RC9000-10/RC8000

SW8010 System Utility

Save, Incsave; Load, Incload

PN 99200389

RC Computer

Keywords:
RC9000-10, RC8000, System Utility, Utility Programs, Save, Load, Incsave, Incload, Tape Back-Up

Abstract:
This manual describes the programs *save, load, incsave* and *incload*, all part of the System Utilities package, which are used for taking back-up on magnetic tape.

Date:
February 1989

PN: 991 11310

Table of Contents

Foreword

In System Utility Package, SW8010/2, Release 1.0, 1984.10.01, four new programs, save, incsave, load and incload have replaced the two programs save and load (issued in the present package, though, with the new names save13 and load13).

The appearance of fixed media disks and the increase in disk storage capacity together with the appearance of high throughput magnetic tape stations make desirable backup utilities which operate fast and with possible option to operate on recently updated parts of the filesystem.

In RC8000, special input/output operations have been developed to reduce the transport overhead and to even render superfluous trans memory data transport.

Built upon the principles of the wellknown save and load programs new ones have been made, offering

- the same parameter structure
- the same multi volume, dual copy option
- the same low level demands on other users/total system
- the same freedom in choise of backup device

but with

- full magnetic tape speed performance
- possibility for backup on incremental basis
- drastically improved file relocation and reload

at the cost of a changed magnetic tape format. The changed format implies that files saved with the previous save program cannot be loaded by the new load program. For that purpose the old load program will be issued, though, having the name load13.

The new program, save, is backwards compatible with the save program in version 1, release 13.0 with regard to

- parameter syntax

i.e. the function of the program and the parameter set is a superset compared to previous one.

This means that any jobfile containing call of the program save will work properly with the new release of save installed.

The two programs save and incsave equal their predecessor, save, in the following points:

- outfile parameter works as for other utility programs
- infile parameter allowed anywhere in the parameter list in a nested way
- no label check in the magnetic tape file before it is overwritten
- up to 32 tape volumes in two copies in one job
- the tapes in the two copies need not have the same length
- backing storage areas are protected during the save
- entries are saved in the same order they are specified
- the search in the catalog for entries specified by name is very fast
- change of scope even for entries saved with their base values
- extensive error and exception reporting

The main differences from earlier releases are:

- the tape format has changed completely
- the dumplabel record contains an identification of the back up
- a save catalog is created in a backing storage area containing a full description of the backup made, including an entry for each file to be saved
- as the backup proceeds, the entries in the save catalog become updated with tape identification and position
- each volume tape in the sequence contains as the first data area after the dump label record a copy of the save catalog, i.e. the catalog is fully updated for all entries backed up so far.
- the program incsave leaves the save catalog as a permanent file ready to be used for later relocation and reload of files as well as for later documentation of the backup performed
- the possibility inherent in the RC8000/IDA disk controller for transfer of backing storage areas from disk to tape connected to the same controller without engaging the RC8000 bus and memory is fully exploited in the two programs
- transfer of backing storage areas from disk to tape not connected to the same RC8000/IDA disk controller bound to engage the data bus and memory takes place by new record procedures performing multi buffered "output of previously input" without in-memory buffer data movement, thus decreasing the cpu load
- transition of mode of operation between the two i/o concepts is done automatically dependent of the actual tape mounting and the actual disk involved
- the program incsave offers a concept of incremental backup based on levels and shortclock/latest changed
- the new program will need at least 6 area processes and at least as many message buffers as area processes with a minimum of 6 and a maximum of 11.
- output of filemarks, which becomes output of two filemarks and a reposition in between, will only happen when a volume tape runs full or when the backup is completed - in the previous release of save it happended after output of the last block of each backing storage area.

On the following pages the new manual for save, incsave, load and incload appear.

Chapters 1 to 20 describe save and incsave; Chapters 21 to 40 describe load and incload.

Chapters 1 and 2 are an introduction with examples.

Chapter 3 and 4 describe the conventions of call and the function of the parameters.

Chapter 5 sums up the questions and answers concerning compatibility regarding the programs save and load in earlier version as well as regarding the new programs inbetween.

Chapter 6 and 7 are descriptions of implementation details and exact formats, and are not prerequisites for normal use of the programs.

Chapter 8 and 9 describe the requirements of a job process in order to be able to run the programs and the error messages comming from the programs.

Chapter 10 gives further examples.

Chapter 11 to 20 are parallel descriptions of the program incsave.

Chapters 21 and 22 are an introduction to the program load with examples.

Chapters 23 and 24 describe the conventions of call and the function of the parameters.

Chapter 25 sums up the questions and answers concerning compatiblity regarding the programs save and load in earlier version as well as regarding the new programs inbetween.

Chapter 26 and 27 are descriptions of implemenation details and exact formats, and are not prerequisites for normal use of the programs.

Chapter 28 and 29 describe the requirements of a job process in order to be able to run the programs and the error messages comming from the programs.

Chapter 30 gives further examples.

Chapter 31 to 40 are parallel descriptions of the program incload.

1. Introduction, Save

The program save transfers catalog entries and backing storage areas to magnetic tape for:

- large or small scale backup of files
- shipment of files to other installations
- shipment of files to other name bases in the directory hierarchy.

The catalog entries and backing storage areas are transferred back to disk by the program load (cf. (3)).

2. Examples

2.1 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 mt62 mtdp0001 0 2
```

the same is obtained by the call:

```
save t.0
```

Using file descriptor name this way instead of magnetic tape name gives a freedom to alter the file descriptors in the catalog and leave the backup jobs unchanged.

In case the file name 'magtapename' contains the text:

```
mtdp0001.2
```

the same job may look:

```
save in.magtapename
```

Using parameter files this way gives a freedom to leave the backup jobs unchanged and just change the contents of the parameter file, e.g. the file 'magtapename' may be perodically changed to contain the names of the actual magnetic tape pool used for multivolume backup.

2.2 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
```

2.3 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
```

2.4 Example 4

In an RC8000, suppose the two logical disks named 'disc5' and 'disc6' are placed on physical ida disks and device number 55 is an ida tape station, all controlled by the same ida mainprocess.

You want to transfer to the tapes 'mtdpooo1', mtdp0002', ... all permanent files (permkey=3) belonging to the two disks, no matter their entry bases, in such a way that they may be reloaded from their respective user processes.

Several files among the ones to be saved are very voluminous data files, so you want to exploit the possibility of high density streaming backup.

In a process with system bases, the command

```
save mt62 mountspec.55 ,
     mt841201.1. mt841202. mt841203,
     segm.3 ,
         disc.disc5.disc6 ,
         scope.perm
```

will do the job.

The parameter segm.3 is not quite necessary since the default value for the blocklength is 3 segments.

The job will run smoothly (no paging) in a job process of 50000 halfwords.

The chances of keeping the tape streaming at high speed, high density with a blocklenth of 3 segments increase by a low overall system load on the actual ida disk controller.

In a BOSS job, the 'device <device number of IDA main process>' command in the job options (allowed by proper BOSS options will be necessary

In a SOS job, the 'mount <name of IDA main process>' command in the job prior to the save will be necessary.

2.5 Example 5

In RC9000-10, or in RC8000 if the disks 'disc2' and 'disc3' are not controlled by the same ida mainprocess which controls the tapestation in example 4 (maybe not ida discs at all), you will have to save the files with a longer blocklenth to be able to keep the tape streaming in high speed, high density, e.g. a blocklength of 21 segments:

```
save mt62 mountspec.55,
mt841201.1.mt841202.mt841203,
disc.disc2.disc3,
scope.perm
```

The process still having std base = system base, will have to have a memory size of 90000 halfwords to progress smoothly.

Note, that the tape may be moved freely to another station of the same type (high and low density meaning the same) during the job.

The chances of keeping the tape streaming in high speed, high density (cf. 6.2) increase by blocklengths above 21 segments.

In connection with ida disks on RC8000 multiples of 21 segments are favorable.

The chances increase, too, by decrease in the overall system load on the system bus and the disks.

In RC8000, note that tapes with long blocks (> = 4 segments at some installations, > = 9 segments at others) must not be reloaded using tape stations connected via RC8301 Device Controller and NCP.

Note further, that for tapes on RC9000-10, or for tapes on RC8000 mounted at stations connected via IDA801 Disc/Tape Controller, the blocklength must not exceed 84 segments.

3. Call

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

[<special param>] [<save specifier>]
```

3.1 Outfile

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

3.2 Mount Parameter

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

<device no>   ::=  <integer>

<modekind>    ::=  (mt62)
                   (mt32)
                   (mt16)
                   (mt08)
```

3.3 Tape Parameter

```
<tape param>      ::= <tape param>.<file no>  ,

                                          0-31
                            (.<next volume>)     [.label.<fpname>]

<tape name>       ::=
<next volume>     ::=  <name>/<filedescriptor>
<name>            ::=  name of magnetic tape
<file descriptor> ::=  name of magnetic tape file descriptor
<file no>         ::=  <integer>/last
<fpname>          ::=  name obeying fp syntax
```

3.4 Special Parameter

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

3.5 Save Specifier

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

3.5.1 Modifier

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

<from disc>  ::=  all/maincatdisc/<disc name>
<to disc>    ::=   no/maincatdisc/<disc name>/0/1/2/3
<new scope>  ::=   no/temp/login/user/project
```

3.5.2 Disc Specifier

```
                                                  1-*
<disc specifier> ::=  disc (.<from disc>)
```

3.5.3 Entry Specifier

```
                                                                  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/al
```

3.6 Infile Parameter

Everywhere the delimiter <s> is allowed in the parameter list according to syntax for FP commands, (cf. [2]), the parameter pair <s> in.<file> is allowed and will be syntactically equivalent to <s>.

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

4. Function

The program will save the catalog entries and possible backing storage areas specified by <disc 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 disc 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.

At tape shift, a message is displayed on current output, specifying the name of the tape just abandoned and the file and block count of the last block before the ending tape mark.

The values of current entry- and segment count are displayed, too, followed by the name of the continuing tape.

When the parameter list is emptied, the magnetic tape file is closed with a couple of tape marks and a message is displayed on current output, specifying the name of the tape, the current position and the total amount of entries and segments saved.

Now the tape is 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.

4.1 Function, Outfile Parameter

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

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

Whether the program terminates through its final end or by a runtime alarm, the current output zone is unstacked again.

4.2 Function, Mount Parameter

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

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

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: mounspec.0 meaning no parent message.

<modekind>
The modekind specified will be used for all volumes in the set specified in <tape param> unless at least one of the tapes in the set is specified by file descriptor, cf. below.

Default: mt62

release. (yes/no)
If release.yes the tape in the set actually used at program termination will be released, i.e. the reservation is cancelled and a release message is sent to the operating system.

