found and the file with information about the length of the record has been found, but has been damaged, or the file is positional.
05 02	FS WARNING : INCONSISTENT DATA BASE Returned by the file system primitives WRITEKEYED and DELETEKEYED. The search made for a key has not been successful, because the index is not consistent with the contents of the file.
05 03	FS WARNING : NO PRIMARY INDEX Returned by the file system primitive OPENKEYED when the primary index has been annulled. All future write and delete operations will be ignored, but file can be read.
05 04	FS WARNING : DUPLICATED KEY Returned by the file system primitive WRITEKEYED, and informs the user that a duplicated secondary key now exists in an index that allows duplicated keys.
05 05	FS WARNING : INCOMPLETE RECORD Returned by the file system primitive READSEQ if an incomplete record has been read.
05 06	FS WARNING : VOLUME NOT COPIED Returned by the file system primitive COPY when a directory containing a volume is copied.
06 00	FS ERROR : TIMEOUT Emitted by the file system primitives LOCKRECORDKEYED and LOCKRECORDPOS and by the OPEN primitives after a time-out has expired.

06 01	FS ERROR : RECORD LOCKED Emitted by the file system primitives DELETEKEYED, DELETEPOS, WRITEKEYED and WRITEPOS. A record cannot be accessed because it has been reserved by another process.
06 02	FS ERROR : DEVICE NOT READY Emitted by file system primitives when a device has been disconnected or switched off incorrectly. If the device is a hard or a floppy disk unit, the disk drive may have been switched off incorrectly (only for free-standing devices), or the wires have been disconnected. Both for free-standing and integrated floppy disk units, it may also be that the floppy was inserted the wrong way round, or that the drive cover was not closed. The user should eliminate the cause of the error and attempt the operation again.
06 03	FS ERROR : INVALID PATHNAME Emitted by file system primitives when the specified pathname is incorrect because it is longer than 60 characters, a central part is over 14 characters long, the name of the file is longer than 12 characters, part of the pathname does not exist.
06 04	FS ERROR : NAME ALREADY EXISTS Emitted by the file system primitive COPY, MAKEVOLUME, MOUNT, RENAME and by the CREATE primitives. A directory, a file or a volume cannot be written again, added, created, etc.
06 05	FS ERROR : NAME NOT FOUND Emitted by the file system primitives when the name of a file, directory or volume cannot be found. This message is emitted when the user tries to remove a file or a directory, to connect a file or a directory, to create or remove indices from a keyed filed unsuccessfully.

06 06	FS ERROR : NOT EMPTY DIRECTORY Returned by the file system primitive REMOVE. A directory cannot be removed because it is not empty, that is it contains files or sub-directories. In this case, to empty the directory, the primitive CLEARDIR will have to be used, or the equivalent MCL command.
06 07	FS ERROR : END OF FILE Returned by the file system primitives READBYTE and READSEQ at the EOF. With READBYTE the program tries to read byte over the end of a byte-stream file; the remainder of the bytes up to the end of the file have been read. With READSEQ the program tires to read the last records, that do not contain an LF, of a sequential file.
06 08	FS ERROR : OUT OF BOUNDS Returned by the file system primitives READBYTE and READPOS when an attempt is made to read over the end of the file.
06 09	FS ERROR : RECORD NOT FOUND Returned by the file system primitives DELETEPOS, READPOS and WRITEPOS (writeMode = rewrite) when there is not valid record in the position that has been specified.
06 10	FS ERROR : RECORD ALREADY EXISTS Returned by the file system primitive WRITEPOS when the write mode specified is 'newwrite' or 'alwayswrite', but the record to insert already exists.
06 11	FS ERROR : KEY NOT FOUND Returns by the file system primitives DELETEKEYED, READKEYED and WRITEKEYED. A keyed file cannot be accessed because the key has not been found in the index.

06 12	FS ERROR : KEY ALREADY EXISTS Returned by the file system primitives CONVERT and WRITEKEYED. In the first case, during conversion of a positional file to a keyed file the user has tried to create a primary index with duplicate keys. In the second case, WRITEKEYED has been used with the option 'newwrite' or 'alwayswrite', but the record cannot be written in the file because it would create a duplicate key.
06 13	FS ERROR : OPERATOR INTERVENTION The operator has modified the operating mode whilst operations were taking place: i.e. a floppy disk has been been replaced whilst it is being used.
07 00	WS WARNING : END OF PAGE Returned by the printer driver primitive WRITEPR, at the end of a page or document that is being printed. For handling end of page, see the specific hardware documentation for the printer.
07 01	WS WARNING : DOCUMENT TOO FAR IN Returned by the printer driver primitives READPR and WRITEPR when the document is not inserted correctly in the front feeder, or when the printer is in local. For correcting this situation, refer to the specific hardware documentation.
07 02	WS WARNING : OPERATOR REQUEST Returned by the printer driver and cash adapter primitives when an event that requires operator intervention occurs. For the printer, a document is in the front feed or the printer tape has nearly finished. The program must display this message in the form of a request to replace the tape, and to continue without a reset. For the cash adapter, see MOS, PMM and Driver Primitives, Reference Manual.

07 03	WS WARNING : NO CHARS Emitted by the terminal driver primitive READINFOTM and by the PIN pad primitive READINFOPP when there are no characters waiting for the circular buffer of the keyboard.
07 04	WS WARNING : BREAK OCCURRED Emitted by the terminal driver primitive READTM, by the printer primitive WRITEPR and the PIN pad primitive READPP. A break function has been requested via the specific key, or a software interruption has been sent to the family that has activated the primitive when it was waiting, or the time-out value set by means of the primitives WRITEINFOTM, WRITEINFOPR, WRITEINFOPP has expired.
07 05	WS WARNING : TERMINAL ON The specified terminal has been turned on.
08 00	WS ERROR : UNIT DOWN Emitted by the driver primitives when the peripheral requested (printer, PIN pad, badge reader/writer, cash adapter, cheque reade/writer) is off.
08 01	WS ERROR : UNIT NOT AVAILABLE Emitted by the driver primitives if the user has try to use a peripheral which is not defined in the configuration of the work station.
08 02	WS ERROR : RESOURCE BUSY Emitted by the driver primitives READINFOTM, READTM and by OPEN when another process is already using the resource requested.
08 03	WS ERROR : LINE ERROR An error has been detected on the line during data exchange with the peripheral connected.

