RC9000-10/RC8000

SW8010 System Utility

User's Guide, Part Two

PN 99200389

RC Computer

Keywords:
RC9000-10, RC8000, System Utility, Utility Programs

Abstract:
This manual describes the programs in the System Utility package that are labelled *utility programs*.

Date:
February 1989

PN: 991 11264

Table of Contents

1. Introduction

System Utility, User's Guide, Part One gives a general introduction to the file processor and utility program system and a detailed description of certain important features of the system.

This manual, *System Utility, User's Guide, Part Two*, contains detailed descriptions of the individual programs.

Chapter 2 contains a list of programs together with a short abstract of each program. The list is divided in three sections, one for each category of programs: catalog handling programs, data handling programs, and job control programs.

Chapter 3 contains the detailed descriptions of the programs, one section for each, and all in alphabetic order.

2. Abstracts

The System Utility Programs may be divided up in categories according to their functionality as below.

* means more than one functionality according to below categories

@ means strictly BOSS functionality

2.1 Catalog Handling Programs

2.1.1 Creating Entries

assign Creates or changes a temporary entry so that the tail of the two specified entries become identical.

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 entries.

set * Creates a new catalog entry with scope temp or changes an already existing entry (with scope temp) according to the parameters.

setmt Creates catalog entries of scope temp describing files one magnetic tape according to the parameters.

2.1.2 Changing Entries

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

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.

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 entries.

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

permanent Changes the permanent key of the specified entry to the specified integer.

rename Changes the names of catalog entries as specified.

scope Changes the scope of catalog entries as specified in the call of the program.

set * Creates a new catalog entry with scope temp or changes an already existing entry (with scope temp) according to the parameters.

2.1.3 Looking Up Entries

lookup Finds and lists catalog entries with specified name.

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

search * Finds and lists all catalog entries with a given scope.

translated Prints the data of translation which is found in all ALGOL/FORTRAN programs.

2.1.4 Surveying Catalogs

cat Works as catsort, except that no sorting of entries prior to output takes place, i.e. the entries specified in the call are listed in the order in which they are found in the catalog

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.

search * Finds and lists all catalog entries with a given scope.

2.1.5 Removing Entries

clear Removes catalog entries with name and scope as specified.

clearmt Removes catalog entries according to the parameters.

2.2 Data Handling Programs

2.2.1 Creating Data in a File

binin * The program can input files generated by the program "binout". The program "binin" and "binout" are primarily used when binary files are stored on paper tape.

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". The program can furthermore output autoload tapes.

char Outputs the specified character the specified number of times.

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

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

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

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

2.2.2 Changing Data in a File

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.

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

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

2.2.3 Moving Data Between Files

binin * The program can input files generated by the program "binout". The program "binin" and "binout" are primarily used when binary files are stored on paper tape.

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". The program can furthermore output autoload tapes.

compresslib Compresses into one single area a number of external ALGOL / FORTRAN procedures or code procedures to occupy a minimum number of segments.

copy 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).

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

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

2.2.4 Verification of Data in a File

binin * The program can input files generated by the program "binout". The program "binin" and "binout" are primarily used when binary files are stored on paper tape.

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

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.

translated * Prints the data of translation which is found in all ALGOL/FORTRAN programs.

2.2.5 Backup and Restore

load The program restores files from a backup made by *save*, cf. [13].

load13 The program can input catalog entries and bs-files from a magnetic tape file generated by the program *save13*.

save The program performs backups of files and file systems, cf. [13].

save13 The program can output catalog entries and bs-files to a magnetic tape file for later reestablishment by the program *load13*.

2.3 Job Control Programs

2.3.1 Job Flow Control

claimtest * Checks the claims of the calling process according to the call parameters and leaves the ok bit true if the claims specified are present, false otherwise.

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 sec. 1.3, newjob and replacejob, in ref. 10.

corelock * @ Sends a corelock message to the parent (the operating system) demanding that the job should stay in the core the specified number of seconds. This feature is only used in connection with process control devices producing data with a high rate, cf. ref. 10, secs. 3.4 and 6.7.

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.

end * Returns current input to previous current input at the positions where it was left.

finis Finis terminates the job.

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 end (or it may correspond to the mode bits as set by a call of the program "mode").

init Forces a reinitialization of FP.

job @ Makes it possible to use tapes containing a BOSS job request in runs directly under "s".

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

newjob Sends a newjob message to the parent (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. 1.3, newjob and replacejob, in ref. 10.

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. ref. 10, sec. 3.2).

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

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. A replace message from an on line is not accepted by BOSS.

skip Bypasses parts of current input as specified in the parameter list.

timer @ Sends a timer message to the parent (the operating system) demanding a provoked interupt after a certain time.

2.3.2 Job Mode Control

corelock * @ Sends a corelock message to the parent (the operating system) demanding that the job should stay in the core the specified number of seconds. This feature is only used in connection with process control devices producing data with a high rate, cf. ref. 10, secs. 3.4 and 6.7.

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.

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. ref. 10, sec. 3.2).

2.3.3 Input / Output Control

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. ref. 10, ch. 3 and sec. 6.2).

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

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").

o Selects a new file as current output.

2.3.4 Name Base Control

base Changes the catalog base of the job process, thereby changing the name base, and displays the process bases of the job process.

2.3.5 Resource Control

claim Lists selected parts of the bs area claims of the process together with area, buf, size and first core.

claimtest * Checks the claims of the calling process according to the call parameters and leaves the ok bit true if the claims specified are present, false otherwise.

2.3.6 Device Control

change @ Sends a change paper message to the parent (operating system). The program is only used when a job executed under BOSS uses job controlled printer. (cf. ref. 10, section 6.2).

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. ref. 10, ch. 3 and sec. 6.2).

kit @ Sends a mount disk message to the parent (the operating system) demanding a disk kit with a specified name to be mounted on a specified disk unit (cf. ref. 10, ch. 3 and sec. 5.3).

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. ref. 10, sec. 6.1). The program does not await the mounting, unless there is asked for mounting of an unspecified worktape.

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 number (cf. ref. 10, sec. 6.1).

release * Sends a release message to the parent (the operating system) tape reel (cf. ref. 10, sec. 6.1). releasing the specified magnetic

rewind Sends a rewind operation to the magnetic tape process, allowing the job process to continue while the tape is rewinding.

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

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. ref. 10, sec. 6.1).

unload Sends an unload operation to the magnetic tape process, allowing the job process to continue while the tape is unloading.

2.3.7 Operator Interaction Control

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. ref. 10, sec. 6.1). The program does not await the mounting, unless there is asked for mounting of an unspecified worktape.

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 number (cf. ref. 10, sec. 6.1).

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.

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.

2.3.8 Accounting

account @ Sends an account message to the parent (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.

3. The Programs

3.1 account

Sends an account message to the parent (the operating system) which is then expected to produce a record in the account file. Only used when jobs running under BOSS wants to produce special account information.

3.1.1 Call

```
                                                    1-3
account <account kind> {<integer>}
```

where the parameters <account kind> and <integer> are integers.

3.1.2 Function

The program sends an account message containing the integers.

3.1.3 Storage Requirements

1536 halfwords plus space for FP.

3.1.4 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 message no account record is produced.

3.2 assign

Creates or changes a temporary entry so that the new entry becomes a sub entry to the old one, if the old one is an area entry, and the two entries become identical if the old one is a non-area entry. The program is used together with the programs entry and nextfile.

3.2.1 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 t
```

The calls:

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

will set progfile equal to the current value of tape.

3.2.2 Call

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

3.2.3 Function

```
<oldname> ::= <name>
              <apostrophized name>
              <generalized name>
              <general text>
```

If <oldname> is an area entry, <resultname> will become a bs-entry, i.e. modekind = 1<23+4 and document name = <oldname>. If <oldname> is a non-area entry, <resultname> will become identical to <oldname>.

3.2.4 Storage Requirements

1536 halfwords plus space for FP.

3.2.5 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 <reultname> 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 disk/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> no room**
The wanted creation of an entry was not possible because the name overflow for the new name in the main catalog exceeded the limit.

*****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.

3.3 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 decribe file 2 and new to describe file 0 and set the warning bit to yes. A following call will change old to descibe file 1 and leave new unchanged - the ok bit is set to no.

If the entry t describes file no. 7 on magtape mt310514 the call

```
backfile t t t t t t
```

will make it describe file no. 1.

3.3.1 Call

```
backfile ( <name> ) 1-*
```

3.3.2 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.

3.3.3 Storage Requirements

512 halfword + space for FP.

3.3.4 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.

3.4 Base

The program changes the catalog base of the job process according to the parameters in the call.

The catalog, standard, user and max bases of the process may be listed on current output before as well as after the change of catalog base.

The catalog base may be changed to any interval which is contained in (or equal to) the max base and which also contains the standard base, equals it or is contained in the standard base.

3.4.1 Call

```
base {what}0-* [<specifier>] [<modifier>] [what]

<specifier> ::= std | user | max | abs

<modifier>  ::= <modif1> [<modif2>]

<modif1>    ::=

<modif2>    ::= <integer> | <int1>.<int2>

<int1>      ::=

<int2>      ::= <integer>
```

However, an empty modifier to the specifier <abs>, i.e. the call:

```
base abs
```

will not be accepted.

3.4.2 Function

If the parameter <what> is met before a possible specifier/modifier, the current values of the catalog base, user base and max base of the job process are listed on current output.

The catalog base of the job is changed to the interval specified. The interval is found as follows:

1) the specifier is determined:

`no specifier`	`the lower limit of the current catalog base`
`std`	`the lower limit of the current standard base`
`user`	`the lower limit of the current user base`
`max`	`the lower limit of the current`

```
                          max base
abs                       the integer zero
```

2) The new interval is derived from the specifier and a possible modifier:

```
no modifier               lower limit, upper limit of the
                          specified base
<mofif1>                  specifier + <modif1>, specifier +
                          <modif1>
<modif1> <modif2>         specifier + <modif1>, specifier +
                          <modif2>
```

The parameter <int1>.<int2> is interpreted as <int1> shift 12 + <int2>.

By specifying <int1> greater than 2047, a negative modifier will be specified.

This may also be done by specifying an integer greater than 8 388 607, e.g. the modifier -1000 may be specified as 16 777 216 - 1000 = 16 776 216.

However, negative integers should of course be specified as signed integers.

Note that an ampty specifier/modifier defines the current catalog base, i.e. no change.

If the parameter <what> is met after a possible specifier/modifier, the values of the bases after the changes of catalog base are listed on current output.

3.4.3 Storage Requirements

512 halfwords plus space for FP.

3.4.4 Error Messages

***** base interval**
The interval specified is not a legal catalog base for the job process. The catalog base remains unchanged.

***** base param**
Parameter error in the call of the program. The catalog base remains unchanged.

3.4.5 Examples

3.4.5.1 base user

The call

```
base user
```

will change the catalog base to equal the user base.

3.4.5.2 base abs

The call

```
base abs 1532 1584
```

will change the catalog base to the interval 1532, 1584, provided it is a legal catalog base for the process.

3.4.5.3 base 1

The call

```
base 1
```

will change the catalog base to the interval lower catalog base + 1, lower catalog base + 1, provided it is a legal catalog base.

3.4.5.4 base what

The call

```
base what
```

will write

```
cat base          :          420 429
std base (login)  :          420 420
user base         :          420 429
max base (project) :         400 499
```

provided the current bases have these values.

3.4.5.5 base what std 0 3 what

The call

```
base what std 0 3 what
```

will write

```
cat base          :          420 429
std base (login)  :          420 420
user base         :          420 429
max base (project) :         400 499

cat base          :          420 423
std base (login)  :          420 420
```

```
user base          :          420 429
max base (project) :          400 499
```

provided the bases of the process initially have the above values.

3.5 binin

The program can input files generated by the program "binout", verifying the contents of the file by checksum control.

3.5.1 Example

A magnetic tape file, described in the catalog by the entry t, was produced by the FP command:

```
t=binout fup
```

It may be loaded by the FP command

```
binin t
```

and thereby the catalog entry 'fup', the area and its contents are reestablished.

3.5.2 Call

```
[<outfile> =]
binin [list.(yes/no)] [disk.<disk>]
<binout file> [.<modifier>]

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

<disk>            ::= (<name> )
                      (0/1/2/3)

<name>            ::= <name of a disk>

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

                      (   <binout segments> )
<modifier>        ::= ( s.<binout segments> )
                      ( c.<binout segments> )

<binout segments>::=
(<no of binout segments>                              )
(<no of binout segments> . <first binout segment>)

<no of binout segments>::=
<first binout segments>::= integer
```

3.5.3 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 decribed 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) <outfile> 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 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. All files created by the commands in the command segment are created on the disk with the name 'disc', unless otherwise specified by the disk.<disk> option.

2) <outfile> **is** present.
   All binout segments of the binout files are interpreted as load segments and loaded to the file specified by <outfile>.

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

If a disk is specified by disc. <name>, all entries will be created on that disk, if specified by disk.(0...3), each entry will be created on the disk with the most resources of the specified permanens, cf. 3.5.5.

Default: disk.disc

3.5.4 Modifier

<binout segment> This modifier has only effect if an outfile 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 preceded by one word in the output. This word is an integer which is the length of the entire segment (no of halfwords).

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.

3.5.5 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 decribes 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 decribes 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 decribed 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.

3.5.6 Storage Requirements

The storage requirement for "binin" is approx. 4096 halfwords plus the space needed by FP.

3.5.7 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> cannot be reserved or is unknown.

*****binin <binout file> core size**
No memory 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> reslut <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 binfile1 binfile2
```

inputs two binout files; command segments are required in the input.

The binout files may e.g. be proceduced by the FP commands:

```
binfile1=binout fpnames.p move.b
binfile2=binout algolprog
```

```
binin binfile.c
```

The binout file is checked, but no catalog functions are called, and no output is produced.

```
copyarea=binin binfile.s
binfile1=binout copyarea.ne.a
```

In this way it is possible to copy binout tapes. Another copy is made by new call of "binout", without reading the tape again.

```
code3=binin bincode1.2 bincode2.1.2 bincode1.4.3
```

loads segments 0,1 from bincode1, segment 2 from bincode2 and segments 3,4,5 and 6 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
```

3.6 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". The output file is added checksum information for verification by 'binin'.

3.6.1 Example

The program file named 'fup' is output on magnetic tape file, described by the entry t, by the FP command

```
t=binout fup
```

(compare with the example under the program "binin").

3.6.2 Call

```
<outfile> = binout {<input description>}1-*

<input description> ::= <name> {mofdifier}0-*

                {.p           }
                {.b [.<halfs>]}
                {.s [.<field>]}
<modifier> ::=  {.a [.<field>]}
                {.np          }
                {.ne          }

<halfs> ::=     <no of halfwords>

                {<no of blocks>                }
<field> ::=     {                              }
                {<no of blocks> . <first block>}

<no of blocks> ::=
<first block>  ::= integer
```

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

3.6.3 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 input to "binin".

3.6.4 binout

The output file is defined by:

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

3.6.5 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 a 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 explicitly, 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.<halfs>` Intended for output of slang programs. Has the same effect as p, except that only the first <halfs> halfwords of the actual file are output. If <halfs> is not present, the last word of the filedescriptor associated with <name> determines the number of halfwords. 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 are 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 e.g. 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.<halfwords>, 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
```

3.6.6 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 halfword 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 halfword.