Default: release.yes

4.3 Function, Tape Parameter

```
<tape param> ::= <tape name>.<file no>  ,
                                      0-31
                 (.<next volume>)    [.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 in the catalog.

The modekind of the last file descriptor used in the set of volume tapes will be used for the entire set of tapes.

<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 empty file 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 or continuation dumplabel record, and it will appear on current output.

4.4 Function, Special Parameter

```
                                                    1-*
<special param> ::=  {segm.<integer>       }
                     {list. {yes/names/no}}
```

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

segm. <integer>
Backing storage areas will be saved in magnetic tape blocks of <integer> segments
If <integer> < = 1, segm will become 2.

In RC8000, for tape stations connected via RC8301 Device Controllers, a blocklength greater than 4 segments, at some installations extended to, e.g., 9 segments, will lead to program termination (device status unintelligible). In RC9000-10 or in RC8000, for tape stations connected via IDA801 Disc/Tape Controllers, the same will happen for blocklengths beyound 84 segments (64 K characters to be exact).

Default: the value found in the file count of the programs own catalog entry tail.
At installation, the value is 3.
The value may be changed, simply by

```
save - changeentry save save save <value> save save save.
```

The legal value interval for default value is 2< = value < = 84.
If the value is outside the legal value interval, the value 3 is used.

list. (yes/names/no)
If list.yes, the entries saved are listed in one of the forms:

```
<name> <modekind> <scope>.<docname>,
     d.<shortclock> d.<latest changed>
<name> <modekind> <key>.<docname> <lower base><upperbase>,
     d.<shortclock> d.<latest changed>
```

The shortclock, i.e. word 6 of the entry tail, is shown only for non procedure entries with a contents key < > 0.

Generally, latest changed is the shortclock for the latest change to the entry or its associated backing storage area recorded by the monitor. The exact meaning used in the program is explained in 7.1.

If list.names, only the entry name is listed.
If list.no, the entries are not listed.

Default: list.yes

4.5 Function, Save Specifier

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

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

A modifier will modify the sate 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 main catalog scan, picking out entries specified belonging to disks currently specified and record them in the save catalog with current modifiers.

Entries picked out which belongs to a disk specified in current disk specifiers will be saved in a temporary save catalog together with its specification (disk, scope) and its modification (changedisk, newscope).

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

After a block containing a version dumplabel record which is listed on current output too, the save catalog and all the files in it are transferred to the tape or tapes specified, in one or two copies, starting in the file

number specified and extending over as many volume tapes among the specified ones as are needed.

Concluding the backup, a tape mark is output, i.e two tape marks are output, giving an empty file.

4.5.1 Modifier

```
                                                              1-*
<modifier> ::=  {changedisc  {.<from disc>.<to disc>}    }
                {newscope.<newscope>                     }
```

A modifier is valid until changed by another modifier.

4.5.1.1 Changedisc

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

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

`<from disc> ::=`	all/maincatdisc/<disc name>
`all`	means all discs
`maincatdisc`	means the disk with the main catalog
`<disc name>`	means the disk with that name
`<to disc> ::=`	no/maincatdisc/<disc name>
`no`	means changedisc. <from disc>.<from disc>
`0/1/2/3`	means that the disk is selected by the monitor at time of reload (the disk with the most resources of the specified permanent key.
`others`	as for <from disc>

Default: changedisc.all.no

4.5.1.2 Newscope

`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

4.5.2 Disc Specifier

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

A disk specifier specifies one or more disks from where the entry specifier will specify entries.

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

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

Default: disc.all

4.5.3 Entry Specifier

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

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>

Default: any name, any docname, scope.temp

4.5.3.1 Name

<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.

Changes the default for scope to 'the best name'.

Default: any name.

4.5.3.2 Scope

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. The terms temp, login, user and project have their 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 listed on current output with permanent key and entry bases instead of the scope name.

The entries are the ones "inside" the standard base of the job process executing the save, i.e. the specification is well suited for backup on system bases or backup on bases overlaying some project's bases.

Each entry is reloadable only in a user process with bases corresponding the entry bases.

The entire backup is, however, reloadable in a process having the same bases as the one doing the backup.

Single files with unique names and/or document names are reloadable in such a process, too.

Entries specified by **`any other scope`** will be listed on current output using a scope name.

Such entries are reloadable in any process (scope.system only in a process with maxbase = system base, though).

Concerning the target scope, cf. the modifier 'newscope'.

Backing storage areas with entry bases outside project bases need a special attention:

They will not be protected against other users write access during the backup.

If other users perform writing in such an area during the backup, the save program will be held up while the writing takes place, and then it will continue.

If the size of the area changes during the backup, the program will continue and give a proper message concerning the incident.

Note that if a filesystem is saved in such a way that more entries have the same name and scope/bases, e.g. "newscope.user scope.own", they cannot be separated at reload.

```
Default:  name specified: the best name
          no name specified: temp
```

The best name means the name with the best scope among the scopes temp, login, user and project or 6ny scope visible changed into one of above by the modifier 'newscope'.

4.5.3.3 Docname

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>.

Default: any document name.

4.6 Function, Infile Parameter

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

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

The infile parameter may be used in a "nested" way:

When <s> 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 (2), except the character 'nl' is equivalent to the character 'sp', and except names of the types: apostrophized name, generalized name and general text.

When the seperator '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.

4.7 Alternative and Dummy Parameter Names

For compatibility reasons, some alternative parameter names are allowed.

The mount parameters mthh, mtlh, mthl and mtll, are allowed, and are equivalent

to mt62 (or mt32): mthh, mtlh
to mt16 (or mt08): mthl, mtll.

The mount parameters mto and nrz are allowed and equivalent to mtlh and mtll, and mte and nrze are still allowed but do not have new names (high density, even parity, low density, even parity).

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

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

The changedisc parameter <from disc> = any is allowed and equivalent to <from disc> = all.

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

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

The disk specifier parameters <from disc> = main and any are allowed and are equivalent to <from disc> = all.

The special parameter reserve.<yes or no> is allowed but is without any function because of the changed strategy concerning backing storage areas (cf. 6.3).

5. Compatibility

The program is backwards compatible with the program save in SW8010/1, release 13.0 and earlier releases as far as

1) parameter names and parameter syntax, and
2) parameter function

concern, which means that any jobfile set up for use with the earlier release will work the same way with the present release.

One should note, however, that tape stations do not necessarily agree on the interpretation of "high density" and "low density" (cf. 6.2).

The main extensions compared with Version 1 are

1) the use of a save catalog to speed up relocation and reload of files,
2) the change to a tape format more suited to "stream" data to and from the tape, and
3) the increase in ressource consumption as a consequence of changed input/output strategy.

Because of these extensions, the tapes produced are not compatible with the tapes produced by earlier releases of save.

It should be noted, then, that

- tapes produced by version 2, release 1.0 or 2.0 or release 3.0 and newer of save must be loaded by version 2 of load of the same release or newer.
- tapes produced by earlier releases of save must be loaded by earlier releases of load (release 13.0 is included in version 2 of the package by the name load13).
- tapes to be shipped to installations with earlier release of load only must be produced by earlier release of save (release 13 is included in version 2 of the package by the name save13).

The programs save, incsave, load and incload are parameter compatible in the sense that any one of them may read any others parameter list. Unused parameters known to be used by another of the programs will simply be ignored.

6. Implementation Details

In the present chapter the implementation details concerning

- the use of the save catalog
- handling of magnetic tape
- handling of backing storage areas
- the input/output strategy

are given.

In the next chapter the exact formats are given. Knowledge of these details is not necessary in order to use the programs save, incsave, load and incload but it helps understand the way they work and the options and possibilities offered by the programs.

6.1 Use of Save Catalog

The use of the save catalog is the core around which the programs save and incsave are built, so the description is somewhat elaborate.

The programs save and incsave scan their parameter list once, including parameters read from a, maybe in multilevel stacked, input zone.

After the special parameters, i.e. before the save specifiers, the program

- save creates a temporary save catalog file with a wrk-name on the disk with the most temporary ressources available, changes its shortclock to describe the dumptime and connects it for output

- incsave gets the shortblock from tail (6) of the main catalog entry describing the incsave on the next lower level found in the catalog and uses it for basetime, except for incsave on level 0 (zero) which uses the time 0 (zero) for basetime

- incsave creates a permanent (user) save catalog file with the name 'level' concatenate levelnumer on the disk with the most permanent resources available, unless already present, in which case the existing one is used, changes its shortclock to describe the dumptime and connects it for output.

Now a save catalog head describing the backup (cf. 7.1) is written in the file and at this point zones to handle disk to tape output are laid out, cf. 6.2.

The parameter scan continues with the save specifier. Each time an entry specifier is ready, main catalog entries are picked out of the main catalog according to the entry specifier and checked with the current disk specifier and, after evaluation of shortclock latest changed (cf. 7.1), with the basetime (incsave).

Together with current modifiers the entry is prepared as a record with zeroed tape position fields in the save catalog, cf. 3.1, and output to the file when the block (1 segment) is full.

When the last entry specifier has been processed, a dummy record (all zeroes) is prepared and output and the save catalog area is cut down to actually used size.

When the tape (or tapes) has (or have) been prepared (cf. 6.2), the save catalog is transferred to tape as the first data area after the version dump label block.

Now the save catalog is connected for input and the entry records are scanned from the start.

As many entry records as recorded as the maximal number in the dump label block (7.2.1) are prepared for a partial catalog and the save catalog records are updated with tape locations fields (7.2.2) designating the current position of the tape (s) and written back to the save catalog whenever the block (1 segment) is full.

For every area entry in the set prepared, an area process is prepared (cf. 6.3).

If successfully, the entry is listed on current output, if not the entry is skipped with an entry warning on current output, and the 'first slice' field of the record in the save catalog as well as the record in the partial catalog is zeroed.

When all records are prepared, the last one from the save catalog is written back to the file before the partial catalog is transferred to tape(s) (cf. 7.2.3)

Should the tape(s) expire during transfer of the partial catalog and a change to next volume take place, the save catalog then is fully updated until the current partial catalog and ready to be transferred to the next volume tape (cf. 7.2.2).
In other words: the save catalog on the last used volume tape in a succession of tapes will contain the most updated save catalog except for the permanent one left by incsave, which is fully updated.

After the transfer of the partial catalog, all backing storage areas prepared successfully are transferred to tape one by one (cf. 7.2.4).

Now the area processes are removed and the save catalog scan is continued, preparing the next partial catalog a.s.o. until all records in the save catalog have been processed.

Finally the use of the magnetic tapes is finished, writing a number of tape marks, i.e. an empty file comes next. Proper accounts of the entries, segments and slices saved are displayed on current output.

Before terminating (even after runtime alarm) the program

- save removes the save catalog file and the partial catalog file
- incsave just removes the partial catalog file

and unstacks a possibly stacked output zone and reconnects it to the file as before the program was called.

6.2 Handling of Magnetic Tape

The handling of magnetic tapes is governed by

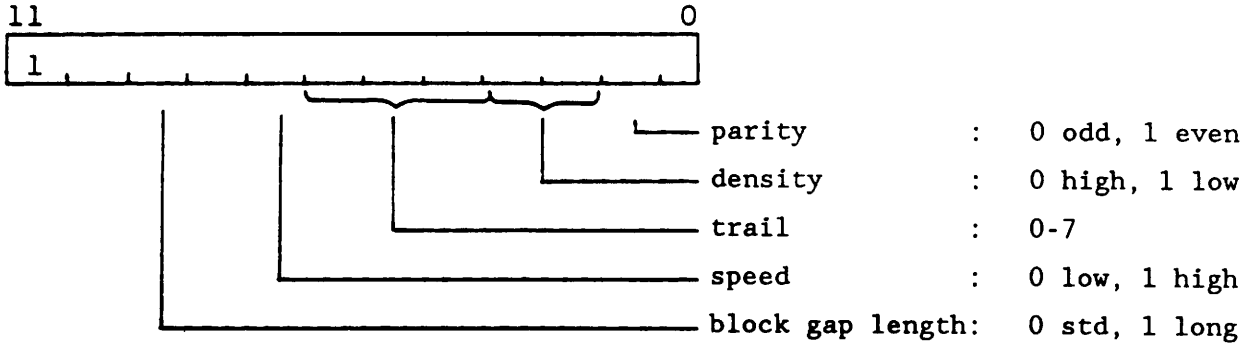
- the mount parameters
- the tape parameters

6.2.1 Mount Parameters

The magnetic tapes are prepared according to the mount parameters

- modekind abbreviation
- mountspec parameter

The modekind specified is converted into the mode word:



The modeword is inserted into the proper zones and is used in all communication with the external processes representing the tapes.

The modekind abbreviations give the following values of the fields in the modeword:
(trail = 0, block gap length = 0):

modekind abbrev.	speed	density	value*	parity
mt62	-	high	.l	odd
mt32	-	high	ll	odd
mthh	high	high	.l	odd
mtlh mto	low	high	.l	odd
mte		high	.l	even
mt16	-	low	..	odd
mt08	-	low	l.	odd
mthl	high	low	..	odd
mtll nrz	low	low	..	odd
nrze		low	..	even

The fields of the modeword select between the following options for the different tape stations:

	RC9255/RC8344	RC9250	RC8343	RC3715
parity	- -	- -	- -	odd even
density/ recording (bpi)	6250 gcr 1600 pe	6250 gcr 3200 ddp 1600 pe	 3200 pe 1600 pe	 1600 pe 800 nrz
speed, high/ (ips) low	75 25	100/50	100/50 25/12.5	45 45

If the parameter 'mountspec' is specified, a mount special message is sent to the parent, specifying the device number where to mount the tape and the name of the tape. The parent (operating system) will then tell the operator where and what to mount.

If the tape is not mounted in advance, two print messages:

```
enable <name of tape>
high   <name of tape> or
low    <name of tape>
```

are sent to the parent to be displayed in order to help the operator select the proper density, if not selectable by program.

6.2.2 Tape Parameters

In the tape parameters up to 32 tape volumes may be specified in two copies, each copy with its own mount parameters, its own starting file number and its own label parameter.

The two copies are written in parallel, block by block, but a change of tape to the next volume may well happen independently in the two copies. Whenever a tape runs full (eot sensed), the outstanding blocks are written to the tape, a tapemark is output, and if another volume is specified, it is prepared as described in 6.2.1. If no more volumes are specified, the program terminates with a runtime alarm.

The file number specification 'last' needs an explanation:

The tapes in the copy are traversed, volume by volume, positioned after filemark by filemark to find the first empty file (cf.7.2.1).

6.3 Handling of Backing Storage Areas

Whenever an area entry from the save catalog is ready to be inserted into the partial catalog, steps are taken to protect the backing storage area during the save.

The catalog base of the job process is changed to equal the entry base of the area. If outside max base, to equal the max base, though.

An area process with the name of the entry is created. If the creation fails, the entry is marked (first word of the entry, the 'first slice' field, is zeroed in the save catalog record, cf. 7.1), and the entry is skipped with a warning on current output.

Next, the area process is write protected (if the current monitor does not offer write protection, the area process is reserved instead) unless its base is outside the max base.

If the protection fails because the area process is already reserved by another process, the entry is marked as above and 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 marked as above and skipped with a warning on current output, telling that the entry is inaccessible.

If the area is the current output document, the entry is marked as above and skipped with a warning on current output, telling that the entry is current output file.

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 according to modifiers and inserted into the partial catalog and listed on current output.

When the partial catalog is ready and has been transferred to tape (s), all successfully prepared backing storage areas belonging to the partial catalog are transferred to tape, block by block.

Foreign write accesses to the area or change of entry size during save may happen to areas not write protected (reserved), i.e. areas with an entry base outside max base of the job process.

To avoid changes in system - or pseudo system files during save in a non system process, the user will have to make special arrangements with the owner of the files or the manager of the installation.

The area processes are removed one by one after the transfer of all the areas in the partial catalog.

Concerning read and write accesses to any area during the save, the following may be stated:

Foreign read access to an area during save will not influence the save.

Foreign write access to an area during the save will hold up the save during the write and maybe damage the file saved but is possible only if the area entry base is outside max base of the job process.

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 the entry base is outside max base of the job process.

Foreign write access during the save will be rejected, i.e. the program trying to write will be held up (algol/fortran) or terminated (slang), unless the entry base is outside max base of job process.

Using one of the entry specifiers scope.perm or scope.all, which are intended for system backup, entries with a base equal to or inside the standard base of the job 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 protection.

6.4 The Input_Output Strategy

The transfer of a backing storage area from disk to tape may take place in one of two ways:

- the transfer takes place in an RC8000, and the area and the tape are controlled by the same ida main process and the transfer takes place locally in the ida801 controller without engaging the RC8000 bus further.

- the takes place in an RC9000-10, or in an RC8000, but the area and the tape are not controlled by the same ida main process or other requirements are not fulfilled so the transfer takes place via the RC8000 bus from area into memory and out to tape.

Transfer of an area starting in the first way may later change to the other as a result of changed conditions.

6.4.1 Local Copy Transfer

If three conditions are fulfilled, the program (save and incsave) will start all transfers as local copy operations:

- the transfer takes place in an RC8000
- only one copy of volume tapes specified
- the blocklength specified is an integer divisor in 21.

A local copy operation is a message specifying a transfer of segments from a backing storage area to a magnetic tape file, cf. [1].

If all three conditions above are fulfilled, a target process to receive the operations is chosen:

- if the tape process exists and has an ida mainprocess as a main process, the ida main process is chosen

- if the tape process does not exist or its main process is not an ida main process, any ida main process is chosen, if any exists

- if no ida main process exist, an illegal name is chosen.

The copy operations are sent and waited for and checked in a multibuffered way exactly as normal output operations are.

The number of buffers used in the communication is chosen this way:

Release 3.0
Single buffered operation.

Before release 3.0
The fp area process is disclaimed.

Of the number of area processes claimed by the job process, four are set aside for use with the program area itself, a possible output file, a possible parameter infile and the save catalog.

Of the remaining area process claim, 1 is used for partial catalog and the rest for possible area processes in the partial catalog.

The number of buffers to be used in the communication, then, is chosen to equal this remaining number of area processes, (at most 9, though) and the maximal number of entries in each partial catalog becomes one less.

If the process does not claim enough area processes to have at least one for entries in partial catalog, the program terminates with an alarm, stating the minimal number of area processes needed (6).

The answers, cf. [1], are checked and they all lead to a call of the block procedure where

- a normal answer, no device status error will just record the position and the number of segments transferred returned in the answer and remove the area process

- all other answers will cause all pending messages to be waited for without check, and

- normal answer, device status error will record the position and the number of segments transferred returned in the answer, repeat the unfinished part of the transfer as described in 6.4.2 and resend all the copy operations which were pending

- dummy answer will record the position as unchanged and

a) in case of result = 2 try to reserve the tape process and write protect/reserve the area process

b) in case of result = 3 change to another target process, if the main process of the tape process is found to be an ida process and another one than the present target process.

- finished by a transfer of the area as described in 6.4.2 and a resend of all the operations which were pending.

Note that result = 2, rejected in RC8000 may be caused by the user process not being a user of the IDA main process, and the action will not be able to change this.
In a BOSS job it requires a 'device <device no of IDA main process>' job option (with BOSS options allowing it), in SOS it takes a 'mount <name of IDA main process>' command to become a user of the main process.

Note that result = 3, unintelligible, may be caused by the choice of blocklength and/or the slicelength of the disk involved.

To be able to start a local copy operation, the following conditions must be fulfilled:

If slicelength < = 21 segments,
- blocksize is an integer divisor in slicelength

If slicelength > 21 segments,
- 21 is an integer divisor in slicelength and
- blocksize is an integer divisor in 21

The action taken on result = 3 is not able to change any of these conditions and the transfer will change to a trans memory transfer if one of them is not fulfilled.

This communication method allows:

- the backing storage areas in the backup to be any mix of areas belonging to or not belonging to ida disks controlled by the same controller as the one controlling the tape, in case the tape is mounted on an ida tape station.

- the backing storage areas in the backup to be any mix of areas belonging to ida disks and areas belonging to other disks, no matter what kind of tape station is used.

- the tapes in the backup to be mounted on any kind of tape station giving the recording density wanted for the modekind specified no matter which kind of disks are involved in the backup

- the tapes in the backup to be remounted on (moved to) any other tape station giving the same recording density for the modekind specified at any time during the backup no matter which kind of disks are involved in the backup

- any device status, disk or tape, to be handled by the normal i/o check and recovery actions, specially end of tape action with change to next volume tape.

6.4.2 Trans Memory Transfer

If not all three conditions in the first paragraph of 5.4.1 are fulfilled the transfers will be started and carried through as trans memory transfers.

In 5.4.2 is described how transfers started as local copy transfers may change to trans memory transfers.

Trans memory transfers are carried out as multibuffered record input/output without in memory data copying.

In case of blocklength greater than 21 segments, input as well as output is double buffered, else both are triple buffered.

With double buffered input/output, three data buffers, with triple buffered five data buffers, are allocated in memory, and by change of block the descriptions are shifted in such a way, that the block output will be the oldest block input, and the block input will overwrite the latest block output.

Furthermore, the block output may be repeated for more output documents in parallel, in this case maybe for the two copies of the volume tape.

Finally, the block output to one or more output documents may be blinded, here used to handle change of volume tape with output of save catalog in one copy of the volume tapes without affecting the other.

The advantages of the method employed are

- standard inblock and outblock procedures are used with the normal error recovery actions taken on transfers

- no need for in memory input buffer to output buffer data copying, saving appr. 1 msec cpu time per segment of data

- a very high mean data transfer rate, increasing with an increase in block length

The disadvantage of the trans memory transfer compared to local copy transfer is the heavy load on the system bus, increasing with an increase in blocklength and with two copies, endangering the transfers to the need of being repeated at data overrun and thus slowing down the overall data transfer rate.

The data buffers for trans memory transfer are allocated in memory and ready for use, even though transfers are started as local copy transfers, so the memory and buffer resources demanded by the job process are the same from job start until finish.

7. Exact Formats

In the present chapter the exact formats of

- the save catalog
- the magnetic tape blocks

are given.

7.1 Save Catalog Format

The catalog head is one block of an integral number of segments containing the fields:

```
+0                              : catalog base lower
+2                              : catalog base upper
+4                              : standard base lower
+6                              : standard base upper
+8                              : user base lower
+10                             : user base upper
+12                             : max base lower
+14                             : max base upper
+16                             : number of disks in the backing storage system
+18                             : max number of volume tapes (32)
+20                             : number of copies specified
+22                             : number of volume tapes specified in copy no 1
+24                             : number of volume tapes specified in copy no 2
+26                             : max number of segments in one block on the
                                  tape (segm)
+20 + 8                         : name of disk no. 1 (8 hwds)
+20 + 8 * 2                     : name of disk no. 2 (8 hwds)
                                .
                                .
                                .
+20 + 8 * no_of_discs           : name of disk no.
                                   no_of_disks (8 hwds)
                                .
                                .
                                .
+22 + 8 * no_of_discs           : name of tape volume 1 copy no. 1 (8 hwds)
                                .
                                .
                                .
                                : name of tape volume max (32) copy no.1 (8 hwds)
                                : name of tape volume 1 copy no. 2 (8 hwds)
                                .
                                .
                                .
                                : name of tape volume max (32) copy no. 2 (8 hwds)
```

The save catalog entry records have a length of

- 58 halfwords when only one copy of volume tapes is specified.
- 64 halfwords when two copies of volume tapes are specified

A record has the fields:

```
+0                          : first slice <12 + namekey <3 + permkey
+2:                         : lower entry base
+4:                         : upper entry base
+6:                         : name of entry (1)
                            : name of entry (2)
                            : name of entry (3)
                            : name of entry (4)
+14                         : modekind (size)
+16                         : document name (1)
                            : document name (2)
                            : document name (3)
                            : document name (4)
+24                         : tail (6)
                            : tail (7)
                            : tail (8)
                            : tail (9)
+32                         : tail (10)
+34                         : scope
+36                         : actual scope
+38                         : new scope
+40                         : disk number
+42                         : new disk name (1)
+44                         : new disk name (2)
+46                         : new disk name (3)
+48                         : new disk name (4)
+50                         : shortclock latest changed
+52                         : volume count copy no. 1 for partial catalog
+54                         : file count copy no. 1 for partial catalog
+56                         : block count copy no. 1 for partial catalog
+58                         : volume count copy no. 2 for partial catalog
+60                         : file count copy no. 2 for partial catalog
+62                         : block count copy no. 2 for partial catalog
```

The leading 34 halfwords are the catalog entry head and tail from the main catalog.

The field 'first slice' may be zeroed at time of transfer to tape of the partial catalog containing the entry if the backing storage area for some reason is not transferred to tape (area process in accessible, reserved by another etc. cf. 6.3).

The fields 'scope', 'actual scope' and 'new scope' are scope keys representing the scope specification used in the specifications of the entry, the actual scope of the entry and the new scope to be given the entry by current modifier.

The scope key has the following value range:

```
0 any scope visible
1 all
2 perm
3 system
4 own
5 project
6 user
7 login
8 temp
```

The field 'disk number' is the number of the disk to which the entry belongs, numbered 1, ..., no of disks in the system.

The field 'new disk name' contains the name of the new disk to which the entry should belong according to current modifier.

The field 'shortclock latest changed' has the meaning:

permanent area entry	shortclock latest changed in entry (9) in auxcat
permanent bs entry	shortclock from associated main entry except: 1) main entry not found in main cat = > value: = 1 2) main entry not found in aux cat = > entry dropped
permanent other entry	shortclock in entry (13) in auxcat
temporary procedure entry	shortclock at time of catalog scan
temporary other entry	shortclock in entry (13) in main cat

The fields 'volume count', 'file count' and 'block count' for copy no 1 and 2 designate the tape or tapes and positions of the sync block ahead of partial catalog containing the entry.

Initially all zero, assigned values when the partial catalog is transferred to tape.

7.2 Magnetic Tape Format

The volume tapes in each copy of the backup contain:

- a version (volume 1) or a continue (later volumes) dump label block as the first block in the file specified (volume 1) or file one (later volumes).

- a save catalog in a number of blocks, updated until the latest output of a partial catalog.

After the save catalog follows in one or more blocks a number of partial catalogs each containing the same maximal number of entries except the last one which may be shorter. Every partial catalog is preceeded by a sync block of a length recorded in the dump label (7.2.1) containing all zeroes.

Every partial catalog is followed by another sync block (of a length recorded in the dump label (7.2.1), and then by the data areas in one or more blocks belonging to the backing storage area entries which may be in the partial catalog. If no area entries exist in a particular partial catalog, it is followed by the next partial catalog (preceeded by a sync block). The last block of every data area is followed by a sync block (as the one after the partial catalog). In each sync block of this kind is recorded the catalog entry of the data area coming next (if any). If the end of tape mark is passed, the last block on the tape, which may be a block of any of the types mentioned above, is followed by a tape mark. After the last block, sync block, partial catalog or data area, a number of tape marks are written, securing that the next file is an empty file.

7.2.1 Dump Label Blocks

A dump label block is 100 halfwords long and is either of the types

- version dump label
- continue dump label

The blocks contain a text record and a non text record each one identifying the backup.

The text record is 58 halfwords long and contains the fields

- program name (save or incsave)
- current tape name and file number
- the text 'vers.' or 'cont.' followed by the dumptime
- the text 'segm.' followed by the maximal block length in segments
- the text 'label.' followed by the label text specified in the call (save) or
- the text 'level.' followed by the dump level number and the dumptime for the next lower level incremental dump found at user base in the main catalog (incsave)
- the characters <0><0><0><0><0>

The text record appear on current output whenever written on tape.

The non text record contains the fields:

```
+58 +0          : maximal block length in segments
+58 +2          : maximal no of entries in partial catalogs
+58 +4          : no of entries in the save catalog
+58 +6          : name of save catalog (1)
                : name of save catalog (2)
                : name of save catalog (3)
                : name of save catalog (4)
+58 +14         : lower entry base for save catalog
+58 +16         : upper entry base for save catalog
+58 +18         : size of save catalog in segments
+58 +20         : shortclock for dumptime (=tail (6) in save catalog
                  entry in main catalog)
+58 +22         : version
+58 +24         : release <12 + subrelease
+58 +26         : length of sync block preceeding partial catalogs (halfwords)
+58 +28         : lenght of sync block following areas (halfwords)
                  (only release 3.0 and newer)
```

7.2.2 Save Catalog Blocks

The save catalog on tape is a number of blocks of the maximal length recorded in the dump label (7.2.1) each containing as many save catalog entry records as permitted by the length of the blocks and the length of the entries (7.1).

The fields of the entry records are described in 7.1.

7.2.3 Sync Blocks

Synchronization (sync) blocks determine the positions of partial catalogs and have a length recorded in the dumplabel block (7.7.1).

The blocks contain all zeroes.

Other synchronization blocks mark the end of data areas, (including partial catalogs). They contain either catalog entry of the next data area or all zeroes.

7.2.4 Partial Catalog Blocks

A partial catalog is a number of blocks containing at most as many entry records as recorded in the dump label (7.2.1).

The entry records have a length of 34 halfwords and are identical to the main catalog entry head and tail for the entry.

The first halfword field, first slice, may be zero, which means that although the entry is an area entry, the backing storage area was not transferred to tape (the area was inaccessible, could not be write protected etc., cf 6.3).

7.2.5 Data Area Blocks

The data area belonging to an area entry record in a partial catalog is a number of blocks each of the maximal length recorded in the dumplabel (7.2.1) except the last one, which may be shorter.

The blocks are taken as an integral number of segments from the disk area and transferred to tape without any change.

8. Requirements

The job process will need ressources of the following types to be able to run the programs save and incsave:

- memory
- message buffers
- area processes
- tempory entries and segments

8.1 Memory

As explained in 6.4.2, the main memory demands are allocated in the stack from the start, and depend very much on the blocksize specified.

The following total process sizes have shown in practice to be sufficient to allow fairly complicated executions without pagings in the central loop for the given blocksize:

```
 2 segments  40- 50000  halfwords  20000 halfwords minimum
 3 segments  45- 55000  halfwords  25000 halfwords minimum
 7 segments  55- 65000  halfwords  35000 halfwords minimum
 9 segments  60- 70000  halfwords  40000 halfwords minimum
21 segments  90-100000  halfwords  70000 halfwords minimum
42 segments 100-110000  halfwords  80000 halfwords minimum
63 segments 130-140000  halfwords 110000 halfwords minimum
84 segments 170-180000  halfwords 150000 halfwords minimum
```

Note that beyound 21 segments per block the demand curve changes.

In case less than absolute minimum memory size is available, the program terminates with a stack alarm.

8.2 Area Process and Buffer Claim

As explained in 6.4.1 and 6.4.2, the number of area processes available determines the maximal number of entries in each partial catalog with a certain minimum and to a certain maximum.

Furthermore this number determines the number of message buffers needed.

The following table gives for 1 and 2 copies of volume tapes the minimal number of area processes needed and upwards the number of message buffers needed.

minimum area processes	message buffers needed		
	1 copy	2 copies	
		segm<=21	segm >21
6	6	8	5
7	7	8	5
8	8	8	5
9	9	8	5
10	10	8	5
11	11	8	5
>11	11	8	5

In case less than minimum area processes or message buffers are available, the program terminates with an alarm stating the need.

8.3 Temporary Entries and Segments

The job process will need one temporary entry and some temporary segments for each of the following purposes:

- infile parameter is used
- outfile parameter is used
- outfile is a temporary backing storage area or does not exist
- save catalog (incsave will need it as a permanent resource)
- partial catalog

In case of shortage of these ressources, the program will

- in the first two cases continue after a parameter warning
- in the third case terminate with the device status alarm
- in the last two cases terminate with a proper alarm message on current output

9. Error Messages

Error messages from the program are written in current output zone.

The following kinds of error messages exist:

- parameter alarm
- parameter warning
- entry specifier warning
- area entry warning
- parameter input syntax message
- area process warning
- area process alarm
- catalog error message

9.1 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	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>

9.2 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.
changedisc param syntax	The parameter 'changedisc .<from disc>' was not followed by .<name> The value becomes <to disc> = 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 unknown	The disc 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 <s><name>. The rest of the parameter list is read, each parameter with this warning as a result.

9.3 Entry and Save Specifier Warnings

The entry specifier is listed in the error message, the mode bits are set 'warning.yes, ok.yes' and the program continues with the next entry specifier.

***** save no entries found according to following specifier**

disc: <disc specifier>
entry: <entry specifier>

Explanation: no entries are found in the main catalog according to the entry specifier.

9.4 Save Specifier Alarm

The modebits are set 'warning.yes, ok.no' and the program terminates.

***** save no entries saved according to any specifier**

Explanation: although entries were found according to specifications, none were saved because a partial catalog on disk could not be connected (cf. 9.).

***** save nothing saved**

Explanation: all entry specifiers have missed with the above 'no entries found' warning for each one as result.

9.5 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: <entry name> skipped <cause>**

Explanation: the area process could not be created, not be protected/reserved or the area was inaccessible for the reason stated in <cause>.

The entry but not the area has been saved.

The warning appears only if list.yes or list.name is specified.

Cause:	**Reason:**
area claims exceeded catalog	create area process failed
i/o error, state of document	create area process failed
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
current output file	the area is connected as current output file
process does not exist,	set write protect/reserve failed
process is not user of area process	set write protect/reserve failed
area process inaccessible	area is inaccessible from executing process

9.6 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. (9), with the exception that the character 'NL' is equivalent to the character 'SP' and the exception that parameter names of the types: apostrophized name, generalized name and general text are not allowed.

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 a 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 reinitalized**

9.7 Area Process Warning

The warning appears on current output following the entry concerned, and is caused by the detection that either foreign write access or entry changes have taken place during the transfer of the area to tape(s).

It can only happen to area entries with an entry base outside the project base of the job process.

The mode bits are set 'warning.yes, ok.yes', and the program continues.

<entry>
***** warning <text> <name> <lower entry base> <upper entry base>**

The <text> is either of:

- write accesses to area during save
- area size changed during save

9.8 Catalog Error Messages

The error messages appear on current output before the first entries are saved.

The messages concern the

- changing or creation of a permanent (incsave) save catalog
- the connecting to the save catalog
- the creation and connecting to a temporary partial catalog

The modebits are untouched and the program continues until output is tried. Then the program terminates with a device status alarm.

***** save <monitor proc> <name> <result>**

Explanation: an existing permanent save catalog could not be changed or could not be created/scoped.

If possible a temporary one is created.

If not created, the next error message will be as below.

The name of the monitor procedure and the result is shown together with the name of the save catalog.

***** save connect <name> <result>**

Explanation: either the permanent save catalog (incsave) could not be connected, a temporary one (save) or a temporary partial catalog could not be created and connected.

The result of the connection together with the name of the save/partial catalog is shown.

10. Further Examples

10.1 Example

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
```