08 04	WS ERROR : BOURRAGE Emitted by the printer, cash adapter and the cheque reader/writer driver primitives. A document is stuck in the front feed of the printer, the cash adapter is jammed, or a cheque is stuck in the cheque reader/writer feed. For recovery see the documentation for the specific peripheral.
08 05	WS ERROR : NO DOCUMENT Emitted by the printer driver primitives when no document has been inserted in the front feed. Emitted by the badge and cheque reader/writer driver primitives, when the time-out has expired before the badge or cheque has been inserted in the badge or cheque reader/writer, or when an attempt has been made to read or write on a document without a magnetic stripe.
08 06	WS ERROR : CARTER IS OPEN Emitted by the printer driver primitives OPENPR, CLOSEPR, READPR and WRITEPR when the carter of the printer is open. If the session was opened in default mode, the process will have to repeat the last and second to last write operation. If the session was opened in "front-feed" mode, the error can be recovered by repeating the last write operation only.
08 07	WS ERROR : LOCAL Emitted by the printer driver primitives OPENPR, CLOSEPR, READPR and WRITEPR when the printer is in local mode. The process must operate as for the previous error.
08 08	WS ERROR : VERIFY OR READ ERROR Emitted by the driver primitives when an attempt at reading or writing to the magnetic stripe fails, or the string that has been read does not meet the conditions of LRC or VRC. The operation must be repeated.

08 09	WS ERROR : VIRGIN STRIPE Emitted by the driver primitives READPR and READBG when the magnetic stripe is empty.
08 10	WS ERROR : PRINT HEAD BLOCKED Emitted by the printer driver primitives OPENPR and WRITEPR when the printer head is jammed (only for the PR 2840).
08 11	WS ERROR : END OF INPUT Returned by the terminal driver primitive READTM when the operator types **CONTROL D** (End Of File) as the first character.
08 12	WS ERROR : BADGE INSERTED Reurned by the badge reader/writer driver primitive READINFOBG if a badge has already been inserted in the badge reader/writer.
08 13	WS ERROR : INVALID INSERTION Emitted by the badge reader/writer driver primitives READBG and READINFOBG when the operator has inserted the badge wrongly.
08 14	WS ERROR : END OF TAPE Returned by the printer driver primitives CLOSEPR and WRITEPR and the cheque reader/writer primitives WRITECR and WRITEINFOCR when the tape of the daisy printer or the cheque reader/writer has finished.

08 15	WS ERROR : OFF OCCURRED Returned by terminal driver primitives. Signals the ON-OFF-ON transaction for the terminal, if when the terminal is OFF calls to the driver have not been arrived. and then the process has not received the signal 'TERMINAL DOWN'. When the terminal is switched ON again, the state of terminal is stored for what concers: split status, screen size, breack key and change window key. The initial state is restored for all the existing windows.
08 16	WS ERROR : PRINTER TIMEOUT Emitted by the printer driver primitives CLOSEPR, OPENPR, READPR and CLOSETEPR when the time-out for waiting for a reply from the printer expires, that is fixed by the WRITEINFOPR primitive.
11 00	LM WARNING : LM END OF FILE This code is returned by the line manager interface primitive READCU if in the data received there is a signal for the logical end of the record. The length of a read operation that returns LMENDOFFILE may be equal to zero. For SNA: this reply-code returns different information depending on the value assigned to the EOF signal (in the StationName table) in configuration. If the EOF signal has been fixed at 1, the reply-code must be interpreted as End of Chain. If the EOF signal has been fixed at 2, the reply-code must be interpreted as End Bracket.
11 01	LM WARNING : OVERFLOW This code is returned by the line manager interface primitives READCU and READUNS if the length of data received exceeds the user input area available. The extra data are passed to the user in a successive read operation.

11 02 LM WARNING : INVERSION REQUEST

This code is returned by the line manager interface primitives READCU and WRITECU and depends on the protocol.
For BSC: it is returned by WRITECU to inform an extended interface user that the handler has received a request to invert the data flow. The primitive is not executed, to allow the application program to take the necessary action.
For SNA: READCU returns this code and no data in the user data area, to inform the user that the change direction indicator has been positioned to invert the data flow. READCU is not executed and the next operation performed by the user must be a WRITECU. WRITECU returns this code to inform the user that the data flow is in receive mode (e.g. a BID command has been received and accepted).

---

11 03 LM WARNING : STATUS CHANGED

This code is returned by the line manager interface primitive WRITECU, for the protocol SNA, if a BID command or an open bracket has been refused.

---

11 04 LM WARNING : COMMAND RECEIVED

This code is returned by the line manager interface primitives READCU, WRITECU, CLOSECU, WRITECUINFO and READCUINFO, to inform an extended interface user that a protocol-dependent command has been sent to the application. The primitive that has returned the warning is not executed to allow the application to read the command and its associated data, if any , by means of the READCUINFO the primitive.

---

11 05 LM WARNING : ERROR CONDITION

This code is returned by the line manager interface primitives READCU, WRITECU, CLOSECU, WRITECUINFO and READCUINFO, to inform an extended interface user that a pending error condition has occurred; the primitive is not executed to allow the application to read the nature of the error, using the primitive READCUINFO, and to take the necessary action if necessary.

---

12 00	LM ERROR : TOO MANY CU CREATED This code is returned by the line manager primitive CREATECU if the number of CUs in use would exceed the overall maximum allowed, or the max number of multiplexted CUs allowed on a single Link Unit (LU). In this case two actions can be taken: repeat CREATECU until a CORRECT reply-code is returned, or offline processing can be continued until a CU is free.
12 01	LM ERROR : CONNECTION FAILED This code is returned by the line manager primitives CREATECU, OPENCU and READCU if an anomalous condition has occurred that stops physical connection with the remote application.
12 02	LM ERROR : LINE DOWN This code is returned by the line manager interface primitives OPENCU, READCU, WRITECU, CLOSECU and WRITECUINFO when the line is inactive.
12 03	LM ERROR : LINE ERROR This code is returned by the line manager interface primitives READCU, WRITECU, CLOSECU and WRITECUINFO, when an error occurs on the line, but the connection is still active; the application program can resyncronise with the remote application to retry the operation again.
12 04	LM ERROR : REMOTE BUSY This code is returned by the line manager interface primitives WRITECU, CLOSECU and WRITECUINFO, when the time-out value defined by the user with OPENCU has expired before the requested operation has finished.
12 05	LM ERROR : DATA NOT AVAILABLE This code is returned by the line manager interface primitives READCU if there is an error in the program; the data to be read have not been received before time-out expires.