3.6.7 Command Segment

The contents of a command segment are a number of commands, sufficient to create a catalog entry and load the load segments in a later call of "binin". 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.

3.6.8 Storage Requirements

The storage requirement for "binout" is approx. 3072 halfwords plus the space needed for FP.

3.6.9 Error Messages

*****binout <name> output impossible**
No left side in the call, or the output file 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, the input description 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 file from which input is not possible, or <name> is unknown.

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

*****binout <name> prog or entry**
The input description demands output of load segments in spite of the fact that <name> does not describe a file, nor does the input description cause any 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.

3.6.10 Examples on the Use of the Modifiers

The contents of the areas textarea and codearea containing a text and a program file respectively (e.g. produced by "edit" and "slang") are output on a file, binfile by the FP command

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

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

```
binin binfile
```

The ALGOL compiler may be moved to a magnetic tape - say mt471100, file 1 - and kept there. If ALGOL is present on the backing storage, this is done by the FP commands:

```
tapealgol=entry mt62 mt471100 0 1 0 algol algol
auxarea=binout algol.ne.s.11; as algol has 11 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 LIBRARY PROCEDURES are of course not moved.

3.7 bossjob

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 sec. 1.3, newjob and replacejob in ref. [10].

3.7.1 Call

```
bossjob <file name> [<name of remote batch printer>]
```

where <file name> is a name of a permanent job file.

```
<name of remote batch printer>::=<name of max 6 char>
```

3.7.2 Function

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

3.7.3 Storage Requirements

1536 halfwords plus space for FP.

3.7.4 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.

3.8 cat

The program works as catsort, except no sorting of entries prior to output takes place, i.e. the entries specified in the call are listed in the order in which they are found in the catalog.

3.8.1 Examples

Example 1

The call

```
cat scope.system
```

will list all entries in the main catalog of scope system in the order they are found in the catalog.

Example 2

The call

```
cat
```

will list all entries with entry base inside or equal to project base from the main catalog in the order they are found.

Example 3

The call

```
cat cat.disc3 name.pip docname.pip
```

will list all entries from the catalog with the document name disc3, which have either the name 'pip' or the document name 'pip'.

3.8.2 Call

Exactly as for catsort, except that in spite of sorting parameters, no sorting takes place.

3.8.3 Format of Output

Exactly as for catsort.

Note that the parameter basesort.no will influence the format of the output.

Since the entries appear unsorted, entries of different bases cause an additional line stating the entry base and permkey in case of basesort.yes.

3.8.4 Function

As for catsort, except that the temporary work file is not sorted.

3.8.5 Error Messages

As for catsort.

3.8.6 Further Examples

Example 1

```
cat sort.no
```

will list all entries in the main catalog, disregarding the parameter sort.no

Examples 2-8

The examples in 3.8.6 apply to cat as well, except that the entries are listed in the order they are found in the catalogs.

3.9 catsort

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

3.9.1 Examples

Example 1

The FP command:

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

will list all entries with a base contained in the project base, e.g. belonging to the actual project. The parameter min causes that only name, segments, docname, date, and scope is output.

Example 2

The FP command:

```
catsort scope.project min.yes
```

will do the same, but only permanent entries with an entry base equal to the project base are output.

Example 3

The FP command:

```
catsort
```

will list all entries with a base inside or equal the to project base from the main catalog sorted according to base and entry name.

See also Further Examples.

3.9.2 Call

```
                                {<catalog spec>} 0-*
[<outfile>= ]     catsort       {<entry spec>  }
                                {<sorting spec>}
```

```
<catalog spec> ::=  { main.{yes/no}          }
                    {                        }
                    {      {yes/no}          }
                    { cat. {<integer>}       }
                    {      {<document name>}}
                    {      {main}            }

<entry spec>   ::=  {        { only }        }
                    {system. { yes  }        }
                    {        { no   }        }
                    {                        }
                    {name.   <entry name>    }
                    {docname.<document name> }
                    {                        }
                    {base.  {<scope>   .min }}
                    {       {<lower>.<upper>}}
                    {                        }
                    {scope. {<scope>   .min }}
                    {       {<lower>.upper> }}
                    {                        }
                    {before. <clock>         }
                    {after.  <clock>         }
                    {size.   <lower><upper>  }
                    {cont.   <lower><upper>  }
                    {min.    yes/no          }

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

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

<lower>        ::=  <integer>
<upper>        ::=  <integer>

<clock>        ::=  <yymmdd>.<hhmmss>
<yymmdd>       ::=  <integer>
<hhmmss>       ::=  <integer>
```

3.9.3 Format of Output

In case of basesort.yes, each entry is listed in one line in the form:

<entryname> <modekind> <document name><remaining entry tail>

If the parameter min is specified the line is:

<entryname> <modekind> <document name>

Before each group of entries with the same base, one line stating the base and permkey is listed.

In case of basesort.no, this line is replaced by a supplement to each entry line. The supplement comes after entry name and consists of the information from the entry head:

```
<first slice> <name key> <catalog key> <lower base> <upper base>
```

If, however, an auxiliary catalog is listed, the document name of area entries is replaced by:

```
-> <write access counter><read access counter> <-
```

and another line follows:

```
-> d.<latest changed>          <-d.<latest read>
```

3.9.4 Function

If an outfile is specified, this file is used for output, otherwise current output file is used.

The catalogs are copied, one by one, into a temporary work file, which is sorted according to the parameters.

The parameters are processed, one by one, before any catalog is accessed.

Any parameter being repeated is a modification to the previous one.

Catalog Specifiers

`main.<yes/no>`	defines whether the main catalog is listed or not. Default: main.yes

`cat.<integer>`
`.<name>`
`.main`
`.yes`
`.no`
	defines whether the auxiliary catalog is listed or not.
	An integer or a document name specifies an auxiliary catalog to be listed.
	The name 'main' specifies the auxiliary catalog for the disk having the main main catalog.
	The name 'yes' means all auxiliary catalogs.
	The name 'no' specifies no auxiliary catalogs.
	The option prevents the main catalog from being listed, unless explicitly specified by main.yes.
	Default: cat.no.

Entry Specifiers:

`system.yes`
`.no`
`.only`
	defining whether the system files are listed or not.
	Default. system.yes

`name.<name>`	only entries with the entry name specified are listed. Default: any name.
`docname.<name>`	only entries containing the document name specified are listed. Default: any document name.
`base.<scope>`	only entries with an entry base contained in the base specified by the scope are listed. Default: base.system
`base.<lower><upper>`	only entries contained in the specified base are listed.
`scope.<scope>`	only entries with the specified scope are listed.
`scope.<lower>.<upper>`	only entries with entry base as specified are listed. (cf. above).
`before.<clock>` `after.<clock>`	only entries updated before (or after) are listed. In the main catalog, shortclock is used as latest update, i.e. only non procedure entries are candidates. In auxiliary catalogs, latest changed in the tail is used for area entries, shortclock is used for non procedure file descriptors. Default: after.zero before.infinity
`size.<lower>.<upper>`	only area entries with the size in the interval specified are output, (lower and upper size included). Default: size.0.infinity
`cont.<lower>.<upper>`	only entries with a contents key in the interval specified are output (lower and upper included). Default: cont.0.infinity.

Sorting Specifiers

`basesort.yes`	sorted after the entry base (which means grouped after project and users).
`docsort.yes`	each area entry is followed by all subentries which have a document name equal to the entry name of the main entry.

`slicesort.yes` 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.

`sort.no` no sorting at all is performed. The total catalog will be output, neglecting all other parameters but main and cat.

3.9.5 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.

3.9.6 Further Examples

Example 1

```
catsort sort.no
```

will list the total main catalog, segment by segment, in unsorted form, empty entries represented by a line marked: -

For each segment is stated the segment number and the number of non empty entries on the segment.

Example 2

```
catsort cat.yes
```

will list all entries in the auxiliary catalogs sorted according to base and entry name.

Example 3

```
catsort name.pip docname.pip basesort.no
```

will list all entries in the main catalog with entry name 'pip' or document name 'pap', sorted according to entry name.

Example 4

```
catsort docname.disc
```

will list all entries from the main catalog with document name 'disc', sorted according to base and entry name.

Example 5

```
catsort cat.disc1
```

will list all entries from the auxiliary catalog with document name 'disc1' sorted according to base and entry name.

Example 6

```
catsort main.yes cat.yes size.5000.8000000
```

will list all area entries from the main catalog and all auxiliary catalogs which have a size greater than 5000 segments.

Example 7

```
catsort after.840506.174500 before.840506.240000
```

will list all entries from the main catalog updated since the date 1984.0506 at 174500 until the date 840506 at 240000, i.e. entries with a shortclock (all entries which are not procedure entries) set in the interval given.

Example 8

```
catsort cat.disc3 after.840601.000000
```

will list all entries from the auxiliary catalog with document name 'disc3' which have been updated since the date 840601 at 000000 hours, i.e. area entries (last changed in the tail is used) or file descriptors which are not procedure entries (the shortclock of the tail is used).

3.10 change

Sends a change paper message to the parent (the operating system). The program is only used when a job uses job controlled printer. (cf. ref. [10], section 6.2].

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".

3.10.1 Call

```
change <device name> <paper type>
```

where the parameter <paper type> is an integer.

3.10.2 Function

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

3.10.3 Storage Requirements

1536 halfwords plus space for FP.

3.10.4 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. ref. [10])

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

3.11 changeentry

Changes an existing catalog entry according to the parameters in the call. The program is a supplement to the program "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.

3.11.1 Example

Suppose that the catalog entry named 'source' contains the name of a magnetic tape 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.

The catalog entry named 'source' containing the name - say mt471100 - may be created by a call of "set":

```
source = set mt16 mt471100
```

3.11.2 Call

```
                                                                    0-4
<newname> = changeentry [<kind> [<docname> [<date> {<word>}  ]]]

{<kind>    }      ::= {<integer>              }
{<docname>}           {<integer>  .<integer>}
                      {<name>                 }


<date>               ::= {<integer>              }
                         {<integer>  .<integer>}
                         {<name>                 }
                         {<name>     .<name>     }
                         {d.<isodate>.<clock>    }

<word>               ::= {<integer>              }
                         {<integer>  .<integer>}
                         {<name>                 }
                         {<name>     .<name>     }

<isodate>            ::= {<yymmdd>   / 0}

<clock>              ::=  <hhmm>

<yymmdd>             ::=
<hhmm>               ::=  <integer>
<newname>            ::=  name
<name>               ::=  name, apostrophized name,
                          generalized name or general text
```

3.11.3 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 program "set".

3.11.4 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 modekind abbreviations and if found here the value found is used. If not found in the modekind table (see ref. [8]) it is searched for in the catalog and the kind of the entry found is used.

Document 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 found above is the modekind 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 document name in the tail of the entry found is used.

The Other Parameters:

A parameter of the form <halfword1> gives separate specifications of the two 12-bit halfwords in the word.

<integer> <integer1>.<integer2>	The value is placed in the tail as the word or halfword in question.
<name> <name>.<name>	The name is looked up in the catalog and the value of the word or halfword in question in the entry tail is used.

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

3.11.5 Storage Requirements

1536 halfwords plus space for FP.

3.11.6 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 to a non-area entry or vica versa was attempted.

*****changeentry <result name> change bs device impossible**

*****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.

3.12 char

Outputs the specified character the specified number of times.

3.12.1 Example

The current output may be 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.

3.12.2 Call

```
[ <outfile> ]  =  char { <iso value>                  } 0-*
                       { <iso value>.<repeat factor> }

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

3.12.3 Function

If no repeat factor is specified the character will be output once, 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, as nl.100 will be changed to nl.64. Other characters will be repeated max. 133 times.

If an outfile is specified, it is used for output, else current output is used.

3.12.4 Storage Requirements

1024 halfwords plus space for FP.

3.12.5 Error Message

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

3.13 claim

Lists some claims of specified processes.

3.13.1 Examples

In an installation with the 2 disks:
`disc and disc1`

the call:

```
claim
```

may print:

```
name: fgs   area:  10  buf:  12  size:  200000  first: 149790

disc:  84 segm/slice
   temp   1596 segm.    19 slices    20 entr.
   login   924 segm.    11 slices    21 entr.
   perm    924 segm.    11 slices    21 entr.

disc1: 42 segm/slice
   temp     42 segm.     1 slices
   login    42 segm.     1 slices     1 entr.
   perm     42 segm.     1 slices     1 entr.
```

The call

```
claim key.disc
```

would print:

```
name: fgs   area:  10  buf:  12  size:  200000  first: 149790

disc: 84 segm/slice
   key 0   1596 segm.   19 slices    20 entr.
   key 1    924 segm.   11 slices    22 entr.
   key 2    924 segm.   11 slices    21 entr.
   key 3    924 segm.   11 slices    21 entr.
```

and the call:

```
claim perm.disc temp
```

would print:

```
name: fgs   area:  10  buf:  12  size:  200000  first: 149790

disc: 84 segm/slice
   perm     924 segm.   11 slices    21 entr.
```

```
disc: 84 segm/slice
   temp    1596 segm.   19 slices    20 entr.

disc1: 42 segm/slice
   temp      42 segm.    1 slices
```

The call

```
claim all buf
```

may print

```
name: s          area:  50  buf:  53  size: 1708290 first: 126714
name: driverproc area:   0  buf: 129  size:  119918 first:      8
name: errorsnoop area:   0  buf:   1  size:    1100 first: 149790
name: acp        area:   7  buf:  14  size:   36000 first: 149790
name: hsp        area:   5  buf:  12  size:   70000 first: 149790
name: ftsserver  area:   8  buf:   6  size:   80000 first: 149790
name: ftsuser    area:   9  buf:  11  size:   80000 first: 149790
name: fgs        area:  10  buf:  12  size:  200000 first: 149790
name: tem        area:   4  buf:  19  size:   23000 first: 149790
name: primo      area:   9  buf:  53  size:   26000 first: 149790
name: sos        area:  61  buf:  50  size:  120000 first: 149790
name: boss       area: 297  buf: 253  size:  559114 first: 149790
```

3.13.2 Call

```
[<outputfile> =] claim {[<process>] {<spec>}0-* }0-*

<outputfile> ::= <name of a file>

<process>    ::= <name of internal process> / all

<spec>       ::= <disk> / <scope> / <scope>.<disk> /
                 <disk>.<scope> / <other>

<disk>       ::= <name of disk>

<scope>      ::= temp / login / perm / key

<other>      ::= area / buf / size / first
```

3.13.3 Function

The program scans the parameter list. For each parameter group, the internal tables in the monitor are scanned. If a name of an internal process is specified, the resources of that process are listed. The name 'all' means all internal processes, one by one. No process name means own process. 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 disks are listed. If <scope> is specified the listing of resources is restricted to the specified scopes.

perm is equal to scope user + project.

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

Note that temp entries are only output for the disk containing 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 claim for own process is already reduced by 1, which is the one used by FP.

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

3.13.4 Storage Requirements

1536 halfwords plus space for FP.

3.13.5 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 param <docname> unknown**
A disk named <disk> does not exist.

3.14 claimtest

Checks the claims of the calling process according to the call parameters and leaves the ok bit true if the requirements specified are met, false otherwise.