10.2 Example

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
```

10.3 Example

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, disc 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.

10.4 Example

In example 1.4, the permanent files on two logical ida disks were saved on a streaming tape unit on the same ida controller, using a blocklength of 3 segments.

The chances of keeping the tape in streaming mode using this blocklength increase if the tape is written with long block gaps, an option available for RC8343 and RC8344 Streaming Tape Units (cf. 6.2).

As in example 1.1, filedescriptiors may be set in the catalog, specifying the modekind wanted:

```
t1= set 2688.18 mt841201; long block gap,
                            ; high speed, high density
t2= set 2688.18 mt841202;
t3= set 2688.18 mt841203;
```

Now example 1.4 becomes:

```
save mountspec. 55,
     t1.1.t2.t3 ,
     scope.perm
```

The halfword, mode, specifying long block gap for the different standard modes is determined this way:

```
long block gap, mthl = 2048 + 512 + 132 = 2692
long block gap, mthh = 2048 + 512 + 128 = 2688
long block gap, mtll = 2048 + 512 + 4 = 2564
long block gap, mtlh = 2048 + 512 + 0 = 2560
```

Tapes with long block gaps may be read by any tape station.

11. Introduction Incsave

The program incsave transfers catalog entries and backing storage entries, i.e. files, to magnetic tape for backup purpose in an incremental procedure based on the concept of levels.

The backup of a file system in one level is based on the backup of the same file system on the next lower level, i.e. all backups on the same level are based on the backup on the same next lower level.

A permanent save catalog identifying and describing the backup on the level given is left in the file system of the job process in order to

- be the base of a backup of the same filesystem on the next upper level
- offer documentation of the backup performed
- ease relocation and reload of files in the backup performed by the program incload

The files in the backup are conveniently relocated and reloaded by the program incload, but the program load may do it.

12. Examples

12.1 Example

All files of scope project, user, login and temp are transferred to the tape mtdp0001 file 1 by the call:

```
incsave mtdp0001.1 level.0 scope.own
```

A level 0 backup of a file system is considered a total backup of the file system, i.e. all files are saved, no matter their age.

A save catalog of name 'level0' of scope user is left in the catalog (cf. 6.1 and 7.1) describing the backup made. Shortclock (word 6 of the entry tail) in the entry 'level0' contains the dumptime, on which incremental backups on level 1 are based.

The parameter level.0 may be omitted, since zero is the default value.

12.2 Example

All files in the filesystem in example 1, which have been updated since the backup in example 1 are transferred to the same tape, next file, by the call:

```
incsave mtdp0001.last level.1 scope.own
```

This level 1 backup of the file system will concern all files in the same file system which have been updated or created since the level 0 backup.

A permanent save catalog of the name 'level1' is left in the catalog, describing the backup.

Now, as long as the level 1 backup is not too voluminous, you may continue with level 1 backups in the next file and the next file a.s.o. or you may use the same file for all level 1 backups.

At all times, then, the latest level 1 backup and the level 0 backup together contains your filesystem as of the time for the level 0 backup and all changes made since then, except file removals.

When the level 1 backup grows too voluminous, you may do a level 0 backup again. e.g. in file 1, or you may proceed on the next level in the next file:

```
incsave mtdp0001.last level.2 scope.own
```

Now a permanent save catalog of the name 'level2' is left in the catalog.

Together, the level 0, the latest level 1 and the latest level 2 backup contain your file system as of the time of the level 0 backup with all changes made since then, except file removals.

To relocate and reload the latest version of a file, you first try the level 2 backup, then the level 1 and the level 0 backup until the file is found, unless you know in which level to look.

12.3 Example

Suppose a pool of tapes for level 0 backups are specified in the file 'level0tapes':

```
mountspec.55
mt62
mtdp0001.1. mtdp0002. mtdp0003...
```

and a pool of level 1 tapes in the file 'level1tapes':

```
mountspec.55
mt62
mtdp0101.last.mtdp0102.mtdp0103...
level.1
```

and a pool of level 2 tapes in the file 'level2tapes':

```
mountspec.55
mt62
mtdp0201.last.mtdp0202.mtdp0203...
level.2
```

and maybe more.

In a job process with standard base = project base of some project, all permanent files belonging to all users in the project (login, user, project) will be saved by the call:

```
incsave in.level0tapes scope.perm
```

Now, with certain intervals level 1 backups are made by the call

```
incsave in.level1tapes scope.perm
```

When the level 1 backup grows too voluminous or the total amount of level 1 backups threaten too overflow the tapes in the pool, a level 2 backup is made:

```
incsave in.level2tapes scope.perm
```

and so forth.

Whenever desired, the total level 0 dump can be made:

```
incsave in.level0tapes scope.perm
```

defining the base for a new succession of incremental backups.

If the job process has its standard base = system base, the file system specified becomes all permanent files in the main catalog.

13. Call

The program is called exactly as save, except for

- the additional special parameter 'level':
- the ignored tape parameter 'label'.

13.1 Tape Parameter

The 'label' parameter is allowed but ignored, so the syntax becomes the same as for save.

13.2 Special Parameter

The parameter group becomes:

```
<special param> ::= { segm.<integer>       }
                    { list. {yes/no/names}}
                    { level.<integer>      }
```

14. Function

The function is the same as for save, except the additional special parameter determines the name of the save catalog to use.

14.1 Tape Parameter

The 'label' parameter is ignored.

14.2 Function, Special Parameter

`level.<integer>`

Defines the backup level. If the integer specified exceeds 9, dumplevel becomes 9.

Default: 0

An entry of scope user defining any next lower level with one of the names level0, level1, level2, ..., level9 is looked up in the catalog.

If found its shortclock is taken as the base time for the backup, if not the time 0 is taken.

A new save catalog of scope user and name <:level:> add 'backup level' is looked up and changed/created with backup time 'now' as shortclock.

15. Compatibility

The programs incsave, save, incload and load are parameter compatible in the sense that any one of them may read any others parameter list and simply ignore unused parameters known to be significant to others.

16. Implementation Details

16.1 Use of the Save Catalog

The save catalog is used in exactly the same way as for save, cf. 6.1, except it is left permanent in the catalog with a known name, fully updated concerning the positions of all files saved.

16.2 Handling of Magnetic Tape

Exactly as for save, cf. 6.2.

16.3 Handling of Backing Storage Areas

Exactly as for save, cf. 6.3.

16.4 The Input/Output Strategy

Exactly as for save, cf. 6.4.

17. Exact Formats

Exactly as for save, cf. 7., especially 7.1 where 'shortclock latest changed' is defined.

18. Requirements

As for save, cf. Chapter 8.

19. Error Messages

As for save, cf. Chapter 9, except the program name is incsave.

20. Further Examples

20.1 Example

In this example is suggested a way of performing incremental backups of a filesystem, which fully exploits the possibilities inherent in the concept of levels.

Suppose pools of tapes are defined in files pool0, pool1, ... and pool91, pool 92, ...

Maybe a poool is just one file on a tape or one tape, maybe a pool is multivolume tape specifications in two copies. Maybe the pools are defined by file descriptors.

Suppose a filesystem is specified in the file 'filesystem'.

Start with a full level 0 backup:

```
incsave in.pool0 in.filesystem
```

Next, periodic level 9 backups should be made on an exponential progression of tape pools (also called Tower of Hanoi progression, 1, 2, 1, 3, 1, 2, 1, 4 ..., pool 1 used every other time, pool 2 every fourth time, pool 3 every eighth time, etc.):

```
incsave in.pool91 level.9 in.filesystem
incsave in.pool92 level.9 in.filesystem etc.
```

These level 9 backups are based on the level 0 full backup.

When the level 9 backup becomes too voluminous (too time consuming, too tapeconsuming), a level 1 backup should be made in the level 1 pool:

```
incsave in.pool1 level.1 in.filesystem
```

Now the exponential series of level 9 backups should progress as uninterrupted.

The level 9 backups now become based on the level 1 backup, which was based on the level 0 full backup.

The progression of backups can be carried as far as desired.

21. Introduction Load

The program load is used to

- relocate and reload catalog entries and backing storage areas from magnetic tape backups created by save or incsave.
- relocate and read files from such tapes as a simple test of the tape format.
- produce documentation of such backups from the save catalog.

22. Examples

22.1 Example 1

All catalog entries and backing storage areas saved in example 1 of Chapter 2 are relocated and restored by the call:

```
load mtdp0001.2
```

in case

```
t=set mt62 mtdp0001 0 2
```

the same is obtained by the call:

```
load t.0
```

In case the file named 'magtapename' contains the text:

```
mtdp0001.2
```

the same job may look

```
load in.magtapename
```

22.2 Example 2

All files saved in example 2 of Chapter 2 may be reloaded by the call

```
load mtdp0001.2 in.savefiles
```

A simple test of the backup made without actual reload of the files will be done by the call:

```
load mtdp0001.2 load.no in.savefiles
```

A record of the files saved obtained from the save catalog on the tape without actually reading the files on the tape may be done:

```
load mtdp0001.2 survey.yes in.savefiles
```

The parameter in.savefiles is not necessary, but is just used here to illustrate that load may read the parameter list of save.

22.3 Example 3

Among the files saved in example 3 of Chapter 2, the ones belonging to the disks known to the job process doing load (the ones included in the backing storage system at the time of load) will be loaded by the call:

```
load mtdp0001.last disc.all
```

The ones in the save which belongs to disks unknown at the time of load will be loaded too, changing to the disk 'disc3', which is known to the job:

```
load mtdp0001.last disc.all.any changedisc.any.disc3
```

The ones belonging to disks unknown will be loaded to disks with the most permanent resources by the call:

```
load mtdp0001.last disc.any changedisc.any.3
```

22.4 Example 4

The system backup in example 4, Chapter 2 contains all permanent files fromt he disks disc5 and disc6, saved over 3 volume tapes.

Suppose a user wants to relocate and reload one file named 'myfile' with scope user.

The call

```
load mt62 mountspec.55,
     mt841203.1       ,
     myfile.scope.user
```

will do the job. If the file is located on one of the tapes prior to mt841203, the program load will know it from the save catalog read from the tape, and it will mount he correct tape, position directly to the file, load it and stop.

If the file is located on the last tape, mt841203, actually mounted, it will search it along the tape, load it when found and stop.

If the user wants to relocate and restore all files of scope below project, the call:

```
load mountspec.55,
     mt841203.1  ,
     scope.user  ,
     scope.login
```

will do the job.

Note that default modekind, mt62, is used. All files among the ones to be loaded, which are located on the tapes prior to the one mounted, will be relocated directly, i.e. the program will mount the first tape and, file by bile, position to the file and load it. The files located on the last tape will be searched along the tape and reloaded, one by one.

The call

```
load mountspec.55,
     mt841203.1
```

will relocate and reload all files which are visible to the job process and not protected (inside max base).

If the job process has std base = system base, the entire backup is reloaded, as it would be by the call:

```
load  mountspec.55,
      mt841203.1 ,
      scope.perm
```

In the last case, had the backup been made by 'scope.all', the load would pick out the permanent files (permkey = 3).

23. Call

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

23.1 Outfile

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

23.2 Mount Parameter

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

<device no>        ::=  <integer>

                        ( mt62 )
                        ( mt32 )
<modekind>         ::= ( mt16 )
                        ( mt08 )
```

23.3 Tape Parameter

```
<tape param>       ::= <tape param>.<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 descriptor
<file no>          ::= (<integer>/last)
<fpname>           ::=  name obeying fp syntax
```

