AL

Title:

SYSTEM 3 UTILITY PROGRAMS

Part Two

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 31-D494 (PG4)

Edition: April 1978

Author: Tove Ann Aris
Hans Rischel

Keywords:

RC 4000, Basic Software, File Processor, Users Guide

Abstract:

This second part of the utility program manual contains detailed descriptions of the individual programs performing catalog handling, data handling, and job control. This 5th edition comprises 57 program names in alphabetic order. 126 pages.

ISBN 87 7557 0319

Printed by A/S Regnecentralen, Copenhagen

CONTENTS

CONTENTS
(RCSL 31-D494)

PREFACE

This second part of the utility program manual contains detailed descriptions of the individual programs except those which have their own manuals (compilers, editors, special programs). These manuals are found in the list of references on the next page.

Each description of a utility program has individual page numbering to facilitate updating. Revised versions of the program descriptions will be distributed separately and may thus be inserted in the manual.

Part I of the Utility Program Manual gives a general introduction to the file processor and utility program system and a detailed description of certain important features of the system.

The file processor and the utility programs in system 3 are based on the system 2 versions. The necessary changes in the programs and coding of new programs was done by Tove Ann Aris, Bo Tveden Jørgensen, Jørgen Zachariassen and the author.

The advices and corrections from Christian Gram and Tove Ann Aris have been of great help during the preparation of this manual.

Hans Rischel
A/S Regnecentralen, April 1973

(first edition: June 1972)

Third edition:

This edition is similar to second edition with below exceptions. Following descriptions have been added: assign, changeentry, char, correct, edit, headpunch and setmt, clearmt. Following descriptions have been changed: account, backfile, bossjob, catsort, claim, convert, copy, entry, head, i, load, move, newjob, o, print, save and scope.

Tove Ann Aris
A/S Regnecentralen, September 1974

PREFACE
(RCSL 31-D494)

Fourth edition:

Following descriptions have been added: procsurvey, rubout.
Following have been totally rewritten: save, load. Following have been changed: binout, catsort, changeentry, char, clearmt, copy, convert, correct, edit, entry, head, headpunch, mode, move, replace, set and setmt.

Tove Ann Aris
A/S Regnecentralen, March 1977

Fifth edition:

Following description has been added: label.
Following have been changed: binin, catsort, copy, finis, load, print and save.

Tove Ann Aris
A/S Regnecentralen, April 1978

REFERENCES

Ref. 1 Søren Lauesen and Rune Einersen: Boss 2, Users Manual, RCSL No. 31-D370.

Ref. 2 Per Brinch Hansen: Multiprogramming System, RCSL No. 55-D140

Ref. 3 Tove Ann Aris and Bo Tveden Jørgensen: RC 8000 MONITOR, Part 2, Reference Manual, RCSL No. 31-D477.

Ref. 4 Egon Højer Pedersen: Definition of external processes, RCSL No. 31-D458.

Ref. 5 Hans Dinsen-Hansen: Algol 6, Users Manual, RCSL No. 31-D322

Ref. 6 Jens Hald and Alan Wessel: Fortran, RCSL No. 31-D392

Ref. 7 Per Brinch Hansen: Slang Assembler, RCSL No. 55-D18

Ref. 8 Peter Kraft: Editor I, RCSL No. 55-D22

Ref. 9 Bjørn Ø. Thomsen: New Version of the Editor, RCSL No. 55-D101

Ref.10 Torkild Glaven: Do, an fp utility program, RCSL No. 31-D280

Ref.11 Hans Rischel: Utility Programs, part I, RSCL No. 31-D364

Ref.12 Rune Einersen and Lars Otto Kjær Nielsen: Boss 2, Operators Manual, RCSL No. 31-D461.

Ref.13 Tove Ann Aris and Hans Rischel: Utility Programs, part III, RCSL No. 31-D379.

Catalog Handling Programs

ASSIGN RCSL No. 31-D305
Creates or changes a temporary entry so that the tail of the two specified entries become identical.

BACKFILE RCSL 31-D279
Subtracts one from the file number (unless it is 0) in the tails of the entries specified and signals reach of file 0.

CATSORT RCSL No. 31-D488
Lists on current output selected parts of the main catalog (or any subcatalog) sorted according to the parameters. At last also total number of entries and segments output are listed.

CHANGEENTRY RCSL No. 31-D424
Changes an existing catalog entry according to the parameters in the call. The program is a supplement to the programs SET and ENTRY and is used when one wants to change some of the elements in the entry tail by copying from the tails of other catalog entries.

CLEAR RCSL No. 31-D235
Removes catalog entries with name and scope as specified.

CLEARMT RCSL No. 31-D425
Removes catalog entries according to the parameters.

ENTRY RCSL No. 31-D426
Creates or changes a temporary catalog entry according to the parameters in the call. The program is a supplement to the program SET and is used when one wants to set some of the elements in the entry tail by copying from the tails of other catalog entries.

LOOKUP RCSL No. 31-D427
Finds and lists catalog entries with specified name.

NEXTFILE RCSL No. 31-D238
Adds one to the file number in the tail of the catalog entries specified.

PROCSURVEY RCSL No. 31-D391
Lists types of procedures and their parameters, as well as the procedure date.

RENAME RCSL No. 31-D239
Changes the names of catalog entries as specified.

SCOPE RCSL No. 31-D331
Changes the scope of catalog entries as specified in the call of the program.

SEARCH RCSL No. 31-D241
Finds and lists all catalog entries with a given scope.

SET RCSL No. 31-D428
Creates a new catalog entry with scope temp or changes an already existing entry (with scope temp) according to the parameters.

SETMT RCSL No. 31-D429
Creates catalog entries of scope temp describing files on magnetic tape according to the parameters.

Data Handling Programs

BININ RCSL No. 31-D243
The program can input files generated by the program BINOUT. The programs BININ and BINOUT are primarily used when binary files are stored on paper tape.

BINOUT RCSL No. 31-D487
The program can output catalog entries and contents of files in a format (a binout file) which may be input by the program BININ or the program INITIALIZE CATALOG. The program can furthermore output autoload tapes.

CHAR RCSL No. 31-D431
Outputs the specified character the specified number of times.

COPY RCSL No. 31-D489
Copies one or several files into another file and calculates the number of characters copied and the sum of their ISO values. Blind characters are not copied. The program can be used instead of EDIT if only a simple copying is wanted. Furthermore the program may be used for check reading of text files (e.g. texts punched on paper tape).

CORRECT RCSL No. 31-D433
The program corrects specified words on the backing storage according to the parameters. The program may also be used to print specified bits as integers.

EDIT RCSL No. 31-D434
Edit is a line oriented program for editing of text files.

HEAD RCSL No. 31-D435
Prints a number of form feeds and a page head containing the name of the job and the date.

HEADPUNCH RCSL No. 31-D436
The program punches a readable text pattern according to the parameters. The same information is also written on current output.

LABEL RCSL No. 31-D467
Outputs a boss label on file 0 of the specified magnetic tape.

LOAD RCSL No. 31-D491
The program can input catalog entries and bs-files from a magnetic tape file generated by the program SAVE.

MESSAGE RCSL No. 31-D248
May be used (together with HEAD) to make nice headings on the output. The parameter list in the call of message is simply output when the program is called.

MOVE RCSL No. 31-D438
Performs blockwise copying of files on backing storage or magnetic tape.

PRINT RCSL No. 31-D492
Prints from a backing storage area or directly from the core store with specified formats. The program is primarily intended for printing of dumped core areas.

RUBOUT RCSL 31-D380
Rubs out the contents of the specified backing-storage files. If demanded the catalog entry is removed after the cleaning.

SAVE RCSL No. 31-D493
The program can output catalog entries and bs files to a magnetic tape file for later reestablishment by the program LOAD.

Job Control Programs

ACCOUNT RCSL No. 31-D336
Sends an account message to the parent (the operating system) who is then expected to produce a record in the account file. Only used when jobs running under BOSS wants to produce special account information (cf. the BOSS2 User Manual ch. 10).

BOSSJOB RCSL No. 31-D337
Sends a newjob message to BOSS (the internal process named BOSS) demanding the specified file enrolled as job file in an off line job. In this way a job running under another operating system may create a BOSS job. The actual job continues with the next FP-command. Further details are found in section 1.7, internal jobs, in the BOSS User Manual.

CHANGE RCSL No. 31-D254
Sends a change paper message to the parent (the operating system). The program is only used when a job executed under BOSS uses job controlled printer. (cf. the BOSS2 User Manual, ch. 6.3 and ch. 10).
Output on printer from a job running under BOSS is normally made either by printing on current output or as off-line printing initiated by the FP-command CONVERT.

CLAIM RCSL No. 31-D338
Lists the bs-area claims of the process.

CONVERT RCSL No. 31-D440
Sends a convert message to the parent (the operating system) who is then expected to print the specified backing storage area. A file with scope login is not accepted and the file must not be in use (for instance the file must not be current output). A temporary file converted will immediately disappear from the reach of the job. Each convert operation performed by BOSS requires a cbuffer which must be reserved in the job specification (cf. the BOSS2 User Manual ch. 6).

CORELOCK RCSL. No. 31-D257
Sends a corelock message to the parent (the operating system) demanding that the job should stay in core the specified number of seconds. This feature is only used in connection with process control devices producing data with a high rate, cf. the BOSS2 User Manual ch.9.3.

COREOPEN RCSL No. 31-D258
Sends a coreopen message to the parent (the operating system) signalling the end of a corelock period (cf. the program CORELOCK). The program is only used on process control installations.

END RCSL No. 31-D259
Returns current input to the previous current input at the position where it was left.

FINIS RCSL No. 31-D490
Finis terminates the job.

I RCSL No. 31-D340
Selects a new file as current input. The former file may later be resumed at the position where it was left (for instance by a call of END).

IF RCSL No. 31-D262
Makes the execution of the next FP-command conditioned by the values of one (or several) mode bits. The condition may reflect the success of the latest program executed as the ok and warning bits are set at program end (or it may correspond to the mode bits as set by a call of the program MODE).

KIT RCSL No. 31-D263
Sends a mount disc message to the parent (the operating system) demanding a disc kit with a specified name to be mounted on a specified disc unit (cf. the BOSS2 User Manual ch. 4.3).

MODE RCSL No. 31-D441
Changes the FP mode bits specified in the call and may thereby change the working cycle of FP.

MOUNT RCSL No. 31-D265
Sends a mount message to the parent (the operating system) who is then expected to ask the operator to mount the tape reel (cf. the BOSS2 User Manual ch. 5 and 10). The program does not await the mounting, unless there is asked for mounting of an unspecified worktape.

MOUNTSPEC RCSL No. 31-D266
Sends a mount special message to the parent (the operating system) limiting a later mounting of the specified magnetic tape reel to the station with the specified device number (cf. the BOSS User Manual ch. 5 and 10).

NEWJOB RCSL No. 31-D341
Sends a newjob message to the parent (the operating system) demanding the specified file enrolled as job file in a new off line job i.e. in this way a new job is created. The actual job continues with the next FP-command. Further details are found in sec. 1.7, internal jobs, in the BOSS User Manual.

O RCSL No. 31-D342
Selects a new file as current output.

ONLINE RCSL No. 31-D269
Turns the job into the conversational mode where the current input to the job is typed on the terminal at run time. A conversational job is very resource demanding and the user must have a special option in the user catalog (cf. the BOSS User Manual ch. 3.5).

OPCOMM RCSL No. 31-D270
Sends the parameter list in the call as a print message to the parent (the operating system) with request for an answer from the operator and types the answer (when received) on current output (cf. the BOSS2 User Manual ch. 10).

OPMESS RCSL No. 31-D271
Sends the parameter list in the call as a print message to the parent (the operating system). If the operating system is BOSS the message is typed on the main console (cf. the BOSS2 User Manual ch. 10).

RELEASE RCSL No. 31-D272
Sends a release message to the parent (the operating system) releasing the specified magnetic tape reel (cf. the BOSS User Manual ch. 5 and 10).

REPEAT RCSL No. 31-D273
The program makes it possible to repeat (a specified number of times) a series of FP-commands placed in brackets.

REPLACE RCSL No. 31-D442
Sends a replace message to the parent (the operating system) defining a file as replacement for the current job file. After termination of the job BOSS will create a new job with the same name and the specified file as job file. A replace message from an on-line job is not accepted by BOSS.

RING RCSL No. 31-D275
Sends a mount ring message to the parent (the operating system). The program is normally not used as the software sends the mount ring message automatically when needed.

SKIP RCSL No. 31-D276
Bypasses parts of current input as specified in the parameter list.

SUSPEND RCSL No. 31-D277
Sends a suspend message to the parent (the operating system) asking for suspension of the specified magnetic tape reel. This is relevant for worktapes only. The station is now available for mounting of another tape reel, but the suspended worktape is still reserved for the job until it terminates or releases the tape reel. Each suspend operation uses a suspend buffer. (cf. the BOSS2 User Manual, ch. 5 and 10).

TIMER RCSL No. 31-D278
Sends a timer message to the parent (the operating system) demanding a provoked interupt after a certain time. The use of the program is described in details in the BOSS2 User Manual ch. 10.

ACCOUNT

Sends an account message to the parent (the operating system) who is then expected to produce a record in the account file. Only used when jobs running under BOSS wants to produce special account information (cf. the BOSS2 User Manual, ch. 10).

Call:
account <s> <account kind>{<s> <integer>}$_1^3$
where the parameters <account kind> and <integer> are integers.

Function:
The program sends an account message containing the integers.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***account call
The program was called with a left hand side.

***account <parameter list> parameter error
Parameter error in the call of the program.

***account <parameter list> kind illegal.
The account kind was not accepted by the operating system

In case of any error no account record is produced.

Assign

Creates or changes a temporary entry so that the tail of the two specified entries become identical. The program is used together with the program entry and nextfile.

Example:
The programmer wants to set an entry in the file longname and instead of

```
new=entry longname longname longname longname longname,
          2.6 longname
```

the following commands are used

```
t=assign longname
new=entry t t t t t 2.6
```

The program calls:

```
nextfile tape
...
...
nextfile tape
progfile=assign tape
```

will set progfile equal to the current value of tape.

Call:

```
<resultname> = assign <oldname>
```

Function:
Creates or changes a temporary entry so that the two entry tails becomes identical. Apart from the parameter treatment the program works exactly like entry.

Storage requirements:
1536 bytes plus space for FP

Error messages:

***assign call
No left hand side in the call of the program.

***assign param <parameter>
Parameter error in the call of the program

***assign <oldname> unknown
The file <oldname> was not found in the catalog

***assign <result name> change kind impossible
A change of an area entry to a non-area entry or vice versa was attempted.

***assign <result name> change bs device impossible
A change of kit/doc name of an area entry was attempted.

***assign <result name> bs device unknown
The bs device specified was not found.

***assign <result name> no resources
The resources of the job did not allow the wanted creation or change of an entry.

***assign <result name> entry in use
The entry could not be changed because another job was using it.

If any message appears no entry is created or changed.

BACKFILE

Substracts one from the file number (unless it is 0) in the tails of the entries specified and signals reach of file 0.

Examples:

If the catalog entries old and new describe file 4 of magtape mt310514 and file 2 of mt310515 respectively, then the command

```
backfile old new
```

will change old to describe file 3 of mt310514 and new to describe file 1 of mt310515. A repeated call will change old to describe file 2 and new to describe file 0 and set the warning bit to yes. A following call will change old to describe file 1 and leave new unchanged - the ok bit is set to no.

Call:

backfile $\left\{ \langle s \rangle \langle name \rangle \right\}_{1}^{\infty}$

Function:

For each name in the list a catalog lookup is made and the file number in the tail of the entry is decreased by one unless it is zero.
If any file number becomes zero then the warning bit is set to yes.
If any file number already was zero then the ok bit is set to no.

Storage requirements:

Space for FP

Error messages:

***backfile call
Left hand side in the call. Program terminates without further actions.

***backfile <name> param
Parameter error. The faulty parameter starting with the name specified is skipped and the program continues with the next parameter.

***backfile param
Same as above except that the faulty parameter does not start with a name.

***backfile <name> unknown
No entry with the specified name was found. The program continues with the next parameter.

***backfile <name> protected
The job was not allowed to change the tail in the entry found. The program continues with the next parameter.

If any error message occurs then the ok bit is set to no.

BININ

The program can input files generated by the program BINOUT. The programs BININ and BINOUT are primarily used when binary files are stored on paper tape.

Example:
A paper tape was produced by the FP-command:

```
tpo=binout fup
```

It may be loaded by the FP-command

```
binin tro
```

and thereby the catalog entry 'fup', the area and its contents are reestablished. When using BOSS one should load the tape by a load command in the job specification like this

```
load trn pip
```

and then get the file 'fup' by the following call of binin in the job file:

```
binin pip
```

Note, that the no parity mode is used in the load command.