3.14.1 Example

The job

```
claimtest perm.disc1.1000.10
if ok.no
finis
```

is terminated if the permanent resources on disc1 are less than 1000 segments and 10 entries.

3.14.2 Call

```
                                                    0-*
claimtest       { <key>.<bs claims>     }
                { buf  .<buffer claim>  }
                { area .<area claim>    }
                { size .<size>          }
                { int  .<internal>      }


<key>           ::= perm/login/tempspec/temp

<bs claims>     ::= <document name>.<segments>.<entries>

<document name>::= <name of disk>

{ <segments>      }
{ <entries>       }
{ <bs claims>     }
{ <buffer claim>  } ::= integer
{ <area claim>    }
{ <size>          }
{ <internals>     }
```

3.14.3 Function

The parameters in the call are examined one by one.

The <key> parameter names mean:

```
temp     : key = 0
tempspec : key = 1
login    : key = 2
perm     : key = 3
```

If an error in the parameter occurs or the claims specified exeed the claims available according to the process description of the calling process, the program terminates setting the ok bit false. If the program reaches the end of the parameter list, the program terminates setting the ok bit true.

Temporary and temporary special entry claims are checked on the disk containing the main catalog no matter the document name specified.

3.14.4 Storage Requirements

```
4096 halfwords + space for FP
```

3.14.5 Error Messages

*****claimtest: parametererror, unknown fpparameter <parameter>**
The parameter is neither of the eight names: perm, login, tempspec, temp, buf, area, size, int.

*****claimtest: parametererror,**
`parameter` must be `( name/integer )` `read` <param>
The parameter is a name or an integer when it should be an integer or a name.

*****claimtest: syntaxerror, separator must be <point> read <sep>**
Separator not a point.

*****claimtest: unknown bs device <name>**
The bs device with the name <name> is not included in the bs system.

3.15 clear

Removes catalog entries with name and scope as specified.

3.15.1 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.

3.15.2 Call

```
                                  1-*
clear <scope spec>  {<name>}

<scope spec>  ::= <scope> [.<disk name>]

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

<disk name>   ::= <name of disk>
```

3.15.3 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.

3.15.4 Scope Specifications

The concept of scope of a catalog is explained in ref. [10], sec. 4.1. A disk name means a further restriction to entries which are either

(a) area entries, where the data area is placed on the specified disk, or

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

3.15.5 Storage Requirements

2048 halfwords plus space for FP.

3.15.6 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 disk was not included in the bs system. No entries removed.

*****clear <scope spec> bs device not ready**
The disk specified was not ready or catalog i/o error. 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.

3.16 clearmt

Removes catalog entries according to the parameters.

3.16.1 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`.

3.16.2 Call

```
<result name> =
clearmt <mtname>. [[ <lower integer>.] <upper integer>]
```

The <mtname> is not used during interpretation of the parameters. If no <lower integer> is specified, the value 1 is used.

3.16.3 Function

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

3.16.4 Storage Requirements

512 halfwords plus space for FP.

3.16.5 Error Messages

*****clearmt call**
No 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.*

3.17 compresslib

The program compresses into one single area a number of external ALGOL / FORTRAN procedures or code procedures to occupy a minimum number of segments. The object area may be an already compressed library of procedures.

3.17.1 Example

The call

```
read = compresslib write arctan ln arcsin
```

collects into one single area the algol library procedures read, write, arctan, ln and arcsin and sets the descriptors of these procedures so that they may be found and accepted by the ALGOL and FORTRAN compilers.

3.17.2 Call

```
                                                              1-*
<collection area> = <compresslib> {<procedure area>}

<collection area> ::= <procedure area>
<procedure area>  ::= <name>
```

A procedure area is an area entry with content 4, and with length < 0. The intervals of the procedure areas must be equal and the procedure areas must be visible to and not protected against the user process.

3.17.3 Function

The contents of the collection area is examined so that it is known how much space the code and the external list take up in the area. Then the procedure areas on the right hand side are scanned from left to right and their code and external lists are placed one after the other as tight as possible after the external list and the code of the collection area. If necessary the collection area is extended as the processing proceeds.
Parameters not accepted in this phase are dropped in the compression.
When all procedure areas have been collected, the accepted areas are removed and the procedures are described as shared main entries (see the appendix), i.e. the content becomes 32 + first segment in the collection area, and the mode kind becomes bs.
Possible sub entries to the procedure areas are not changed.

3.17.4 Storage Requirements

900 halfwords + room for FP + room for the largest external procedure.

3.17.5 Error Messages

All error messages have the following form:

*****compresslib <param> <further explanation>**

After an error message of this form, the standard actions are as follows:

If <param> is absent or it is the name of the collection area, the program terminates immediately. None of the procedure areas to the right will be changed.
If <param> is one of the right hand parameters, this parameter is dropped from the compression, but the other parameter will not be effected.

Further explanations may be:

connect error:	The area could not be connected for input or output. the reason may be that it is reserved for output by another process.
unknown:	The area could not be found.
interval:	The interval of the procedure area is not equal to the interval of the collection area.
content:	The parameter does not denote an external procedure.
not area:	The parameter does not denote an area (with length > 0).
intervals:	The interval of the collection area is not equal to the catalog base of the process or the catalog base is not contained in or equal to the standard interval of the process.
no collection area:	The program call has no left hand side.
transport error:	Input from a procedure area or input to the collection area is impossible. Usually because of claims exceeded or bs_fault.
process too small:	It is imposible to hold the entire procedure in the process area. If parameter is the collection area, it is impossible to hold the external list in the process area.
too many segments:	The collection area has exceeded 4063 segments.

3.18 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.

3.18.1 Example

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

```
convert out1
```

3.18.2 Call

```
convert <name> [<name of remote batch printer>] [integer]

<name of remote batch printer>::= <name of max. 6 char>
```

3.18.3 Function

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

3.18.4 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.

3.18.5 Storage Requirements

1536 halfwords plus space for FP.

3.18.6 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.

3.19 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 (ISO values 0 and 127) are not copied. Furthermore the program may be used for check reading of text files (e.g. texts on paper tape).

3.19.1 Example

The text files 'text1' 'text2' are output as one magnetic tape file 'text' by the FP command

```
text=copy text1 text2
```

and the number and the sum of the characters are printed on current output. One may then check the text by reading it in another job by the FP command

```
copy text
```

3.19.2 Call

```
[ <outfile>= ]  copy [list. {yes/no}]    ,
                                                    1-*
     { <infile>                                    }
     { <lines>                                     }
     { [<infile>.] <iso value>.<appearances>       }
     { message.{yes/no}                            }

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

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

<appearances>    ::= <integer>
```

3.19.3 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>
Thespecified 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 (default: message.yes)

1. after each parameter:
   <infile> segm. <number of segments>
   number of characters < 128
   sum of characters
   number of characters > 128 (if any)
   number of blind characters (0, 127) - (if any)
   number of sub characters (26) - (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)
   total number of sub characters (26) - (if any)

3.19.4 Storage Requirements

1536 halfwords plus space for FP.

3.19.5 Error Messages

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.

3.20 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 may, e.g., be used in connection with process control devices producing data with a high rate, cf. ref. [10], secs. 3.4 and 6.7.

3.20.1 Example

The FP command:

```
corelock 5
```

demands corelock for a period of 5 seconds.

3.20.2 Call

```
corelock <seconds>
```

where <seconds> is an integer.

3.20.3 Storage Requirements

1536 halfwords plus room for FP.

3.20.4 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.

3.21 coreopen

Sends a coreopen message to the parent (the operating system) signalling the end of a corelock period (cf. the program "corelock"). The program may, e.g., be used on process control installations.

3.21.1 Example

The program is called without parameters:

```
coreopen
```

3.21.2 Call

```
coreopen
```

3.21.3 Storage Requirements

1536 halfwords plus room for FP.

3.22 correct

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

3.22.1 Example

The FP call:

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

will make the following corrections on segment 4 of 'bsfile':

```
halfword 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.

3.22.2 Call

```
correct <bsfile>.<segmno>  ,

{ address.<addr>
{[<bitspec>] if <oldvalue>  ,
                                  1-* 0-*
               then <newvalue>}   }

<bitspec>      ::=  bits.<firstbit>.<lastbit>

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

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

3.22.3 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. [8].

During syntax check only the first 3 letters in the words:

`address, bits, then, negative`

are tested, i.e adr is accepted for address.

Odd addresses are reduced by 1.

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

Bits are numbered 0, ..., 23 with the most significant bit being no. 0.

The word 'shortclock' in the catalog entry is updated. In case <bsfile> describes an external procedure, the internal date is updated.

3.22.4 Storage Requirements

1024 halfwords plus space for FP.

3.22.5 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 connected**
<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 value found.

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 in 'shortclock' of the entry tail, or the internal date of an external procedure is incorrectly described either in the catalog entry or in the code. The correction has been performed.

3.23 delete

Finds and removes all catalog entries with a given scope, possibly filtered by filters given. The filters work on the entry name and on the document name as for the program search.

The program should be used with extreme care, e.g. execute the program *search* with the same parameters before executing delete.

3.23.1 Example

By the fp command

```
delete user
```

all entries of scope user are listed on current output and **removed**.

By the fp command

```
delete own
```

all entries of scope temp, login, user, or project are listed on current output and **removed**, while the command

```
delete user ret.tx
```

will list all entries of scope user which contain both the substring 'ret' and the substring 'tx' in either name or document name and **remove** them.

3.23.2 Call

```
[<out file> = ] delete  <scope spec> {<filter>} 0-*

<scope spec> ::=          <scope>  [.<disk name>]

<scope>      ::=                   {temp         }
                                   {login        }
                                   {user         }
                                   {project      }
                                   {own          }
                                   {<low>.<upp>  }

<filter>     ::=       <substring> {.<substring>} 0-*

<substring>  ::=       {<name>                   }
                       {<apostrophized name>     }
                       {<generalized name>       }
                       {<general text>           }
```

```
<low>       ::=
<upp>       ::=     integer
```

3.23.3 Function

The main catalog is scanned, and a subset of it is listed with an output format as for *lookup* and the entries in the subset are **removed**. If an outfile is specified, the list of catalog entries is printed on that file, otherwise current output is used. Messages from *delete* are always printed on current output. If no filters are given, all entries from the main catalog according to the scope specification are listed and **removed**, otherwise, the set of catalog entries is further delimited by means of filters (see filter specification below).

3.23.4 Scope Specification

The scope concept is explained in ref. [10], section 4.1. The scope *own* means any scope in the set *temp, login, user,* or *project* (cf. the example above). If a disc name is specified, only entries in the auxiliary catalog on that disc are candidates. The scope given by <low>.<upp> means all entries with entry interval equal to the interval <low>.<upp>, which will have to be contained in but not equal to the std. interval of the process.

3.23.5 Filter Specification

A filter consists of one or more substrings concatenated by period. If a list of filters exists, an entry selected for listing and removing will only be accepted if either its name or its document name contain all the substrings of at least one of the filters. The order of the substrings in a filter is irrellevant. Thus, in a possible list of filters, you may consider space as "or" and period as "and", where the precedence of "and" and "or" is as in algol.

3.24 edit

"edit" is a line oriented program for editing of text files.

3.24.1 Example

The FP call and edit commands:

```
                                ;    COMMENTS
betterfinal=edit finaltext      ;    fp call
l/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.

Ref. [10] shows several very relevant examples of the use of "edit".

3.24.2 Call

```
                                                0-*
[ <outfile>= ]    edit ( <source> )
```

3.24.3 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.
```

3.24.4 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".

3.24.5 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.

```
l
/
*
,
:
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.

3.24.6 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".

3.24.7 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
17              Move line pointer 7 lines forwards.
1-4             Move line pointer 4 lines backwards.
1 t             Line top, move line pointer to line 1.
1 b             Move line pointer to line bottom, i.e.
                the line containing EM.
1./find/        The line pointer is moved forward 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.

3.24.8 Delete

```
                COMMENTS
d               Delete current line.
d 4             Delete current and 4 following lines.
d-2             Delete current and 2 preceding lines.
d t             Delete current and all in front.
d b             Delete current and all following.
d./find/        Delete current up to and including the
                line containing the textstring find.
```

After deletion the line pointer points at the line following the last deleted line.
Note: NL see Line.

3.24.9 Insert

	COMMENTS
i/	The two lines are inserted in front of
elephants	the current line. After insertion the
monkeys	line pointer points at the line in front
/	of the terminating delimiter. (here in 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.

3.24.10 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 the position is not found, the line pointer points at the next line.

3.24.11 Global

	COMMENTS
g/bad/good/	In the current line any appearance of the string bad is replaced by the string good. The line pointer is unchanged. If, however, the replacing string is two or more lines, the line pointer is changed. It will point at the last line before the terminating delimiter, as for 'insert'. In this case only the first occurrence of the string is replaced, since later occurrences are no longer in current line.
g2/bad/good/	In the current and 2 following lines any appearance of the string bad is replaced by the string good. The line pointer is moved 2 lines forward. If, however, the replacing string is two or more lines it must be noticed that only occurrences in the original lines are replaced, and

only as long as the line pointer does not not exceed the initial value of current line + the number of lines specified, which is where the line pointer will end.

g-6/bad/good/ In current and 6 preceding lines any appearance of the string bad is replaced by the string good. The line pointer is not moved.

The effect is the same as 1-6, g 6/bad/good/

If the replacing string is two or more lines, this is important, because it means that the interval for the line pointer is frozen to initial value of current line up to initial value + number of lines specified, which is where the pointer will end. The contents of that line interval is changed for each replacement, moving some of the original lines beyond the interval limit.

g t/bad/good/ In current and all preceding lines any appearance of the string bad is replaced by the string good. The line pointer is not moved.

If, however, the replacing string is two or more lines it must be remembered that the effect is the same as:

1 t, g<current line>/bad/good/

g b/bad/good/ In current and all following lines any appearance of the string bad is replaced by the string good. The line pointer points at the line following the last line.

The effect is the same for replacement strings or more lines.

g b/unwanted// Remove the string unwanted from current line and until bottom.

Note: NL see Replace

3.24.12 Finis

	COMMENTS
f	"edit" copies to EM and exits.

3.24.13 Print

	COMMENT
p	Prints current line.
p2	Current and 2 following lines are printed.
p-2	Current and 2 preceding 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.

Note: NL see Line.

3.24.14 Source

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

	COMMENTS
s 2	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:                          text2:
begin real a,b,c,d;             begin integer i,j,k;
procedure testoutput;           boolean ok;
begin                           procedure error(i);
write(out,<:<10>:>,a,b,c);      integer i;
```

```
end testoutput;                    begin
procedure calculate(x);            write(out,<:<10>alarm
                                   :>,i;
real x;                            end error;
begin                              procedure merge(a,b,x);
...                                ...
```

```
                                   COMMENTS

new=edit text1 text2 text3         Edit call with 3 sources.
1./ure calculate/,                 Copy until this line from
                                   source 1.
s 2                                Continue from source 2.
d./boolean ok/,                    Delete inclusive this
                                   line.
1./end error/,11,                  Copy until this line.
s1                                 Continue from source 1.
d./end testoutput/,                Delete inclusive this
                                   line.
l b                                Copy to last line.
s 3                                Continue from source 3.
f                                  Copy and exit.
```

3.24.15 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.
                                   <12>.
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//bå                            Erase faulty line.