23.4 Special Parameter

```
                                                        1-*
                        { copy.{1/2}                    }
<special param>  ::= { vol.<integer>                    }
                        { list.{<yest/no/names>}        }
                        { load.{yes/no}                 }
                        { survey.{yes/no}               }
                        { connect.{yes/no}              }
```

23.5 Load Specifier

```
                                                    1-*
                        { <modifer>             }
<load specifier> ::= { <disk specifier>  }
                        { <entry specifier> }
```

23.5.1 Modifier

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

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

<to disk>      ::= {no/maincatdisc/<disk name>/0/1/2/3}

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

23.5.2 Disk Specifier

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

23.5.3 Entry Specifier

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

<entry factor>   ::= {<entry name>/scope.<scope>
                        /docname.<docname>}
<entry name>     ::= <name>
<scope>          ::= {temp/login/user/project/own
                        /system/perm/all}
```

23.6 Infile Parameter

Everywhere the delimiter <s> is allowed in the parameter list according to syntax for FP commands (cf.[2]) the parameter pair <s> in.<file> is allowed and will be syntactially equivalent to <s>.

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

24. Function

The program will either

- relocate and reload the catalog entries and backing storage areas specified by <disk specifier> and <entry specifier> after having modified them according to the modifications in <modifier> from the tape(s) specified in <tape param> mounted according to <mount param>, or
- relocate and read as above to test the tape format, or
- relocate as above to produce documentation of the backup catalog for the files specified.

The tapes specified may be the first or the second out of two copies of maybe more volume tapes.

The program interprets the parameter groups, one by one.

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

If a proper version dumplabel record is read as the first block it is displayed on current output, and the program continues by transferring the save catalog from the tape to a backing storage area.

Each <entry specifier> starts a save catalog scan, picking out the entries specified, and the entries satisfying the current disk specifications are transferred to a load catalog together with its save modifiers. Now the load catalog is scanned, and each entry is relocated and loaded after a possible change according to its modifiers.

During the reload, suceeding magnetic tape volumes, as far as needed, are mounted whenever actual volume is used up.

At tape shift, a message is displayed on current output specifying the name of the tape just abandoned and the file and block count at the last block read before the tape shift is initiated. The values of current entry count and segment count are displayed, too, and the name of the next tape to read.

When the load catalog is exhaused, the program finishes with a message on current output specifying the name of the current tape, the position of the tape and the total amount of entries and segments read/loaded.

At last the program releases the tape if so specified in mount parameters and writes on current output an account of the files loaded.

If two sets of tapes were used in the backup, the one specified is used for the reload.

24.1 Function, Outfile Parameter.

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

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

Whether the program terminates through its final end or by a runtime alarm, the current output zone is unstacked again.

24.2 Function, Mount Parameter

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

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

`<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 speciefied in <tape param> is mounted. Default: mountspec.0 meaning no parent message.

`<modekind>`
The modekind specified will be used with first read operation. In case of mode error, the next mode is tried. When all modes have been tried without luck, the program terminates with an alarm. Default: mt62.

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

24.3 Function, Tape Parameter

```
<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.

The first tape is determined by the special parameters copy and vol, by default first copy, first volume. The tape thus selected, determines the actual copy used by load, if two were used in the save.

<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> ::= <integr>/last
The file number of the first tape in the set where the load 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 empty file is searched along the tapes in the set, even if they are specified by <file descriptor>'s, and the last one before that, which contained a version or a continue dump label is selected. If necessary the tape is mounted before it is positioned.

label.<fpname>
The label text is read, but ignored.

24.4 Function, Special Parameter

```
                                                    1-*
                    { copy    . {1/2}           }
                    { vol     . {<integer>}     }
<special param>::= { list    . {yes/no/names}  }
                    { load    . {yes/no}        }
                    { survey  . {yes/no}        }
                    { connect. {yes/no}         }
```

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

copy. {1/2}
If two copies of volume tapes were specified in tape parameters the one specified by copy will be used by load. If only one copy was specified, that one will be used by load.
Default: 1

vol.<integer>
If more volumes of the copy were specified in tape params the one specified by vol will be used by load as the first volume tape to mount.
Default: 1

list. {yes/no/names}
If list.names, only the entry names will be listed on current output.
If list.no, the entries are not listed.
If list.yes, they are listed as for save.
Default: list.yes

`load. {yes/no}`
If `load.no`, the backup is read as if `load.yes`, but the entries and backing storage areas are not inserted in the catalog or transferred to disk.
Default: `load.yes`

`survey. {yes/no}`
If `survey.yes`, only the save catalog transferred to disk is read and all entries specified in the call which are found in the catalog are listed as if they were found on the tape(s).
Default: `survey.no`

`connect. {yes/no}`
If `connect.no`, the files are reloaded under working names to the target disk and renamed, thus

- spending extra ressources during reload
- postponing the destruction of the possibly existing file until the new one is actually reloaded.

If `connect.yes`, the files are reloaded with their real names, connecting to possibly existing areas before reload, thus

- saving the need for extra ressources during reload
- riscing the contents of the possibly existing file in case of termination before reload in completed.

Default: `connect.no`

24.5 Function, Load Specifier

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

The elements of the load specifier are treated one by one, until the parameter list is exhausted.

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 save catalog scan, picking out entries.

Entries picked out which belongs to a disk specified in current disk specifiers will be transferred to a load catalog with its save modifiers. When all entry specifiers have been processed, the entries from the load catalog will be reloaded after having been modified according to their load modifiers.

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

24.5.1 Modifier

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

A modifier is valid until changed by another modifier.

24.5.1.1 Changedisc

```
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/any/maincatdisc/<disk name>}
all                 means all disks known in the system
                    at time of reload
any                 means any disk not known in the
                    system at time of reload
maincatdisc         means the disk with the main catalog
<disk name>         means the disk with the name
<to disk>::=        {no/maincatdisc/<disk name>/0/1/2/3}
no                  means changedisc. <from disk>.<from
                    disk>
0/1/2/3             means that the monitor will select
                    the disk at reload, i.e. the one with the
                    most resources of the specified
                    permanent key
others              as for <from disk> exept 'any'.
```

Default: changedisc.all.no.any.no

24.5.1.2 New Scope

```
newscope. <new scope>
```

All entries will be changed to have the scope <newscope>

```
<newscope>   ::= no/temp/login/user/project
```

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

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

24.5.2 Disk Specifier

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

Specifies entries which at time of save, maybe changed by changedisc modifier, belonged to the disk or disks specified.

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

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

24.5.3 Entry Specifier

`<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>}`
Default: any name, any docname, any scope

24.5.3.1 Name

`<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.
Default: any name

24.5.3.2 Scope

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

The attribute is the scope.
All entries with the scope specified have the attribute.
The terms temp, login, user, project have the usual meaning
`own` means one of above.
`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 at load by

`scope.perm or scope.all`

are the ones which

- were given bases, possibly changed by newscope modification, which are inside or equal to the standard base of the job process loading, and for scope.perm with a permkey of 3.

It makes no difference, that the backup was done by `scope.perm or scope.all`.

Entries specified at load by

`default scope (any scope)`

are the ones which

- if backed up by scope.perm or scope.all were given bases, possibly changed by newscope modification, which are visible to and not protected against the job process loading.
- if backed up by scope key were given scopes, possibly changed by newscope modification, among the ones temp, login, user or project.

Entries specified at load by

`scope key`

are the ones which
- if backed up by scope.perm or scope.all were given bases, possibly changed by newscope modification, which together with permkey give the scopes specified
- if backed up by scope key were given scopes, possibly changed by newscope modification, which are among the scopes specified.

Note that if a filesystem is saved in such a way that more entries have the same name and scope/bases (flat filesystem), they cannot be separated at load. They will all be loaded into the same file and the last one wins.
Default: any scope

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

24.6 Function, Infile Parameter

Every where the delimiter `<s>` 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 `<s> 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 (2)), except the character 'n1' is equivalent to the character 'sp' and except names of the types apostrophized name, generalized 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.

24.7 Alternative and Dummy Parameter Names

For compatibility reasons, some alternative parameters are allowed.

The mount parameters mthh, mtlh, mthl and mtll are allowed, being equivalent to
mt62 (or mt32): mthh, mtlh
mt16 (or mt08): mthl, mtll

The mount parameters 'mto' and 'mte' are allowed and equivalent to 'mtlh' and 'mtll'.

The mount parameters 'mte' and 'nrze' are still allowed, but have no new names (high density, even parity and low density, even parity).

The special parameter 'check.yes/no' is still allowed, but has no function.

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

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

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

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

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

Parameters known to be used by save, incsave or incload but not by load are simply ignored.

25. Compatibility

The program is backwards compatible with the program load in SW8010/1, release 13.0 and earlier releases as far as

1) parameter names and parameter syntax, and
2) parameter function

concern, which means that any jobfile set up for use with the earlier release will work the same way with the present releease.

One should note, however, that tape stations do not necessarily agree on the interpretation of "high density" and "low density" (cf. 6.2).

The main extensions compared with earlier releases are

1) the use of a save catalog to speed up relocation and reload of files,
2) the change to a tape format more suited to "stream" data to and from the tape, and
3) the increase in ressource consumption as a consequence of changed input/output strategy.

Because of these extensions, the tapes read by this release are not compatible with the tapes read by earlier releases of load.

It should be noted, then, that

- tapes produced by version 2 release 1.0, release 1.1 or 2.0 or release 3.0 and newer of save must be loaded by version 2 of load of the same release or newer.
- tapes produced by earlier releases of save must be loaded by earlier releases of load (release 13.0 is included in version 2 of the package by the name load13).
- tapes to be received by installations with Version 1 release of load only, must be produced by Version 1 release of save (release 13.1 is included in version 2 of the package by the name save13).

The programs save, incsave, load and incload are parameter compatible in the sense that any one of them may read any others parameter list. Unused parameters known to be used by another of the programs will simply be ignored.

26. Implementation Details

In the present chapter the implementation details concerning

- the use of the save catalog,
- the handling of magnetic tape,
- the handling of backing storage areas,
- the input/output strategy

are given.

Knowledge of these details is not necessary in order to use the programs, save, incsave, load and incload, but it helps understand the way they work and the options and possibilities offered.

26.1 Use of the Save Catalog

The use of the save catalog is the core around which the programs load and incload are built, so the description is somewhat elaborate.

The programs load and incload scan their parameter list twice, including parameters read from a, maybe in multilevel, stacked input zone.

First scan is to get the mount, tape and special parameters and to

- count the number of entry specifiers
- count the number of disks specified which are unknown to the system.

Second scan is to build entry specifier tables, one of each entry specification, with associated modifier tables.

When the tape specified is mounted, positioned and the proper dumplabel read, the program

- incload searches the main catalog for a save catalog of scope user with the name and shortclock given in the dumplabel record and tries to connect it for input. If no success, it does as load
- load creates a temporary working name backing storage area and transfers the save catalog from the tape mounted to the area and connects it for input.

Now the program reads the save catalog head to

- get the description of the tapes specified in the save job
- position the save catalog to the first entry record (cf. section 7.1).

A temporary working name file is created and connected for output of a load catalog.

The save catalog is scanned, record by record, until the all zero end record is met, and the records, which satisfy the specifications in some entry specifier, maybe after the modifications recorded in the save catalog record, are transferred to the load catalog together with the modifiers belonging to the entry specification.

The save catalog is disconnected, the load catalog cut down and reconnected for input and a tempory working name file is created and connected for output as a partial catalog (cf. subsection 7.2.3).

Now the load catalog is scanned, record by record, until all records have been processed.