Call:

$$\left\{\text{<other output>} = \right\}_0^1 \quad \text{binin} \left\{\text{list.} \left\{\begin{matrix}\text{yes}\\ \text{no}\end{matrix}\right\}\right\}_0^1 \left\{\text{<binout file>} \left\{\text{.<modifier>}\right\}_0^1\right\}_1^\infty$$

<other output> ::= <name of output file>

<binout file> ::= <name of input file>

$$\text{<modifier>} ::= \left\{\begin{matrix}\text{<binout segments>}\\ \text{s.<binout segments>}\\ \text{c.<binout segments>}\end{matrix}\right\}$$

$$\text{<binout segments>} ::= \left\{\begin{matrix}\text{<no of binout segments>}\\ \text{<no of binout segments> . <first binout segment>}\end{matrix}\right\}$$

The elements <no of binout segments> and <first binout segment> are integers.

Function:

If the parameter list.yes is specified, all entry names found are listed on current out.
The input to BININ is a number of binout files, each consisting of a number of binout segments. A binout segment is a stream of 8-bit characters with odd parity, the second bit of each character being 0. A binout segment is terminated by a sum character, a character with the second bit being 1. A binout segment input by BININ is transformed to a number of words, each composed of the rightmost 6 bits of 4 characters. The rightmost 6 bits of the sum character form the sum modulo 64 of all other characters in the binout segment; this sum is checked by BININ.
BININ scans the parameter list from left to right, and loads the sequence of binout segments defined by the binout files. When a file is exhausted, the input is continued from the file described by the next element in the parameter list, and when it is exhausted, the execution of BININ is terminated.
The left side in the call of BININ determines how the binout segments are interpreted:

1) <other output> is not present.
   The very first binout segment is input and interpreted as a command segment. The commands in the command segment are executed one by one, and when the command segment is exhausted, the next binout segment is input and interpreted as a command segment. If a command segment includes a load command, a number of binout segments following the present command segment is input and moved to backing store or magnetic tape as defined by the load command. The following segment is interpreted as a command segment and so on. A tape produced by BINOUT may be read in this way.
2) <other output> is present.
   All binout segments of the binout files are interpreted as load segments and loaded to the file described by <other output>.

A command segment must not exceed 256 words; a load segment can be of any length.

Modifier

<binout segments> This modifier has only effect if other output is specified (left side in call occurs). The first <first segment> binout segments of the actual in-file are skipped, and only <no of binout segments> binout segments are loaded to the output file. If <first segment> does not occur, no segments are skipped.

.s The modifier causes each load segment to be preceeded by one word in the output. This word is an integer which is the length of the entire segment (no of bytes). The last segment is terminated by a word being 0.

.c This modifier causes the binout file to be checked only; i.e. the commands in the command segments are checked for syntax errors, and only the <:end:> command is executed. The sums of all binout segments are checked, but no load segments are output to the files specified.

Commands:
BININ uses the same command language as the program INITIALIZE CATALOG (cf. ref. 2).
A command in a command segment is identified by a textstring consisting of at most 6 ISO characters (including NULL characters). This textstring may be followed by a fixed number of parameters. Parameters can be catalog entry names and words. A name is a textstring of 12 ISO characters beginning with a small letter followed by a maximum of 10 small letters or digits terminated by NULL characters. The possible commands are:

<:newcat:> has no effect
<:oldcat:> has no effect.
<:end:> terminates BININ.

<:create:>,<name>,<entry tail of 10 words>
Creates a temporary catalog entry with the name and contents as specified. If the first word of <entry tail> is positive, an area of that size is reserved on the backing store. If the entry already exists, it is first removed.

<:change:>,<name>,<entry tail of 10 words>
Changes an existing catalog entry with a given name as specified. If the entry describes an area on the backing store, the number of segments is reduced to the value specified by the first word of entry tail.

<:rename:>,<name>,<newname>
The catalog entry, <name>, is renamed to <newname>.

<:remove:>,<name>
Removes the catalog entry specified; if the entry describes a backing store area, this is removed too.

<:perman:>,<name>,<catalog key>
Makes the catalog entry specified permanent with the catalog key <catalog key>. If <catalog key> equals 3, then the entry base is changed to the user base i.e. the entry becomes user scope.

<:load:>,<name>,<no of binout segments>
Loads a number of binout segments following the present command-segment to the file described by <name>. On magnetic tape each binout segment is output as one block. On backing store the boundaries of backing store segments are ignored. The sum characters are not transferred to the output file.

Storage requirements:
The core storage requirement for BININ is approx. 4096 bytes plus the space needed by FP.

Error messages:
***binin param <erroneous parameters>
Parameter error in call of BININ. The program proceeds, ignoring the erroneous parameters.

***binin <binout file> exhausted
The last character of <binout file> is not a sum character, when <binout file> is the last input file.

***binin input name missing
The parameter list does not include a <binout file> or <end of parameter list> is found before a normal termination of BININ.

***binin <binout file> input impossible
<binout file> is unknown or the input process can not be initialized.

***binin <output file> output impossible
<output file> can not be reserved or is unknown.

***binin <binout file> core size
No core space for buffers etc.

***binin <binout file> sizeerror
A command segment from <binout file> occupies more than 256 words in core store.

***binin <binout file> sumerror in command segment

***binin <binout file> sumerror in load <output file>

***binin <text string> syntaxerror
The <textstring> is not recognized as a command.

***binin <binout file> create <name> result <result>
Create entry, result <> 0 (monitor function).

***binin <binout file> remove <name> result <result>
Remove entry, result <> 0 (monitor function).

***binin <binout file> change <name> result <result>
Change entry, result <> 0 (monitor function).

***binin <binout file> rename <name> result <result>
Rename entry, result <> 0 (monitor function).

***binin <binout file> perman <name> result <result>
Permanent entry, result <> 0 (monitor function).

If an error is detected BININ continues with the next parameter in the list.

Further examples:

binin tro tro
inputs two paper tapes; command segments are required in the input.
The tapes may f.inst. be produced by the FP-commands:
tpo=binout fpnames.p move.b
tpo=binout algolprog

binin tro.c
The binout paper tape is checked, but no catalog functions are called, and no output is produced.

copyarea=binin tro.s
tpo=binout copyarea.ne.a
In this way it is possible to copy binout tapes. Another copy is made by a new call of BINOUT, without reading the tape again.

code3=binin bincode1.2 bincode2.1.2 bincode1.4.3
Loads segments 1,2 from bincode1, segment 3 from bincode2 and segments 4,5,6,7 from bincode1 thus merging two binouts of slang programs into code 3.
Possible command segments are regarded as load segments, because <other output> is specified.
The areas bincode1 and bincode2 may e.g. be produced by the FP-commands:
bincode1=binout code1.s.ne
bincode2=binout code2.s.ne

BINOUT

The program can output catalog entries and contents of files in a format (a binout file) which may be input by the program BININ or the program INITIALIZE CATALOG (cf. ref. 3). The program can furthermore output autoload tapes.

Example:
The program file named 'fup' is output on paper tape by the FP-command
tpo=binout fup
(compare with the example under the program BININ).

Call:
<out file> = binout $\{$ <s> <input description>$\}_1^\infty$

<input description> ::= <name> $\left\{ \begin{array}{l} .p \\ .b \ \{.<bytes>\}_0^1 \\ .s \ \ .<field> \\ .a \ \ .<field> \\ .np \\ .ne \end{array} \right\}_0^\infty$

<bytes> ::= <no of bytes>

<field> ::= $\left\{ \begin{array}{l} <no\ of\ blocks> \\ <no\ of\ blocks> \ . \ <first\ block> \end{array} \right\}$

The elements <no of bytes>, <no of blocks> and <first block> are integers. The elements .p, .b.<bytes> , .s.<field> , .a.<field> , .np, and .ne are in the following called modifiers.

Function:
The output from BINOUT is a binout file consisting of binout segments.

The binout file is a stream of 8-bit characters on magnetic tape, in a backing store area, or on paper tape. Each binout segment is terminated by a special character, called the sum character.
Normally each <input description> causes the output of a number of binout segments. The first of these consists of the catalog entry defined by <name>, and determines the number of the remaining binout segments. This first binout segment is called a command segment. If the <input description> defines a program file, the command segment is followed by a number of binout segments, being the contents of this file. The latter segments are called load segments.
Depending on the modifiers of the input description, either the command segment or the load segments may be omitted, and it is also possible to output text files as load segments. The output from BINOUT is normally used as follows:

1) As input to BININ.
2) As input to INITIALIZE CATALOG, as described in ref. 2, chapter 14-15. In this case, the output from BINOUT must be a paper tape or a magnetic tape file, including the command segments.
3) As an AUTOLOAD PROGRAM. The output must then be a paper tape without command segments.

Binout file:
The output file is defined by:

<out file>, which must be the name of a catalog entry describing a paper tape punch, a backing store area, or a file on magnetic tape. If the output file is paper tape, BINOUT will select the output mode to odd parity, independent of the mode defined by the file descriptor.

Input description:
The <input description> is a name, which may be followed by a set of modifiers; it defines the binout segments to be output:

<name> is the name of an arbitrary catalog entry. If the <input description> consists of the name only, the corresponding catalog entry determines the format of the output:
The command segment is output but load segments are only output, if <name> describes a file containing a program. A file on magnetic tape, and a backing store area, which is organized as logical blocks, is output as a number of load segments, each load segment being a block of the file.
Other program files are output as a single load segment.

The format of the output may also be chosen explicitely, by means of the modifiers. The effects of these modifiers are as follows:

p Intended for output of text files. The <name> must describe a file on magnetic tape or a backing store area. The contents of this file is output as a single load segment.

b.<bytes> Intended for output of slang programs. Has the same effect as p, except that only the first <bytes> bytes of the actual file are output. If <bytes> is not present, the last word of the filedescriptor associated with <name> determines the number of bytes. This number may be set by SLANG, just after translating a program.

s.<field> Intended for output of SLANG programs fulfilling below requirements. The <name> must describe a file on magnetic tape or a backing store area, which is assumed to be organized as logical blocks (i.e. the first word of each block defines the length of the entire block; a block with a non-positive length terminates the area.). The contents of the file is output as <no of blocks> load segments, and if <first block> is present, the first <first block> blocks of the file are skipped. In this case the modifier .ne is normally used too. If the <field> specification is empty, all blocks of the file are output.

a.<field> Intended for output of autoload programs.
Has the same effect as s.<field>, except that the first word of each block is not output.

np No program, i.e., no load segments are output.
Normally not used.

ne No entry, i.e., the command segment is not output.
Used for instance for output of files which may later be loaded to defined areas (fuss=binin tro).

Note, that in a sequence of modifiers, only the latest of the modifiers:
p, b.<bytes>, s.<field>, a.<field>, and np
has effect; e.g. the <input description>:
jza.s.ne.a.1.3.p
has the same effect as the <input description>:
jza.ne.p

Binout segment:
A binout segment is a stream of 8-bit characters with odd parity, the left-most bit of each character being the parity bit. The last character in the segment is a sumcharacter, which is characterized by the second bit being one. The right-most 6 bits of this character form the sum modulo 64 of all other characters in the segment.
Each byte of the input is output as two characters. The second bit of these is always 0, whereas the right-most 6 bits are a copy of the corresponding 6-bit group of the byte.

Command segment:
The contents of a command segment is a number of commands, sufficient to create a catalog entry and load the load segments in a later call of BININ or INITIALIZE CATALOG. The command segment, as output by BINOUT, consists of at most 3 commands, which are the output of the following words:

```
    <:create:>                  ; 2 words, text string
    <name of entry>             ; 4 words, text string
    <entry tail>                ; 10 words
    <:perman:>                  ; 2 words, text string
    <name of entry>             ; 4 words, text string
    <catalog key>               ; 1 word, integer
    <:load:>                    ; 2 words, text string
    <name of entry>             ; 4 words, text string
    <no of load segments>       ; 1 word, integer
```

The <:perman:> command is omitted if the catalog entry has catalog key 0; and the <:load:> command is only included if load segments are output.

Storage requirements:
The core storage requirement for BINOUT is approx. 3072 bytes plus the space needed for FP.

Error messages:

***binout <name> output impossible
No left side in the call, or the output device defined by <name> is reserved or does not exist, or <name> does not describe a binary file.

***binout <name> <list of erroneous parameters>
Parameter error in call of BINOUT. If the parameters are part of an input description, this is ignored.

***binout input name missing
End of parameter list is found before an expected input description.

***binout <name> unknown
<name> is not name of a catalog entry.

***binout <name> input impossible
<name> describes an input device from which input is not possible, or <name> is unknown.

***binout core size
The core store space needed for buffers etc. is too small.

***binout <name> prog or entry
The input description demands output of load segments in spite of that <name> does not describe a file, or the input description causes no output.

***binout <name> segments <integer>
The input description demands more output than possible; only <integer> load segments from the file described by <name> are output.

If an error is detected BINOUT continues with the next parameter in the list.

Examples on the use of the modifiers:
The contents of the areas textarea and codearea containing a text and a program file respectively (for instance produced by EDIT and SLANG) are output on a paper tape by the FP-command

```
tpo=binout textarea.p codearea.b
```

Only the part of codearea which contains code is output.
The tape may be input later by the FP-command:

```
binin tro
```

A binary paper tape may be copied by the FP-commands

```
copyarea=binin tro.s
tpo=binout copyarea.ne.a
```

(cf. the description of BININ).

The ALGOL compiler may be moved to magnetic tape - say mt471100, file 1 (this may be useful if the backing storage is very small). If ALGOL is present on the backing storage, this is done by the FP-commands:

```
tapealgol=entry mto mt471100 0 1 0 algol algol
auxarea=binout algol.ne.s.12      ; as algol has 12 logical segments
tapealgol=binin auxarea
```

Now the areas algol and auxarea may be cleared and tapealgol renamed to algol and permanented in the catalog. (The tape reel may now be dismounted and will be requested whenever ALGOL is called.)
The ALGOL STANDARD PROCEDURES are of course not moved.

BOSSJOB

Sends a newjob message to BOSS (the internal process named BOSS) demanding the specified file enrolled as job file in an off line job. In this way a job running under another operating system may create a BOSS job. The actual job continues with the next FP-command. Further details are found in section 1.7, internal jobs in the BOSS User Manual.

Call:
bossjob <s> <file name> {<name of remote batch printer>}$_0^1$
where <file name> is a name of a permanent job file.
<name of remote batch printer>::=<name of max 6 char>

Function:
A newjob message containing the specified name(s) is sent to BOSS.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***bossjob call
Left hand side in the call of the program.
***bossjob <parameter list> parameter error
Parameter error in the call of the program.
***bossjob <filename> <error cause>
Error during creation of the new job. The cause may be any of the following:
job queue full
job file not permanent
job file unknown
job file unreadable
user index too large
illegal identification
user index conflict
job file too long
temp claim exceeded
option unknown
param error at job
syntax error at job
line too long
attention status at remote batch terminal
device unknown
device not printer
parent device disconnected
remote batch malfunction
In case of any error no new job is created.

CATSORT

Lists on current output selected parts of the main catalog (or any subcatalog) sorted according to the parameters. At last also total number of entries and segments output are listed.

Example:
The FP-command:

```
catsort base.project.min
```

will output all files with a base contained in the project base, i.e. belonging to the actual project. The parameter min causes that only name, segments docname, date and scope is output.
The FP-command:

```
catsort
```

will output all non-system entries in the main catalog sorted according to base and entry name.
See also Further Examples.

Call:

```
{<outfile>=}(1,0) catsort {<catalog spec>
                           <limit spec>
                           <sorting spec>}(inf,0)

                    maincat. {yes}
                             {no}
<catalog spec> ::=
                    subcat.  {yes
                              no
                              <integer>}

                             {system. {only}
                                      {yes}
                                      {no}  }
                             {name.<entry name>
<limit spec> ::=             {docname.<document name>
                             {base.   {<scope> {.min}  }
                                      {<baselow>.<baseup>}}

                             {basesort}
<sorting spec> ::=           {docsort }  .{yes}
                             {slicesort}  {   }
                             {nosort  }   {no }

                             {project}
<scope> ::=                  {user   }
                             {login  }
                             {temp   }

{<baselow>}
{<baseup> } ::= <integer>
```

Format of output:
Each entry is output on one line in the form:
<entryname> <first slice> <name key> <catalog key> <lower entry base>
<upper entry base> <mode.kind or segments> <kit/doc name>
<remaining entry tail>
If the parameter min is specified the output is:
<entryname> <segments> <kit/docname>

Function:
If an outfile is specified, this file is used for output, otherwise current output file is used.
The catalogs are one by one copied into a working bs file, which is sorted according to the parameters.
The sorting parameters are:

basesort.yes — meaning sorted after the entry base (which means grouped after project and users).

docsort.yes — meaning that each area entry is followed by all sub-entries (which have a kit/document name equal to the entry name of the main entry).

slicesort.yes — meaning sorted according to first slice. This parameter will cancel the parameter docsort.yes and has the same priority.

The priority of the sorting parameters are basesort, docsort. The last sorting criterion will always be alfabetic sorting on entry name.

nosort.yes — meaning that no sorting at all is performed. The total catalog will be output, neglegting all other parameters but maincat and subcat.