12 06	LM ERROR : RESOURCE BUSY This code is only returned for SNA, by the line manager interface primitive, READCU, when resource at protocol level, needed to execute the user request is being used by another application.
12 07	LM ERROR : RESOURCE NOT AVAILABLE This code is only returned for SNA, by the line manager interface primitive, READCU, CLOSECU and WRITECUINFO, when a resource at protocol level, needed to execute a user request is not available (temporarily or permanently).
12 08	LM ERROR : INPUT ABORT This code is returned by the line manager interface primitive READCU if the sequence of input messages has been aborted.

REPLY-CODES FOR THE PORT/UPORT PRIMITIVES

-2	WAIT SUSPENDED Returned by the primitive RECEIVEPORT as a kernel message specifying that control has returned to the receiving process, forced by **CONTROL K.**
-100	FATAL ERROR Returned by the primitive NOTIFYSTATUS if no port can be obtained to receive the reply, and by the primitive NOTIFYSWITCH if the input parameters are inconsistent. Reurned by the primitived UPORT in reply to the protocol when the result of checks on some of the constant value fields is negative. The user must check the electrical connection of the line.
-101	INVALID PORT The input parameter ObjectId is incorrect.
-102	PARAMETER ERROR Returned by the primitive NOTIFYSWITCH when the machine from which the notification has been requested, is the local machine, and by REMOVENOTIFY when the parameter Notifyld is incorrect.
-103	NO TABLE SPACE Returned by the primitives CONNECTPORT, NOTIFYSWITCH, OPENRCVPORT and OPENXMTPORT, when there is not enough space allocated for the internal tables. Emitted by the primitive UPORT when the session that has been opened cannot be established, because there are not private ports. The user must reconfigure the system with more private ports.
-104	PORT OPEN IN RECEIVE Returned by the primitive OPENRCVPORT when the port has been already opened in receive mode. This is only a warning: the port is already open, so there is no point in repeating the primitive.

-105	PORT1 LINKED Returned by the primitive CLOSERCVPORT when the specified port cannot be closed because there are one or more ports linked to it, i.e. it is considered to be ObjectId2 in one or more link-port primitives. To close the port, the ports linked to it must first be disconnected and neutralized. Returned by LINKPORT when the port identified by the parameter ObjectId1 is already connected to another port, or one or more other ports are connected to it. Only one linking level is allowed. The Distinguished Port is an exception; it can be linked to others if the existing links are implicit, but not if they are explicit. Returned by READPORTDATA and RECEIVEPORT when the port is linked to another port (it has been specified as ObjectId1 in the preceding LINKPORT) and therefore cannot receive messages, because they are all forwarded to another port.
-106	PORT2 LINKED Returned by the primitive LINKPORT when the port identified by the parameter ObjectId2 is already linked to another port and, as only one level link is allowed, no further link is allowed, unless the current port is first unlinked.
-107	INVALID MESSAGE CONTROL BLOCK Returned by the primitives BROADCAST, READPORTDATA, RECEIVEPORT and SENDPORT when the message control block is incorrect or inconsistent; e.g. when one of its elements is greater than that defined by the system, or the number of elements is lesser that 1 or greater than the max value allowed. Returned by the primitive UPORT when requesting to receive and/or transmit a message mapped on an inadmissible control block.
-108	MESSAGE TOO LONG Returned by the primitives BROADCAST and SENDPORT when the total length of the message to be sent is greater than the max value defined by the system.

-109	NO ROUTE Returned by the primitive OPENXMTPORT when the port cannot be opened because its position in the network cannot be identified, or because it cannot be reached; all the routes are blocked (e.g. because certain links are down). Returned by SENDPORT when the message cannot be send because there are no route connecting the system on which the destination port is situated. This usually means that the route chosen when OPENPORT was executed cannot be used because the line is down.
-110	LINK DOWN Returned by the primitive UPORT when data exchange has failed because of an anomaly at one of the communication levels that reaches the driver line from the port. The actions to take in this case are: check the electrical connections of the machines involved in the data exchange, and that data have not been sent to a machine which is off; check the configuration parameters of the line and the port.
-111	RECEIVE TIMEOUT Returned by the primitive RECEIVEPORT when the time-out specified by the OPENRCVPORT (or modified by RESPORTINFO) has expired without a message having arrived at the port.
-112	NO DATA LEFT Returned by the primitive READPORTDATA when the message is finished and therefore no data have been copied. A ReceivePort primitive may be executed in order to read the next message.
-114	ABORT REQUEST Returned by the primitive UPORT, indicates the lack of an acknowledge relative to a message. This code confirms that the part of the communication that goes from the port to the line driver is working properly. The error can be caused by overloading of the machines exchanging messages or by errors in the application programs.

-126	MACHINE DOWN Returned by the primitive NOTIFYSTATUS, shows that the state of the machine is DOWN.
-127	MACHINE UP Returned by the primitive NOTIFYSTATUS, shows that the state of the machine is UP.