For each record which requires a new partial catalog, the proper tape is positioned, maybe after a mount, and the partial catalog is transferred from tape to disk.

An entry record requires a new partial catalog when the values of its position fields change compared to its predecessor, or when its position fields are zero but the predecessor had its counterpart as the last one in the current partial catalog.

Now the partial catalog is searched for the counterpart (same name and bases) of the current load catalog entry.

For each partial catalog entry, which is rejected but marked to carry data, its data area on the tape is read in order to keep the position on the tape updated.

When the counterpart entry in the partial catalog is found, the tape will be in position for its possible data area.

Now the entry from the partial catalog is changed according to the modifiers recorded in the load catalog and the entry is listed on current output, unless the area is an area entry, but

- its data area was never saved (cf. section 6.3)
- a working name area entry of the scope wanted could not be created on the target disk (connect.no) or
- an existing area entry of the name, scope and disk wanted (connect.yes) could not be connected for output.

In this case an entry warning is listed on current output instead.

The data area is transferred from the tape and if a working name area entry was the object file, it is renamed after a possible removal of the existing entry.

When all records of the load catalog have been processed, the use of the magnetic tape is finished by

- sending a release message to the parent, if specified by release.yes
- writing accounts of the position on the tape just left, segments and entries read from the tape during the job and segments and entries actually loaded in the job.

Before terminating (even after runtime alarm), the program load removes

- the save catalog
- the load catalog
- the last partial catalog

files, incload only the two last, and a possibly stacked output zone is unstacked and reconnected to the file it was before the program was called.

26.2 Handling of Magnetic Tapes

The handling of magnetic tapes is governed by

- the mount parameters
- the tape parameters

26.2.1 Mount parameters

The magnetic tapes are prepared according to the mount parameters, modekind, mountspec and release, as for save, cf. subsection 6.2.1, except that all modes will be tried sucessively in case of mode error.

26.2.2 Tape Parameters

In the tape parameters, one volume tape may be specified by name or file descriptor or up to two copies of up to 32 volume tapes may be specified.

If more tapes are specified, the special parameters 'copy' and 'vol', maybe by default (copy one, volume one), determine the first volume tape to mount. The tape, then, is mounted and positioned.

The file number parameter 'last' needs an explanation: the tape is positioned at file number 1, block number 0, and the first block is read.

As long as the block is not empty, the file number is increased by one, the block number is zeroed and the first block is read. The procedure may extend over more volume tapes. First time an empty file is met, the search ends and latest file identified by continue or dumplabel record becomes the tape and position to mount and position.

If the backup extended over more tapes and the rest of the set of volume tapes used, from the one first mounted and on, are specified in

the call, then, the last one actually used becomes the tape to mount first, and the save catalog, which controls the relocation and reload, becomes the most updated version to get from any tape.

From the tape pointed out to be the first one as described above, the save catalog is transferred to a working name temporary disk area (26.1), except for incload, which first tries to connect to an existing save catalog of name, base, size and shortclock as recorded in the dump label (cf. subsection 7.2.1).

The tape or set of tapes recorded in the save catalog head (cf. section 7.1) becomes the tape or set of tapes specified to the program, with the one actually mounted as the current volume tape, and if it occurs in one of two copies of volume tapes, if determines the copy to be used all over in the reload.

If the reloading process extends over more volume tapes, a change of volume tape to the next one in the set takes place at end of document status, nothing input.

The save catalog in the next volume tape mounted is just bypassed, as it is of no better use than the one already in use.

26.3 Handling of Backing Storage Areas

Whenever an area entry from the load catalog is relocated in its partial catalog, steps are taken to allocate a backing storage area:

- if the special parameter connect.yes is specified, an entry of the name and bases is looked up in the catalog. If it exists, no matter what disk, it is tried to change its entry tail into the object one, except for the document name. If sucessfull, the file is connected for output. If not, or
- if the special parameter connect.no is specified, maybe by default, it is tried to create an area entry with a working name, but with bases, permanent key and entry tail as specified on the disk specified (if new disk name = 0,1 2, or 3, the monitor chooses the disk with the most resources of the specified permanent key). If sucessfull, the area is connected for output. If not, a possibly created entry is removed again.

Connection for output involves the creation and reservation of an area process.

If any step fails, a proper message is issued and the entry is listed on current output with a message, that the entry has been skipped.

If nothing failed, the area is transferred from tape to disk and the entry is renamed.

If it cannot be renamed because of name overlap, the entry in the way is renamed and another rename is tried. If still no success, the entry is removed instead with a proper message on current output.

Since the area process is reserved during the reload, the following may be said about foreign read - and write accesses to the area during the reload:

- connect.yes is specified:
  Any continuing foreign read/write access to the area at the time of connection are unaffected by the reload, but will cause the reservation to fail and the entry to be skipped. Any foreign read/write access to the area after the connection will be rejected (i.e. read accesses from algol programs will be repeated until success, all other accesses will terminate their programs) and not affect the reload.

- connect.no is specified, maybe by default:
  The new area has a working name, and no foreign read/write access is likely to happen. Any continuing foreign read/write access to the old area at the time of removal/renaming are not affected by the reload, but will cause the removal to fail and the entry to be skipped.
  Any new foreign read/write accesses after the renaming are ok, since the reload is complete.

26.4 The Input/Output Strategy

The transfer of backing storage areas from tape to disk takes place as described in section 6.4.2, Trans Memory Transfer.

Persistent parity errors in input from tape are ignored and the missing tail of the harmed block is zeroed. For further details, cf. 29.8.

27. Exact Formats

Described in Chapter 7.

28. Requirements

The job process will need ressources of the following types to be able to run the program load or incload:

- memory
- message buffers
- area processes
- temporary entries and segments.

28.1 Memory

As described in section 8.1.

28.2 Area Processes and Message Buffer Claim

The job process will need at least

6 area processes

and in case the blocklength on the tapes is < = 21 segments, it will need at least

6 message buffers

else it will need only

4 message buffers.

In case less than minimum is available, the program terminates with an alarm stating the need.

28.3 Disk Resources

The job process needs one temporary entry and some temporary segments for each of the following purposes:

- infile parameter is used
- outfile parameter is used

- outfile is a temporary backing storage area or does not exist
- save catalog (not incload, if save catalog exists)
- load catalog
- partial catalog

If connect.no is specified, maybe by default, the job process will need one temporary entry in the main catalog and for each object disk in the reload and for each permanent key to be used on that disk, it will need one entry and as many segments as the largest file to be reloaded on that disk for that permanent key. If connect.yes is specified, no extra resources will be needed for the areas to be reloaded.

29. Error Messages

Error messages from the program are written in current output zone.

The following kinds of error messages exist:

- parameter alarm
- resource alarm
- connection alarm
- monitor alarm
- other alarm
- parameter warning
- save specifier warning and message
- area entry warning
- parameter input syntax error message
- mode shift message

29.1 Parameter Alarms

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.

***** load 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

entry specifiers not properly recorded	parameter list not the same in two scans (properly infile changed)

29.2 Resource Alarms

At resource alarm, the program terminates with 'warning.yes, ok.no':

***** load area claim, at least needed: <needed>, claim: <claim>**
***** load buffer claim, needed: <needed>, claim: <claim>**

<needed> is the amount of resources needed, <claim> is the amount claimed.

29.3 Connection Alarms

After connection alarm, the program continues just to terminate with a 'device status' alarm concerning output to the area not connected.

The modebits then become 'warning.yes, ok.no'.

***** load connect <filename> <cause>**

where <filename> is the name of the area concerned and <cause> is one of the following:

- no resources
- malfunction
- not user, not exist
- convention error
- not allowed
- name format error

The area concerned will be either
- the save catalog (maybe a working name)
- the load catalog (working name)
- the partiel catalog (working name)

Explanation:
The area concerned could not be connected for output for the reason stated.

***** load <monitor call> <entry name> <result>**

where <monitor call> is either of

- create entry
- permanent entry (maybe into auxiliary catalog)
- set entry base

- rename entry
- remove entry

<entry name> is the name of the entry concerned and <result> is the result of the monitor call.

Explanation:
During reload of the entry concerned, the object entry is prepared by a combination of the three first monitor calls (in case of connect. yes when the object entry did not exist). The entry name will be a working name.

After reload of the entry, the object entry is renamed, maybe after removal of the existing one (in case of connect.no). One of the monitor calls returned a positive result for the reason stated.

After the warning, the entry is listed, followed by an area entry warning without <cause>.

29.4 Other Alarms

All other alarms from the program concern the reading and contents of dumplabels on tape and the transfer/bypassing of save/partial catalog from tape and the contents of the save catalog head.

After the alarm the program terminates with 'warning.yes, ok.no'

The alarms have the common form:
***** load <text> <name> <value> <text> <value>**

Following an explanation for each possible <text>.

no dumplabel found on volume tape, file, block number:
On the first tape mounted, no proper dumplabel was found.
(blocklength <> 100 halfwords or not text starting with 'save' or 'incsave' or neither 'vers.' nor 'cont.' dumplabel).
The name of the tape and the position is given as <name> and two <value>s.

Incomplete save catalog transferred from tape:
The size of the save catalog recorded in the dumplabel did not agree with the number of segments actually transferred or the transfer suffered from persistent i/o troubles.
The name of the tape and the size of the save catalog recorded as well as the number of segments actually transferred are shown in <name> and <value>s.
The reason is i/o troubles and the alarm should not occur.

no proper dumplabel on volume tape, file, block number:
On a new volume tape mounted, no dumplabel was found, cf. above.

dumplabel incompatible on volume tape, file, block number:
The fields of the dumplabel record found on the next volume tape do not agree with the ones found in the first volume tape mounted.
The new volume tape has been used for backup since the one in the first volume tape mounted.

The new volume tape name and the file number are shown in <name> and <value>s.

max no of volumes in save catalog incompatible with load program:
The maximal number of volumes tapes in one copy recorded in the save catalog head is not the one used by the loading program (at present 32). The name of the tape from which the save catalog came and the number recorded there is shown in <name> and <value>.

incomplete partial catalog transferred from tape:
Not the number of segments in a partial catalog as recorded in the dumplabel were transferred from tape or the transfer suffered from persistent i/o troubles.
The name of the tape and the number of segments, recorded as well as transferred, are shown in <name> and <value>s.
The reason is i/o troubles and the alarm should not occur.

incomplete save catalog bypassed on tape:
During bypass of a save catalog on the next volume tape mounted not as many segments as recorded in the dumplabel were bypassed or the transfer suffered from persistent i/o trouble.
Reason as above.

29.5 Parameter Warning

The parameter warning skips current parameter, displaying it in the error message and continues, setting the modebits:
'warning.yes, ok.yes'

***** load 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 one 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.
load param unknown survey param unknown connect param unknown	The parameter displayed 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.
scope param syntax	The scope parameter was not followed by .<name>. No entries will be loaded 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 loaded according to this entry specifier.
docname param syntax	The 'docname' parameter was not followed by .<name>. No entries will be loaded according to this entry specifier.
name illegal	The name specified in entry secifier was 'c', 'v' or 'primout'. The entry specifier is not recorded.
name double defined	A name was already specified. The new name is ignored and the program continues with the current entry specifier.
load spec param unknown	Syntactically the parameter would

start a load specifier, but the parameter is not <s> <name>. The rest of the parameter list is read, each parameter with this warning as a result.

29.6 Area File Warning

The warning concerns the transfer of the data blocks from the tape to a backing storage area, or just reading the data blocks in case of load.no.