Other parameters:

maincat — defining whether the maincatalog is output

subcat — defining whether the subcatalogs are output (if any), an integer specifies a subcatalog to be output (0 corresponds to maincat).

system — defining whether the system files are output.

name.<entry name>: only entries with the name <entry name> are output. Only 1 name parameter is allowed.

docname.<document name>: only entries containing the kit/doc name <document name> are output. Only 1 docname parameter is allowed.

base.<scope>
base.<baselow>.<baseup>: only entries contained in the specified base are output. A negative value of <baselow> or <baseup> must be given as the positive complement e.g. the integer -1000 is specified as 16777216-1000=16776216.

The parameters are initialized as follows:

maincat.yes
subcat.no
system.no
basesort.yes
docsort.no
slicesort.no

Error messages:
***catsort error param <erroneous and following parameters>
Parameter error in the call.
***catsort, create sortarea impossible
It was impossible to create an area for sorting.

In case of any error message, the program terminates.

Further examples:

catsort nosort.yes

will output the total main catalog in unsorted form.

catsort maincat.no subcat.yes system.yes

will output all entries in the subcatalogs sorted according to base and entry name.

catsort name.pip docname.pap basesort.no

will output all non-system entries in the main catalog with entry name pip or document name pap, sorted according to entry name.

catsort docname.disc system.yes

will output all entries in the main catalog with document name disc, sorted according to base and entry name.

CHANGE

Sends a change paper message to the parent (the operating system). The program is only used when a job executed under BOSS uses job controlled printer. (cf. the BOSS2 User Manual, ch. 6.3 and ch. 10).
Output on printer from a job running under BOSS is normally made either by printing on current output or as off-line printing initiated by the FP-command CONVERT.

Call:

change <s> <device name> <s> <paper type>
where the parameter <paper type> is an integer.

Function:

A change message containing the specified device name and paper type is send to the parent who is then expected to perform the necessary actions (message to the operator etc.)

Storage Requirements:

1536 bytes plus space for FP.

Error Messages:

***change call
The program was called with a left hand side.

***change <parameter list> parameter error
Parameter error in the call of the program.

***change <parameter list> <error cause>
The change message was not accepted by BOSS for one of the following causes:
1. no buffers
2. job printer not allowed (cf. the BOSS2 User Manual).

In case of any error the change action is not performed by BOSS.

CHANGEENTRY

Changes an existing catalog entry according to the parameters in the call. The program is a supplement to the programs SET and ENTRY and is used when one wants to change some of the elements in the entry tail by copying from the tails of other catalog entries.

Example:
Suppose that the catalog entry named 'source' contains the name of a magtape reel in the document name field. By the FP-commands

```
filex=changeentry filex source filex filex filex filex filex
```

the entry filex is changed to contain the name of the tape reel.

A catalog entry named 'source' containing the name - say mt471100 - may be created by a call of SET:

```
source = set mto mt471100
```

Call:

```
<result name> = changeentry {<s> <kind> {<s> <kit/doc name>
                            {<s> <free param> {<s> <file> {<s> <block>
                            {<s> <contry> {<s> <length>}o}o}o}o}o}o}o}o
```

```
{ <kind>          }
{ <kit/doc name>  }   ::=  { <integer>                  }
                           { <integer1> . <integer2>    }
                           { <name>                     }

<free param>          ::=  { <word>                     }
                           { <byte1>.<byte2>            }
                           { d.<isodate>.<clock>        }

<isodate>             ::=  { <yymmdd>                   }
                           { 0                          }   0 is interpreted as now

<clock>               ::=  <hhmm>                          may be omitted in case no
                                                           entry named d exists

{ <file>          }
{ <block>         }   ::=  { <word>                     }
{ <contry>        }        { <byte1> . <byte2>          }
{ <length>        }

{ <word>          }
{ <byte1>         }   ::=  { <integer>                  }
{ <byte2>         }        { <name>                     }
```

Function:
The left hand side is looked up. If it does not exist, the program terminates. Otherwise the parameters are interpreted as described below yielding the wanted entry tail. From this point the program continues exactly as the program SET.

Parameters
Kind:
<integer>: The value is placed in the tail.
<integer1> . <integer2>: The value <integer1> shift 12 + <integer2> is placed in the tail.
<name>: First the name is searched for in the table of mode-kind abbrevations and if found here the value found is used. If not found in the mode-kind table (see Utility Programs, part 1, Appendix) it is searched for in the catalog and the kind of the entry found is used.
Kit/doc name:
<integer>: The value is placed in the tail.
<integer1> . <integer2>: The value <integer1> shift 12 + <integer2> is placed in the tail.
<name>: If the kind just found is the mode-kind bs (2048 shift 12 + 4) the name itself is used in the tail. For all other kinds the name is looked up in the catalog and the kit/doc name in the tail of the entry found is used.
The other parameters:
A parameter of the form <byte1> . <byte2> gives separate specifications of the two 12-bit bytes in the word.
<integer>: The value is placed in the tail as the word or byte in question
<name>: The name is looked up in the catalog and the value of the word or byte in question in the entry tail found is used.

If the parameter list does not specify all of the tail, the rest of the tail is set to zero.

Storage requirements:
1536 bytes plus space for FP

Error messages:
***changeentry call
No left side in call of the program
***changeentry param <parameter>
Parameter error in call of the program
***changeentry <name> unknown
Lefthand side or a parameter was searched in the catalog but not found.

***changeentry <result name> change kind impossible
A change of an area entry to a non-area entry or vice versa was attempted.
***changeentry <result name> change bs device impossible
A change of kit/doc name of an area entry was attempted.
***changeentry <result name> bs device unknown
The bs device specified was not found.
***changeentry <result name> no resources
The resources of the job did not allow the wanted creation or change of an entry.
***changeentry <result name> entry in use
The entry could not be changed because another job was using it.

If any message appears no entry is changed.

CHAR

Outputs the specified character the specified number of times.

Example:
The current output is divided in groups by the call
```
char nl.8
```
which produces 8 newlines on current output.
```
char ff nl
```
produces a top of form and a newline on current output.

Call:

$$\left\{\text{<outfile> =}\right\}_{0}^{1} \quad \text{char} \quad \left\{\begin{array}{l}\text{<iso value>}\\ \text{<iso value>.<repeat factor>}\end{array}\right\}_{0}^{\infty}$$

<iso-value> ::= <integer>|nl|ff|em|sp
<repeat factor> ::= <integer>

Function:
If no repeat factor is specified the character will be output one time else the character will be output as many times as specified by repeat factor.

The repeat factor may be changed by the program, e.g. ff.19 will be changed to ff.6 and nl.100 will be changed to nl.64. Other characters will be repeated max. 133 times.

If an outfile is specified this is used for the output else current output is used.

Storage requirements:
1024 bytes plus space for FP.

Error messages:
***char param <parameter>
Parameter error in the call. The program continues in the parameter list.

CLAIM

Lists some claims of the process.

Examples:
In an installation with two bs-devices, named drum and disc, the call:

```
    claim
```

may print:

```
area 6   buf 4   size 16384   first core 25492

drum: 1 segm/slice
  temp       0 segm   19 entr
  login      0 segm    0 entr
  perm       0 segm    0 entr

disc: 36 segm/slice
  temp     900 segm
  login    432 segm    4 entr
  perm       0 segm    0 entr
```

The call:

```
  claim perm.disc temp
```

will print:

```
area 6   buf 4   size 16384   first core 25492

disc: 36 segm/slice
  perm       0 segm    0 entr

drum: 1 segm/slice
  temp       0 segm   19 entr

disc: 36 segm/slice
  temp     900 segm
```

Call:

$$\left\{\text{<output file=>}\right\}_{0}^{1}\ \text{claim}\ \left\{\begin{array}{l}\text{<scope>}\\ \text{<docname>}\\ \text{<scope>.<docname>}\end{array}\right\}_{0}^{\infty}$$

<docname> ::= <name of drum or disc kit>

$$\text{<scope> ::=}\ \left\{\begin{array}{l}\text{temp}\\ \text{login}\\ \text{perm}\\ \text{key}\end{array}\right\}$$

Function:
The program scans the parameter list. For each parameter group, the internal tables in the monitor are scanned. If a document name is specified in the parameter group, the resources of catalog entries and segments for each permanent key on that device are listed, else the resources on all bs-devices are listed. If <scope> is specified, the listing of resources is restricted to the specified scopes.

perm is equal to scope user + project.

If <scope> is specified to key, the permanent keys will be output instead of the scope names.

Note that temp entries are only output for the main catalog, since all temporary entries are counted only here, cf. ref. 2 and 3.

If claim is called in the beginning of a job, the value of area is already reduced by 1, which is the one used by FP.

An empty parameter list means: all bs-devices, all scopes.
If there is a left side in the call of claim, the output will appear on <output file> otherwise on current output.

Storage Requirements:
900 bytes plus space for FP.

Error messages:
***claim connect <output file>
The specified output file could not be connected. Current output is chosen as output.

***claim param <list of erroneous parameters>
Parameter error in call of claim.

***claim <docname> unknown
A bs-device named <docname> does not exist.

CLEAR

Removes catalog entries with name and scope as specified.

Example
By the FP-command
clear user text4
the catalog entry (if any) with scope user and name text4 is removed from the catalog. A catalog entry with the same name but another scope is not affected.

Call:
clear <s> <scope spec> $\{$<s> <name>$\}_1^\infty$

<scope spec> ::= <scope> $\{$.<device name>$\}_0^1$

<scope> ::= { temp | login | user | project }

<device name> ::= <name of drum or disc kit>

Function:
The scope specification is interpreted and then the name list is scanned. For each name in the list the name is searched in the catalog. If an entry with the specified name and scope is found, it is removed from the catalog.

Scope specification:
The concept of scope of a catalog entry is explained in the BOSS2 User Manual ch. 4.1. A device name means a further restriction to entries which are either

(a) area entries, where the data area is placed on the specified bs device

or (b) non-area entries, which are present in the auxiliary catalog on the device cf. ref. 3.

Storage requirements:
2048 bytes plus space for FP.

Error messages:
***clear call
The program was called with a left hand side. No entries removed.
***clear <scope spec> illegal scope
The scope specification was illegal. No entries removed.
***clear <scope spec> bs device unknown
The specified device was not on the computer. No entries removed.
***clear param <parameter>
Illegal parameter. The rest of the parameter list is skipped.
***clear <scope spec> <name> unknown
The entry to be removed was not found. The program continues with the next name in the parameter list.
***clear <scope spec> <name> entry in use
The entry could not be removed because another job was using it. The program continues with the next name in the parameter list.

CLEARMT

Removes catalog entries according to the parameters

Example:
The FP-command:
```
pap=clearmt mt004711.3
```
will remove the entries pap1 pap2 pap3.
The FP-command:
```
f=clearmt f.3.5
```
will remove the entries f3 f4 f5.

Call:
<result name> = clearmt <mtname>.{ <upper integer> | <lower integer>.<upper integer> }
The <mtname> is not used during interpretation of the parameters.
If no <lower integer> is specified, it is set to 1.

Function:
Entry names <resultname> followed by <lower integer> to <upper integer> are removed.

Storage Requirements:
512 bytes plus space for FP

Error Messages:
*** clearmt call
No left hand side or left hand side of more than 9 characters

*** clearmt param
Parameter error in the call, e.g. <integer> greater than 99.

*** clearmt <resultname> catalog error
Error in catalog, monitor or hardware

In case of above error messages the program terminates

*** clearmt <resultname> unknown
The specified entry was not found. The program continues

CONVERT

Sends a convert message to the parent (the operating system) who is then expected to print the specified backing storage area. A file with scope login is not accepted and the file must not be in use (for instance the file must not be current output). A temporary file converted will immediately disappear from the reach of the job. Each convert operation performed by BOSS requires a cbuffer which must be reserved in the job specification (cf. the BOSS2 User Manual ch. 6).

Example:
A program has produced a text file in the area out1 . It is printed by the FP-command

```
convert out1
```

Call:

convert <s> <name> $\{$ <name of remote batch printer>$\}_0^1\{$<s> <integer>$\}_0^1$
<name of remote batch printer> ::= <name of max. 6 char>

Function:
The convert message with the specified name(s) and integer (or zero if no integer is specified) is sent to the parent.

Paper Types:

0	Standard paper, i.e. monitor format, one copy. A page is 64 lines of 133 positions.
1	A4 upright, one copy. A page is 64 lines of 72 positions.
2	A4 across, one copy. A page is 42 lines of 112 positions.
3	Monitor, two copies.
4	A4 upright, two copies
5	A4 across, two copies.
6	Monitor, three copies.
7	A4 upright, three copies.
8	A4 across, three copies.
9-99	for extensions.
100-999	special forms. Requires agreement with the operator.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:

***convert call
Left hand side in call of the program

***convert <parameter list> parameter error
Parameter error in call of the program.

***convert <parameter list> <error cause>
The convert message was not accepted by BOSS for one of the following causes:

1. no cbuffers
2. file does not exist
3. file has login scope
4. no resources
5. file in use
6. file is not area
7. attention status at remote batch terminal
8. device unknown
9. device not printer
10. parent device disconnected
11. remote batch malfunction
12. not textfile

In case of any error the convert operation is not performed by BOSS.

COPY

Copies one or several text files into another file and calculates the number of characters copied and the sum of their ISO values. Blind characters are not copied. The program can be used instead of EDIT if only a simple copying is wanted. Furthermore the program may be used for check reading of text files (e.g. texts punched on paper tape).

Example:
The text files 'text1' 'text2' are output as one paper tape file by the FP-command

```
tpe=copy text1 text2
```

and the number and the sum of the characters are printed on current output. One may then check the tape by reading it in a later job by the FP-command

```
copy tre
```

Under BOSS the tape should be input by a load command

```
load tre pip
```

in the job specification. The check reading is then performed in the job file by the FP-command

```
copy pip
```

Call:

{ <outfile>= }$_0^1$ copy { list.{ yes / no } }$_0^1$ { <infile> / <lines> / { <infile>. }$_0^1$ <iso value>.<appearances> / message.yes / message.no }$_1^\infty$

<infile> ::= <name>
<lines> ::= <integer>

iso value> ::= { <small letter> / <integer> }

<appearances> ::= <integer>

Function:
If the parameter list.yes is specified, the input is listed on current out. The program interpretes one parameter at a time as follows:

<infile> The file is copied on <outfile> if any. If no <outfile> is specified only the calculation of number and sum of characters is performed.

<lines> This number of visible lines are copied from current input on <outfile> if any.

<iso value>.<appearances> and
<infile>.<iso value>.<appearances>
The program copies from <infile> if specified, else from current input on <outfile> if any.
Copying stops when the specified number of appearances of the iso character are met. The last character is not output.

message.yes or message.no
Determines whether the following should be output on current output (standard: message.yes)
1. after each param:
   <infile> segm. <number of segments>
   number of characters < 128
   sum of characters
   number of characters >= 128 (if any)
2. at program end (only if the call contains an <outfile> and more than one param):
   <outfile> segm. <number of segments>
   total number of characters < 128
   total sum of characters
   total number of characters >= 128 (if any).

Storage Requirements:
1536 bytes plus space for FP.

Error Message:
All errors cause the warning bit to be set.

***copy connect <outfile> <cause>
The output file cannot be connected for output. The ok bit is set to no and the program is terminated.
<cause> may be:
1. no resources
2. not found
3. In use
   maybe file is the job file
4. convention error
   output attempted on input device or vice versa
5. error
   catalog, monitor or hardware error

***copy connect <infile> <cause>
An input file cannot be connected for input. The parameter is ignored.

***copy param <illegal parameter>
Illegal parameter syntax. The parameter is ignored.

***copy end medium
Current input is exhausted because the parameter <lines> or <iso value>.<appearances> demands reading past EM. The program continues with the next parameter.

***copy no core
The call is not executed because the process is too small.

CORELOCK

Sends a corelock message to the parent (the operating system) demanding that the job should stay in core the specified number of seconds. This feature is only used in connection with process control devices producing data with a high rate, cf. the BOSS2 User Manual ch. 9.3.

Example:
The FP-command:

```
    corelock 5
```

demands corelock for a period of 5 seconds.

Call:

```
    corelock <s> <seconds>
```

where <seconds> is an integer.

Storage requirements:
1536 bytes plus room for FP.

Error Messages:

```
***corelock call
    Left hand side in the call of the program
***corelock <parameter list> parameter error
    Parameter error in the call of the program.
```

In case of any error no corelock message is sent.

COREOPEN

Sends a coreopen message to the parent (the operating system) signalling the end of a corelock period (cf. the program CORELOCK). The program is only used on process control installations.

Example:
The program is called without parameters:

```
coreopen
```

Call:

```
coreopen
```

Storage Requirements:
1536 bytes plus room for FP.

CORRECT

The program corrects specified words on the backing storage according to the parameters. The program may also be used to print specified bits as integers.