SYSTEM ERRORS IN ALPHABETICAL ORDER AND BY CLASS

CLASS	STRING	CODE
DEBUG	BAD SCOPE	02 07
DEBUG	BAD STATE	02 08
DEBUG	CHANNEL BUSY	02 11
DEBUG	CHANNEL DESTROYED	02 10
DEBUG	ERROR IN THE STRING	02 04
DEBUG	ILLEGAL INDEX	02 09
DEBUG	INVALID IDENTIFIER	02 02
DEBUG	INVALID OPERATION	02 01
DEBUG	INVALID PARAMETERS	02 03
DEBUG	NOT YET IMPLEMENTED	02 06
DEBUG	RECORD TOO LONG	02 00
DEBUG	SECURITY VIOLATION	02 05
DEBUG	NO WRITERS	02 14
FS ERROR	DEVICE NOT READY	06 02
FS ERROR	END OF FILE	06 07
FS ERROR	INVALID PATHNAME	06 03
FS ERROR	KEY ALREADY EXISTS	06 12
FS ERROR	KEY NOT FOUND	06 11
FS ERROR	NAME ALREADY EXISTS	06 04
FS ERROR	NAME NOT FOUND	06 05
FS ERROR	NOT EMPTY DIRECTORY	06 06
FS ERROR	OPERATOR INTERVENTION	06 13
FS ERROR	OUT OF BOUNDS	06 08
FS ERROR	RECORD ALREADY EXISTS	06 10
FS ERROR	RECORD LOCKED	06 01
FS ERROR	RECORD NOT FOUND	06 09
FS ERROR	TIMEOUT	06 00
FS WARNING	BYTE-STREAM OPERATION	05 00
FS WARNING	DUPLICATED KEY	05 04
FS WARNING	INCOMPLETE RECORD	05 05
FS WARNING	INCONSISTENT DATA BASE	05 02
FS WARNING	NO PRIMARY INDEX	05 03
FS WARNING	POSITIONAL OPERATION	05 01
FS WARNING	VOLUME NOT COPIED	05 06

CLASS	STRING	CODE
LM ERROR	CONNECTION FAILED	12 01
LM ERROR	DATA NOT AVAILABLE	12 05
LM ERROR	INPUT ABORT	12 08
LM ERROR	LINE ERROR	12 03
LM ERROR	LINE DOWN	12 02
LM ERROR	REMOTE BUSY	12 04
LM ERROR	RESOURCE BUSY	12 06
LM ERROR	RESOURCE NOT AVAILABLE	12 07
LM ERROR	TOO MANY CU CREATED	12 00
LM WARNING	COMMAND RECEIVED	11 04
LM WARNING	ERROR CONDITION	11 05
LM WARNING	INVERSION REQUEST	11 02
LM WARNING	LM END OF FILE	11 00
LM WARNING	OVERFLOW	11 01
LM WARNING	STATUS CHANGED	11 03
PMM ERROR	CONCURRENT ERROR	04 01
PMM ERROR	MEMORY OVERFLOW	04 00
PMM ERROR	TOO LONG	04 02
PMM WARNING	BREAKPOINT	03 10
PMM WARNING	FORMAT ERROR	03 03
PMM WARNING	ILLEGAL SEGMENT LENGTH	03 04
PMM WARNING	INVALID SEGMENT NUMBER	03 05
PMM WARNING	MEMORY CONFLICT	03 00
PMM WARNING	NO HALT	03 07
PMM WARNING	NON-EXISTENT FILE	03 01
PMM WARNING	NOT EMPTY	03 11
PMM WARNING	PROCESS DESTROYED	03 08
PMM WARNING	PROCESS SUSPENDED	03 09
PMM WARNING	SOFTWARE INTERRUPT	03 06
PMM WARNING	TYPE CONFLICT	03 02

CLASS	STRING	CODE
SYSTEM	DIAGNOSTIC ERROR	01 01
SYSTEM	FATAL ERROR	01 07
SYSTEM	FATAL SEGMENT FAULT	01 15
SYSTEM	HARDWARE ERROR	01 03
SYSTEM	INHIBIT SEGMENT FAULT	01 11
SYSTEM	INVALID INSTRUCTION	01 13
SYSTEM	KERNEL SEGMENT FAULT	01 09
SYSTEM	NO PROCEDURE	01 16
SYSTEM	NO ROUTING	01 17
SYSTEM	OPERATION ABORTED	01 19
SYSTEM	OUT OF DISK SPACE	01 06
SYSTEM	PRIVILEGED INSTRUCTION	01 14
SYSTEM	RECEIVE ERROR	01 02
SYSTEM	RESULTS DOUBTFUL	01 18
SYSTEM	READ-ONLY FAULT	01 08
SYSTEM	SEGMENT LENGTH FAULT	01 10
SYSTEM	SYSTEM ERROR	01 04
SYSTEM	TABLE OVERFLOW	01 05
SYSTEM	TERMINAL DOWN	01 00
SYSTEM	EXECUTE FAULT	01 12
WS ERROR	BADGE INSERTED	08 12
WS ERROR	BOURRAGE	08 04
WS ERROR	CARTER IS OPEN	08 06
WS ERROR	END OF INPUT	08 11
WS ERROR	END OF TAPE	08 14
WS ERROR	INVALID INSERTION	08 13
WS ERROR	LINE ERROR	08 03
WS ERROR	LOCAL	08 07
WS ERROR	NO DOCUMENT	08 05
WS ERROR	OFF OCCURRED	08 15
WS ERROR	PRINT HEAD BLOCKED	08 10
WS ERROR	PRINTER TIMEOUT	08 16
WS ERROR	RESOURCE BUSY	08 02
WS ERROR	UNIT DOWN	08 00
WS ERROR	UNIT NOT AVAILABLE	08 01
WS ERROR	VERIFY OR READ ERROR	08 08
WS ERROR	VIRGIN STRIPE	08 09
WS WARNING	BREAK OCCURRED	07 04
WS WARNING	DOCUMENT TOO FAR IN	07 01
WS WARNING	END OF PAGE	07 00
WS WARNING	NO CHARS	07 03
WS WARNING	OPERATOR REQUEST	07 02
WS WARNING	TERMINAL ON	07 05

PORT/UPORT REPLY-CODES GIVEN IN ALPHABETICAL ORDER AND BY STRING

STRING	REPLY-CODE
ABORT REQUEST	-114
FATAL ERROR	-100
INVALID MESSAGE CONTROL BLOCK	-107
INVALID PORT	-101
LINK DOWN	-110
MACHINE DOWN	-126
MACHINE UP	-127
MESSAGE TOO LONG	-108
NO DATA LEFT	-112
NO ROUTE	-109
NO TABLE SPACE	-103
PARAMETER ERROR	-102
PORT OPEN IN RECEIVE	-104
PORT1 LINKED	-105
PORT2 LINKED	-106
RECEIVE TIMEOUT	-111
WAIT SUSPENDED	-2

B. DESCRIPTION OF THE ERROR MESSAGES OF THE SHELL MAINTENANCE COMMANDS

This appendix contains a detailed description of the error messages emitted by the following maintenance utilities of the system software:

- DISKCHECK
- DISKEDIT
- TREECHECK
- TREEMEND
- VOLGC

The messages are listed in alphabetical order.