It appears on current output following the area entry concerned.

The modebits are set 'warning.yes, ok.yes' and the program continues.

The warning will have the form

*****load warning <expl>**
<entry> <size> transferred: <segs>

where <expl> will be one of:

- too many segments of area transferred from tape
- not all segments of area transferred from tape

<entry> will be the entry name
<size> will be the size given in the entry, and
<segs> will be the actual number of segments transferred.

In case of actually loading (load.yes), the entry loaded will get the size of the actual number of segments loaded.

The reason for missing data blocks or having too many is that the entry base of the original entry was outside the max base of the process doing the save, and the entry was changed by someone else while the segments of the area were transferred to tape, cf. 9.7.

The size of the entry after the load will be the number of segments actually loaded, while the remaining entry tail will look as before someone else changed it during the save.

29.7 Persistent I/O Warning

The warning occurs when the transfer of an area from tape is bothered with persistent i/o troubles.

The modebits are set 'warning.yes, ok.yes'.

The warning is

***** load parity: persistent parity error in input from tape**

<tape name> file, block no <file> <block> loading to
<file name> last <no> halfwords of segments <no>-<no> would be/are zeroed.

The trouble is parity error in spite of a number of retries reading the same block from tape. The missing part of the block is zeroed and the result depends on conditions:

load.yes, connect.no: the file is a wrk.... file and stays a wrk.... file, i.e. it is not renamed and it is not recurred, it stays at the user's disposal. The text is "are zeroed".

load.yes, connect.yes: the file ends up with the original name, i.e. if it existed in advance, it gets its new contents (with zero values in it). The text is "are zeroed".

load.no: the file is not loaded at all and the text becomes "would be zeroed".

If the file tranferred is the save catalog or a partial catalog, the program terminates with one of the alarms:

incomplete save catalog / partial catalog transferred from tape.

In case of any other file, the program continues.

29.8 Monitor Alarm

After monitor alarm the program terminates with 'warning.yes, ok.no' after having removed the entry concerned.

29.9 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: <entry name> skipped <cause>**

Explanation:
The area process could not be created, not be reserved or the area was inaccessible at save for the reason stated in <cause>.

The entry and the area have not been loaded.

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	create area process failed

does not permit call	
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	reserve failed
not user, cannot be reserved	reserve failed
does not exist	reserve failed

29.10 Load Specifier Warning

The load specifier warning may occur just before program termination.

The load specifier is listed in the error message, the mode bits are set 'warning.yes, ok.yes'.

***** save no entries found according to following specifier**
disk: <disk specifier>
entry: <entry specifier>

Explanation:
No entries are found in the save catalog according to the load specifiers or the entries specified have been skipped, cf. above.

29.11 Load Specifier Message

The load specifier message may occur just before program termination. The modebits are untouched.

The message is:

```
nothing loaded because of <survey.yes/load.no>
```

Explanation:
No entries are loaded because survey.yes or load.no were used.

29.12 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. (2), with the exception that the character 'NL' is equivalent to the character 'SP' and the exception that the name types apostrophized names, generalized names and general text are not allowed.

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.

***** load syntax <parameter>**
*** read from <file>**
*** selected from <file>**
.
.
.
***** load reinitialized**

29.13 Mode Shift Message

The message occurs when the mode specified by mount parameters or by default does not agree with the mode set at the tape station where the tape is mounted, as experienced at the first try to read the dumplabel. The message will not set the mode bits.

The message is

*** mode error on <tapename>, trying <next mode>**

After the message, another try is made with the next mode in the set

```
mt62/mtlh, mte, mt16/mtll, nrze,
mt32     , mt08, mthh     , mthl
```

If still mode error, another mode message occurs.

When all modes in the set have been tried without success, the program terminates with a 'device status' alarm with 'warning.yes, ok.no'.

30. Further Examples

30.1 Example 1

Continuing the example in 22.4, where the volume tapes mt841201, mt841202 and mt841203 had been used for a backup of all permanent files in the system (cf. section 2.4), written in high density, and a user wants to, e.g. reload all of his user files. Suppose the 'high density' at backup time was 1600 bpi and now at load time the user may have access to stations only where 1600 bpi is 'low density'.

The user does not know which tape was the last one actually used, so he submits the job:

```
load mt16 mt841201.last connect.yes scope.user
```

The job will mount the tape mt841201, search along the tapes until mt841203, from where the save catalog is taken.

Since the user files wanted may be scattered all over the tapes, the job may now require the first tape mounted (if still not mounted), position from user file to user file and load them, require the second tape mounted and so forth until the last user file is reloaded, which is where the last tape will be abandoned.

If the tapes involved stay mounted throughout the job, no time is wasted 'going back' on the tapes, since release.yes is specified by default.

The connect.yes parameter implies that any existing file will be directly connected at reload, no matter what disk it belongs to, and that only the resources needed for the not existing files are required.

30.2 Example 2

Consider the example in section 10.3, where all files in the system, temporary or permanent, belonging to the disks 'disc', 'disc1' and 'disc2', are backed up on two copies of tapes, each specified to hold 5 volume tapes.

Suppose all permanent files belonging to a certain project is to be reloaded from the second copy set of tapes, and that the fourth volume is known to be the last one actually used in the backup.

In a process with std base = project base, the job

```
load in.magtapes copy.2 vol.4 scope.perm
```

will do the job.

The first tape to be mounted will be mtdp0009, which is the fourth volume in copy set no.2, and the save catalog, which is taken from this tape, will tell where on the tapes among the three first in the copy the files wanted are saved.

The tapes are mounted in sequence and the files reloaded by direct position and load, until the fourth volume. If it contains any of the wanted files, the files are searched along the tape and reloaded in the order they were saved, until the last one has been reloaded, which is where the tape is abandoned.

As long as the stations involved are set in the density recorded on the tapes, the program will read the tapes despite no specification of modekind in the job.

If the four tapes involved are all mounted from the start, or at least become mounted while the predesessor is being used, no time is waisted during change of volume tape.

31. Introduction Incload

The program incload is used to

- relocate and reload catalog entries and backing storage areas from magnetic tape backups created by incsave.
- relocate and read files from such tapes as a simple test of the tape format.
- produce documentation of such backup from the save catalog resident in the file system.

The program may do the same for tapes created by save, only the save catalog used will be taken from the tape in the same way as by the program load.

32. Examples

32.1 Example 1

In the example in section 12.1, all 'own' files are backed up in a level 0 backup, i.e. total backup, on the tape mtdp0001 leaving a permanent save catalog by the name 'level0'.

Suppose the job process has the same user bases as the backup job had.

The job

```
incload mtdp0001.1 survey.yes
```

will list all files described in the save catalog, without reading anything on the tape but the dumplabel and the save catalog.

The job

```
incload mtdp0001.1 load.no
```

will do the same, but the tape will be read, partial catalog by partial catalog, data area by data area, without actually loading any file.

The job

```
incload mtdp0001.1
```

will actually reload all the files in the backup, while the job

```
incload mtdp0001.1 scope.user
```

will reload the user files by direct position and load, file by file.

In a job process with any other user base, but with bases which make the save catalog visible, the jobs will do the same. Files loaded, though, will end up with the new user scope.

In a job process with bases, which make the save catalog invisible, the job will degrade to act as if the program load was used, i.e. get the save catalog from the tape and reload the files as they are met along the tape.

32.2 Example 2

In the example in section 12.2, a level 1 and a level 2 backup of the same file system were made on the base of the level 0 backup in 12.1, all along the tape 'mtdp0001'.

If you want to reload a certain user file on the same user bases in the 'latest edition' saved, but you dont know in which level to look, you submit the job:

```
incload mtdp0001.last scope.user.myfile
```

Now the job will reload the file, if it is found in the level 2 save catalog.

If not, only the catalog has been searched, and the program will print the warning:

***** incload no entries found according to...**

```
<load specifier>
```

and terminate with 'warning yes.ok.yes'.

Now, from the output from this job, you know in which file the latest level 1 backup is found, say file no 3, and you submit the job

```
incload mtdp0001.3 scope.user.myfile
```

with the same possible results as above.

If still not found, you submit the job

```
incload mtdp0001.1 scope.user.myfile
```

Note that the same jobs with survey.yes specified would make a survey of the save catalog alone and come up with the same warning and 'warning.yes, ok.yes' bits set if the file is not found in the save catalog.

Conditioned by the warning and ok bits, the job might continue with the next survey or with an actual reload from present file.

32.3 Example 3

In the example in section 12.3, incremental backups in different levels are made with std base of job process = project base of the project for which the file system backed up consists of all permanent files with bases inside the project base.

The backups are placed on tape volumes specified, level by level, in files named 'level0tapes', 'level1tapes' - etc.

Suppose you want to reload all files belonging to a certain user in their latest editions.

In the users job process, submit the jobs:

```
incload in.level0tapes scope.user
incload in.level1tapes scope.user
incload in.level2tapes scope.user
etc.
```

If the level 0 dump is the latest backup, stop here, If the level 1 dump is the latest backup, stop here, and so forth.

A simple lookup of the files 'level0', 'level1', 'level2' etc., will show which one contains the latest backup.

If the backups were made in a job process with std base = system base, the same jobs would work, provided no files 'level0', 'level1', 'level2', ... existed of bases better than system bases.

For each job, with a 'better' save catalog than system, the program would act as the program load and pickup the save catalog from the tape instead.

33. Call

The program is called exactly as the program load.

34. Function

The function is the same as for the program load, except the save catalog recorded in the dumplabel of the first tape mounted will be looked up in the main catalog and connected if found.

If it is not found or cannot be connected (covered by a better name), the save catalog is taken from the tape exactly as for the program load.

35. Compatibility

The programs incsave, save, incload and load are parameter compatible in the sense that any one of them may read any others parameter list and simply ignore unused parameters known to be significant to others.

36. Implementation Details

Described in Chapter 26.

37. Exact Formats

Described in chapter 7.

38. Requirements

Described in Chapter 28.

39. Error Messages

As for load, except the program name is 'incload'.

40. Further Examples

In the example in section 20.1, a file system is backed up periodically in level 9 backups.

The base of the level 9 backups is initially the full level 0 backup, later it becomes a level 1 backup and maybe even later it becomes a level 2 backup, and so forth.

Suppose a user wants to 'go hunting' for the latest backup of his file 'myfile' of scope user.
First, a simple lookup of the files 'level0', 'level1', ..., 'level9' tells which level contains the latest backup.
Starting with that level and going 'backwards', the user perfomrs surveys of the save catalogs. All he has to know is the name of the pool of tapes, or just a single tape in the pool latest used for each of the levels.

Suppose 'level9' contains the latest backup and only level 1 backups were made since the level 0 full backup.

Suppose 'pool92' contains the latest level 9 backup,
'pool1' contains the latest level 1 backup and
'pool0' contains the full level 0 backup

The job:

```
incload in.pool92 survey.yes scope.user.myfile
if warning.no
(incload in.pool92           scope.user.myfile
finis)
if ok.no
finis
incload in.pool1 survey.yes  scope.user.myfile
if warning.no
(incload in.pool1            scope.user.myfile
finis)
if ok.no
finis
incload in.pool0 survey.yes  scope.user.myfile
if warning.no
(incload in.pool0            scope.user myfile
finis)
if ok.no
finis
```

```
message myfile not found
finis
```

will 'hunt down' the file and load it if found.

Appendix A. References

1) RC8000/IDA801 Main process 991 09773

2) *System Utility, User's Guide, Part One* 991 11263

Part of SW8010I-D System Utility Manual Set