Example:
The FP-call:

```
correct bsfile.4 addr.0 bits.0.11 if 700 then neg.456,
   bit.12.23 if neg.1234 then 4000,
   adr.8 if 0 then 1
```

will make the following corrections on segment 4 of bsfile:

byte	0	is changed to	-456	(in case it is	700)
-	1	-	4000	-	-1234
-	8:9	-	1		0

No corrections are made if <oldvalue> is not correct in all cases.

Call:

correct <bsfile>.<segmno>

{address.<addr> {<bitspec> if <oldvalue> then <newvalue>}$_1^\infty$}$_0^\infty$

<bitspec> ::= {empty (identical to bits.0.23)
bits.<firstbit>.<lastbit>}

{<oldvalue>
<newvalue>} ::= {<integer>
negative.<integer>}

{<segmno>
<addr>
<firstbit>
<lastbit>} ::= <integer>

Function:

Segment number <segmno> is input and for each address it is tested whether the specified <oldvalue> is found, in which case it is replaced by <newvalue>. If no errors are found the segment is output.

Note that the file will be connected in the standard way for utilities, i.e. segmno is calculated as segmno + block count cf. Utility Programs, Part I, section 5.5.

During syntax check only the first 3 letters in the words: address, bits, then, negative are tested. adr is accepted for address.

Odd addresses are reduced by 1.

<segmno> and <addr> are counted from 0.

Shortlock in catalog entry is updated. In case <bsfile> describes an external procedure, the internal date is updated.

Storage requirements:

726 bytes plus space for FP.

Error messages:

***correct call
Left hand side in the call.

***correct param <faulty parameter>
syntax error in the call

***correct param missing
end of parameter list when more parameters are expected

***correct <bsfile> not conneted
<bsfile> could not be connected, maybe not present or not kind bs

***correct segm.<segmno>
<segmno> >= size of <bsfile>

***correct addr.<addr>
<addr> > 511

***correct addr.<addr> bits.<firstbit>.<lastbit>
<firstbit> > <lastbit> or <lastbit> >23

***correct addr.<addr> bits.<firstbit>.<lastbit> oldvalue=<oldvalue>
<oldvalue> is greater than the specified bits allow

***correct addr.<addr> bits.<firstbit>.<lastbit> newvalue=<newvalue>
<newvalue> is greater than the specified bits allow

***correct addr.<addr> bits.<firstbit>.<lastbit>
oldvalue=<oldvalue> ,found=<oldvalue found>
the specified <oldvalue> is not equal to the found value

In the last case the program continues in the parameter list (but no corrections will be made), in all other cases the program terminates immediately.

In case of any of above error messages no corrections are made.

***correct entry inconsistent
***correct code inconsistent
The date of an external procedure is incorrectly described either in the catalog entry or in the code. The correction has been performed.

EDIT

Edit is a line oriented program for editing of text files.

Example:
The FP-call and edit commands:

```
                                    COMMENTS
    betterfinal=edit finaltext      fp call
    1./bad/,r/bad/good/,f           edit command
```

will produce in betterfinal a corrected version of the text finaltext.

The FP-call:

```
    (i corrfile
    newtext=edit oldtext
    end)
```

will correct the text in oldtext with the edit commands in corrfile. The FP-command end ensures that FP will not read from corrfile in case edit exits before the finis command.

BOSS User Manual shows several very relevant examples of the use of EDIT.

Call:

$$\left\{\text{<outfile>=}\right\}_{0}^{1}\ \text{edit}\ \left\{\text{<source>}\right\}_{0}^{\infty}$$

Function:
The program will edit the text in <source> by the commands in current input and store the resulting text in outfile.

<outfile> can be any kind of document. If no outfile is specified, no text is stored.

<source> if no source is specified this is interpreted as an empty source.

When EDIT is loaded and prepared for input of commands the message

```
    edit begin
```

is printed, and before EDIT exits, it prints the message:

```
    edit end.
```

Edit commands:
The editing is performed by means of the following commands. Only the first letter in the word is tested by EDIT:

```
    line
    delete
    insert
    replace
    global
    finis
and less important
    print
    source
    mark
    verify
    where
    ; <comments>      (this line is skipped by EDIT)
```

The commands are separated by NL or COMMA. Superfluous NLs are blind. SPs between commands are blind. Commands separated by COMMA form a sequence. At end of each single command or sequence, the line on which the line pointer points is printed (unless the command: v n is given) see verify.

Delimiters:
A special feature of edit is that the delimiter is chosen each time as the first symbol following the command letter(s), e.g.

```
    1
    /
    *
    ,
    :
    2
    p
```

In the last case p was the first letter in the following word. Illegal symbols, SP, NL and EM cannot be used as delimiters.
The delimiter must not be a part of the string to be searched, the string to be removed or the replacing or inserted string.

In all following examples only the delimiter / is shown.

Warning æ ø å
Those letters have a special meaning and cannot be used in the strings unless the following command is given:

```
                                    COMMENTS
    m e                             mark empty
```

see Mark.

Line:
All corrections are made in the current line, so first of all this must be found. At start the line pointer points at the first line.

	COMMENTS
l7	Move line pointer 7 lines forwards
l-4	Move line pointer 4 lines backwards
l t	Line top, move line pointer to line 1
l b	Move line pointer to line bottom, i.e. the line containing EM.
l./find/	The line pointer is moved forwards to point at the first line containing the string find

Empty lines are not counted. They have the same number as the following line. This is the case for all commands. The line pointer points at the first of the empty lines.
In case the search string consists of several lines, the line pointer will point at the last line. A NL must not be specified by æ10æ as NL has a special representation. This is also the case for the commands delete and print.

Delete:	COMMENTS
d	Delete current line
d 4	Delete current and 4 following lines
d-2	Delete current and 2 preceeding lines
d t	Delete current and all in front
d b	Delete current and all following
d./find/	Delete current and including the line containing the textstring find

After deletion the line pointer points at the line following the last deleted line.
Re. NL se Line.

Insert:	COMMENTS
i/ elefants monkeys /	The two lines are inserted in front of the current line. After insert the line pointer points at the line in front of the terminating delimiter (here the line monkey)

Note that it is a syntax error if the first delimiter is not followed by NL. SPs between the first delimiter and the NL are blind.

Replace:	COMMENTS
r/bad/good/	In the current line the first string bad is replaced by the string good.
r/something//	Remove the first string something from the line
r//something/	Before anything else on the line place the string something

The string, which is to be replaced, must be within one line, i.e. a NL character can only be used in connection with empty lines. A NL character must not be specified by æ10æ. The replacing string can be of any number of lines. The line pointer points at the last line in the replacing string.
If position not found, the line pointer points at the next line.

Global:	COMMENTS
g/bad/good/	In the current line any bad is replaced by good. The line pointer is unchanged.
g 2/bad/good/	In the current and 2 following lines any bad is replaced by good. The line pointer is moved 2 lines forwards.
g-6/bad/good/	In current and 6 preceeding lines any bad is replaced by good. The line pointer is not moved.
g t/bad/good/	In current and all preceeding lines any bad is replaced by good. The line pointer is not moved.
g b/bad/good/	In current and all following lines any bad is replaced by good. The line pointer points at the line following the last line.
g b/unwanted//	Remove the string unwanted from current line and until bottom.

Re. NL see Replace.

Finis:	COMMENTS
f	EDIT copies to EM and exits

Print:	COMMENTS
p	Prints current line
p2	Current and 2 following lines are printed
p-2	Current and 2 preceeding lines are printed with normal direction.
p t	All lines in front of and including current line are printed with normal direction
p b	Current and all following lines until EM are printed
p./find/	The current and all lines inclusive the line with the string find are printed

The line pointer points at the last printed line.
Re. NL see Line.

Source:
The sources are the parameters to the call of edit and are numbered from 1

```
    s 2                          COMMENTS
                                 Edit with input from source number 2
```

Example: the programmer wants to produce a textfile new which is text1 with the procedure error from text2 placed between procedure testoutput and procedure calculate and to link text3 to text1:

```
text1:
begin real a,b,c,d;
procedure testoutput;
begin
write(out,<:<10>:>,a,b,c);
end testoutput;
procedure calculate(x);
real x;
begin
...
```

```
text2:
begin integer i,j,k;
boolean ok;
procedure error(i);
integer i;
begin
write(out,<:<10>alarm :>,i);
end error;
procedure merge(a,b,x);
...
```

```
new=edit text1 text2 text3
1./ure calculate/,
s 2
d./boolean ok/,
1./end error/,11,
s1
d./end testoutput/,
1 b
s 3
f
```

```
COMMENTS
Edit call with 3 sources
Copy until this line from source 1
Continue from source 2
Delete inclusive this line
Copy until this line
Continue from source 1
Delete inclusive this line
Copy to last line
Continue from source 3
Copy and exit.
```

Mark:

EDIT is initialized to:	COMMENTS
m s	Mark standard, which is equivalent to the 3 following commands
m n æ	Mark numeric æ The character æ is here chosen to be used to specify a character by its numeric code, i.e. an integer between 0 and 127, e.g. æ12æ
m c ø	Mark character ø The character ø is here chosen to be used as character replace mark, see example.
m l å	Mark line å The character å is here chosen to be used as line erase mark, i.e. the total line containing the letter å is erased.

If those 3 characters should be treated like other letters, use the following command

m e	Mark empty

Any other characters may be chosen as mark numeric, mark character or mark line, e.g.

m n z	Mark numeric z

The selected characters should not be used in any other context in the edit commands.

Examples of use of mark characters:

	COMMENTS
r/formfeed/æ12æ/	Replace the text formfeed by the character formfeed
r/a**b/a//ba	Erase faulty line (on a console or terminal % should be used as this causes the monitor to erase the line. Under BOSS % only works until timeout, later BEL can be used)
r/abgde<BS><BS><BS>ø<BS>c<SP><SP>fg/alphabet/	Result: r/abcdefg/alphabet/ only to be used when typed on devices which has a backspace BS character. used for correction of one character without changing the rest of the line.

Verify:
Normally the line is listed at end of each command sequence. This may be omitted by the EDIT command

	COMMENTS
v n	Verify no

and reset by

v y	Verify yes

Where:
The EDIT command

```
    w
```

prints the number of the current source text line, e.g.

```
    3 line.
```

Matching strings:
The characters SP, NL and non-graphic characters are blind for identification, i.e. they are skipped by the matching procedure when met in the source string.
In case those characters are part of the search string, they will take part in the matching.
Two strings are considered identical, if the source text has a minimum the same number of SP and NL (and other blind characters) as the search string, e.g.

```
    r/a b/ak/
```

will accept

```
    a b
    a       b
```

but not

```
    ab
```

Parity errors:
When a parity error is met in the source text, the message

```
    parity error on <source>
```

is typed on current out, and edit continues and copies the character 26.
During verification and printing of a line, the character will be printed as the character 38 (ampersand).
The character may be changed as any other symbol, by using the numerical value of the character, e.g.

	COMMENTS
1./æ26æ/,r/æ26æ/g/,f	The faulty character is replaced by g

Error messages:

1. initial alarms

***edit end. no core
The current process is too small

***edit end: param.
The parameters to the call of edit are not syntactically correct.

***edit end: connect object.
The object document cannot be connected by the file processor. If an output area should be created the alarm may indicate that there is no room on the backing store.

***edit end: work area.
There is no room on the backing store for the work area needed for intermediate storage of commands.

2. alarms concerning communication with peripheral devices

***edit <command no.> connect source.
The source document can not be connected by the file processor.
Note: when the source command is used to select a source outside the source given in the parameter list, the source document is defined as empty.

***edit <command no.> source unknown.
The source is not found.

***edit <command no.> work area.
Not enough backing storage for output

***edit <command no.> character
A character with a code greater than 127 has been input either from the source document or the command document.

***edit <command no.> correction area
Not enough backing storage area for a long correction.

<command no.> is reset to 1 at start of each sequence.

Other errors in connection with the transfer of characters and blocks are handled by the file processor and treated as hard errors.

3. alarms caused by errouneous commands:

***edit <command no.> syntax.
A syntax error in the command format is found.

***edit <command no.> position not found
A line position cannot be found, or no match with the string in a replace command can be obtained. The string looked for is printed.

***edit <command no.> backspace error
If random access to the text is not allowed. i.e. when the outfile is not backing storage, backspacing is only allowed a limited number of lines. The alarm is given when backspacing is attempted beyond this number of lines, which among other things is dependent on process size.

<command no.> is reset to 1 at start of each sequence.

REFERENCES:

RCSL 55-D22 Editor 1
RCSL 55-D101 New version of the editor.

END

Returns current input to the previous current input at the position where it was left.

Call:
    end

Function:
The function is the same as when an EM character is read by FP from current input. The actual current input is unstacked, and FP continues reading from the previous current input.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:
***end call
    Left hand side in the call. The end action is still performed.

***end param <parameter>
    Wrong parameter in the call. The end action is still performed.

ENTRY

creates or changes a temporary catalog entry according to the parameters in the call. The program is a supplement to the program SET and is used when one wants to set some of the elements in the entry tail by copying from the tails of other catalog entries.

Example:
Suppose that the catalog entry named 'source' contains the name of a magtape reel in the document name field. By the FP-commands

```
file1=entry mto source 0 1
file2=entry mto source 0 2
file3=entry mto source 0 3
```

one gets catalog entries 'file1', 'file2', 'file3' which serves as file descriptors for file 1, 2 or 3 on the tape reel.

A catalog entry named 'source' containing the name - say mt471100 - may be created by a call of SET:

```
source = set mto mt471100
```

Call:

```
<result name> = entry {<s> <kind> {<s> <kit/doc name> {<s> <free param>
                      {<s> <file> {<s> <block> {<s> <contry>
                      {<s> <length>}0 }0 }0 }0 }0 }0 }0
```

```
{<kind>          }
{<kit/doc name>} ::= {<integer>
                     {<integer1> . <integer2>
                     {<name>
```

```
<free param>  ::= {<word>
                  {<byte1>.<byte2>
                  {d.<isodate>.<check>
```

```
<isodate>     ::= {<yymmdd>
                  {0          0 is interpreted as now
```

```
<clock>       ::= <hhmm>      may be omitted in case no entry
                              named d exists.
```

```
{<file>  }
{<block> }      ::= {<word>
{<contry>}          {<byte1> . <byte2>
{<length>}
```

```
{<word>  }      ::= {<integer>
{<byte1> }          {<name>
{<byte2> }
```

Function:
The parameters are interpreted as described below yielding the wanted entry tail. From this point the program continues exactly as the program SET.

Parameters
Kind:
<integer>: The value is placed in the tail.
<integer1> . <integer2>: The value <integer1> shift 12 + <integer2> is placed in the tail.
<name>: First the name is searched for in the table of mode-kind abbrevations and if found here the value found is used. If not found in the mode-kind table (see Utility Programs, part 1, Appendix) it is searched for in the catalog and the kind of the entry found is used.
Kit/doc name:
<integer>: The value is placed in the tail.
<integer1> . <integer2>: The value <integer1> shift 12 + <integer2> is placed in the tail.
<name>: If the kind just found is the mode-kind bs (2048 shift 12 + 4) the name itself is used in the tail. For all other kinds the name is looked up in the catalog and the kit/doc name in the tail of the entry found is used.
The other parameters:
A parameter of the form <byte1> . <byte2> gives separate specifications of the two 12-bit bytes in the word.
<integer>: The value is placed in the tail as the word or byte in question.
<name>: The name is looked up in the catalog and the value of the word or byte in question in the entry tail found is used.

If the parameter list does not specify all of the tail, the rest of the tail is set to zero.

Storage requirements:
1536 bytes plus space for FP

Error messages:
***entry call
No left side in call of the program
***entry param <parameter>
Parameter error in call of the program
***entry <name> unknown
A parameter was searched in the catalog but not found.

***entry <result name> change kind impossible
A change of an area entry to a non-area entry or vice versa was attempted.
***entry <result name> change bs device impossible
A change of kit/doc name of an area entry was attempted.
***entry <result name> bs device unknown
The bs device specified was not found.
***entry <result name> no resources
The resources of the job did not allow the wanted creation or change of an entry.
***entry <result name> entry in use
The entry could not be changed because another job was using it.

If any message appears no entry is created or changed.

FINIS

Finis terminates the job.

Call:

finis $\left\{ \text{output.} \left\{ \begin{matrix} \text{yes} \\ \text{no} \end{matrix} \right\} \right\}_{o}^{1}$

Function:
The current output file is terminated (emptying of buffers etc.) and a finis job message is send to the parent (the operating system), who is then expected to remove the job.
If parameter output.no is specified, and primout is empty, then nothing is output.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:
***finis call
The program was called with a left hand side - the finis action is still performed.

***finis param <parameter>
Erroneous parameter in the call - the finis action is still performed.

HEAD

Prints a number of form feeds and a page head containing the name of the job and the date and clock.

Example:
The output from two programs is separated in a nice way by calling HEAD in between:

```
     head 1
```

This command prints one form feed and a page head on current output.

```
     head iso cpu
```

This command prints a page head with the date in iso form (i.e. year month day), followed by the cpu time used by the job.