r/abgde<BS><BS><BS>ø<BS>c<SP><SP>fg/alphabet/
                                        Result:
                                        r/abcdefg/alphabet/
                                        only to be used when typed
                                        on terminals writing on
                                        paper which has a
                                        backspace, BS, character.
                                        Used for correction of one
                                        character without changing
                                        the rest of the line.
```

3.24.16 Verify

Normally the line is listed at the 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
```

3.24.17 Where

The "edit" command

```
w
```

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

```
3 line.
```

3.24.18 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 as 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
```

3.24.19 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.`

3.24.20 Error Messages

3.24.20.1 Initial Alarm

*****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 backing storage.

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

3.24.20.2 Alarms Concerning Communication with Peripheral Devices

*****edit <command no.> connect source**
The source document cannot 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 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.

3.24.20.3 Alarms Caused by Erroneous 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 specified or 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 the start of each sequence.

3.25 end

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

3.25.1 Examples

See 3.24.1, edit.

3.25.2 Call

```
end
```

3.25.3 Function

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

3.25.4 Storage Requirements

1024 halfwords plus space for FP.

3.25.5 Error Messages

*****end call**
Left hand side in the call. The end is still performed.

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

3.26 entry

Creates a temporary or changes an existing 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 tail by copying from the tails of other catalog entries.

3.26.1 Example

Suppose that the catalog entry named 'source' contains the name of a magnetic tape reel in the document name field. By the FP commands

```
file1=entry mt16 source 0 1
file2=entry mt16 source 0 2
file3=entry mt16 source 0 3
```

one gets catalog entries 'file1', 'file2', 'file3' which serve 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 mt16 mt471100
```

3.26.2 Call

```
                                                                    0-4
<newname> = entry [<kind> [<docname> [<date> {<word>}    ] ] ]

{<kind>    }     ::=   {<integer>              }
{<docname>}            {<integer1> .<integer2>}
                       {<name>                }

<date>           ::=   {<integer>              }
                       {<integer1> .<integer2>}
                       {<name>                }
                       {<name1>    .<name2>   }
                       {<d.isodate>.<clock>   }

<word>           ::=   {<integer>              }
                       {<integer1> .<integer2>}
                       {<name>                }
                       {<name1>    .<name2>   }

<isodate>        ::=   {yymmdd>    / 0}
<clock>          ::=   <hhmm>
<yymmdd>         ::=
<hhmm>           ::=   <integer>
<newname>        ::=   name
<name>           ::=   name, generalized name,
                       apostrophized name, or general
                       text
```

3.26.3 Function

The parameters are interpreted as described below yielding the wanted entry tail. From this point the program continues exactly as the program "set".

3.26.4 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 modekind abbreviations and if found here the value found is used. If not found in the modekind table (cf. ref. 8, Appendix) it is searched for in the catalog and the kind of the entry found is used.

Docname:

`<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 modekind 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 <integer> . <halfword> or <name1>.<name2> gives separate specifications of the two halfwords in the word.

`<integer>:`	The value in the tail as the word or halfword in question.
`<name>:`	The name is looked up in the catalog and the value of the word or halfword 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.

Note:
If <free> = d.<isodate> and an entry named d exists, the left half of <free> will be taken from d and the right half will become <isodate>.

3.26.5 Storage Requirements

1536 halfwords plus space for FP.

3.26.6 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 to a non-area or vice versa was attempted.

*****entry <result name> change bs device impossible**
A change of document name of an area was attempted.

*****entry <result name> bs device unknown**
The disk 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> no room**
Name overflow in the main catalog exceeded the limit.

*****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.

3.27 finis

finis terminates the job and removes the process.

Example

See the program *if*, and the program *claimtest*.

3.27.1 Call

```
finis  [output.{yes/no}]
```

3.27.2 Function

The current output file is terminated (emptying of buffers etc.) and a finis job message is sent to the parent (the operating system), who is then expected to remove the job.

If parameter output.no is specified, and the parent is BOSS, the finis message will specify that output is not wanted.

3.27.3 Storage Requirements

1024 halfwords plus space for FP.

3.27.4 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.

3.28 head

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

3.28.1 Example

The output from two programs is separated in a nice way by calling *head* in between:

```
head 1
```

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

```
head old cpu
```

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

3.28.2 Call

```
                                          0-*
                          { <integer> }
[ <outfile> = ]  head     { cpu       }
                          { iso       }
                          { old       }
```

3.28.3 Function

In an integer is given as parameter that many form feeds are printed. Next, one line consisting of job name, date, and clock is printed. In 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.

3.28.4 Storage Requirements

1024 halfwords plus space for FP.

3.28.5 Error Messages

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

3.29 i

Selects a new file as current input. The former file may later be resumed at the position where it was left (e.g. by a call of *end*).

3.29.1 Example 1

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 fiel 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`.

If, on the other hand, we have the following commands

```
(i commds2
 i commds1)
```

the first will select the file `commds2` as currrent input, but since FP immediately executes the next command without reading from current input, the next command will select the file `commds1` as current input. Now FP will start reading from the file `commds1`, and when it is exhausted or the command *end* executed, current input will return to the previous, i.e. `commds2` and FP will continue reading from `commds2`. When this file in turn is exhausted or the command *end* executed, current input will return to the file where the composite command was read, and reading will continue from here.

3.29.2 Example 2

edit reads the editorial commands from current input. The commands to *edit* may be kept in a separate file `editcomds` if the editing is done by the following composite FP command:

```
(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`, even if *edit* should exit before emptying the file `editcomds`.

3.29.3 Call

```
i <file name>
```

3.29.4 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.

3.29.5 Storage Requirements

1024 halfwords plus space for FP.

3.29.6 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`	forbidden 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).

3.30 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).

If more bit values are specified, the value of the expression will be a logical and of the bit values.

3.30.1 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 job. Proceed as follows:

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

3.30.2 Call

```
                                    1-*
if { <mode bit> . (yes/no) }

<mode bit>  ::= { ok        }
                { warning   }
                { initmess  }
                { bswait    }
                { error     }
                { pause     }
                { listing   }
                { all       }
                { <integer> }
```

3.30.3 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 successive calls of "if".

3.30.4 Storage Requirements

1024 halfwords plus space for FP.

3.30.5 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.

3.31 init

The command, which is an entry into FP, forces an initialization of FP.

3.31.1 Example

By the command

```
init
```

a reinitialization of FP is done, giving the output on current output after having reconnected it to primary output document :

```
*** fp reinitialized

fp version 2 release 4.00
```

provided the FP modebit 'initmess' is true (cf. ref [8], 4.2).

3.31.2 Call

```
init
```

3.31.3 Function

The job starts the reinitialization of FP with the information about primary input and output handed over by the parent at job start. During the reinitialization of FP the job creates catalog entries 'v' and 'c' describing the primary input and output files resp. If such entries are already present they are removed by the job unless they describe the proper files. The zone initialization ends by connecting the current input and output zones to the primary input and output files.

The reinitialization count is zeroed, allowing FP itself to reinitialize up to 10 times before giving up with a FINIS parent message, cf. (8), 3.3.

The catalog base is reset to equal the standard base of the job and the name and name table address of the parent are renewed.

The overflow/underflow interrupts are masked off (integer overflow, floating under/overflow).

A possible current input zone stack chain is zeroed and the '***fp reinitialized' message is written on current output.

If the modebit 'initmess' is 'true', the initialization message stating version and release of FP is written, too, on current output, and in any case the current output zone is emptied with a NL character.

The modebits 'warning' and 'ok' are set 'warning.yes' and 'ok.no', while the rest are left unchanged.

At last the area processes in the monitor are scanned, and any area process with the job process as user is removed (the job process is

removed as user). The event queue of the job process is scanned, too, and any pending operation not belonging to current input zone is waited for.

Now the FP command reading routine is entered.

3.31.4 Storage Requirements

1024 halfwords plus space for FP.

3.31.5 Error Messages

```
*** fp init troubles

*** fp trouble : c or v
```

Cf. ref. [8], Appendix B.4

3.32 job

Makes it possible to use files containing a BOSS job specification in the first line as job files in other operating systems than BOSS.

When the operating system is BOSS, the program makes it possible to enroll the current BOSS edit file as a terminal job by the command "go", regardless of a possible job specification in the first line.

3.32.1 Example

Assume you have a file, 'jobfile', containing:

```
job fgs 1 274001
p=algol tp
p pip
finis
```

If you type

```
i jobfile
```

FP will simply skip the line

```
job ...
```

and read the next line.

If you execute under BOSS and enroll the same file as a terminal job by the command 'run', BOSS will read the job specification and FP will start reading the next line.

If you enroll the same file as a terminal job by the command "go", FP will read the job specification, skip it, and read the next line.

3.32.2 Call

```
job <parameter list>
```

<parameter list> may consist of any sequence of parameters obeying the FP syntax.

3.32.3 Function

"job" returns to 'FP end program' with ok.yes and warning.no.

3.32.4 Storage Requirements

512 halfwords plus space for FP.

3.32.5 Error Messages

None.

3.33 kit

Sends a mount disk message to the parent (the operating system) demanding a disk with a specified name to be mounted on a disk storage module; cf ref [9], chapters 3 and 5.

3.33.1 Example

The FP command

```
kit 12 disc5
```

asks for mounting of the disk 'disc5' on the disk storage module with device number 12.

3.33.2 Call

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

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

3.33.3 Function

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

3.33.4 Storage Requirements

1536 halfwords plus space for FP.

3.33.5 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.

3.34 label

Outputs a BOSS Volume Header Laber (ISO 1001 - 1986, VOL1 label) in file 0 of the specified tape to make the tape an initialized volume.

3.34.1 Example

The FP call:

```
label mtlh mt123456 p 789012
```

outputs a BOSS label on mt123456.

If you have a filedescriptor, e.g.:

```
f=set mt16 mt123456 0 1
```

the call:

```
label f f p 789012
```

will have the same effect.

3.34.2 Call

```
label <modekind> <mtname> [<access> <project number>]

<modekind>        ::= { mt62 }
                      { mt32 }
                      { mt16 }
                      { mt08 }

<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, the 4 last must be digits.

3.34.3 Function

A label in the format accepted by BOSS (cf. ref. [10], sec. 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.

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

3.34.4 Alternative Parameter Names

For compatibility reasons, the modekind parameter names mthh, mthl, mtll, mto, mte, nrz, and nrze are still valid, giving:

modekind abbreviation		speed	density	value*	parity
mt62		-	high	.1	odd
mt32		-	high	11	odd
mthh		high	high	.1	odd
mtlh	mto	low	high	.1	odd
	mte		high	.1	even
mtl6		-	low	..	odd
mt08		-	low	1.	odd
mthl		high	low	..	odd
mtll	nrz	low	low	..	odd
	nrze		low	..	even

*) value of density field in modeword

3.34.5 Error Messages

*****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 is not a known modekind abbreviation, or it is not the name of a a file descriptor.

*****label, <parameter> illegal tapename**
Tapename is illegal.

*****label, <parameter> illegal access kind**
Access must be p, r, or w.

*****label, project number missing**
If access 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.

3.35 load13

The program can input catalog entries and bs files from magnetic tape files generated by the program *save13*. Unless the magnetic tape is explicitly targeted for restore by the program *load13*, the program *load* should be used, cf. [13].

3.35.1 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:

```
load13 t.0
```

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

```
load13 mt471100.1 scope.temp pap
```

See also: Further examples.

3.35.2 Call

```
[ <outfile>= ]
load13 [<mountparam>.]
<tape parameter> [<special param>]  [<load spec>]

<mountparam>      ::=
[mountspec.<deviceno>.] [<modekind>.] [release .(yes/no)]

<tape parameter> ::=                                  0-9
<tapename>.<fileno> {.<tapename for next volume>}

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


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

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

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

<modifiers        ::=                                  0-*
{changekit.<bs device spec>.<bs device spec>}
{changekit.all.<bs device spec>                        }
{newscope.<newscope spec>                              }

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

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

<entry spec>      ::=
                                          0-*
{<name>                                  }
{<name>.scope.<scope spec>               }
{docname.<docname>                       }
{docname.<docname>.scope.<scope spec>}
{scope.<scope spec>                      }
```

3.35.3 Function

The contents of the dump label (see "save13") are checked and listed on current out.

Next the program loads from the magnetic tape all the entries and bs files 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 disk name for non-area entries, the kit into which the entry is permanented.

3.35.3.1 Function, mountparam

If no mountparam is specified, the program will use a standard magtape station, modekind=mt62 (or in case mtname is a filedescriptor, then the modekind of this file will be released at end of program.

```
e.g. mountspec.10.nrz.release.no.
  mountspec.10.
  nrz.
  release.no.
```

3.35.3.2 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 are 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.

3.35.3.3 Function, special param

`check. (yes/no)`	`Default 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)`	`Default is survey.no` `If survey.yes, then all entries from file 1 to <fileno> are listed but not loaded.`
`load. (yes/no)`	`Default 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)`	`Default is list.yes, all loaded entries are 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.`

3.35.3.4 Function, modifiers

If `<bs device spec> = 0`,
the area is created on the disk with the most temporary resources.

If `<bs device spec> = 1`,
the area is created on the disk with the most resources for key = 1.

If `<bs device spec> = main`,
the area is created on the disk 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

`changekit.all.<loaded kit>`
As above, except all entries specified, no matter their document name, are loaded on <loaded kit>.

`newscope.<newscope spec>`
Default 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`

3.35.3.5 Function, kit spec

General about <bs device spec>, see modifiers.

`kit.<saved kit>` `Default` is main, meaning all disks. Only entries which in the tape is described as addressing this disk will be loaded (or entries which after a changekit parameter is addressing this disk). The parameter is valid for all following <entry spec> until a new kit parameter is specified. If a kit parameter specifies a not connected disk, a changekit, changing this kit to a connected kit name, must be specified earlier in the parameter list. E.g. kit.disc2

3.35.3.6 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.

3.35.4 Tape Format

See *save13*.

3.35.5 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.

3.35.6 Storage Requirements

12000 halfwords.

3.35.7 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.

3.35.8 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
```

3.36 lookup

Finds and lists all visible catalog entries with a specified name.

3.36.1 Example

The FP command

```
lookup pip
```

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

```
pip    -set 16 disc d.880523.1021 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.

3.36.2 Call

```
                                                    1-*
[ <outfile> = ]     lookup ( <name> )
```

3.36.3 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.

3.36.4 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

```
<scope> [.<device name>]
```

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. 1 and 2).

3.36.5 Storage Requirements

2560 halfwords plus space for FP.

3.36.6 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.

3.37 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.

3.37.1 Example

The FP command

```
message program execution no.1
```

outputs the text

```
program execution no.1
```

on current output.

3.37.2 Call

```
[ <outfile> = ]   message <parameter list>
```

<parameter list> may consist of any sequence of parameters obeying the FP syntax.

3.37.3 Function

The parameter list is copied on <outfile> or current output (if no outfile is specified). The output is terminated by an NL character.

3.37.4 Storage Requirements

512 halfwords plus space for FP.

3.37.5 Error Messages

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

3.38 mode

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

3.38.1 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.