BAD POINTER TO NEXT RECORD

The following messages are displayed if errors occur during execution.

BLOCK SIZE NOT MULTIPLE OF 'BYTE_PER_SECTOR' IN VOLUME DESCRIPTOR

The volume descriptor gives a size in blocks that is not a multiple of the number of blocks per sector.

DUPLICATE POINTER

PDD pointed to twice in free list.

ERROR : DISK BIT MAP LENGTH EXCEEDS VOLUME DIMENSION

Self explanatory

ERROR : DUPLICATE FILE ID: 'check Ent.id'

The same PDD is associated to two files: one of the directory entries (usually the second) is released.

ERROR : PDD ID OUT OF RANGE : 'check Emt.id'

There are more PDDs than have been dimensioned for the current volume.

ERROR : TRACING STOPPED AT RECORD N. 'extNumber' DUPLICATED POINTER

A duplicated directory has been found during extent tracing.

ERROR IN "." ENTRY

Error in the current directory: entry missing, or the wrong PDD has been associated to the name.

ERROR IN ".." ENTRY

Error in the father directory: entry missing, or the wrong PDD has been has been associated to the name.

ERROR IN BACK_LINK IN EXTENT TABLE CHAIN

Incorrect back link in an extent.

ERROR IN DATA SET 'pddNumber' : BLOCK MULTIPLY ALLOCATED.

More than one file descriptor refers to a specific number of blocks as its own.

ERROR IN FREE EXTENT LIST; STOPPED AT RECORD N. 'extNumber'

Some of the pointers to the free extent list are wrong.

ERROR IN FREE PDDLIST; STOPPED AT RECORD N. 'pddNumber'

Some of the pointers to the free PDD list are wrong.

ERROR IN READ EXTENT TABLE - EXTENT NOT TRACEABLE

The information related to the file cannot be found.

ERROR IN READ OF PDD - DATA SET NOT TRACEABLE

In the extent the space described for the current file is not consistent with the addresses of the volume containing it.

EXTENT INTEGRITY ERROR - BLOCK NOT TRACEABLE

In the extent the space described for the current file is not consistent with the addresses of the volume containing it.

FULL LENGTH OF DATA SET NOT TRACEABLE FROM EXTENTS

The length of the file is < 0 or > of the space allocated.

INVALID LINK TO NEXT EXTENT

The pointer "next" in an extent directory is a non-significant number.

MISSING BLOCKS SPACE (ALLOCATED BUT NOT SET)

The space is referred to in the file descriptor, but shown as free in the Bit Map.

MISSING BLOCKS SPACE (SET BUT NOT ALLOCATED)

The space is occupied, but it is not associated to a file descriptor.

NUMPDDS <0 IN VOLUME DESCRIPTOR

Incorrect information in the volume descriptor.

NUMPDDS INCORRECT IN VOLUME DESCRIPTOR

Incorrect number of PDDs: max. number of PDDs allocated exceeded.

PDD FIRST BYTE INVALID

The file is empty by the field 'firstByte' in the PDD has not been reset.

PDD INTEGRITY : ERROR - VALUES OUT OF RANGE

The space described for the file is outside the volume.

RECORD FLAGGED AS ALLOCATED

Extent wrongly allocated in free list

RECORD POINTED TO IS FLAGGED ALLOCATED

The record pointed is in free list by mistake

SPACE ALLOCATION ERROR - BLOCK MULTIPLY ALLOCATED

More than one PDD refers to the same space.

UNDEFINED CLASS

The field of a PDD dedicated to the type of file contains an nonsignificant value.

UNDEFINED ERROR REPORT THIS CASE, PLEASE

Call the Olivetti Software Assistance Service

VOLSIZE <0 IN VOLUME DESCRIPTOR

Wrong information in the volume descriptor.

WARNING : PDD FOR ROOT DIRECTORY IS ON FREE LIST

The PDD of the Root directory is in the free list. The situation can be recovered using the DISCHECK.

BAD ERROR NUMBER

Error not recognised by the utility.

COMMAND UNKNOWN

The utility has specified a command which is not valid for the utility.

ERROR IN OPTION

Error in specifying the options.

ERROR IN PARAMETER(S)

Error in specifying the pathname.

ERROR: DIGIT EXPECTED

A digit is expected that the user has not specified.

ERROR: OUT OF RANGE

An element of the Bit-map or the PDD-table does not exist.

ERRORE IN UNMOUNT OF DEVICE

An error has occurred during the UNMOUNT operation on a physical device.

HARDWARE ERROR

An anomaly has occurred on a device.

INVALID COMMAND

A Write command has been specified without setting the option "W".

INVALID POINTER IN PDD OF DIRECTORY

A PDD points out of the space in that volume.

Action: execution of the utilities DISKCHECK and VOLGC

NO SUCH ENTRY IN DIRECTORY

Attempt at addressing an entry in a directory which is not present in the handling table of the directory.

NO VALID POINTER IN PDD OF ROOT DIRECTORY

The PDD of the volume directory is outside the space of the volume.

Action: execution of the utilities DISKCHECK and VOLGC.

NOT A DIRECTORY

When the command "D" is specified a PDD-id is obtained that is for a directoy.

PDD-ID OUT OF RANGE

A PDD has been specified that is out of the PDDs range.

SYNTAX ERROR

Error in specifying the command.

SYSTEM ERROR

The volume is not consistent.

Action: if the father volume exists check its consistency using DISKCHECK and VOLGC, otherwise this is the result of anomalous system operating.

A FATHER PAGE LINK IS WRONG

The pointer field in the father page of the page of the index file, does not point to the beginning of a file, but to the middle of a higher level page.

Action: If the index refers to an internal key, it can be deleted and reconstructed with the commands DELINDEX and MAKEINDEX applied to that index.

A FATHER PAGE LINK IS OUT OF BOUNDS

Link damaged in the B-tree structure. The pointer field to the father page of a page, contains an offset that is outside the limits of the file.