Call:

$$\left\{ \text{<out file>} = \right\}_{0}^{1} \qquad \text{head} \left(\begin{array}{l} \text{<s> <integer>} \\ \text{<s> cpu} \\ \text{<s> iso} \mid \text{old} \end{array} \right)_{0}^{3}$$

Function:
If an integer is given as parameter, that many form feeds are printed. Next, one line consisting of job name, date and clock is printed. If an outfile is specified this is used for the output, else the current output file is used. Date in iso form is standard. The parameter old will cause the date to be printed as day month year.

Storage requirements:
1024 bytes plus space for FP.

Error messages:
***head param <parameter>
Parameter error in the call. A page head is still output.

HEADPUNCH

The program punches a readable text pattern according to the parameters. The same information is also written on current output.

Examples:

The FP-call:

```
headpunch
```

punches a textpattern consisting of: jobname, date and clock

```
headpunch data 2
```

punches the text pattern: data 2

```
headpunch in.textarea
```

punches a text pattern corresponding to the contents of textarea.

The FP-calls:

```
o top
head
message tre
copy textprogram
o c
headpunch in.top
tpe=copy taxprogram
```

will cause the output to start with an optically readable text, e.g.
ta0 1977.03.22 11.12 tre taxprogram 7 segm. 12345/678901.

Call:

headpunch $\left\{ \text{<parameter list>} \right\}_{0}^{1}$

<parameter list> ::= $\left\{ \begin{array}{l} \text{in.<bs-area>} \\ \text{<any sequence obeying the fp-syntax>} \end{array} \right\}$

Function:

The text pattern is output on punch in tpn mode, and the text is written on current output. Current output should not be punch.

If no parameters are specified, the output will be: jobname, date and clock.

If the parameter is in.<bs-area> and the program succeeds to connect to this area, the contents of this area is output until a character=25 (EM) or >127 is found or until 120 characters have been output. NL is punched as space.

In all other cases the parameter list is copied.

Storage requirements:

1536 bytes plus space for FP.

Error messages:

***headpunch call
Left hand side in the call of the program.

***headpunch connect tpn
tpn cannot be connected. The program terminates.

I

Selects a new file as current input. The former file may later be resumed at the position where it was left (for instance by a call of END).

Example:
If we have the following FP-commands in a job file

```
i commds1
i commds2
```

the first will cause FP to start reading from the file 'commds1'. When this file is exhausted FP will return to the job file and read the next call of I, which in turn causes FP to read commands from the file 'commds2'.

EDIT reads the editorial commands from current input. The commands to EDIT may be kept on a separate file 'editcomds' if the editing is done by the following composite FP-commands:

```
(i editcomds                ; the file 'editcomds' is connected
                            ; as current input file.
newtext=edit oldtext        ; call of EDIT
end)                        ; reselects the previous current
                            ; input file
```

The parentheses are essential here. If they were omitted FP would immediately start reading from the file 'editcomds' instead of calling EDIT.
(The END command is not necessary if EDIT reads and accepts all of the file 'editcomds'. It ensures however that FP does not start reading from 'editcomds'.)

Call:

```
I <s> <file name>
```

Function:
The current input file is stacked so that reading may be resumed later (when the new file is exhausted or by a call of END). Next the specified file is connected as current input.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:

***i call
Left hand side in the call.

***i param
Parameter error in the call.

***i <document name> <cause>
The specified file could not be connected for some reason which is explained by <cause> as follows:

no resources	forbidded by the parent (the operating system)
disconnected	device disconnected
name unknown	the file did not exist
kind illegal	the file could not be used for input
reserved	the file was used by another job

In case of any error FP forgets about all previous current input files and returns to the primary input file (the job file).

IF

Makes the execution of the next FP-command conditioned by the values of one (or several) mode bits. The condition may reflect the success of the latest program executed as the ok and warning bits are set at program end (or it may correspond to the mode bits as set by a call of the program MODE).

Example:
If the translation of an Algol source program goes wrong, you want to do the translation once more with listing of the program. If the translation error is serious you want to terminate the run. Proceed as follows:

```
prog1=algol text                ; translate
if warning.yes                  ; if syntactical errors
prog1=algol text list.yes       ; then translate and list
if ok.no                        ; if serious errors
finis                           ; then terminate job
prog1                           ; else execute the program
```

Call:

if { <s> <mode bit> . { yes / no } }$_1^{\infty}$

<mode bit> ::= { <integer> / ok / listing / warning / error / pause / all }

Function:
The next (possibly composite) FP-command is executed if each of the mode bits mentioned in the parameter list has the specified value 'yes' or 'no'. If not, the next FP-command is skipped. The program IF does not change any mode bit (even not the ok and warning bits) hence repeated questions may be asked on the same mode bits by several succesive calls of IF.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:
***if call
Left hand side in the call - does not affect the function of the program.
***if param <parameter>
Wrong parameter in the call. The erroneous parameter is skipped and the program continues with the next parameter.

KIT

Sends a mount disc message to the parent (the operating system) demanding a disc kit with a specified name to be mounted on a specified disc unit (cf. the BOSS2 User Manual ch. 4.3).

Example:
The FP-command

```
kit 12 disc5
```

asks for mounting of the disc kit 'disc5' on the disc unit with device number 12.

Call:

```
kit <s> <device no> <s> <kit name>
```

where <device no> is an integer and <kit name> is a name.

Function:
A mount kit message containing the device number and name specified is sent to the parent.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:

***kit call
Left hand side in the call of kit.

***kit <parameter list> parameter error
Parameter error in the call of kit.

***kit <parameter list> not available
The kit specified by the kit name is not available for the job.

In case of any error no mount kit message is sent.

LABEL

Outputs a boss label on file 0 of the specified magnetic tape.

Example:

The fp-call:
    label mto mt123456 p 789012
outputs a boss label on mt123456.

If you have a filedescriptor, e.g.:
    f=set mto mt123456 0 1
the call:
    label f f p 789012
will have the same effect.

Call:

label <modekind> <mtname> { <access> <project number> }$_0^1$

<modekind> ::= { mto | mte | nrz | nrze | <name of filedescriptor> }

<mtname> ::= { <name of magnetic tape> | <name of filedescriptor> }

<access> ::= { p | r | w }

<project number> ::= <integer, max. 999999>

Name of magnetic tape must start with mt followed by exactly 6 characters, the first 2 may be letters or digits, 4 last must be digits.

Function:

A label in the format accepted by boss (cf. BOSS2 Users Manual, ch. 5.3) will be written in file 0 of the tape. Next, two tapemarks are written (i.e. an empty file 1), and approx. 5 inches of tape is erased.

Notice: this means that the first part of the previous contents of the tape will unconditionally be destroyed.

Error messags:

***label, call
    Left hand parameter in the call
***label, <parameter> param
    Illegal parameter in the call
***label, <parameter> modekind error
    Filedescriptor does not describe a magnetic tape.
***label, <modekind param> unknown
    Modekind does not describe mto, mte, nrz, nrze cr a filedescriptor.
***label, <parameter> illegal tapename
    Tapename is illegal.
***label, <parameter> illegal access kind
    Access must be p, r og w
***label, project number missing
    If access is specified, a project number is demanded
***label, <parameter> illegal project number
    Project number must be max. 999999
***label, too many parameters
    The program accepts max. 4 parameters
***label, parameter missing
    The program demands at least 2 parameters.
***label, connect tape unsuccessful
    Hard error.

LOAD

The program can input catalog entries and bs-files from magnetic tape files generated by the program SAVE.

Example:

All catalog entries and bs-files saved on mt471100 file 1 are reestablished by the FP-command:

```
load mt471100.1
```

In case: t=set mto mt471100 0 1
the same is obtained by the command:

```
load t.0
```

All catalog entries and bs-files of scope temp plus the entry by name pap are loaded by the FP-command:

```
load mt471100.1 scope.temp pap
```

See also: Further examples.

Call:

{<outfile>=}$_0^1$

load {<mountparam>.}$_0^1$ <tape parameter> <special param> <load spec>

<mountparam> ::= {mountspec.<deviceno>.}$_0^1$ {<modekind>.}$_0^1$ {release.{yes / no}}$_0^1$

<tape parameter> ::= <tapename>.<fileno>{.<tapename for next volumen>}$_0^2$

<tapename> ::= { <filedescriptor describing a magnetic tape file> / <name of magnetic tape> }

<fileno> ::= { last / <integer> }

<special param> ::= { check {.yes / .no} / survey {.yes / .no} / load / list {.yes / .no / .name} }

<load spec> ::= { <modifiers> / <kit spec> / <entry spec> }

<modifiers> ::= { changekit.<bs device spec>.<bs device spec> / newscope.<newscope spec> }$_0^\infty$

<kit spec> ::= kit.<bs device spec>

<bs device spec> ::= { <bs device name> | main | 0 | 1 }

<entry spec> ::= { <name> | <name>.scope.<scope spec> | docname.<docname> | docname.<docname>.scope.<scope spec> | scope.<scope spec> }$_{0}^{\infty}$

Function:

The contents of the dump label (see SAVE) is checked and listed on current out.

Next the program loads from the magnetic tape all the entries and bs-file specified by <load spec>. If <entry spec> is empty all the entries are loaded.

Each entry is created with scope and <bs device spec> as defined in the entry record. For area entries <bs device spec> defines the kit name for non-area entries, the kit into which the entry is permanented.

Function, mountparam

If no mountparam is specified, the program will use a standard magtape station, modekind=mto (or in case mtname is a filedescriptor, then the modekind of this filedescriptor) and the tape will be released at end of program.
e.g. mountspec.10.nrz.release.no.
mountspec.10.
nrz.
release.no.

Function, tapeparameter.

In case mtname is a filedescriptor, filenumber will be understood relative to the filenumber in the filedescriptor. The modekind of the filedescriptor will be used.

If <fileno>=last, the file in front of the first file which does not contain a version label is loaded. This parameter gives a longer run time than an integer parameter.

Tapenames of following volumes is only necessary in case the following volume has a name different from what is stated in the continuation block. This may be the case if saving was performed with 2 parallel tapes.

Function, special param

check.{yes / no} Std. is check.yes.
If check.no, then the program continues, if the mtname or the fileno in the dumplabel is wrong. Also when the dumplabel is a continuation label.

survey.{yes / no} Std. is survey.no
If survey.yes, then all entries from file 1 to <fileno> are listed but not loaded.

load.{yes / no} Std. is load.yes, the specified entries are loaded.
If load.no, then all specified entries in the file are listed but not loaded.

list.{yes / no / name} Std. is list.yes, all loaded entries are listed.
If list.no, the entries are not listed.
If list.name, then only the names of the entries are listed.
If an outfile is specified, this file is used for output, otherwise current output file is used.

Function, modifiers

If <bs device spec> = 0, the area is - if possible - created on a drum, otherwise on a disc.
If <bs device spec> = 1, the area is - if possible - created on a disc, otherwise on a drum.
If <bs device spec> = main, the area is created on the bs-device containing the main catalog.
If <bs device spec> = a name <> main, the area is created on the bs-device with this name.

changekit.<saved kit>.<loaded kit>
This parameter is valid for the total call.
Each entry, which on the tape is described as an entry on <saved kit> will be loaded on <loaded kit>.
e.g. changekit.disc1.disc2

newscope.<newscope spec>
Std. is newscope.std, meaning no change in scope.
This parameter is valid for the following entry specifications. They will all be created with <newscope spec>.
<newscope spec> ::= temp |login |user |project |std
e.g. newscope.temp

Function, kit spec

General about <bs device spec>, see modifiers.

kit.<saved kit> Std. is main, meaning all devices.
Only entries which in the tape is described as addressing this kit will be loaded (or entries which after a changekit parameter is addressing this kit). The parameter is valid for all following <entry spec> until a new kit parameter is specified. If a kit parameter specifies a not

connected kit, a changekit parameter, changing this kit to a connected kit name, must be specified earlier in the parameter list.
e.g. kit.disc2

Function, entry spec

<scope spec> ::= temp |login |user |project |own |system |perm |all

<name> All entries of the name are loaded.

scope.<scope spec>
All entries with this scope are loaded.

<name>.scope.<scope spec>
An entry of specified name and scope is loaded.

docname.<docname>
All entries with specified docname (maybe kitname) are loaded

docname.<docname>.scope.<scope spec>
All entries with specified docname (maybe kitname) and specified scope are loaded.

Tape format: see SAVE.

Load of systemdump.

Entries which are saved at scope.perm (or scope.all) may be loaded in a boss job. Entries with bases corresponding to scopes temp, login and user will be loaded if their name is specified. For sake of security it is decided that entries of scope project must be specified by <name>.scope.project.

Storage requirements:
12000 bytes.

Error messages:

***load: error in tapeparam <erroneous and following parameters>
Parameter error in the call. The program terminates.
***load: error in modekind spec.
<tapename> describes an entry of kind 18, but mode is neither 0 nor 4.
***load: error in param <erroneous and following parameters>
Parameter error in the call. The program terminates.
***load param, kitnames exceeded
The program does not accept more than 10 bs-device names different from the bs-devices connected. The program terminates.
Remedy: two calls of load.
***load: no dumplabel on file <fileno>
The file contains no dumplabel. The program terminates.

***load: dumplabel <specification>
Error in the specified part of the dumplabel. The program terminates.
<name> entry inconsistent
<name> code inconsistent
The date of an external procedure is incorrectly described, either in the catalog entry or in the code. The entry is loaded.
<name> bad tape: <pattern>
Hard error during run. The pattern shows the status word.
<name> bad tape: <pattern> blocklength = <blocklength>
Blocklength error on the tape.
<name> bad tape, blocks skipped <skipped blocks>
The blocks could not be interpreted by the program.
<name> bad tape, segm. loaded <segments>
The segments loaded do not correspond to the number of segments specified in the entry record.
bad tape, entry no. <d> missing
<name> bad tape, segm. no. <d> missing
<name> monitor <xx> result <y> <explanation>
The call of the algol procedure monitor with the parameter <xx> gave the unwanted result <y>.
<explanation>:
device not mounted
process base error
no work resources
no perm resources
entry in use
impossible (catalog error)
***not found <entry spec>
<entry spec> was not found on the tape.
***load not ok <d>
This message occurs at program exit in case of any error.
<d> is the number of errors.

Further examples:

1) The programmer wants to load the entries and bs-files pr1 and pr2 from a saved file, which contains several other entries, furthermore he wants to change the scope of pr2 to user:

```
load mt471100.2 pr1 newscope.user pr2
```

2) The programmer wants to load the entry named pip and all his entries which belong to the catalog on kit5, from a file which contains other entries as well:

```
load mt471100.3 pip kit.kit5 scope.own
```

3) The programmer wants to check the contents of mt471100

```
load mt471100.last survey.yes
```

4) The programmer wants to load file 8 but gets the output

```
dump  mt471100    006   vers. 130473.12   s=1   unhappydays
***load dumplabel fileno
```

at repeated calls. This suggests that the start of the magnetic tape has been overwritten and probably the wanted file will be found on file 10. Try the FP-command:

```
load mt471100.10 check.no load.no
```

LOOKUP

Finds and lists catalog entries with specified name.

Example:
The FP command

```
lookup pip
```

finds and lists the entries with name 'pip' and prints something like:

```
pip         =set 16 disc d.770523.1021 0 0 0 0   ; temp
            ;  92 17 0 -56 -56
```

The first line gives the tail and the scope of the entry, the second line gives the entry head.

Call:

$$\left\{\text{<outfile> =}\right\}_{0}^{1} \quad \text{lookup} \quad \left\{\text{<s> <name>}\right\}_{1}^{\infty}$$

Function:
Each name in the list is searched in the catalog and all entries with this name which may be accessed by the job are listed. If an <outfile> is present this file is used for the output - otherwise the current output file is used.

Format of the output:
Each catalog entry is listed as two lines:

```
<name>  =set <entry tail> ; <scope spec>
        ;  <entry head>
```

The name and entry tail appear exactly as in a call of the program SET for creating the entry.
The scope specification has the form

$$\text{<scope>} \quad \left\{.\text{<device name>}\right\}_{0}^{1}$$

where <scope> is one of temp, login, user, project, system or *** (the last one means scope undefined) and where a <device name> tells that the entry is permanented into the auxiliary catalog on this device.

The entry head is output as the five integers

<first slice> <name key> <catalog key> <interval lower> <interval upper>

as described in the manuals for the monitor (ref. 2 and 3).

Storage requirements:
2560 bytes plus space for FP.

Error messages:
***lookup connect <outfile>
The specified output file could not be connected - current output is used instead.
***lookup param <parameter>
Parameter error. The remainder of the parameter list is skipped.
***lookup <name> unknown
No entries with the given name was found. The program continues with the next name in the list.
***lookup <name> no resources
The program has terminated because the job has too few area processes.

MESSAGE

May be used (together with HEAD) to make nice headings on the output. The parameter list in the call of message is simply output when the program is called.