3.38.2 Call

```
                              0-*
{<mode bit>. {yes/no}}    [what]

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

The integer values of the mode bit names are as follows:
bswait = 13, initmess = 14, 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 ref. [8], section 4.2.

3.38.3 Function

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

3.38.4 Storage Requirements

1024 halfwords plus space for FP.

3.38.5 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.

3.39 mount

Sends a mount message to the parent (the operating system) which is then expected to ask the operator to mount the tape reel (cf. ref. 10, section 6.1). The program does not await the mounting, unless there is asked for mounting of an unspecified worktape.

3.39.1 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 mt62 mt280007 0 3
p8=set mt62 mt280008 0 1
p9=set mt62 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 mt62 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 workfile 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").

3.39.2 Call

```
mount <name>
```

3.39.3 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 is 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.

3.39.4 Storage Requirements

1536 halfwords plus space for FP.

3.39.5 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.

3.40 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. ref. [10], section 6.1).

3.40.1 Example

If the installation has some standard magnetic tape stations and a non standard with device number 12 and one has a tape reel named mt123456, which must be mounted on device no. 12, the FP command

```
mountspec 12 mt123456
```

ensures that BOSS will accept the reel only when mounted on station No 12. If 'pip' is the name of a file on a magtape e.g.

```
pip=set mt62 mt123456 0 7
```

the same result is obtained by the FP command

```
mountspec 12 pip
```

3.40.2 Call

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

where <device no> is an integer.

3.40.3 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.

3.40.4 Storage Requirements

1536 halfwords plus space for FP.

3.40.5 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.

3.41 move

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

3.41.1 Example

The contents 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 mt62 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 mt62 mt312223 0 3
tofile =set mt62 mt312224 0 7
tofile =move fromfile.4
```

3.41.2 Call

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

```
<bs file> = move {<param>}0-* {<bs file> / <mt file>}
```

or

```
<mt file> = move {<param>}0-* {<bs file> / <mt file set>}0-*

<bs file> ::= name of a backing storage area entry

<mt file> ::= name of magnetic tape entry

<param>       ::= {message.{yes/no}}
                  {blockl.<segm>   }

<mt file set> ::= <mt file> [.<no of files> [.<skip>] ]

<segm>        ::=
<no of files> ::=
<skip>        ::= integer
```

3.41.3 Function

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

The parameter 'message.yes' will cause output of the number of halfwords copied and the word checksum.

```
<mt file> = move <mt file> / <mt file set>
```

As many files as specified will be written, separated by tape marks. <no of files> specifies the number of files to copy, <skip> specifies how many files to skip before copying.

The blocklengths on the input files are kept on the output files, except when the inputblock is not a three character multiple. Then the outputblock is added one or two zeroed characters up to the nearest word boundary.

```
<mt file> = move <bs file>
```

The parameter 'blockl.<segm>' defines the blocklength on the magnetic tape, given as segments per block, with a maximal value of 84 segments per block. Default is 1 segment per block.

```
<bs file> = move <mt file>
```

If the input blocklength is not a multiple of 512 halfwords, the output blocks will be filled with zeroes up to the nearest segment boundary. If the input blocklength is not a multiple of three characters, the last word will be added one or three characters up to the nearest word boundary.

The length of the bs area is cut to the number of segments actually copied.

```
<bs file> = move <bs file>
```

The blocklength used for input and output will be the greatest number of segments possible in the job process.

The length of the output area is cut to the actual number of segments copied, and the last 5 words of the entry tail will be copied from the input area entry.

3.41.4 Storage Requirements

The minimum memory requirements for move is 3318 halfwords plus space for FP (4854 halfwords).

When copying from magnetic tape with blocklengths greater than one segment, more space will be needed, namely 1536 halfwords for each extra segment in the blocklength.

The greatest possible blocklength when copying to or from from magnetic tape is 84 segments per block, so no matter the memory space available, greater blocks will not be used for magnetic tape.

When copying from disc to disc or from disc to tape, blocklengths greater than one segment are attractive. For each extra segment in the blocklength, move will need extra 1536 halfwords of memory.

Blocklength (segments):	Minimum memory size (halftwords):	
1	4854	(incl FP)
2	6390	
7	12534	
9	15606	
15	24822	
21	34038	
63	98550	
84	130806	
132	204800	(200 K)
681	1048576	(1 M)
1364	2097152	(2 M)

3.41.5 Error Messages

*****move: no core**
The process area is too small to contain the minimum input and output buffers or input blocklenght from magnetic tape exceeds 84 segments.

*****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 an 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.

*****device status <inputfile>**
Blocklength error. The process area could not contain the magnetic tape input blocks.

3.41.6 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.

*) 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.

```
move mt1
```

causes the alarm:

```
***move call
```

because no output file is specified.

```
mt1 = 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.

3.42 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 1.3, newjob and replacejob, in ref. [10].

3.42.1 Call

```
newjob <file name> [ <name of remote batch printer> ]
```

where <file name> is a name of a permanent job file.

```
<name of remote batch printer>::= name of max 6 characters
```

3.42.2 Function

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

3.42.3 Storage Requirements

1536 halfwords plus space for FP.

3.42.4 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.

3.43 nextfile

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

3.43.1 Examples

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.

If the catalog entry 't' describes file no. 0 at a magnetic tape, the FP command

```
nextfile t t t t t t
```

will change it to file no. 6.

3.43.2 Call

```
                          1-*
nextfile ( <name> )
```

3.43.3 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.

3.43.4 Storage Requirements

1536 halfwords plus space for FP.

3.43.5 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.

3.44 o

Selects a new file as current output.

3.44.1 Example

The text output from an ALGOL translation may be written in 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
```

3.44.2 Call

```
o <file name>
```

3.44.3 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 previously 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 with the most temporary resources 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 "c" connects current output to 'o' (either because of the command 'o c' or because of some error) the following is done: if a catalog entry named 'c' is present with the proper document name (primary output process), 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.

3.44.4 Storage Requirements

1024 halfwords plus space for FP.

3.44.5 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.

3.45 online

Turns a BOSS job into the conversational mode where the current input to the job is typed on the terminal at run time. A conversational job is more resource demanding and the user must have a special option in the BOSS user catalog (cf. ref. [10], section 3.2).

3.45.1 Call

```
online.
```

3.45.2 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.

3.45.3 Storage Requirements

512 halfwords plus space for FP.

3.45.4 Error Messages

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

3.46 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.

3.46.1 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).

3.46.2 Call

```
opcomm <parameter list>
```

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

3.46.3 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.

3.46.4 Storage Requirements

1536 halfwords plus space for FP.

3.46.5 Error Messages

*****opcomm call**
Left hans side in call of the program. No message is sent and no waiting is performed.

3.47 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.

3.47.1 Example

An example of the use is given in the description of the program "opcomm".

3.47.2 Call

```
opmess <parameter list>
```

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

3.47.3 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.

3.47.4 Storage Requirements

1536 halfwords plus space for FP.

3.47.5 Error Messages

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

3.48 permanent

The program changes the catalog key of the specified entries to the specified integer.

3.48.1 Example

The fp call:

```
permanent pip.3
```

will change the catalog key of the entry pip to 3. Normally the program scope should be used.

3.48.2 Call

```
                                    0-*
permanent { <name>.<integer> }
```

3.48.3 Function

For each name a catalog lookup is made and the catalog key of the entry found is changed to the specified value. This may cause an illegal scope.

3.48.4 Storage Requirements

2048 halfwords plus space for FP.

3.48.5 Error Messages

*****permanent call**
Left hand side in the call. The program terminates.

*****permanent param <parameter>**
Parameter error in the call. The faulty parameter is not treated.

*****permanent <name> unknown**
No entry with the specified name was found.

*****permanent <name> protected**
The job was not allowed to change the entry key of the specified entry.

*****permanent <name> no resources**
The job has no permanent resources left on the relevant disk.

*****permanent <name> error**
Catalog or hard error.

3.49 print

Prints from a backing storage area or directly from memory with specified formats. The program is primarily intended for printing of dumped areas and for disassembling.

3.49.1 Example

The memory 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 0.16 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 16 contain the start address of the memory area and the registers at the time of the dump).

If the area is described with contents 7 in the area entry tail (dumped memory areas should always have this contents, which also is set by s and BOSS when making a dump) the output is listed with absolute addresses, corresponding to the logical allocation in memory. One can select the part to be printed by specifying such absolute addresses: The command

```
print image 179766.179874.a
```

prints the part of the dump originating from the memory addresses 179766 to 179874.

3.49.2 Call

```
[ <outfile> = ]  print <source>    ,

                                                  0-*
[<modifier>]   ( <format list> <field> )

<source>      ::= ( <bs area name>           )
                  ( <internal process name> )
                  ( <memory address>         )

<modifier>    ::= (.<first number>)
                  (.   s          )
```

```
                                                                    0-*
<format list>::= { integer                                        }
                 { word                                           }
                 { half                                           }
                 { abshalf                                        }
                 { char                                           }
                 { octal                                          }
                 { hex                                            }
                 { code                                           }
                 { text                                           }
                 { bits <pattern>                                 }
                 { all                                            }
                 { words . <words per line>                       }

<pattern>    ::= { .<first bit> . <last bit> }0-*

<field>      ::=

{ <from - to>  [.i          ]                                      }
{              [.c.<center>]                                      }
{              [.r          ]                                     }
{              [.a          ]                                     }
{ <from addr>.<to addr>.<from segm>                               }
{ <from addr>.<to addr>.<from segm>.<to segm>                     }

<from - to>   ::= { <from addr>                 }
                  { <from addr>.<to addr>       }

<memory address> ::=
<first number>   ::=
<words per line> ::=
<first bit>      ::=
<last bit>       ::=
<center>         ::=
<from addr>      ::=
<to addr>        ::=
<from segm>      ::=
<to segm>        ::= integer
```

3.49.3 Function

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

The parameter list is scanned.

If <source> is the name of a backing storage area, *print* prints from this area. If it is the name of a bs-entry, i.e. its documents name is the name of a backing storage area entry, *print* prints from the subarea pointed out by the bs-entry.

If <source> is the name of an internal process, *print* prints from the memory area of the process, provided the cpa limit of the calling process is greater then the last physical address of the process specified.

If <source> is an integer, *print* prints from the memory area with the integer as physical base address.

Concerning the source modifiers, .s and .<first number>, cf. 3.49.5 and 3.49.6.

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

1) When a <format list> is recognized 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.

3.49.4 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 address and word is printed in octal. If code is also specified, the final address is printed in octal, too.
`hex`	current address and word is printed in hexadecimal. If code is also specified, the final address is printed in hexadecimal, too.
`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, control characters, ISO 0...31, and 127, 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 be 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 below. 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> <octal> <hex> <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> <unsigned left-most halfword> <instruction>.`

3.49.5 Field Specification

The area to be printed is limited by the integers <from addr> and <to addr>, <from addr> alone meaning <to addr> = <from addr>. If no <from addr> and <to addr> is specified in the field specification, the area is limited by the addresses <from addr> and <to addr> in an address space being:

- the address space of the area, if it is a backing storage area not containing a dumped internal process (contents <> 7), the base address being zero
- the logical address space of the dumped internal process, if the area is a backing strage area containing a dumped internal process (contents = 7), the base address being the first logical address of the process
- the physical address space of the internal process, if the area is the memory area specified by the name of an internal process, the base address being the first physical address of the process (logical address space and base address being the first logical address if the process is the calling process itself)
- the physical address space of the entire memory, if the area is a memory area specified by an integer, the base address being the physical address specified by the integer.

Specification of <from segm> and <to segm> normally makes sense for backing storage area only, but in special cases it could make sense for memory areas as well. When specified, a source area is considered divided into physical segments of 512 halfwords each, or, when the source modifier *.s* is used, into logical segments, where the first word of each segment contains the length of the segment. The areas fp, algol and fortran are such areas, most utility programs too, but they consist of one logical segment only. Data structures in memory areas may be organized that way, too.

When <from segm> and <to segm> are specified, <from segm> alone meaning <to segm> = <from segm>, the part of the segments limited by the relative addresses <from addr> and <to addr> in each segment is printed with addresses relatively numbered inside each segment.

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 backing storage 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 become <from addr> below the center and <to addr> above the center. (If the source is a backing storage 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 backing storage 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, cf. below.

3.49.6 Numbering of Addresses

The addresses listed in the left column of the output will be numbered **relatively** from zero and up inside each segment, when <from segm> and <to segm> are specified in the field specification.

If no <from segm and <to segm> is specified, the addresses are numbered **consecutively** from the base address of the area, starting by :

- zero if the area is a backing storage area not containg a dumped internal process (contents < > 7)
- first logical address if the area contains a dumped internal process (contents = 7)
- first physical address if the area is a memory area specified by the name of an internal process
- zero if the area is a memory area specified by a physical address.

If the source modifier .<first number> is used, the consequtive numbering of addresses will instead start with :

- first number, if the area is a backing storage area with any contents key
- first number + first logical address, if the area is a memory area specified by an internal process name (probably only first number = 0 makes sense)
- first number, if the area is a memory area specified by a physical address (probably only first number = 0 makes sense)

If the field specifier .r is used, it overrides the use of .<first number>, renumbering the addresses relatively to the base address, starting with zero.

Survey of numbering options with no <from segm> and <to segm> :

Options :		First Number: Last Number:	
print <bs area>	<from>.<to>	<from> - <to>	*
print <bs area>	<from>.<to>.r	<from> - <to>	
print <bs area>.no	<from>.<to>	no +<from> - no +<to>	
print <dump>	<from>.<to>	<l.f.> - <l.t.>	*
print <dump>	<from>.<to>.r	<from> - <to>	*
print <dump>.no	<from>.<to>	no +<from> - no +<to>	
print <proc>	<from>.<to>	<p.f.> - <p.t.>	*
print <proc>	<from>.<to>.r	<from> - <to>	*
print <proc>.no	<from>.<to>	no +<l.f.> - no +<l.t.>	**
print <addr>	<from>.<to>	addr+<from> - addr+<to>	*
print <addr>	<from>.<to>.r	<from> - <to>	*
print <addr>.no	<from>.<to>	no +<from> - no +<to>	

<l.f.> means 'logical from address'
<l.t.> means 'logical to address'
<p.f.> means 'physical from address'
<p.t.> means 'physical to address'

* means usefull
** means usefull if no = 0

3.49.7 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 limit violation**
The field specification together with the modifier attempts to define an address exceeding legal limits.

If printing from an internal process or from memory addresses, make sure the CPA limit of the printing process is above the locations to be printed.

*****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 memory size**
No memory 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.

3.49.8 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 algol.s text all 10.20.0.4
```

prints halfwords 10 to 20 of the first 5 passes of ALGOL as text integer bits.0.11 code.