Action: execution of the utility TREEMEND.

CHECK CONSISTENCE BETWEEN INDICES - RECORD POSITION <position>

The index is not consistent with the other file data indices.

Action: the indices that caused the message must be removed and reconstructed. These operations are carried out using the Shell utilities DELINDEX and MKINDEX (see SHELL - Operating Commands - Reference Manual). If there is a power failure when the MKINDEX is building a primary index, an error condition is generated that may be recovered by replacing the current file with a backup copy.

DATA FILE CORRUPTED

The data file has been truncated, and so it does not make sense to recreate the index.

Action: use a back-up copy of the file.

DOUBLE REFERENCE TO PAGE OFFSET : <offset> - LEVEL : <level>

Within the page linking mechanism, there are two references to the same page.

Action: the index that caused the message must be removed using the Shell procedure DELINDEX.

DOUBLE REFERENCE TO POSITION : <position> - OFFSET : <offset>
- LEVEL : 1

A record position has been referenced twice, i.e. the record position associated with that key is not unique.

Action: the index that caused the message must be removed using the Shell procedure DELINDEX.

DUPLICATED KEY

There is a duplicate key in the index file.

Action: if the key is a secondary key, remove it using DELINDEX of the duplicate secondary key.
If the key is a primary key, no recovery is possible.

EMPTY FILE - NO ACTION

The data file is empty.

EMPTY INDEX

The index that is being examined is empty.

Action the index that caused the message must be removed using the Shell procedure DELINDEX.

ERROR IN CONNECT

An error has occurred when connecting a file.

Action: check the consistency of the pathname given in input.

ERROR IN CONNECT FATHER DIRECTORY

Self-explanatory.

Action: check the consistency of the pathname given in input.

ERROR IN CONNECT FILE

Self-explanatory.

Action: check the consistency of the pathname given in input.

ERROR IN CONNECT FILE INFO

Self-explanatory. It is linked to the message "FILE INFO NON FOUND".

Action: use of the command for reconstructing the structure of the keyed file (e.g.: CONVERT on a data file).

ERROR IN CONNECT HOME DIRECTORY

Self-explanatory.

action: check the consistency of the pathname given in input.

ERROR IN CONNECT INDEX

The index file cannot be connected.

Action: removal of the index by means of the Shell procedure DEL-INDEX

ERROR IN CONNECT RENAMED DATA FILE

Error in connecting the renamed data file.

Action: reactivate the utility.

ERROR IN CONNECT TEMPORARY INDEX

Error when connecting the temporary index.

Action: move to another volume and reactivate the utility.

ERROR IN CONNECT TERMINAL INPUT

There is an error in the standard input.

Action: activate the utility from another WS.

ERROR IN CONNECT TERMINAL OUTPUT

There is an error in the standard output.

Action: activate the utility from another WS.

ERROR IN CONNECT WORKING DIRECTORY

Self-explanatory

ERROR IN CREATE OPEN INDEX

Error when opening the temporary index.

Action: move to another volume and reactivate the utility.

ERROR IN CREATE TEMPORARY FILE

There is inconsistency on the volume being processed.

Action: re-run the utility from another volume.

ERROR IN CREATE TEMPORARY INDEX

Error when creating the temporary index.

Action: move to another volume and reactivate the utility

ERROR IN FIND_KEY

The specified key cannot be found on the file. The key is lost.
This error is not displayed, but is emitted if the SHVAR command is executed straight after the utility.

ERROR IN GET INDEX FOR CONNECT OR OPEN

Error following "NAME NOT FOUND".

ERROR IN GET TYPE

The file connect operation has not been carried out properly: the utility cannot continue, because the file type cannot be found.

Action: transfer the file to another machine and retry.

ERROR IN OPEN FILE

An error has occurred when opening a file.

ERROR IN OPEN FILE INFO

Self-explanatory. This is linked to the message "FILE INFO NON FOUND".

ERROR IN OPEN TERMINAL INPUT

There is an error in the standard input.

Action: run the utility on another WS.

ERROR IN OPEN TERMINAL OUTPUT

There is an error in the standard ouput.

Action: run the utility on another WS.

ERROR IN OPTION

Error in specifying the options.

Action: reactivate the command following the specifications

ERROR IN PARAMETERS

Error in specifying the pathname of "file-name": the utility must check the name of the index file.

ERROR IN READING TEMPORARY FILE

A reading error has occurred on the temporary file.

Action: check the reliability of the volume, the disk, etc.

ERROR IN REMOVE TEMPORARY FILE

Self-explanatory

Action: reactive the utility.

ERROR IN RENAME DATA FILE

The volume from which the utility is activated or the volume on which the file is resident, is corrupted.

Action: use the utility VOLGC

ERROR INVALID KEY_LENGTH

The length of the key is greater than the length of the record (for internal keys).

Action: remove the index using the procedure DELINDEX.

ERROR INVALID PAGE SIZE

The PAGE SIZE of the Info file is not within the range.

Action: remove the index using the procedure DELINDEX.

ERROR INVALID START BYTE

The start byte of an index is outside the length of a record.

Action: remove the index using the procedure DELINDEX. Use of the utility TREEMEND is inadvisable.

ERROR INVALID STRATEGY

The strategy is not within the range of valid values. The Info file is completely corrupted.

Action: remove the index using the procedure DELINDEX.

ERROR ON FILE INFO

Error in reading the Info file.

Action: the error cannot be recovered; if possible use a back-up copy of the Info file.

FILE IN USE BY ANOTHER ENVIRONMENT

The file is being used in another environment so it cannot be used exclusively.

Action: wait until the end of the blocking processes.

FILE INFO CORRUPTED

The information file has been truncated for some reason, and at least one index may be recreated, but not all.

Action: execute the TREEMEND utility.

FILE INFO NOT FOUND

The information file cannot be found for the specified file. If this is the case, this message is the only text output by TREECHECK.

I/O ERROR

Reading or writing error on any file being used by the utility: both for reference or as part of the index file.

Action: check the possible causes that have generated this error (LMS can return significant signals to the Master WS).

INFO COMPLETELY CORRUPTED

The information file has been truncated for some reason, and there is not sufficient information to recreate the indices.