Example:
The FP command

```
message program run no.1
```

outputs the text , program run no.1 , on current output.

Call:
{<outfile> = }1 message <s> <parameter list>
<parameter list> may consist of any sequence of parameters obeying the FP-syntax.

Function:
The parameterlist is copied on <outfile> or current output (if no outfile is specified). The output is terminated by a NL character.

Storage requirements:
512 bytes plus space for FP.

Error messages:
***message connect <outfile>
The specified output file could not be connected. Current output is used instead.

MODE

Changes the FP mode bits specified in the call and may thereby change the working cycle of FP.

Example:
The FP-command
    mode list.yes
causes FP to change to list mode i.e. each FP-command is listed on current output just before execution.
The FP-command:
    mode what
causes all modebits to be listed.

Call:

mode { <s> what | <s> <mode bit>. { yes | no } }∞,1

<mode bit> ::= { <integer> | listing | warning | ok | error | pause | list | all }

The integer values of the mode bit names are as follows: listing=15, warning=17, ok=18, error=19, pause=20, list=23. Bit 16 is used internally by FP.

The mode bits are explained in Utility Programs, part one, ch. 4.2.

Function:
The FP mode bits are changed as specified in the call.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:
***mode call
    Left hand side in the call - does not affect the function of the program.

***mode param <parameter>
    Wrong parameter in the call. The parameter is skipped and the program continues with the next parameter.

MOUNT

Sends a mount message to the parent (the operating system) who is then expected to ask the operator to mount the tape reel (cf. the BOSS2 User Manual, ch. 5 and 10). The program does not await the mounting, unless there is asked for mounting of an unspecified worktape.

Example:
When a program needs a magnetic tape reel which is not mounted, the mounting is automatically requested and the job waits for it. The scheduling of a job which uses several tape reels is however improved if the tape reels are requested right at the beginning of the job, i.e. if the tape reels named 'mt280007', 'mt280008' and 'mt280009' are needed during the job one may start the job file with the FP-commands

```
mount mt280007
mount mt280008
mount mt280009
```

If p7, p8, p9 are names for files on these magtapes, e.g.

```
p7=set mto mt280007 0 3
p8=set mto mt280008 0 1
p9=set mto mt280009 0 2
```

the same result is obtained by the FP-commands:

```
mount p7
mount p8
mount p9
```

An unspecified worktape is requested as follows:

```
workfile=set mto 0 0 1
mount workfile
```

This call of MOUNT asks for mounting of a worktape and places the name of the magtape reel in the entry. File number 1 on the tape is now available under the name 'workfile'.
The worktape is released and made available to other users when the job terminates or if the tape is released during the job. One may suspend the use of the worktape by a SUSPEND command (cf. the description of SUSPEND).

Call:

```
mount <s> <name>
```

Function:
A mount message is sent to the parent.
The name in the message is found as follows: The name is looked up in the catalog. If an entry describing a magnetic tape file (kind=18) is found and if this entry if not protected (e.g. not of scope system) the document name in the entry is used. Otherwise the name specified is used. The document name in the entry may be empty (zero). In this case a worktape is mounted and the name of the worktape placed as document name in the entry.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***mount call
Left hand side in call of mount.

***mount <parameter list> parameter error
Parameter error in call of the program.

In case of any error no mount message is sent.

MOUNTSPEC

Sends a mount special message to the parent (the operating system) limiting a later mounting of the specified magnetic tape reel to the station with the specified device no. (cf. the BOSS2 User Manual ch. 5 and 10).

Example:
If the installation has some standard magnetic tape stations (say 9-track) and a non standard (say 7-track) with device number 6 and one has a 7-track tape reel named mt123456 the FP-command

```
mountspec 6 mt123456
```

ensures that BOSS will accept the reel only when mounted on station no. 6. If 'pip' is the name of a file on a magtape e.g.

```
pip=set mto mt123456 0 7
```

the same result is obtained by the FP-command

```
mountspec 6 pip
```

Call:

```
mountspec <s> <device no> <s> <name>
```

where <device no> is an integer.

Function:
The name is looked up in the catalog. If an entry describing a magnetic tape file (kind=18) is found and if this entry is not protected (e.g. not of scope system) the document name in the entry is used. Otherwise the name specified is used. Next a mount special message containing the specified device number and the name is sent to the parent.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***mountspec call
Left hand side in the call of the program.

***mountspec <parameter list> parameter error
Parameter error in the call of the program..

***mountspec <parameter list> tape name missing
The entry specified has a zero document name.

In case of any error no mountspec message is sent.

MOVE

Performs blockwise copying of files on backing storage or magnetic tape.

Examples:
The content of the backing storage area with name , text4 , is moved to file 5 on the magnetic tape reel named , mt314711 , by the FP commands:

```
file5=set mto mt314711 0 5
file5=move text4
```

Files number 3, 4, 5, 6 on the magtape , mt312223 , will be copied to the magnetic tape , mt312224 , starting at file number 7 by the FP commands

```
fromfile=set mto mt312223 0 3
  tofile=set mto mt312224 0 7
  tofile=move fromfile.4
```

Call:
The program may be called in two ways depending on the kind of the left hand side:

<bs-file> = move $\{$message.yes$\}_0^1$ $\{$<bs-file> | <mt-file>$\}$

or

<mt-file> = move $\{$message.yes$\}_0^1$ $\{$<bs-file> | <mt-file-set>$\}_0^\infty$

<bs-file> must be a catalog entry describing a file on the backing storage.
<mt-file> must be a catalog entry describing a file on a magnetic tape.
<mt-file-set> ::= <mt-file> $\{$.<no-of-files> $\{$.<skip>$\}_0^1\}_0^1$
<no-of-files> is an integer defining how many files to copy.
<skip> is an integer defining how many files to skip before the copying.

Function:
Move performs blockwise copying of files on backing storage or magnetic tape.

The parameter message.yes will cause output of bytes and checksum.

If the output file specifies magnetic tape, as many files as specified in the input parameters will be written, separated by tape marks.

If the files on both sides of the move-call specifies magnetic tape, the block lengths of the input file(s) are kept on the output file(s).

If the output file describes a bs-area, the length of the area will be decreased corresponding to its new contents. If the input file describes a bs-area too, the last 5 words of the catalog entry tail will be inserted in the output tail.

Storage requirements:
The core storage required for move is 2850 bytes plus the space for FP.

When copying from a magnetic tape with block lengths greater than 512 bytes, more core storage will be needed. In a process with 10000 bytes of core storage, the maximum block length is about 1600 bytes.

Error messages:
***move: no core
The process area is too small to contain the input and output buffers.

***move call
No left hand side is specified in the call

***move param: <parameter list>
An input specification has an erroneous format. The specification is shown as <parameter list>. *)

***move: input kind
***move: output kind
The specified file is neither a bs-file nor a mt-file. *)

***move: connect input
***move: connect output
It has not been possible to connect an input or an output file.

***move: too many parameters
It is attempted to copy more than one file to a bs-file.

***move: change error
It is not possible to change the catalog entry describing the output bs-file.

*) The parameters will be checked and handled one by one. Therefore one or more files may have been copied even if the program is terminated by an alarm.

Further examples of use:
In the following the catalog entries mt1 and mt2 describe file number one on two magnetic tapes, and bs1, bs2 --- describe areas on the backing storage.

```
move mt1
causes the alarm:
***move call
because no output file is specified

mt1 = move bs1 bs2 trf
   The areas bs1 and bs2 will be moved to file number 1 and 2 on the
   tape the last parameter causes the alarm:
*** move: input kind

mt1 = move mt2.2 bs3 mt2.2.3
   After this call, the tape described by mt1 will contain:
         file no          contents from
            0             unchanged
            1             mt2 file 1
            2             mt2 file 2
            3             bs3
            4             mt2 file 4
            5             mt2 file 5

mt1 = move mt2.1.1.1
causes the alarm:
***move param: mt2.1.1.1
   because of the erroneous parameter, and no copying is performed.
```

NEWJOB

Sends a newjob message to the parent (the operating system) demanding the specified file enrolled as job file in a new off line job i.e. in this way a new job is created. The actual job continues with the next FP-command. Further details are found in section 1.7 internal jobs in the BOSS2 User Manual.

Call:
newjob <s> <file name> $\{$ <name of remote batch printer>$\}_0^1$
where <file name> is a name of a permanent job file.
<name of remote batch printer> ::= <name of max 6 char>

Function:
A newjob message containing the specified name(s) is sent to the parent.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***newjob call
Left hand side in the call of the program.
***newjob <parameter list> parameter error
Parameter error in the call of the program.
***newjob <filename> <error cause>
Error during creation of the new job. The cause may be any of the following:
job queue full
job file not permanent
job file unknown
job file unreadable
user index too large
illegal identification
user index conflict
job file too long
temp claim exceeded
option unknown
param error at job
syntax error at job
line too long
attention status at remote batch terminal
device unknown
device not printer
parent device disconnected
remote batch malfunction
In case of any error no new job is created.

NEXTFILE

Adds one to the file number in the tails of the catalog entries specified.

Example:
If the catalog entries 'to' and 'from' describe file 3 of the magtape 'mt312223' and file 6 of the magtape 'mt312224', respectively, the FP command

nextfile to from

will change them to describe file 4 and 7 of the tapes in question.

Call:

nextfile $\left\{ \text{<s> <name>} \right\}_{1}^{\infty}$

Function:
For each name in the list a catalog lookup is made and the file number in the tail of the entry is increased by one.

Storage requirements:
1536 bytes plus space for FP.

Error messages:

***nextfile call
Left hand side in the call. The program terminates without further actions.

***nextfile param <parameter>
Parameter error. The faulty parameter is skipped and the program continues with the next parameter.

***nextfile <name> unknown
No entry with the specified name was found. The program continues with the next parameter.

***nextfile <name> protected
The job was not allowed to change the tail in the entry found. The program continues with the next parameter.

O

Selects a new file as current output.

Example:
The text output from an algol translation may be put on a special file in the following way:

```
o list                          ; the file 'list' is chosen as current output
program=algol text list.yes     ; translation of the algol program
o c                             ; current output is shifted back to the
                                ; primary output file
```

Note, that in case of larger programs, the area 'list' may be too small and unable to be extended. Remedy: set a sufficiently larger area before the call of the program '.' e.g. list=set 150.

Call:
o <s> <file>

Function:
The actual use of the current output file is terminated (emptying of buffers) and the file given as parameter is connected as current output.
There is no stacking and unstacking of previous used output files as for current input files.
If <file> is not found in the catalog an area with this name on the backing storage (preferably on a disc) is created and connected as current output.
The name 'c' - however - is used for the primary output file and is treated in the following way. Whenever the program O connects current output to 'c' (either because of the command 'o c' or because of some error) the following is done: If a catalog entry named 'c' is present, the file described by this entry is connected. If the catalog entry is not present it is created as describing the primary output file and current output is connected to the file.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:

***o call
Left hand side in the call.

***o param <param>
Parameter error in the call.

***o <document name> <cause>
The file could not be connected. The reason is explained by <cause>:

no resources	the job resources are exceeded
disconnected	the device is disconnected
kind illegal	the file could not be used for output
reserved	the file was used by another job.

In case of any error the primary output file is connected as current output file.

ONLINE

Turns the job into the conversational mode where the current input to the job is typed on the terminal at run time. A conversational job is very resource demanding and the user must have a special option in the user catalog (cf. the BOSS User Manual ch. 3.5).

Call:

online

Function:

The process 'terminal' is connected as current input and selected as a new primary input.
Contrary to the FP-command

i term

the FP-command

online

has the advantage that an FP syntax error will not return current input to the job file.

Storage Requirements:

512 bytes plus space for FP.

Error Messages:

***online connect terminal
The job does not have the option 'online yes'

OPCOMM

Sends the parameter list in the call as a print message to the parent (the operating system) with request for an answer from the operator and types the answer (when received) on current output (cf. the BOSS2 User Manual, ch. 10).

Example:
A user with initials hsr and project number 47 is placed at a terminal and needs a new project tape reel. The labeling of the reel and an answer back telling the reel name may be requested by the FP-commands:

```
opmess label new p 47 reel
opcomm return name of reel
```

This causes the following lines to appear (among the other messages from BOSS) on the main console

```
message hsr0 label new p 47 reel
pause hsr0 return name of reel
```

When the operator has labeled the reel - say with the name 'mt271536' - he returns the name to hsr by typing

```
answer hsr0 mt271536
```

on the main console.
In the meantime OPCOMM has been waiting for the answer. The answer is now output as the text

```
*operator answer: mt271536 0
```

on current output (for hsr0).

Call:

```
opcomm <s><parameter list>
```

The parameter list may consist of any sequence obeying the FP syntax.

Function:
The first 21 characters (if that many are present) in the parameter list are packed as a print message and sent to the parent.
The answer is then awaited and when it arrives printed on current output in the form

```
*operator answer: <name> <integer>
```

where <name> and <integer> are the answer as typed by the operator (cf. the BOSS2 User Manual ch. 10).

Storage Requirements:
1536 bytes plus space for FP.
Error Messages:
***opcomm call
Left hand side in call of the program. No message is sent and no waiting is performed.

OPMESS

Sends the parameter list in the call as a print message to the parent (the operating system). If the operating system is BOSS the message is typed on the main console (cf. the Boss2 User Manual ch. 10).

Example:
An example of the use is given in the description of the program OPCOMM.

Call:
    opmess <s> <parameter list>
The parameter list may consist of any sequence of parameters obeying the FP syntax.

Function:
The first 21 characters (if that many are present) in the parameter list are packed as a print message and sent to the parent (cf. the Boss2 User Manual, ch. 10).

Storage Requirements:
1536 bytes plus space for FP.

Error Message:
***opmess call
    Left hand side in call of the program. No message is sent.

PRINT

Prints from a backing storage area or directly from the core store with specified formats. The program is primarily intended for printing of dumped core areas.

Example:
The core of the job process has been dumped into a backing storage area named 'image' (under BOSS this is for instance provoked by the FP command 'mode pause.yes' just before the call of the program we are going to debug).
By the FP command

print image 0.14 1536.1600

the words number 0 to 14 and 1536 to 1600 of the area are printed on current output as integers, halfwords and code. (The words 0 to 14 contains the start address of the core area and the registers at the time of the dump).
If the area is described with contents 7 (dumped core areas should always have this contents. When BOSS makes a core dump the contents is set to 7) the output is numbered with absolute addresses (as the program was placed in core when the dump was made). One can select the part to be printed by specifying such absolute addresses: The command

print image 45236.45344.a

prints the part of the dump originating from the core addresses 45236 to 45344.

Call:

$$\left\{\text{<out file> =}\right\}_{0}^{1} \quad \text{print <source>} \quad \left\{\text{<format list> <field>}\right\}_{0}^{\infty}$$

$$\text{<source> ::=} \left\{\begin{array}{l}\text{<bs area name>}\\ \text{<internal process name>}\\ \text{<integer>}\end{array}\right\}$$

$$\text{<format list> ::=} \left\{\begin{array}{l}\text{integer}\\ \text{word}\\ \text{half}\\ \text{abshalf}\\ \text{char}\\ \text{octal}\\ \text{code}\\ \text{text}\\ \text{bits <pattern>}\\ \text{all}\\ \text{words . <words per line>}\end{array}\right\}_{0}^{\infty}$$

<pattern> ::= $\{$. <first bit> . <last bit>$\}_0^\infty$

<field> ::= $\left\{ \begin{array}{l} \text{<from - to>} \left\{ \begin{array}{l} \text{.i} \\ \text{.c.<center>} \\ \text{.r} \\ \text{.a} \end{array} \right\}_0^1 \\ \text{<from addr>.<to addr>.<from block>} \\ \text{<from addr>.<to addr>.<from block>.<to block>} \end{array} \right\}$

<from - to> ::= $\left\{ \begin{array}{l} \text{<from addr>} \\ \text{<from addr>.<to addr>} \end{array} \right\}$

<words per line>, <first bit>, <last bit>, , <from addr>, <to addr>, <from block> and <to block> are integers.

Function:
The format list is initialized to all (see below).

The parameter list is scanned and the printing source is determined. If <source> is the name of an area on the backing storage PRINT prints from this area. If <source> is the name of an internal process PRINT prints from core with the start address of the process as base address. If <source> is an integer the printing takes place from core with this integer as base address.

The program enters the following cycle until the end of the parameter list:

1) When a <format list> is recognised the printing format is changed accordingly.
2) When a <field> is recognized the printing is activated. The printing is done with the current format.

The output occurs on <outfile> if specified - otherwise on current output.