```
print image.0 0.16 16.10.c.12 1594.1604 1614.1616 1598.1582.i,
bits.0.0.1.23 1604.1636.i
```

prints relevant parts after break of ALGOL program (cf. ref. [12]).

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

```
14 interrupt cause
16 sb

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

1594 UV
1596 UV
1598 lastused
1600 last of program
1602 first of program
1604 segment table base
1614 saved stack ref
1616 saved w3

1598.1582.i total stack

1604.1634.i entire segment table
```

```
p=slang list.yes ptx

     1    0
     1    0  b. e2, g1                ; begin block insertproc
     2    0  w.
     3    0
     3    0  d.
     7    0  fpnames

   110    0
   110    0
   110    0  s. a2                    ; begin segment for program
   111    0  w.
   112    0
   112    0  k = h55
   113 1536
   113 1536       0                   ;
   114 1538  a0:  0                   ;
   115 1540
   115 1540  e0:  al. w0  a0.-2       ;
   116 1542       al. w1  a1.         ;
   117 1544       ds. w1  a0.         ;
   118 1546
   118 1546       jl.    -1           ;
   119 1548       jl.    -2           ;
   120 1550       jl.    -3           ;
   121 1552       jl.    -4           ;
   122 1554       jl.    -5           ;
   123 1556  a1:  jl.    -6           ;
   124 1558
   124 1558  e.                       ; end segment
   125 1558
   125 1558  e1 = k  - h55            ; load length
   126 1558  e2 = e0 - h55            ; entry point
   127 1558
   127 1558  g0:
   128 1558  g1:  (:e1+511:) > 9      ; size
   129 1560       0, r.4              ; name
   130 1568       s2                  ; date
   131 1570       0, 0                ; file, block
   132 1574       2<12+e2             ; contents.entry point
   133 1576       e1                  ; load length
   134 1578
   134 1578
   134 1578  d.
   138 1578  insertproc

slang ok 1/22/1

p

att s
break
ready

from fgs
```

```
***break 8 151312

from fgs break fp

att s
dump imagep
ready

att s
start
ready

from fgs
***fp reinitialized

fp version 2 release 4.00

print imagep integer half 0.12

  imagep
149766.    149766    36 -1786
149768.    151302    36  -250
149770.    151322    36  -230
149772.    248672    60 -1184
149774.    248672    60 -1184
149776.         1     0     1
149778.    151312    36  -240

print imagep integer 0.12.r

  imagep
       0.    149766
       2.    151302
       4.    151322
       6.    248672
       8.    248672
      10.         1
      12.    151312

print imagep integer 1536.1538 code 1536.1538.i all 2.4.c.12 2.c.12

  imagep
151302.    151302
151304.    151322


  imagep
151302.  00  w2 (  -250)
151304.  00  w2 (  -230)
151306.  al. w0    -4   151302
151308.  al. w1     14   151322
151310.  ds. w1    -6   151304
151312.  jl.       -1   151311
151314.  jl.       -2   151312
151316.  jl.       -3   151313
151318.  jl.       -4   151314
```

```
151320.  jl.       -5   151315
151322.  jl.       -6   151316


 imagep
151310.  -2256902 3544 ds. w1    -6   151304
151312.   3444735  840 jl.       -1   151311
151314.   3444734  840 jl.       -2   151312
151316.   3444733  840 jl.       -3   151313

 imagep
151310.  -2256902 3544 ds. w1    -6   151304
151312.   3444735  840 jl.       -1   151311
151314.   3444734  840 jl.       -2   151312


print imagep 151312.151316.a

 imagep
151312.   3444735  840 jl.       -1   151311
151314.   3444734  840 jl.       -2   151312
151316.   3444733  840 jl.       -3   151313


print imagep 151312.a.r

 imagep
  1546.   3444735  840 jl.       -1   1545
```

3.50 procsurvey

For each procedure or standard ALGOL or FORTRAN identifier in the parameter list, the program displays on current output

- the procedure type
- the parameter specifications
- the procedure date
- the identifier type or
- the Run Time System entry number

3.50.1 Example

The FP command:

```
procsurvey invar in
```

will produce the output:

```
integer   procedure   invar      :   d.880822.1005   (algol)
  param 1: zone
zone              in         , rs entry no.: 26
```

3.50.2 Call

```
procsurvey { <name>      }0-*
```

The (algol) or (fortran) given for each procedure represents information internally used by the runtime system, and does not state anything about source code etc.

3.50.3 Function

Each name is looked up in the catalog, if several entries with the same name exist, 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.

3.50.4 Storage Requirements

2500 halfwords plus space for FP.

3.50.5 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 or FORTRAN standard identifier.

*****procsurvey <name> entry inconsistent**
The start external list in the entry description contains a halfword > 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.

3.51 release

Sends a release message to the parent (the operating system) releasing the specified magnetic tape reel (cf. ref. [10], section 6.1).

3.51.1 Example

If the total number of tape reels used during a BOSS job exceeds the number 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 mt16 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 no longer required.

3.51.2 Call

```
release <name>
```

3.51.3 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.

3.51.4 Storage Requirements

1536 halfwords plus space for FP.

3.51.5 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.

3.52 rename

Changes the names of catalog entries as specified.

3.52.1 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 a possibly associated data area remain unchanged.

3.52.2 Call

```
                                                1-*
rename { <oldname> . <newname> }
```

3.52.3 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).

3.52.4 Storage Requirements

1536 halfwords plus space for FP.

3.52.5 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 <newname> no room**
The name overflow in the main catalog exceeded the limit.

*****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.

3.53 repeat

The program makes it possible to repeat (a specified number of times) an FP compound command, i.e. a series of FP commands placed in brackets, or the last part of an FP command.

3.53.1 Example

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

```
t1=set mt16 mt471100
t2=set mt16 mt471200

(repeat   20
 nextfile t1 t2
 copy     t1 t2)
```

3.53.2 Call

```
({<FP command>}1-*

[<outfile>=] repeat <total number of times>  ,

<parameter list> <NL>

{<FP command>} )1-*

<total number of times>::= an integer greater than 0

<parameter list>       ::= any sequence obeying the FP syntax

<FP command>           ::= {<simple command>  }
                           {<compound command>}
```

3.53.3 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.

3.53.4 Storage Requirements

512 halfwords plus space for FP.

3.53.5 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.

3.54 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 specified file as job file. BOSS only accepts replace messages from off line jobs, not from on line jobs.
The operating system s removes the job process as if the command had been 'finis' and reads from the file as by the s command 'read'.

3.54.1 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 and the identification to be changed according to the job head in the file 'pip'.

3.54.2 Call

```
replace <job file>  (oldid/newid)
```

where <job file> is a name of a permanent disk file. Default is 'oldid'.

3.54.3 Function

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

3.54.4 Storage Requirements

1536 halfwords plus space for FP.

3.54.5 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.

3.55 rewind

Sends a rewind operation to a magnetic tape document and returns.

3.55.1 Example

By the fp command

```
rewind mt280007
```

the magnetic tape mt280007 starts rewinding, and the program returns to let you perform other jobs while the rewinding takes place.

By the fp command

```
rewind t
```

the same action is performed, provided t is a file descriptor, like :

```
t=set mt62 mt280007
```

If you know that the tape is mounted on device number 12, you could as well go :

```
rewind 12
```

3.55.2 Call

```
rewind <tape spec>

                        {<document name>}
<tape spec>       ::= {<file     name>}
                        {<device number>}

<document name> ::=
<file     name> ::= name

<device number> ::= integer
```

3.55.3 Function

A rewind operation is sent to the magnetic tape document. The name of the document is found as follows : if the parameter is an integer, and the external process with that device number is a magnetic document (kind = 18), the name of the process is used, else the name is looked up in the catalog. If an entry describing a magnetic tape is found (kind = 18 and mode even), the document name of the entry is used, else the name specified is used. The magnetic tape process is reserved and the operation is sent, waited for and when the answer returns, the process is released. If any significant status bits are raised, the program returns with device status error, else it returns normally.

3.55.4 Error Messages

*****rewind call**
Left hand side in call of rewind.

*****rewind parameter missing**
No magnetic tape specification parameter.

*****rewind not device**
The device number specified does not specify an external process.

*****rewind device not mt**
The device number specified is not a magnetic tape process.

*****rewind modekind error**
The filedescriptor found does not specify a magnetic tape.

*****rewind no process**
The <tape spec> did not specify an existing process.

3.56 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.
The function of the program is taken over by the program 'enable'.

3.56.1 Call

```
ring <name>
```

3.56.2 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.

3.56.3 Storage Requirements

1536 halfwords plus space for FP.

3.56.4 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.

3.57 rubout

Overwrites the contents of the specified backing storage files with a 'rubout' fill pattern. If specified, the catalog entry is removed after the rubbing out.

3.57.1 Examples

By the FP command

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

the file text4 is overwritten with the rubout fillpattern 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.

3.57.2 Call

```
                                        0-*
rubout <scope>  (name              )
                (clear.(yes/no))

<scope>::= ( temp    )
           ( login   )
           ( user    )
           ( project )
           ( own     )
```

3.57.3 Function

The files are filled with a fillpattern after which it is cleared in case the value of the parameter is yes. The fillpattern is a double word consisting of 3 NUL characters and 3 EM characters. Scope own means all of temp, login, user, and project.

3.57.4 Storage Requirements

1536 halfwords plus space for FP.

3.57.5 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.

3.58 save13

The program transfers catalog entries and backing storage areas to magnetic tape for backup purpose.

The catalog entries and backing storage areas are transferred back to disk by the program load13.

The program is replaced by the program save and should only be used to produce back up tapes to be read by the program load13.

3.58.1 Examples

3.58.1.1 Example 1

All catalog entries and backing storage areas of scope temp are transferred to the magnetic tape mtdp0001 file 2 by the call:

```
save mtdp0001.2
```

In case

```
t = set mtl6 mtdp0001 0 2
```

the same is obtained by the call:

```
save t.0
```

3.58.1.2 Example 2

All catalog entries and corresponding backing storage areas specified in the file 'savefiles' are transferred to file number 2, overwriting the ones saved in example 1, by the call:

```
save mtdp0001.2 in.savefiles
```

3.58.1.3 Example 3

All catalog entries with corresponding backing storage areas of scope project, those of scope user on the disk named 'disc3' and finally the best entry with the name 'pap' are saved after the ones in example 2 by the call:

```
save mtdp0001.last scope.project,
     disc.disc3 scope.user,
      pap
```

3.58.2 Call

```
[ <outfile> = ]  ,
                                                    1-2
save [<mount param>] (<tape param>)   [<special param>]   ,

[<save specifier>]
```

3.58.2.1 outfile

```
<outfile> ::= name of any filedescriptor
```

3.58.2.2 mountparam

```
                                                          1-*
<mount param>     ::= { mountspec. <device no> }
                      { release. (yes/no)      }
                      { <modekind>             }

<device no>       ::=    <integer>

<modekind>        ::=  { mthh }
                       { mtlh }
                       { mthl }
                       { mtll }
```

3.58.2.3 Tape param

```
<tape param>      ::= <tape name>.<file no> (.<next volume>)0-31,
                      [.label.<fpname>]

<tape name>       ::=

<next volume>     ::= (<name>/<filedescriptor>)

<name>            ::= name of magnetic tape

<file descriptor> ::= name of magnetic tape file descritor

<file no>         ::= (<integer>/last)

<fpname>          ::= name obeying fp syntax
```

3.58.2.4 Special param

```
                                                    1-*
<special param> ::   { segm. <integer>      }
                     { list. (yes/no/name)  }
                     { reserve.(yes/no)     }
```

3.58.2.5 Save specifier

```
                                                    1-*
<save specifier>  ::= {<modifier>        }
                      {<disk specifier>  }
                      {<entry specifier>}

                                                                  1-*
<modifier> ::=  {changedisc {.<from disk>.<to disk>}1-* }
                {newscope. <new scope>                  }

<from disk>       ::=  {all/maincatdisc/<disk name>}

<to disk>         ::=  {no/maincatdisc/<disk name>}

<new scope>       ::=  {no/temp/login/user/project}

                                                 1-*
<disk specifier>  ::=  disk {.<from disk>}

                                                                0-*
<entry specifier> ::=  <entry factor> {.<entry factor>}

<entry factor>    ::= {<entry name>
                      /scope.<scope>/docname.<docname>}

<entry name>      ::=  <name>

<scope>           ::= {temp/login/user/project/
                       own/system/perm/all}
```

3.58.2.6 Infile parameter

Everywhere the delimiter <s> is allowed in the parameter list according to syntax for FP commands (ref. 8), the parameter pair <s> in.<file> is allowed and will be syntactically equivalent to <s>.

```
<file> ::= name of any filedescriptor.
```

3.58.3 Function

The program will save the catalog entries and possible backing storage areas specified by <disk specifier> and <entry specifier> with the modifications in <modifier> on the magnetic tapes specified in <tape param>, i.e. maybe in two copies and maybe extending over more volumes.

The program interprets the parameter groups, one by one.

The tape is mounted according to possible <mount param> and positioned to the file number(s) specified.

A version dumplabel record (cf. below) is output as the first block and displayed on current output.

Each <entry specifier> starts a catalog scan, picking out the entries specified, and the entries satisfying the current disk specifications are saved after a possible change according to the actual state of modifiers. During the save, succeeding magnetic tape volumes, as far as specified, are mounted whenever actual volume is filled up.

The last block in each volume will be a continue record, specifying the number of entries and segments saved so far and the name of the next volume. The first block of the next volume will be a continuation dump label record, which is displayed on current output as well.

When the parameter list is emptied, the magnetic tape file is closed with an end record as the last block, specifying the amount of entries and segments saved.

The tape is positioned to the next file, an empty dump label record is output in the file and displayed on current output, the file is closed and the tape released if so specified in <mount param>.

If two sets of tapes are specified they are treated in parallel, except for different <mount param> and except for volume change, which allows for tapes of different lengths in the two sets.

3.58.3.1 Function, outfile parameter

```
<outfile> ::= name of any file descriptor
```

Current output zone is stacked and connected to the file specified at program start.

If the program terminates through its final end, the current output zone is unstacked again.

If the program terminates with a runtime alarm, the current output zone remains connected to <outfile>.

3.58.3.2 Function, mount param

```
                                                    1-*
<mount param> ::= (mountspec .<device no>)
                  (<modekind>              )
                  (release. (yes/no)       )
```

<mountspec. <device no>
A mount special parent message with the device number and proper tape name will be sent each time a new volume in the set of tapes specified in <tape param> is mounted.
Default: mountspec.0
meaning no parent message.

<modekind>
The modekind specified will be used for all volumes in the set specified in <tape param>.
Default: mto

release. (yes/no)
If release.yes the tape in the set actually used at program termination will be released. Default: release.yes

The mount parameters may be repeated, but the last one will stand.

3.58.3.3 Function, tape param

```
<tape param> ::= <tape name>.<file no> (.<next volume>)0-31,
                 [.label.<fpname>]
```

One tape parameter specifies a set of magnetic tapes, consisting of one to thirtytwo tapes.

<tape name> ::=
<next volume> ::= <name>/<file descriptor>
The name of the tape. If <file descriptor> is used, the name and modekind are taken from the descriptor.

<file no> ::= (<integer>/last)
The file number of the first tape in the set where the save shall start.

The save will start in file number one in all succeeding volumes. If the first tape is specified by <file descriptor>, its file count will be added to <file no>.

If <file no> = last, the first file, which does not start with a version or a continue dump label, is searched along the tapes in the set, even if they are specified by <file descriptor>'s.

label.<fpname>
If a label is specified, the name will be written as the last field in the version dumplabel record, and it will appear on current output.

3.58.3.4 Function, special param

```
                                                    1-*
<special param> ::= (segm.<integer>      )
                    (list. (yes/name/no))
                    (reserve. (yes/no)   )
```

The special parameters may be repeated, but the last one will stand.

segm. <integer>
Backing storage areas will be saved in magnetic tape blocks of <integer> segments, 1<=<integer><=9.
Default: the value found in the file count of the programs own catalog entry tail. If the value is outside the legal value interval, the value one is used.

list. (yes/name/no)
If list.yes, the entries saved are listed on current output in the form:

<name><modekind> <scope>/<permkey><docname> ,
[<base>] [<shortclock>]

If list.name, only the entry name is listed.
If list.no, the entries are not listed.
Default: list.yes

reserve.(yes/no)
If reserve.yes is specified, the program will try to reserve the area process for an area entry to be saved, unless the monitor offers write protection, cf. 3.58.4.
If reserve.no is specified, no reservation is tried.
If the monitor offers write protection, the parameter has no function.
Default: reserve.yes

3.58.3.5 Function, save specifier

```
                                                    1-*
<save specifier> ::= (<modifier>                   )
                     (<disk specifier>             )
                     (<entry specifier>            )
```

The elements of the save specifier are treated one by one, until the parameter list is emptied.

A modifier will modify the state of current modifiers.

A disk specifier will cancel current disk specifiers and define a new set of disk specifiers.

An entry specifier will start a catalog scan, picking out entries specified.

Entries picked out which belongs to a disk specified in current disk specifiers will be saved after having been modified according to current modifiers.

If no <entry specifier> is found in the parameter list after a modifier or a disk specifier or no save specifier is found, a default entry specifier will be used.