INVALID FIELD NUMBER

The user has specified an index that does not exist in the index file.

Action: reactivate the utility specifying one of the indices that is present.

INVALID NEXT PAGE LINK

The pointer field at the "next-page" of a page of the index file does not point to the beginning of a page.

Action: remove the index using the DELINDEX utility. If there is a back-up copy of the file, restore the copy of the file.

INVALID OPERATION

The specified file is not a keyed or packed positional file.

INVALID OPTION(S)

An illegal option has been specified.

Action: reactivate the command following the specifications.

INVALID PATHNAME

The specified pathname is greater than 60 characters.

KEY ALREADY EXIST

Is a duplicate key: the lower level (0) of the index file has been damaged.
This error is not displayed, but is emitted if the command SHVAR is executed after executing the utility.

Action: recover a backup copy of the file, if this is consistent.

KEYS DUPLICATED IN PRIMARY/SECONDARY INDEX

Duplicated keys are in non-duplicated primary or secondary indices.

Action: the index that caused the message must be removed using the Shell procedure DELINDEX.

KEYS NOT ORDERED - OFFSET : <offset> - LEVEL : <level>

The keys on a particular page are not in ascending order.

Action: The index that caused the message must be removed using the Shell procedure DELINDEX.

KEYS NOT ORDERED ON LEVEL 1

The level 1 keys are not in ascending order.

Action: remove the index using the Shell procedure DELINDEX.

MAX KEY NOT EXISTS LEVEL <level>

The max. key for the level being examined does not exist.

Action: execution of the utility TREEMEND for the index that caused the message.

NAME NOT FOUND

This message may have been caused by the following:

- The specified pathname cannot be found.
- The index file for an index is missing. In this case the error has been preceded by the message "FILE INFO CORRUPTED", returned by the primitive Connect, which is not consistent with the Info file and the indices.

Action: execute the utility TREEMEND to recover consistency between the Info file and the undamaged Info file. The contents of the index file that generated the error have been lost.
If there is a complete back-up copy of all the index files, the index file for the one that has been damaged can be recopied, so recovering the entire situation.

NEXT PAGE LINK OUT OF BOUNDS - OFFSET : <offset> - LEVEL : <level>

Within the page linking mechanism, the address of a page indicated by the pointer to it is not within the bounds of the index.

Action: The index that caused the message must be removed using the Shell procedure DELINDEX.

NO REFERENCE TO POSITION <position>

A record position associated with a particular key is not referenced in the index file.

Action: the index that caused the message must be removed using the utility TREEMEND.

OUT OF DISK SPACE

There is not enough space to create temporary back-up files in the volume from which the utility has been activated.

Action: move to another volume and reactivate the utility.

POSITION OUT OF BOUNDS: <position> - OFFSET : <offset> - LEVEL : 1

This message is only output when level 1, the last level of the tree, is being checked. Each key has associated with it a position relative to the offset of the record in the data file. In this case, the record position is not within the bounds of the data file. **offset** indicates the page offset in the index file in which the error has been detected.

Action: the index that caused the message must be removed using DELINDEX utility.

POSITION OUT OF INDEX : <position> - OFFSET :<offset> - LEVEL : <level>

This message has the same meaning as the previous one, but it is output for the higher levels of the tree. A record position is not within the bounds of the index.

Action: execute the utility TREEMEND for the index referenced by the message.

PREVIOUS PAGE LINK CORRUPTED - OFFSET : <offset> - LEVEL : <level>

The link between a page and the page preceding it has been corrupted.

Action: execution of the utility TREEMEND for the index that caused the message.

A reading error has occurred on the index file.

Action: check the type of error (LMS)

REMOTE FILE NO REMOTE ACTION

A file is being referenced that does not reside on the machine from which the utility has been activated, but on another one in the same network.

THE RECOVERY IS NOT POSSIBLE WITH THIS ERROR.
YOUR BACK COPY MAY BE BETTER THAN THE ACTUAL

Self-explanatory.

WRITE I/O ERROR ON TEMPORARY INDEX

Self-explanatory.

Action: move to another volume and reactivate the utility.

WRONG FILETYPE

The filename input in the activation command is not a positional packed or a keyed file. If this is the case, this message is the only text output by TREECHECK.

WRONG KEY IN FATHER PAGE - OFFSET: <offset> - LEVEL: <level>

This key is not associated in the father page with the offset of the current page.

Action: execution of the utility TREEMEND for the index that caused the message.

WRONG NUMBER OF BYTES USED - OFFSET : <offset> - LEVEL : <level>

In each page header is given the number of bytes used in the page. The number of bytes actually used in a page does not correspond to the number in the header.

Action: the index that caused the message must be removed using the Shell utility DELINDEX.

WRONG NUMBER OF LEVELS IN FILE INFO

The number of levels given in the information file does not correspond to the actual number of levels in the index.

Action: execution of the utility TREEMEND on the index that caused the message.

WRONG POSITION IN FATHER PAGE - OFFSET: <offset> - LEVEL: <level>

The offset at which the current page is located is associated on the father page with a key whose value is different from the maximum key value of the current page.

Action: execution of the utility TREEMEND for the index that caused the message.

WRONG ROOT PAGE OFFSET IN FILE INFO

The location of the root page of the index as given in the information file does not correspond to the actual location of the root page.

Action: execution of the utility TREEMEND for the index that caused the message.

BLOCK SIZE NOT MULTIPLE OF 'BYTE_PER_SECTOR' IN VOLUME DESCRIPTOR

The volume descriptor shows a dimension in blocks that is not a multiple of the number of blocks per sector.

BUSY FLAG NOT OKAY

The field 'next_available_pdd'of an allocated PDD is other than zero.

CANNOT SCAN

The directory is not connected.

ERROR : DISK BIT MAP LENGTH EXCEEDS VOLUME DIMENSION

Self-explanatory.

ERROR : DUPLICATE FILE ID : 'check Ent.id'

The same PDD is associated to two files: one of the entries in the directory (usually the second) is released.

ERROR IN BACK_LINK IN EXTENT TABLE CHAIN

Incorrect back link in an extent.

ERROR IN FILE 'pddNumber' : BLOCK MULTIPLY ALLOCATED

More than one file descriptor refers a specific number of blocks as its own. The situation may be rectified using VOLGC.