Format list:
The elements of a <format list> defines how the current word of the actual field appears in the output:

integer	current word is printed as a signed integer.
word	current word is printed as a signed integer.
half	current word is printed as two signed integers, being the two halfwords of the word.
abshalf	current word is printed as two positive integers, being the two halfwords of the word.
char	current word is printed as three unsigned integers i.e. the iso values of a text is printed.
octal	current word and address is printed as octal. If code is also specified, the final address is printed as octal.

code — current word is printed as an instruction in symbolic form. If the instruction includes relative addressing, the output is supplied with the corresponding final address according to the numbering of words:

final address = displacement + number of current word

This final address is printed immediately after the displacement.

text — current word is printed as 3 ISO characters, non-graphic characters replaced by SP.

bits.<pattern> — current word is printed as a number of unsigned integers according to <pattern>. Denoting the bits from 0 to 23, each integer is the value of the bit group defined by <first bit> and <last bit>. The value of <pattern> is initialized to:

0.0.1.1, ... , 22.22.23.23

which causes the current word to the printed as 24 integers, being the value of each bit of the word.

words.<words per line> — determines the number of words to be printed in each line. The line is headed by an integer corresponding to the numbering of words, as explained above. The value of <words per line> is initialized to 1.

all — is equivalent to the <format list> integer bits.0.11 code

If a <format list> consists of more elements, the current word is printed in all forms, as defined by the elements of this list. The different forms occur in a certain order in the output, according to the following sequence:

<text> <integers, halfwords and bit patterns> <instruction>;

<integers, halfwords and bit patterns> are printed in the same order as the corresponding elements in <format list>.

The <format list> is initialized to:

integer bits.0.11 code

which causes current word to be printed in the 3 forms:

<integer> <left-most halfword> <instruction>.

Field specification:

The limits for the printing are determined from the integers <from addr> and <to addr> by rules depending on the other part of the field specification. (A <from addr> alone means <from addr>.<from addr>.) If only <from addr> and <to addr> are in the field specification they give the limits relative to the start of the backing storage area or to the base address for the core area.

Specification of <from block> and <to block> is significant only for bs areas. The area is considered divided into segments (each on 512 bytes) and from each segment with segment number between <from block> and <to block> (<from block> alone means just this single block) the part of the segment determined by <from addr> and <to addr> is printed.

The modifier .i (indirect addressing) causes the contents of the words specified by <from addr> and <to addr> to be interpreted as absolute addresses and used as limits. The values <from addr> and <to addr> are interpreted relative to the base address. (If the source is a bs area it should have contents = 7.)

The modifier .c. (indirect addressing around a center): The contents of the word with relative address (relative to the area start or the base address) is interpreted as an absolute address and taken as center for printing and the printing limits becomes <from addr> below the center and <to addr> above the center. (If the source is a bs area it should have contents = 7).

In case of the modifier .a (absolute addressing) the printing limits are the integers <from addr> and <to addr> taken as absolute addresses. (A bs area source should have contents = 7).

The modifier .r (relative addresses in output) belongs in a way to the format specification. It causes the absolute addresses used as numbering in the output to be replaced by relative addresses.

Storage requirements:
The core store space needed by PRINT is approx. 2048 bytes plus the space needed by FP.

Error messages:

***print param <erroneous parameters>
parameter error in call of PRINT. If the parameters are part of a syntax element, this has no effect.

***print numbering
The field specification attempt to define words outside area.

***print <name> area
area process cannot be created or trouble during input data transfer.

***print connect out
output file cannot be connected.

*** <name> unknown
<name> is neither name of a catalog entry or an internal process.

***print core size
No core space for segment buffers; at most 512 halfwords more are needed.

In the first two cases PRINT continues with the next parameter in the list.
In the other cases PRINT terminates.

Further Examples:

print sin
prints the total area sin as integer bits.0.11 code

print datas integer 0.510.1
prints the second segment of datas as integers

print algol text all 10.20.0.4
prints halfwords 10 to 20 on the first 5 segments of algol as
text integer bits.0.11 code

print image 0.14 16.10.c.12 1594.1604 1614.1616 1598.1582.i
prints relevant parts after break of algol program
(see RCSL No. 31-D199, Code Procedures and Run Time Organization
of Algol Programs). Corrections in Running System may change
these numbers in which case a correction for above manual will
be issued.

0	first address
2	w0
4	w1
6	w2
8	w3
10	exeption register
12	instruction counter
14	interrupt cause

16.10.c.12 8 words before and 5 words after breakpoint

1594	UV
1596	UV
1598	lastused
1600	last of program

PROCSURVEY

Lists types of procedures and their parameters, as well as the procedure date.

Example:

The FP-command:

```
procsurvey invar in
```

will produce the output:

```
integer procedure invar: d.760830.1405
  param 1: zone

zone in, rs entry no.: 26
```

Function:

Each name is looked up in the catalog, if several entries with the same name exists, only the one with the smallest scope will be listed. Procedures and standard variables will be listed, as above, other entry types will cause an error message.

Storage Requirements:

2500 bytes plus space for FP.

Error messages:

***procsurvey call
Left hand side in the call.
***procsurvey <integer> param
Integer parameter.
***procsurvey <name> unknown
The name was not found in the catalog.
***procsurvey <name> connect error
The area could not be connected.
***procsurvey <name> not procedure
The name does not describe a procedure or an algol std. identifier.
***procsurvey <name> entry inconsistent
The start external list in the entry description,
contains a byte>500, i.e. the entry does not describe a legal procedure.
***procsurvey <name> code inconsistent
Illegal contents of the internal list in the code body.

In case of error, procsurvey continues after the error message.

RELEASE

Sends a release message to the parent (the operating system) releasing the specified magnetic tape reel (cf. the BOSS2 User Manual ch. 5 and 10).

Example:
If the total number of tape reels used during a job exceeds the numbers of stations available one has to release one of the tapes during the job in order to tell BOSS that the reel could be dismounted. The FP-command

```
release mt123456
```

tells BOSS that mt123456 can be dismounted.
If 'pip' is a name of a file on the magtape e.g.

```
pip=set mto mt123456 0 7
```

the same result is obtained by the FP-command

```
release pip
```

In general it is good manners to release a tape reel as soon as it is not longer required.

Call:

```
release <s> <name>
```

Function:
A release message is sent to the parent. The name in the message is found as follows: The name is looked up in the catalog. If an entry describing a magnetic tape file (kind=18) is found and if this entry is not protected (e.g. not of scope system) the document name in the entry is used. Otherwise the name specified is used.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***release call
Left hand side in the call of the program.
***release <parameter list> parameter error
Parameter error in the call of the program.
***release <name> tape name missing
The entry specified has a zero document name.

In case of any error no release message is sent.

RENAME

Changes the names of catalog entries as specified.

Example:
By the FP command

```
rename pip.fup
```

the name of the catalog entry named 'pip' is changed to 'fup'. The scope, entry tail and the contents of an associated data area remains unchanged.

Call:

rename { <s> <oldname> . <newname> }$_1^\infty$

Function:
Each <oldname> in the list is looked up in the catalog and the name of the entry found is changed to the corresponding <newname>.

Remark: if several entries with the same name are present, the catalog lookup will find the entry with the 'smallest' scope (corresponding to the order: temp, login, user, project).

Storage requirements:
1536 bytes plus space for FP.

Error messages:

***rename call
Left hand side in the call. The program terminates without further actions.

***rename param <parameter>
Parameter error. The remainder of the parameter list is skipped.

***rename <oldname>.<newname> name conflict
The entry could not get the name changed because an entry named <newname> already exists.

***rename <oldname>.<newname> unknown
No entry named <oldname> was found.

***rename <oldname>.<newname> protected
The job was not allowed to change the name of the entry.

***rename <oldname>.<newname> entry in use
The entry could not be renamed because another job was using it.

In the last four cases the program continues with the next parameter.

REPEAT

The program makes it possible to repeat (a specified number of times) a series of FP-commands placed in brackets.

Example:

By the following FP-commands files number 1 to 20 on mt471100 and mt471200 are checked by to program COPY (which outputs the number and sum of characters for each of the 40 files):

```
t1=set mto mt471100
t2=set mto mt471200
(repeat 20
nextfile t1 t2
copy t1 t2)
```

Call:

{<outfile>=} ({<FP-command>}$_0^{\infty}$ repeat <s> <total number of times> <parameter list> <NL> {<FP-command>}$_1^{\infty}$)

<total number of times> ::= an integer greater than 0
<parameter list> ::= any sequence obeying the FP-syntax

Function:

The program augments the command stack so that the rest of the compound command containing the call of repeat, will be executed the specified number of times.

<outfile> and <parameter list> have no effect at all, but in mode list.yes they may be used to identify the repeat call to be executed.

Storage requirements:

512 bytes plus space for FP.

Error messages:

***repeat no core
There is no room in the process area for the augmentations of the command stack made by repeat. (The command to be repeated must be exceptionally long.)

***repeat no factor
Either there are no right hand parameters to the call or the first right hand parameter is not an integer.
***repeat factor 0
The integer <total number of times> is equal to 0
***repeat nothing to repeat
The call of repeat is the last command in the compound command containing repeat.

In case of error messages, the commands following the repeat call will be executed once.

REPLACE

Sends a replace message to the parent (the operating system) defining a file as replacement for the current job file. After termination of the job BOSS will create a new job with the same name and the specified file as job file. BOSS accepts only replace messages from off-line jobs, not from on-line jobs.

Example:
The FP-command
    replace pip
defines the file 'pip' as replacement for the job file.
The FP-command
    replace pip newid
defines the file 'pip' as replacement for the job file and the identification to be changed according to the job head in the file 'pip'.

Call:
    replace <s> <job file> <s> $\left\{\begin{matrix}\text{oldid}\\ \text{newid}\end{matrix}\right\}_{\circ}^{1}$

where <job file> is a name of a permanent bs-file (cf. the BOSS2 User Manual p. 4-2). Oldid is standard.

Function:
A replace message containing the specified name is sent to the parent.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:
***replace call
    Left hand side in the call of the program.

***replace <parameter list> parameter error
    Parameter error in the call of the program.

***replace <parameter list> not allowed from on-line job
    The replace message was not accepted as the job is an on-line job.

In case of any error no replace message is sent.

RING

Sends a 'mount ring' message to the parent (the operating system). The program is normally not used as the software sends the mount ring message automatically when needed.

Call:

ring <s> <name>

Function:

A 'mount ring' message is sent to the parent. The name in the message is found as follows: The name is looked up in the catalog. If an entry describing a magnetic tape file (kind=18) is found and if this entry is not protected (e.g. not of scope system) the document name in the entry is used. Otherwise the name specified is used.

Storage Requirements:

1536 bytes plus space for FP.

Error messages:

***ring call
Left hand side in the call of the program.

***ring <parameter list> parameter error
Parameter error in the call of the program.

***ring <name> tape name missing.
The entry specified has a zero document name.

In case of any error no mount ring message is sent.

Rubs out the contents of the specified backing-storage files. If demanded the catalog entry is removed after the cleaning.

Examples
By the FP-command

```
rubout user clear.yes text4
```

the file text4 is filled with a fill-pattern after which the entry is cleared.
The following two FP-commands

```
rubout user text4
rubout user clear.no text4
```

are identical, since the clear parameter is initialized to no.
The entry is not removed.

Call:

rubout <s> <scope> { <s> { <name> / clear. { yes / no } } }$_0^\infty$

<scope> ::= { temp / login / user / project / own }

Function:
The files are filled with a fill-pattern after which it is cleared in case the value of the parameter clear is yes. The fill-pattern is a long consisting of 3 NUL-characters and 3 EM-characters.
Scope own means all of temp, login, user and project.

Storage requirements:
1536 bytes plus space for FP.

Error messages:
***rubout call
The program was called with a left hand side.
No file rubout.
***rubout param <parameter>
Illegal parameter.
The rest of the parameter list is skipped.
***rubout <scope> illegal scope
The scope was illegal.
No file rubout.

***rubout <scope> <name> unknown
The entry was not found.
The program continues with the next parameter in the list.
***rubout <scope> <name> entry in use
The entry was not changed or removed because another job was using it.
The program continues with the next parameter in the list.
***rubout <scope> <name> not bs-area
The entry did not describe a backing-storage area.
The program continues with the next parameter in the list.
***rubout <scope> <name> catalog error
Catalog- monitor- or hard error.
The rest of the parameter list is skipped.

SAVE

The program can output catalog entries and bs-files to magnetic tape files for later reestablishment by the program LOAD.

Example:

All catalog entries and bs-files of scope login are output to mt4471100 file 2 by the FP-command:

```
save mt4471100.2
```

In case: t=set mto mt4471100 0 2
the same is obtained by the command:

```
save t.0
```

All catalog entries specified in the bs-area savefiles are saved, destroying a possibly earlier saved file 2, by the command

```
save mt4471100.2.label.account4 in.savefiles
```

(as a standard the label parameter should always be used on file 1).
File 2 and the following will be labelled account4.
All catalog entries and corresponding bs-files of scope project and scope user and the entry named pap (in case several entries are named pap, the entry of the smallest scope will be selected) are saved by the FP-command:

```
save mt4471100.2 scope.project scope.user pap
```

Compare with the examples under the program LOAD. See also further examples.

Call:

$\{$ <outfile>= $\}_0^1$

save $\{$ <mountparam>. $\}_0^1$ $\{$ <tape parameter> $\}_1^2$ <special param> <save spec>

<mountparam> ::= $\{$ mountspec.<deviceno> $\}_0^1$ $\{$ <modekind> $\}_0^1$ $\{$ release.$\{$yes / no$\}$ $\}_0^1$

<tape parameter> ::= <tapename>.<fileno> $\{$.<tapename for next volumen> $\}_0^9$
$\{$.label.<dump-label name> $\}_0^1$

<tapename> ::= <filedescriptor describing a magnetic tape file>
<name of magnetic tape>

<fileno> ::= last
<integer>

<special param> ::= $\{$ list.$\{$yes / no / name$\}$ / segm.<integer> $\}$

<save spec> ::= { <modifiers> | <kit spec> | <entry spec> }

<modifiers> ::= { changekit.<bs device name>.<bs device spec> | newscope.<newscope spec> }

<kit spec> ::= kit.<bs device name>

<bs device name> ::= { main | <name of bs device> }

<bs device spec> ::= { <bs device name> | 0 | 1 }

<entry spec> ::= { <name> | <name>.scope.<scope spec> | docname.<docname> | docname.<docname>.scope.<scope spec> | scope.<scope spec> | in.<bs-file> }

Function:

After label check (see below) the catalog entries and bs-files specified by <save spec> are saved on the magnetic tape file.

If <entry spec> is empty, the program acts as if the parameter scope.login was specified.

A possible dump-label is read and listed on current out. If it is a version label (see Tape Format) and a label parameter is not specified, the program terminates.

After the label check a version label containing current date and hour is written, overwriting the former one and also listed on current out.

The program terminates by writing an empty-label on the following file.

Function, mountparam

If no mountparam is specified, the program will use a standard magtape station, modekind=mto (or in case mtname is a filedescriptor, then the modekind of this filedescriptor) and the tape will be released at end of program.

e.g. mountspec.10.nrz.release.no.
mountspec.10.
nrz.
release.no.

Function, tapeparameter.

In case mtname is a filedescriptor, filenumber will be understood relative to the filenumber in the filedescriptor. The modekind of the filedescriptor will be used.

If <fileno>=last, the first file which does not contain a version label is saved. This parameter gives a longer run time than an integer parameter.

If two tape parameters are specified, they are treated completely in parallel and after saving the two magnetic tape files will contain exactly the same information, except for the tapenames.

Function, special param

list.{yes / no / name} Std. is list.yes, all loaded entries are listed.
If list.no, the entries are not listed.
If list.name, then only the names of the entries are listed.
If an outfile is specified, this file is used for output, otherwise current output file is used.

segm.<integer> Integer must be between 1 and 9. Integer segments are saved in each magnetic tape block. Std. is 1 segm.

Function, modifiers

changekit.<actual kit>.<saved kit>
Each entry on <actual kit> will be saved with bs device name <saved kit>.
changekit.main.main, means: change all devices to main.
For <bs device spec> = 0 or 1, see LOAD.

newscope.<newscope spec>
Std. is newscope.std, meaning no change in scope.
This parameter is valid for the following entry specifications. They will all be saved with <newscope spec>.
newscope ::= temp |login |user |project |std
e.g. newscope.temp

Function, kit spec

kit.<saved kit> Std. is main, meaning all devices.
Only area entries with this kit/docname and non-area entries which are permanented into this device will be saved. The parameter is valid for all following <entry spec> until a new kit parameter is specified.
e.g. kit.disc2

Function, entry spec

<scope spec> ::= temp |login |user |project |own |system |perm |all

<name> The entry of the name is saved. If several entries exists, only the entry of smallest scope is saved. <name> must not be one of the following reserved names: c v primout fp

scope.<scope spec>
All entries with this scope are saved.

<name>.scope.<scope spec>
An entry of specified name and scope is saved.

docname.<docname>
All entries with specified docname (maybe kitname) are saved.

docname.<docname>.scope.<scope spec>
All entries with specified docname (maybe kitname) and specified scope are saved.

in.<bs-file> <save spec> is read from <bs-file>. In this file a <NL> is allowed as separator between the parameters.

The parameters

scope.all
and
scope.perm

meaning all or all non-temporary files (contained in the standard base, i.e. temp base) are intended for system maintenance.