```
                                                              1-*
<modifier>::= (changedisc (.<from disk> .<to disk>)1-*       )
              (newscope. <newscope>                           )
```

A modifier is valid until changed by another modifier.

`changedisc {.<from disk>.<to disk>}1-*`

An entry belonging to <from disk> will be changed as if it belongs to <to disk>.

`<from disk>::=`	`{all/maincatdisc/<disk name>}`
`all`	means all disks
`maincatdisc`	means the disk with the main catalog
`<disk name>`	means the disk with that name
`<to disk> ::=`	`{no/maincatdisc/<disk name>}`
`no`	means `changedisc. <from disk>.<from disk>`
`others`	as for `<from disk>`

Default, `changedisc.all.no`

`newscope. <new scope>`
All entries will be changed to have the scope <newscope>
`<newscope> ::= {temp/login/user/project/no}`

If the entries are saved using a scope key (cf. specifier 'scope') they will be saved with the one denoting <newscope>.

If they are saved using bases they will be saved with permkey and bases corresponding to <newscope>.
`<newscope> = no` means no change of scope.
Default: newscope.no

`<disk specifier> ::= disc {.<from disk>}1-*`

A disk specifier is valid until the next disk specifier, which will cancel it.

`<from disk> ::= {all/maincatdisc/<disk name>}`
The parameters have the same meaning as for 'changedisc'.
Default: disc.all

`<entry specifier> ::= <entry factor> {.<entry factor>}0-*`
An entry specifier is composed of one or more entry factors of which three kinds exist, each of which specify an entry attribute.

If one kind of entry factor is repeated, the last one stands, except for the factor <name>, where a warning will be given and the first name stands.

The entry specifier specifies a set of entries, each of which have all the attributes specified

`<entry factor> ::= {<name>/scope.<scope>/docname.<docname>}`

`<name>`
The attribute is the entry name.

All entries visible with this name have the attribute. The entry names 'c' 'v' and primout cannot be specified.

Entries of name 'c' or 'v' with permkey = 0 and entries with name 'primout' and permkey = 2 are considered to have no name attribute.

```
scope. <scope>
<scope> ::= (temp/login/user/project/own/system/perm/all)
```

The attribute is the scope.
All entries with the scope specified have the attribute.

temp, login, user, project	have the usual meaning
own	means one of above.
system	means permkey = 3 and entry base = system base
perm	means permkey = 3 and entry base inside or equal to the standard base of the process
all	means any permkey and entry base inside or equal to the standard base of the process.

Entries specified by the attribute scope.perm or scope.all will be saved using permanent key and entry base. Entries specified by any other scope will be saved using a key denoting the scope, making them loadable in any process (scope.system in a process with maxbase = system base, though).

cf. the modifier 'newscope'.
Default: name specified: the best name
no name specified: temp

```
docname. <docname>
```
The attribute is the document name of the entry. All entries visible with a document name equal to <docname> have the attribute.
Changes the default for scope to <any visible scope>.

The best name means the name with the best scope among the scopes temp, login, user and project or any scope visible changed into one of above by the modifier 'newscope'.

Default entry specifier: scope. <temp>

3.58.3.6 Function, infile parameter

Every where the delimiter is syntactically correct in the parameter list, the parameter pair <s>in.<filename> is allowed and syntacitally equivalent to <s>.

```
<filename> ::= name of any file descriptor
```

When in.<filename> is met in the parameter list, current input zone is stacked and connected to the file specified by the file descriptor, and the parameter reading is continued in current input zone.

The parameter reading takes place using the special fp input alphabet and according to normal fp parameter syntax (cf. Ref. [9]), except the character 'nl' is equivalent to the character 'sp', and except names of the types: apostrophized name, general name and general text.

When the separator 'em' is met, current input zone is unstacked and parameter reading continues from current input zone, except when unstacked to the initial level, in which case parameter reading continues in the fp command stack.

In case of illegal character or fp syntax error a syntax alarm is written on current output zone, current input zone stack chain is emptied, listing the chain on current output zone, and parameter reading continues in fp command stack.

The fp command listing governed by the mode bit 'list' will list the parameters in the fp command stack, not the parameters in any file.

3.58.3.7 Alternative parameter names

For compatibility reasons, some alternative parameter names are allowed.

The modifier 'changekit' is allowed and is equivalent to 'changedisc'.

The changedisc parameter <from disk> = main is allowed and equivalent to <from disk> = all.

The changedisc parameter <to disk> = main is allowed and is equivalent to <to disk> = maincatdisc.

The disk specifier 'kit' is allowed and is equivalent to the disk specifier 'disc'.

The disk specifier parameter <from disk> = main is allowed and is equivalent to <from disk> = all.

3.58.4 Entries and backing storage areas

Catalog entries are picked out of the main catalog according to <entry specifiers>, checked with <disk specifier>, changed according to <modifier> and transferred to a record, which is output on the tape as one block (cf. 3.58.5, Tape Format).

If, however, the entry is an area entry, steps are taken to protect the area during the save.

Before the entry is changed, the catalog base of the process is changed to equal the entry base. If, however, the entry base is outside the max base of the process, to equal the max base.

An area process with the name of the entry is created. If the creation fails, the entry is skipped with a warning on current output, telling the reason of the failure.

Next, the area process is write protected, unless its base is outside the max base of the process.

If the current monitor does not offer write protection, the process is reserved instead, unless reserve.no is specified.

If the protection fails because the area process is already reserved by another process, the entry is skipped with a warning on current output.

If the base of the area process created does not equal the entry base, i.e. the entry base is outside max base and a better entry exists, the entry is skipped with a warning on current output, telling that the entry is inaccessible.

At last, the write access counter and the name table address of the area process are read and the catalog base of the executing process is reset.

Now the entry is changed and saved, followed by the segments of the backing storage area.

When the area has been saved, the write access counter of the area process is compared with its value before the area was saved. If the value has changed, i.e. the entry base is outside the max base of the existing process (it was not protected) and another process has had write access to it, or the executing process itself has had write access to it (area connected to current output zone), the entry is listed on current output followed by a warning, that the area has been changed during the save.

Finally, the number of segments saved is compared to the size of the area in the entry tail, and if not equal, the entry is listed on current output followed by a warning, that the area and the entry are inconsistent.

The area process is removed, unless the area is the program itself.

Foreign read accesses to the area will not influence the save.

If the current monitor does not offer write protection, foreign write accesses will only hold up the save if reserve.no is specified or the entry base is outside max (scope.system).

If the current monitor offers write protection, foreign read accesses will not be affected by the save.

If the current monitor does not offer write protection, foreign read accesses will be held up by the save, unless reserve.no is specified or the entry base is outside max base.

Foreign write access during the save will be rejected unless reserve.no is specified or the entry base is outside max base of executing process.

Using one of the entry specifiers scope.perm or scope.all, which are intended for system back-up, entries with a base equal to or inside the standard base of the executing process are specified.

Areas saved this way (as well as areas saved using temp, login, user, project or own) will never be changed by other processes during the save. Either they are protected during the save or they are skipped because they are reserved by another process at the time of reservation.

3.58.5 Tape Format

The magnetic tape file created by save starts with a block containing a

- version dump label record

followed by a number of blocks each containing

- an entry record or

- a segment record

ending with a block containing

- an end record.

The file may extend over more tapes, in which case each tape ends with a block containing

- a continue record

and each new volume tape starts with a block containing

- a continuation dump label record.

After the file, save creates another file with only one block containing

- an empty dump label record.

The segment records are 8+segm*512 halfwords long, all other records are 100 halfwords long.

The format and contents of the records are as follows.

Dump label records:
version-, continuation- and empty label records contain a number of characters, terminated by an em-character (the records may be read by edit or copy):

```
hw addr :  contents
 0 -  3 :  <:dump :>
 4 - 11 :  tapename followed by a number of spaces
12 - 15 :  file number followed by a number of
           spaces
16 - 19 :  <:vers. :>,<:cont. :> or <:empty :>
20 - 23 :  <<ddmmyy>, date,
24 - 27 :  <<.hh >, hour,
28 - 31 :  <:s=<segments per block>:>
32 - 39 :  true, 12, label.<fpname>,
40 - 47 :  <:release:>, <<dd.d>,
48 - 51 :  <:<10><0><0><0><0><25>:>
52 - 99 :  null characters
```

Entry record:
An entry record contains parts of a catalog entry head and all of the tail:

```
hw addr :  contents
 0 -  3 :  1, if saved using bases then 52 else 48
 4 -  7 :  entry count, no of segments
 8 - 15 :  entry name
16 - 35 :  entry tail
36 - 39 :  if saved using bases then permanent key
           else scope key (temp=8, login=7, user=6,
           project=5, system=3)
40 - 47 :  disk name
48 - 51 :  if saved using bases then (entry base
           lower, entry base upper) else 1,48
52 - 99 :  if saved using bases then (1,52) else
           (1,48),...
```

Segment record:
A segment record contains some saved segments:

```
hwd addr : contents
  0 -  3 : 2, 8+512* number of segments in record
  4 -  7 : entry count, segment count
  8 -519 : contents of one segment
520-1031 : contents of one segment
     .
     .
     .
```

End record:
An end record contains the entry and segment counts:

```
hwd addr:  contents
 0 - 3  :  3,8
 4 - 7  :  entry count, segment count
 8 -99  :  3,8,...
```

Continue record:
A continue record contains the name of the continuation tape:

```
hwd addr:  contents
 0 - 3   :  4, 16
 4 - 7   :  entry count, segment count
 8 -23   :  name of the continuation tape
 24-99   :  4, 16, ...
```

3.58.6 Requirements

The program will allocate memory space to internal data buffers of at least 2*segm*512 halfwords.

If that does not leave space for two program segments, the program will terminate with a stack alarm, in which case the size of the process must be increased.

If free memory space at time for buffer allocation leaves space for more than enough program segments to avoid program paging in inner loops, the memory surplus will be allocated to databuffers, giving maybe double buffered tape zone and all the rest to extend a single buffered disk zone.

The program needs:

```
4  area processes (fp, save, catalog, area to be saved)
   +
1  area process if current output zone is connected to a
   backing storage area +
1  area process as long as current input zone is
   connected to a backing storage area for parameter
   reading.
```

If the program fails to have the necessary number of area processes, it will

- write a parameter alarm and continue writing or reading in the present file, if the resource was missing at time for zone connection
- skip area entries with a warning (cf. 3.58.7).

The program may need temporary entries and segments to perform zone stacking in case of parameter input from backing storage area. In case of shortage of these resources, the program will terminate with a 'break 10' alarm.

3.58.7 Error Messages

Error messages from the program are written in current output zone.

The following kinds of error messages exist:

- parameter alarm
- parameter warning
- save specifier warning
- area entry warning
- parameter input syntax error message

3.58.7.1 Parameter alarm

At parameter alarm, the parameter list is emptied, listing the parameters from current parameter and on in the alarm message.

The modebits are set: 'warning yes, ok.no'.

Since the parameter list is emptied, the program terminates.

***** save alarm <text> <parameter list>**

Text:	Explanation:
`mountspec param syntax`	mount param not followed by .<integer>
`tape param too many volumes`	tape param specifies more than 32 volumes
`label param syntax`	label not followed by .<name>
`tape param missing`	no tape parameter found
`segm param syntax`	segm not followed by .<integer>

3.58.7.2 Parameter Warning

The parameter warning skips current parameter, displaying it in the error message and continues, setting the modebits:
`'warning yes, ok.yes'`

***** save warning <text> <current parameter>**

Text:	Explanation:
`outfile param connect impossible <cause>`	The current output zone could not be connected to the file specified for the reason explained in <cause>. The stacked output zone is unstacked again and output continues.
`infile param connect impossible <cause>`	The current input zone could not be connected to the file specified for the reason explained in <cause>. The stacked input zone is unstacked again and input continues, maybe from fp command stack.
`mountspec param syntax`	The parameter 'mountspec' was not followed by .<integer>. The value remains the latest read or

	default.
`release param syntax`	The parameter 'release' was not followed by .yes or .no. The value remains the latest read or default.
`list param unknown`	The parameter 'list' was not followed by .yes, .no or .name. The value remains the latest read or default.
`reserve param unknown`	The parameter 'reserve' was not followed by .yes or .no. The value remains the latest read or default.
`changedisc param syntax`	The parameter 'changedisc .<from disk>' was not followed by .<name> The value becomes <to disk> = no.
`newscope param syntax`	The parameter 'newscope' was not followed by .<name> The value is not changed.
`newscope param unknown`	The parameter to newscope was neither of temp/login/user/project /no The value is not changed.
`disc spec param unknow`	The disk specified in disk specifier was unknown. Previous disk specifier is cancelled, all other disks specified in this disk specifier become specified.
`scope param syntax`	The scope parameter was not followed by .<name> No entries will be saved according to this entry specifier.
`scope param unknown`	The scope specified was neither of temp/login/user/project/ own/system perm/all. No entries will be saved according to this entry specifier.
`docname param syntax`	The 'docname' parameter was not followed by .<name> No entries will be saved according to this entry specifier.
`name illegal`	The name specified in entry specifier was 'c', 'v' or 'primout'. The entry is not saved.

`name double defined`	A name was already specified. The new name is ignored and the program continues with the current entry specifier.
`save spec param unknown`	Syntactically the parameter would start a save specifier, but the parameter is not <name>. The rest of the parameter list is read, each parameter with this warning as a result.

3.58.7.3 Save Specifier Warning

The save specifier is listed in the error message, the mode bits are set 'warning.yes, ok.yes' and the program continues.

***** save no entries found/saved according to following specifier**

```
disc: <disk specifier>
entry: <entry specifier>
```

Explanation:
No entries are found in the main catalog according to the save specifier or the entries specified have been skipped cf. below.

3.58.7.4 Area entry warning

The warning appears in current output following the entry concerned.

The modebits are set 'warning.yes, ok.yes' and the program continues.

<entry>
***** warning: area changed during save**

Explanation:
Some other process has had write access to the backing storage area during the save (or the area is connected to current output zone of the program itself and own process has had write access during the save).

The entry and the area have been saved.

The warning appears no matter what the list parameter specifies.

<entry>
***** warning: area and entry inconsistent, area length <integer> segments**

Explanation:
The number of segments actually saved do not equal the size in the catalog entry tail.

The entry and the area have been saved.

The warning appears no matter what the list parameter specifies.

<entry>
***** warning: entry skipped <cause>**

Explanation:
The area proces could not be created, not be protected/reserved or the area was inaccessible for the reason stated in <cause>.

The entry and the area have not been saved.
The warning appears only if list.yes or list.name is specified.

Cause:	Reason:
area claims exceeded	create area process failed
catalog i/o error, state of document does not permit call	create area process failed
entry not found	create area process failed
entry not an area entry	create area process failed
name format illegal	create area process failed
reserved by another process	set write protect/reserve failed
process does not exist, process is not user of area process	set write protect/reserve failed
area process inaccessible	area is inaccessible from executing process

3.58.7.5 Parameter input syntax error message

This message is caused by a syntax error in the parameter list read from a file connected to current input zone.

The parameter list must follow the syntax of an fp parameter list and must be coded in the special fp input alphabet, cf. Ref. 9, with the one exception that the character 'NL' is equivalent to the character 'SP'.

The current parameter is listed in the error message and the zone stack chain is emptied, listing the chain on current output the same way fp does in an fp syntax error message.

The parameter reading is continued in the fp command stack.

The modebits are left unchanged.