ERROR IN FREE EXTENT LIST; STOPPED AT RECORD N. 'extNumber'

Some of the pointers to the free extent list are wrong. The situation is rectified using VOLGC.

ERROR IN FREE EXTENT LIST; STOPPED AT RECORD N. 0

Some of the pointers to the free extent list are wrong. The situation is rectified using VOLGC.

ERROR IN FREE PDDLIST; STOPPED AT RECORD N. 'pddNumber'

Some of the pointers to the free Pdd list are wrong. The situation may be rectified using VOLGCOOL

ERROR IN READ EXTENT TABLE - EXTENT NOT TRACEABLE

The information related to the file cannot be found.

ERROR IN READ PAR_DIR_ID

Error whilst the father directory of the lost directory is being read.

ERROR IN READ PDD - FILE NOT TRACEABLE

The information about the file cannot be found.

ERROR : PDD ID OUT OF RANGE : 'check Emt.id'

There are more PDDs than have been dimensioned for the current volume.

ERROR : TRACING STOPPED AT RECORD N. 'extNumber'

A duplicated directory has been found during extent tracing. The situation may be rectified by VOLGC.

EXTENT INTEGRITY ERROR - BLOCK NOT TRACEABLE

In the extent, the space described for the current file is not consistent with the addresses of the volume containing it.

FULL LENGTH OF FILE NOT TRACEABLE FROM EXTENTS

The length of the file is < 0 or > of the space allocated.

ERROR IN "." ENTRY

Error in the current directory: entry missing, or the wrong PDD has been associated to the name.

ERROR IN ".." ENTRY

Error in the father directory: entry missing, or the wrong PDD has been has been associated to the name.

INVALID LINK TO NEXT EXTENT

The pointer "next" in an extent directory is a nonsignificant number.

LENGTH OF FILE GREATER THAN ITS SPACE

Self explanatory.

MISSING BLOCKS SPACE (SET BUT NOT ALLOCATED)

The space is occupied but it is not associated to a file descriptor. The situation may be rectified by using VOLGC.

MISSING BLOCK SPACE (ALLOCATED BUT NOT SET)

The space is referred to in the file descriptor, but shown as free in the Bit Map. The situation may be rectified by VOLGC.

NO SPACE TO LOSTFILES DIRECTORY - USE THE OPTIONS "F TO REMOVE THE LOSTPDD".

The is no more space for the lost file directory. Use the option "F to remove the missing PDDs.

NUMPDDS <0 IN VOLUME DESCRIPTOR

Incorrect information in the volume descriptor.

NUMPDDS INCORRECT IN VOLUME DESCRIPTOR

Incorrect number of PDDs: max. number of PDDs allocated exceeded.

PDD FIRST BYTE INVALID

The file is empty but the field 'firstByte' in the PDD has not been reset.

PDD INSERTED IN LOST_FILE DIRECTORY

PDD omitted with consistent information inserted in the recovery directory with the name 'lost_NumPdd'.

PDD IDENTIFIER OUT OF RANGE

The number of PDDs is <0 or > of the number of files that the volume can contain.

PDD INTEGRITY : ERROR - VALUES OUT OF RANGE

The space described for the file is outside the volume.

SPACE ALLOCATION ERROR - BLOCK MULTIPLY ALLOCATED

More than one PDD refers to the same space.

SPACE ALLOCATION EXCEEDS VOLUME DIMENSIONS

The PDD of a file has more space allocated that the dimensions of the volume.

UNDEFINED CLASS

The field of a PDD dedicated to the type of file contains a nonsignificant value.

UNDEFINED CLASS : THIS INFORMATION IS UP TO YOU IF YOU LIKE USE DISKEDIT

The type of file that the PDD refers to is undefined. The following action can be taken:
find which file the PDD refers to and set the type in the PDD using the DISKCHECK.

UNDEFINED ERROR REPORT THIS CASE, PLEASE

Call the Olivetti Software Assistance Service.

VALUES OUT OF RANGE - BLOCKS NOT TRACEABLE

The values given for the current extent for the file are out of the range of the volume.

VOLSIZE <0 IN VOLUME DESCRIPTOR

Incorrect information in the volume descriptor.

WARNING : ERROR IN WRITE LOSTD-XXX

The lost directory cannot be written in. If the option "F has been used the damaged PDDs are inserted in the free list.

WARNING : NO MORE SPACE TO LOSTFILE DIR

If the option "F has been used the damaged PDDs are inserted in the free list.

WARNING : ERROR IN WRITE THE LOST FILE

Error during a write operation in the lost file directory.

WARNING : PDD FOR ROOT DIRECTORY IS ON FREE LIST

The PDD of the Root directory is in the free list. The situation may be recovered by using the DISKCHECK.

C. DESCRIPTION OF THE LIMITS OF THE SHELL MAINTENANCE COMMAND

This appendix contains a description of the limits for some of the maintenance utilities.

These are as follows:

- DISKCHECK
- TREECHECK
- TREEMEND
- VOLGC

DISKCHECK and VOLGC

The number of objects that the utilities DISKCHECK and VOLGC can handle depends on the volume size. The limit certified for release 5.2. is 10000 extents, even though these utilities can operate on a higher number of extents.

TREECHECK and TREEMEND

The max number of records, certified in release 5.2, which can be handled by TREECHECK and TREEMEND is 650000 records.

If the number of records to be processed is very high, it is unadvisable to invoke option "L" that produces more detailed output, as the information emitted increases execution time of the utility, which is very lengthy already.

Example

An example in which the execution times for the utilities TREECHECK, TREEMEND and CMPK is given below.

The test has been carried out on a file whose characteristics are as follows:

- it resides on a 120 Mbyte SMD Hard-disk.
- it consists of a single extent.
- it contains 650000 records.

- it has an allocation unit which is consistent with the number of records inserted
- the records have been inserted in random mode.
- the key is internal and it is made up of 10 characters.
- the file is undumaged.
- the space occupied is 649001 bytes.

The result of the test is as follows:

UTILITY	TIME TAKEN
TREECHECK (on unpacked files)	approx 34 hours
CMPK	approx 16 hours
TREECHECK (on packed files)	approx 24 hours
TREEMEND	approx 88 hours