Entries saved by these parameters are identified by base and permanent key, whereas other entries are identified by scope and these may thus be transferred to another user or project.

Tape format:

The first block of a save file and the first block in the file after a save file is always a dump-label (the last one may have been destroyed later), containing 25 double words holding an iso textstring terminated by <EM>, so that it can be read by edit.

1: <:dump :>
2-3: <tapename normalized by spaces>
4: <<zdd >,filenumber
5: <:empty :>, <:vers. :> or <:cont.:>
6: <<ddmmyy>,date
7: <<.hh >,hour
8: <:s=<segment per block>:>
9-10: <dump-label name>

```
   11:   <:   in.:> or <:      :>
12-13:   <bs file name> or <:      :>
   14:   <:<NL><EM>:>
15-25:   <::>
```

According to double word no. 5, the dump-label is called an empty label, a version label. or a continuation label.
In case <save spec> is read from a file, double words 11-13 contain the name if this input file.

The rest of the save file consists of logical records written with the physical blocklength 25 or 2+128*segm double words. The records are of the following 4 types:

entry-record: (heading each entry)

```
    1:   1, if system dump then 52 else 48
    2:   entryno, number of segments
  3-4:   entry name
  5-9:   entry tail
   10:   if system dump then permanent key
         else scopekey
         (3=system, 5=project, 6=user, 7=login, 8=temp)
11-12:   bs device spec
   13:   if system dump then (entry base low, entry base up)
         else (1,48)
14-25:   if system dump then (1,52) else (1,48)
```

segment-record: (containing a saved segment)

```
    1:   2,8+512*segm
    2:   entry number, segment number
3-130:   contents of one segment
131-258: contents of one segment
...
```

end-record: (terminating each file)

```
    1:   3,8
    2:   total number of entries, total number of segments
 3-25:   3,8
```

continuation-record: (when a tape overflows)

```
    1:   4,16
    2:   entry number, total number of segments until now
  3-4:   name of next tape
 5-25:   4,16
```

Example of picture of file on tape:

After save of the two files ta1 (3 segments, scope temp) and ta2 (2 segments, scope login) saved by the call:

```
    save mt123456.1.label.picture ta1 ta2
```

the tape mt123456 looks as follows

```
file 1
(block 0 - version label)
<:dump  mt123456    001   vers. 130473.12   s=1 picture              :>
```

```
(block 1 entry record)
1,48  1,3  <:ta1:>  <entry tail of ta1>  8  disc  1,48  1,48  ...
(block 2 - segment record)
2,520  1,0  <contents of 0. segment of ta1>
(block 3 - segment record)
2,520  1,1  <contents of 1. segment of ta1>
(block 4 - segment record)
2,520  1,2  <contents of 2. segment of ta1>
(block 5 - entry record)
1,48  2,2  <:ta2:>  <entry tail of ta2>  7  disc  1,48  1,48  ...
(block 6 - segment record)
2,520  2,0  <contents of 0. segment of ta2>
(block 7 - segment record)
2,520  2,1  <contents of 1. segment of ta2>
(block 8 - end record)
3,8  2,5  3,8  3,8  ...
FILEMARK
file 2
<:dump  mt123456    002    empty 130473.12    picture                    :>
```

Multivolumen file conventions:

When end of tape is encountered during save, a continuation-block is written, and the next tape of the tape list is mounted.
The continuation tapes contain no dump-label.
When a continuation-block is encountered during LOAD, the next volumen is mounted and the run continued. If a tape-parameter describing the volume is present it will be used, otherwise the name in the continuation record is used.
Example:

```
    save mt1.1.mt2 mt3.1.mt4
```

saves two copies each on two volumes.

```
    load mt1.1.mt2
```

or

```
    load mt1.1
```

loads from the first copy, the second volume of which is mt2.

```
    load mt1.1.mt4
```

loads from volume 1 of the first copy and volume 2 of the second copy.

Storage requirements:
12000 bytes.

Error messages:

***save error in tapeparam <erroneous and following parameters>
    Parameter error in the call. The program terminates.
    The magnetic tape is unchanged.
***save: error in modekind spec.
    <tapename> describes an entry of kind 18, but mode is neither 0 nor 4.
***save, infile <name> unknown
    The program terminates. The magnetic tape is unchanged.
***save error param <erroneous and following parameters>
    Parameter error in the call. The program terminates.
    The magnetic tape is unchanged.

***save mode error
The tape is not in the mode defined by <tapename>.
The program terminates. The magnetic tape is unchanged.
***save dumplabel <specification>
Error in the specified part of the dumplabel.
The program terminates. The magnetic tape is unchanged.
<name> bad area: <pattern>
Hard error during run. The pattern shows the status word.
<name> bad area, segm. saved = <segmno>
Number of segments saved does not correspond to the size specified in the catalog.
<name> entry in use.
<name> is in use and cannot be saved.
<name> not allowed
<name> must not be c v primout fp
<name> entry inconsistent
<name> code inconsistent
The date of an external procedure is incorrectly described either in the catalog entry or in the code. The entry is saved.
***not found: <entry spec>
<entry spec> was not found in the catalog.
***save not ok <d>
This message occurs at program exit in case of any error.
<d> is the number of reasons for the alarm.

Further examples:

1) The programmer wants to save all his scope user files and his two files data1 and data2, so that they will be loaded as scope login:

```
save mt471100.1.label.try newscope.login scope.user data1 data2
```

The files can be loaded by:

```
load mt471100.1
```

2) The programmer wants to switch his files on kit1 and kit2 and move his disc files to kit3:

```
save mt471100.1.label.switch changekit.kit1.kit2 changekit.kit2.kit1,
     changekit.disc.kit3 kit.kit1 scope.own kit.kit2 scope.own,
     kit.disc scope.own
```

The file can be loaded by:

```
load mt471100.1
```

SCOPE

Changes the scope of catalog entries as specified in the call of the program.

Example:
By the FP command
scope user pip
the scope of the catalog entry named 'pip' is changed to 'user'. The catalog entry is now a permanent entry and is not removed when the job terminates.

Call:
scope <s> <scope spec> $\{$<s> <name>$\}_1^{\infty}$

<scope spec> := <scope> $\{$. <device name>$\}_0^1$

<scope> ::= { temp | login | user | project }

<device name> ::= <name of drum or disc kit>

Function:
The scope specification is interpreted and then the name list is scanned. For each name a catalog lookup is made and the scope of the entry found is changed to the specified scope. The entry may hereby replace a catalog entry with the same name (this 'old' entry is removed from the catalog).
Remark: if several entries with the same name are present, the catalog lookup will find the entry with the 'smallest' scope (corresponding to the order: temp, login, user, project).

Scope specification:
The concept of scope of a catalog entry is explained in the BOSS2 User Manual ch. 4.1.
A device name in the scope specification means that the catalog entry should be permanented into the auxiliary catalog on the device mentioned and thereby occupy permanent claims on the device mentioned, but not in the main catalog.
This is meaningful for the scopes user and project only and the entry should be either a non-area entry or an area entry where the data area is situated on the specified bs device.

Storage requirements:
2048 bytes plus space for FP.

Error messages:
***scope call
Left hand side in the call. The program is terminated without further actions.
***scope <scope spec> illegal scope
The scope specification is illegal.
***scope <scope spec> bs device unknown
The bs device specified in the scope specification is not on the computer.

In all cases above the program terminates without changing the scope of any catalog entry.

***scope param <parameter>
Parameter error in the call. The rest of the name list is skipped.
***scope <scope spec> <name> unknown
No entry with the given name was found. 1).
***scope <scope spec> <name> protected
The job was not allowed to change the scope of the entry found. 1).
***scope <scope spec> <name> entry in use
Another job was using the entry and hence the scope could not be changed. 1).
***scope <scope spec> <name> no resources
The resources of the job did not allow the change of the entry scope 1).
***scope <scope spec> <name> change bs device impossible
The entry could not be permanented into the specified auxiliary catalog. 1).
***scope <scope spec> <name> catalog error
catalog error, monitor error or hardware error.

1) The program continues with the next name in the name list.

SEARCH

Finds and lists all catalog entries with a given scope.

Example:
By the FP command
search user
one gets a list on current output of all catalog entries which have scope user under the actual job.
By the FP command
search own
one gets all entries with scope temp, login, user or project listed on current output.

Call:

{<out file> =} search <s> <scope spec>

<scope spec> ::= <scope> {. <device name>}$_0^1$

<scope> ::= { temp | login | user | project | system | own }

<device name> ::= <name of drum or disc kit>

Function:
The catalog is scanned and all catalog entries with the specified scope are listed. If an <out file> is present it is used for the output - otherwise current output is used.

Scope specification:
The scope concept is explained in the BOSS2 User Manual, chapter 4.1
The scope own means belonging to the project and available for the

job i.e. all of temp, login, user or project (cf. the example above). If a device is specified, only area entries where the data area is on this device and non-area entries which are in the auxiliary catalog on the device are listed.

Output format:
Each entry found is listed exactly as described under the program LOOKUP.

Storage requirements:
2560 bytes plus space for FP.

Error messages:
***search connect <outfile>
The output file could not be connected - current output is used instead.
***search param <parameter>
Parameter error in the call. No entries listed.
***search <scope spec> no entries found
No entries with the specified scope (and specified device) was found.
***search <scope spec> illegal scope
Incorrect scope specification. No entries listed.

SET

Creates a new catalog entry with scope temp or changes an already existing entry (with scope temp) according to the parameters.

Example:
An area entry named 'pip' with an area size of 20 segments on the bs device 'disc3' and with date now, is created by the FP command:

```
pip=set 20 disc3
```

(Actually the area may get a slightly larger size because the size is always a multiple of the slice length on the device, cf. ref. 3).
A non-area entry 'file7' which may serve as file descriptor for file 7 on the magtape with name 'mt314711' is created by the FP command

```
file7=set mto mt314711 0 7
```

An area named 'image' on disc (intended for core store dump) is created by the FP command

```
image=set 40 1 0 0 0 7.0
```

(The parameters 0 0 0 7.0 may be omitted as BOSS will automatically set contents 7 when the dump is made).

Call:

<result name> = set {<s> <kind> {<s> <kit/doc name> {<s> <free param>
{<s> <file> {<s> <block> {<s> <contry>
{<s> <length>}. }. }. }. }. }. }.

<kind> ::= { <integer> | <integer1> . <integer2> | <mode kind abbreviation> }

<kit/doc name> ::= { <name> | 0 | 1 }

<free param> ::= { <integer> | <integer1> . <integer2> | d. <isodate> | d. <isodate> . <clock> }

<isodate> ::= { yymmdd | 0 } 0 is interpreted as now

<clock> ::= <hhmm>

{ <file> | <block> | <contry> | <length> } ::= { <integer> | <integer1> . <integer2> }

Function:
The parameters are interpreted as described below yielding the wanted entry tail. Next, creation of the catalog entry <result name> with this

tail is attempted. If the result is 'entry already exists' (cf. ref. 2 and 3) the existing entry is changed to get the entry tail wanted.

Each element in the entry tail except <kit/doc name> is a 24 bits word.
1. <integer> : The integer is placed in the tail
2. <integer1> . <integer2> : Is interpreted as two bytes i.e. as the binary number <integer1> shift 12 + <integer2>
3. <mode kind abbreviation> : Only relevant for <kind>. The table of mode kind abbreviations is scanned and the value found is used.
4. <name> : Only relevant for <kit/doc name>. The name is placed in the tail.

If the parameter list does not specify all of the tail, the rest is set to zero.

When an area entry is created, the bs device is determined by <kit/doc name>:
If <kit/doc name> is 0, the area is - if possible - created on a drum, otherwise on a disc.
If <kit/doc name> is 1, the area is - if possible - created on a disc, otherwise on a drum.
If <kit/doc name> is a name, the area is created on the bs device with this name.

Storage requirements:
1536 bytes plus space for FP.

Error messages:
***set call
No left hand side in the call.
***set param <parameter>
Parameter error in the call.
***set <result name> change kind impossible
Change of an area entry to a non-area entry or vice versa was attempted.
***set <result name> change bs device impossible
A change of <kit/doc name> on an area entry was attempted.

***set <resultname> bs device unknown
The bs device specified was not on the computer.
***set <result name> no resources
The resources of the job did not allow the wanted creation of a catalog entry.
***set <result name> entry in use
The entry could not be changed because another job was using it.

If any error message appears, no entry is created or changed.

SETMT

Creates catalog entries of scope temp describing files on magnetic tape according to the parameters.

Examples:
The FP-command
```
pap=setmt mt004711.3
```
creates the same catalog entries as the FP-commands:
```
pap1=set mto mt004711 d.0 1
pap2=set mto mt004711 d.0 2
pap3=set mto mt004711 d.0 3
```
If t is a name of a file on this magtape, e.g.
```
t=set mto mt004711
```
the same result is obtained by
```
pap=setmt t.3
```
The FP-command:
```
f=set nrz mt004711 0 2
f=setmt f.3.5
```
creates the same catalog entries as the FP-commands:
```
f3=set nrz mt004711 d.0 5
f4=set nrz mt004711 d.0 6
f5=set nrz mt004711 d.0 7
```

Call:
```
<result name> = setmt <mtname>.  <upper integer>
                                 <lower integer>.<upper integer>
```
If no <lower integer> is specified, it is set to 1.

Function:
Entries describing files on the magnetic tape <mtname> are created with names <resultname> followed by <lower integer> to <upper integer>. If a temporary entry already exists, it is first removed.
The mtname is looked up in the catalog. If an entry describing a magnetic tape file (kind=18) is found, the document name and the modekind of this entry is used and the files will be addressed relative to the file number. Otherwise the name specified is used.

Storage Requirements:
512 bytes plus space for FP.

Error Messages:

*** setmt call
No left hand side or left hand side of more than 9 characters

*** setmt param
Parameter error in the call, e.g. an integer greater than 99

*** setmt <resultname> no resources
The resources of the job did not allow creation of the catalog entry

*** setmt <resultname> catalog error
Error in catalog, monitor or hardware

In case of any error message the program terminates.

SKIP

Bypasses parts of current input as specified in the parameter list.

Example:

```
(test2=edit text1
skip æ)
1./error/,r/er/err/
12,r/sory/sorry/
f
æ
```

In case of an error during editing the remaining edit commands will be skipped, i.e. when skip is called the input position in current input is forwarded to just after the second æ character.

Call:

skip { <s> <lines> | <s> <iso value>.<appearances> | <s> <small letter> }$_{1}^{\infty}$

<lines> ::= <integer>

<iso value> ::= { <integer> | <small letter> }

<appearances> ::= <integer>

Function:
The program interpretes one parameter at a time and skips current input as follows:

<lines> This number of graphical lines are skipped.

<iso value>.<appearances>
Skips until the specified number of appearances of the iso character are bypassed.

<small letter>
Skips up to and inclusive this letter.

Storage Requirements:
1024 bytes plus space for FP.

Error Messages:

***skip call
An output file has been specified in the call. This is ignored.

***skip param <illegal parameter>
Illegal parameter syntax. The parameter is ignored.

***skip end medium
Current input is exhausted. The program is terminated. Notice: current input is not unstacked.

SUSPEND

Sends a suspend message to the parent (the operating system) asking for suspension of the specified magnetic tape reel. This is relevant for worktapes only. The station is now available for mounting of another tape reel but the suspended worktape is still reserved for the job until it terminates or releases the tape reel. Each suspend operation uses a suspend buffer. (cf. the BOSS2 User Manual, ch. 5 and 10).

Example:
A worktape has been mounted by the FP-commands

```
workfile=set mto 0 0 1
mount workfile
```

The job has produced some output on 'workfile' but needs now the station for another purpose. The worktape is therefore suspended by the FP-command

```
suspend workfile
```

When the name 'workfile' is referred to later in the job, the worktape is demanded. In the meantime no other job is allowed to use the tape.

Call:

```
suspend <s> <name>
```

Function:
A suspend message is sent to the parent. The name in the message is found as follows: The name is looked up in the catalog. If an entry describing a magnetic tape file (kind=18) is found and if this entry is not protected (e.g. not of scope system) the document name in the entry is used. Otherwise the name specified is used.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:

***suspend call
    Left hand side in the call of the program.
***suspend <parameter list> parameter error
    Parameter error in the call of the program.
***suspend <name> tape name missing
    The entry specified has a zero document name.

In case of any error no suspend message is sent.

TIMER

Sends a timer message to the parent (the operating system) demanding a provoked interrupt after a certain time. The use of the program is described in details in the BOSS2 User Manual ch. 10.

Example:
The FP-call

```
timer 30 2
```

will provoke an interrupt after 30 seconds.

Call:

```
timer <s> <run time> <s> <break time>
```

where <run time> and <break time> are integers, denoting time in seconds.

Function:
A timer message containing the two integers is sent to the parent.

Storage Requirements:
1536 bytes plus space for FP.

Error Messages:

***timer call
Left hand side in the call of timer.

***timer <parameter list> parameter error
Parameter error in the call of timer.

In case of any error no timer message is sent.