```
*** save syntax <parameter>
  * read from <file>
  * selected from <file>
  .
  .
  .
*** save reinitialized
```

3.58.8 Further Examples

3.58.8.1 Example 1

The file pip of scope user and all files with documentname pip and scope user are saved on mtdp0001 file 1 by the call:

```
save mtdp0001.1 pip.scope.user docname.pip.scope.user
```

3.58.8.2 Example 2

All files of scope temp, login, user or project on the disks: disc, disc1 and disc2 are saved changing their disk names:

```
disc  becomes disc3
disc1 becomes disc2
disc2 becomes disc1
```

by the call:

```
save mtdp0001.1,
changedisc.disc.disc3.disc1.disc2.disc2.disc1,
disc.disc.disc1.disc2.scope.own
```

3.58.8.3 Example 3

All files in the main catalog, except

- the main catalog itself
- the auxiliary catalogs
- files of name c or v with permkey = 0
- files of name primout with permkey = 2
- files belonging to other disks than disc, disc1 and disc2

are saved, disk by disk by the call:

```
save in. magtapes,
disc.disc scope.all,
disc.disc1 scope.all,
disc.disc2 scope.all
```

provided the executing process has standard base = system base and the file 'magtapes' contain a tape parameter, e.g.

```
mtdp0001.1.mtdp0002.mtdp0003.mtdp0004.mtdp0005.
mtdp0006.1.mtdp0007.mtdp0008.mtdp0009.mtdp0010.
```

The files are saved in two copies, each copy containing at most 5 volumes.

3.59 scope

Changes the scope of catalog entries as specified in the call of the program.

3.59.1 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.

3.59.2 Call

```
                                    1-*
scope <scope spec> { <name> }

<scope spec> ::= <scope>  [. <device name>]

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

<device name>::= <name of disk>
```

3.59.3 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).

3.59.4 Scope Specification

The concept of scope of a catalog entry is explained in the ref. 10, section 4.1.

A device name in the scope specifications 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 where the data area is situated on the specified bs device.

3.59.5 Storage Requirements

2048 halfwords plus space for FP.

3.59.6 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 included in the backing storage system.

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. *)

*****scope <scope spec> <name> protected**
The job was not allowed to change the scope of the entry found. *)

*****scope <scope spec> <name> entry in use**
Another job was using the entry and hence the scope could not be changed. *)

*****scope <scope spec> <name> no resources**
The resources of the job did not allow the change of the entry scope. *)

*****scope <scope spec> <name> change bs device** impossible
The entry could not be permanented into the specified auxiliary catalog. *)

*****scope <scope spec> <name> catalog error**
Catalog error, monitor error, or hardware error.

*) The program continues with the next name in the name list.

3.60 search

Finds and lists all catalog entries with a given scope, possibly filtered by filters given. The filters work on the entry name and on the document name.

3.60.1 Example

By the fp command

```
search user
```

all entries of scope user are listed on current output.

By the fp command

```
search own
```

all entries of scope temp, login, user, or project are listed on current output, while the command

```
search user ret.tx
```

will list all entries of scope user which contain both the substring 'ret' and the substring 'tx' in either name or document name.

3.60.2 Call

```
[<outfile> =] search <scope spec> {<filter>}0-*

<scope spec> ::=      <scope>  [.<disc name>]

<scope>      ::=               {temp        }
                               {login       }
                               {user        }
                               {project     }
                               {system      }
                               {own         }
                               {<low>.<upp> }

<filter>     ::=   <substring> {.<substring>} 0-*


<substring>  ::=   {<name>                }
                   {<apostrophized name>}
                   {<generalized name>   }
                   {<general text>       }

<low>        ::=
<upp>        ::=   integer
```

3.60.3 Function

The main catalog is scanned, and a subset of it is listed with an output format as for *lookup*. If an outfile is specified, the list of catalog entries is printed on that file, otherwise current output is used. Messages from *search* are always printed on current output. If no filters are given, all entries from the main catalog according to the scope specification are listed, otherwise, the set of catalog entries is further delimited by means of filters (see filter specification below).

3.60.4 Scope Specification

The scope concept is explained in ref. [10], section 4.1. The scope own means any scope in the set temp, login, user, or project (cf. the example above). If a disk name is specified, only entries in the auxiliary catalog on that disk are candidates. The scope given by <low>.<upp> means all entries with entry interval equal to the interval <low>.<upp>, which will have to be contained in but not equal to the std. interval of the process.

3.60.5 Filter Specification

A filter consists of one or more substrings concatenated by period. If a list of filters exists, an entry selected for listing will only be listed if either its name or its document name contain all the substrings of at least one of the filters. The order of the substrings in a filter is irrellevant. Thus, in a possible list of filters, you may consider space as "or" and period as "and", where the precedence of "and" and "or" is as in algol.

3.61 set

Creates a new catalog entry with scope temp or changes an already existing entry (with scope temp) according to the parameters.

3.61.1 Example

An area named 'pip' with an area size of 20 segments on the disk 'disc3' and dated 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 mt62 mt314711 0 7
```

An area named 'image' on disc (intended for job process 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 s and BOSS will automatically set contents 7 when the dump is made).

3.61.2 Call

```
<newname> = set [<kind> [<docname> [<date> (<word>)0-4] ] ]

<kind>     ::= (<integer>                )
               (<integer1>  .<integer2>)
               (<modekind abbreviation>)

<docname>  ::= (<name>  )
               ( 0/1/2/3)

<date>     ::= (<integer>                )
               (<integer1>  .<integer2>)
               (d.<isodate>              )
               (d.<isodate>.<clock>      )

<word>     ::=  <integer>
                <integer>  .<integer>

<isodate>  ::= (<yymmdd>   / 0)
<clock>    ::=  <hhmm>
<yymmdd>   ::=
<hhmm>     ::=  <integer>
<newname>  ::=  name
```

```
<name>    ::=  name, apostrophized name,
                 generalized name or general text
```

3.61.3 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 <docname> is a 24 bits word.

1. <integer> : the integer is placed in the tail

2. <integer1> . <integer2> : is interpreted as two halfwords 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 is used.

4. <name> : only relevant for <docname>. 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 <docname>:

If <docname> is 0,
the area is, if possible, created on the disk with the most resources at key zero (temporary).

If <docname> is 1,
the area is, if possible, created on the disk with the most resources at key one (special temporary).

If <docname> is 2,
the area is, if possible, created on the disk with the most resources at key 2 (login).

If <docname> is 3,
the area is, if possible, created on the disk with the most resources at key 3 (permanent).

If <docname> is a name, apostrophized name, generalized name or general text,
the area is created on the bs device with this name.

3.61.4 Storage Requirements

1536 halfwords plus space for FP.

3.61.5 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 <result name> bs device unknown**
The bs device specified was not included in the backing storage system.

*****set <result name> no resources**
The resources of the job did not allow the wanted creation of a catalog entry.

*****set <result name> no room**
The name overflow in the main catalog exceeded the limit.

*****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.

3.62 setmt

Creates catalog entries of scope temp describing files on magnetic tape according to the parameters.

3.62.1 Examples

The FP command

```
pap=setmt mt004711.3
```

creates the same catalog entries as the FP commands

```
pap1=set mt62 mt004711 d.0 1
pap2=set mt62 mt004711 d.0 2
pap3=set mt62 mt004711 d.0 3
```

If t is a name of a file on this magtape, e.g.

```
t=set mt62 mt004711
```

the same result is obtained by

```
pap=setmt t.3
```

The FP command

```
f=set mt16 mt004711 0 2
f=setmt f.3.5
```

creates the same catalog entries as the FP commands

```
f3=set mt16 mt004711 d.0 5
f4=set mt16 mt004711 d.0 6
f5=set mt16 mt004711 d.0 7
```

3.62.2 Call

```
<resulting name> =
setmt <mtname>.{ <upper integer>                    }
               {                                    }
               { <lower integer>.<upper integer> }
```

If no <lower integer> is specified, it is set to 1.

3.62.3 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 moved.

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 file will be addressed relative to the file number. Otherwise the name specified is used.

3.62.4 Storage Requirements

512 halfwords plus space for FP.

3.62.5 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> no room**
The name overflow in the main catalog exceeded the limit.

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

3.63 skip

Bypasses parts of current input as specified in the parameter list.

3.63.1 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 remainder 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.

3.63.2 Call

```
                                        1-*
skip { <lines>                          }
     { <iso value>.<appearances>        }
     { <small letter>                   }

<lines>          ::= <integer>

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

<appearances>    ::= <integer>
```

3.63.3 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 is bypassed.

`<small letter>`

Skips up to and inclusive this letter.

3.63.4 Storage Requirements

1024 halfwords plus space for FP.

3.63.5 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.

3.64 suspend

Sends a suspend messsage to the parent (the operating system BOSS) 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. ref. 10, section 6.1).

3.64.1 Example

A worktape has been mounted by the FP commands

```
workfile=set mt62 0 0 1
mount workfile
```

The job has produced some output on 'workfile' but now needs 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.

3.64.2 Call

```
suspend <name>
```

3.64.3 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.

3.64.4 Storage Requirements

1536 halfwords plus space for FP.

3.64.5 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.

3.65 timer

Sends a timer message to the parent (the operating system BOSS) demanding a provoked interrupt after a certain time.

3.65.1 Example

The FP call

```
timer 30 2
```

will provoke an interrupt after 30 seconds.

3.65.2 Call

```
timer <run time> <break time>
```

where <run time> and <break time> are integers, denoting time in seconds.

3.65.3 Function

A timer message containing the two integers is sent to the parent. If BOSS is the parent <run time> will be the number of seconds to the interrupt, <break time> the number of seconds allowed the job to respond to the interrupt.

3.65.4 Storage Requirements

1536 halfwords plus space for FP.

3.65.5 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.

3.66 translated

The program prints the date of translation which is found in all SLANG, PASCAL or ALGOL/FORTRAN programs.

3.66.1 Example

The call

```
translated pppp q se
```

may produce the following output:

```
pppp translated by slang   89.01.16 12.33
q    translated by algol   88.12.09 14.56
q    translated by pascal  87.11.30
```

3.66.2 Call

```
translated { <name> }0-*
```

3.66.3 Function

If <name> describes a program, the file is connected and the program searches for the date, which will be output.

3.66.4 Storage Requirements

2048 halfwords plus space for FP.

3.66.5 Error Messages

*****translated call**
Left hand side in the program call.

*****translated param <parameter>**
Parameter error in the program call.

*****translated <name> not found**
The parameter was not found in the catalog.

*****translated <name> not program**
The catalog entry <name> does not describe a program.

*****translated <name> error**
The file <name> could not be connected.

*****translated <name> date not found**

3.67 unload

Sends an unload operation to a magnetic tape document and returns.

3.67.1 Example

By the fp command

```
unload mt280007
```

the magnetic tape mt280007 starts unloading, and the program returns to let you perform other jobs while the unloading takes place.

By the fp command

```
unload t
```

the same action is performed, provided t is a file descriptor, like :

```
t=set mt62 mt280007
```

If you know that the tape is mounted on device number 12, you could as well go :

```
unload 12
```

3.67.2 Call

```
unload <tape spec>

                        {<document name>}
<tape spec>      ::= {<file     name>}
                        {<device number>}

<document name> ::=
<file     name> ::= name

<device number> ::= integer
```

3.67.3 Function

A unload operation is sent to the magnetic tape document. The name of the document is found as follows : if the parameter is an integer, and the external process with that device number is a magnetic tape document (kind = 18), the name of the process is used, else the name is looked up in the catalog. If an entry describing a magnetic tape is found (kind = 18 and mode even), the document name of the entry is used, else the name specified is used. The magnetic tape process is reserved and the operation is sent, waited for and when the answer returns, the process is released. If any significant status bits are raised, the program returns with device status error, else it returns normally.

3.67.4 Error Messages

*****unload call**
Left hand side in call of unload.

*****unload parameter missing**
No magnetic tape specification parameter.

*****unload not device**
The device number specified does not specify an external process.

*****unload device not mt**
The device number specified is not a magnetic tape process.

*****unload modekind error**
The filedescriptor found does not specify a magnetic tape.

*****unload no process**
The <tape spec> did not specify an existing process.

Appendix A. References

Part numbers in references are subject to change as new editions are issued and are listed as an identification aid only. To order, use package number.

1) For RC8000
Monitor, Part 1, System Design 991 03577

For RC9000:
RC9000-10 System Software 991 11255
Included in SW9910I-D

2) For RC8000
Monitor, Part 2, Reference Manual 991 03588

For RC9000:
Monitor, Reference Manual 991 11259
Included in SW9890I-D Monitor Manual Set

3) *Monitor, Part 3, Definition of External Processes* 991 03435

4) *Operating System s, Reference Manual* 991 11260

Included in SW9890I-D, Monitor Manual Set

5) *ALGOL8, User's Guide, Part 2* 991 11280
Included in SW8585I-D, Compiler Collection Manual Set

6) *RC FORTRAN User's Manual* 991 11292
Included in SW8585I-D, Compiler Collection Manual Set

7) *RCSLANG Assembler Programming Guide* 991 11295
Included in SW8585I-D, Compiler Collection Manual Set

8) *System Utility, Part 1* 991 11263
Included in SW8010I-D, System Utility Manual Set

9) *BOSS Operating Guide* 991 11275
Included in SW8101I-D, BOSS Manual Set

10) *BOSS User's Guide* 991 11274
Included in SW8101I-D, BOSS Manual Set

11) *System 3, Utility Programs, Part 3* 991 11294
Included in SW8585I-D, Compiler Collection Manual Set

12) *Code Procedures and Run Time Organization of ALGOL Programs* 991 11296

13) *Save, Incsave; Load, Incload* 991 11313
Included in SW8010I-D, System Utility Manual Set

System Utility Programs, User's Guide, Part Two
PN: 991 11 264