RCSL No: 991-10080

Edition: March 1985

Author: Finn Gustav Strøbech

Title:

RC8000 Utility Programs, Version 2
Save, Incsave
Users Manual

RCSL Nr. 46-F 0087

A/S REGNECENTRALEN
af 1979

Keywords:

RC8000, utility, save, backup, incsave, incremental backup, backup level, magnetic tape, streaming.

Abstract:

The program save performs backup on magnetic tape of single files or file systems.

The program incsave performs backup on magnetic tape of files on an incremental basis.

(90 printed pages)

RCSL Nr. 46-F 0087

Printed by A/S Regnecentralen af 1979, Copenhagen

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 discs and the increase in disc 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.

Special input/output operations have been developed to reduce the transport overhead and to even render superfluous RC8000 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 cotaining 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 in herent in the RC8000/IDA disc controller for transfer of backing storage areas from disc to tape connected to the same controller without engaging the RC8000 bus and memory is fully expoited in the two programs
- transfer of backing storage areas from disc to tape not connected to the same RC8000/IDA disc controller bound to engage the RC8000 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 disc 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 in the tape driver in the device controller as well as in IDA801 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 happend after output of the last block of each backing storage area.

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

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.

Finn G. Strøbech
A/S Regnecentralen, March 1985.

TABLE OF CONTENTS PAGE

TABLE OF CONTENTS (continued) PAGE

TABLE OF CONTENTS (continued) PAGE

TABLE OF CONTENTS (continued) PAGE

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 disc 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 mtlh 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 disc 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

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

You want to transfer to the tapes 'mtdpoool', mtdp0002', ... all permanent files (permkey=3) belonging to the two discs, 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 mthh mountspec.55              ,
     mt841201.1. mt841202. mt841203,
     segm3                          ,
     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 disc controller.

2.5 Example 5

If the discs '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 mthh 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 discs multiples of 21 segments are favorable.

The chances increase, too, by decrease in the overall system load on the RC8000 system bus and the discs.

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 mounted at stations connected via IDA801 Disc/Tape Controller, the blocklength must not exceed 84 segments.

3. CALL

```
{ <outfile> = }1
{           }0 ,
```

save

```
{{ <mount param> }1     <tape param> }2
{{               }0                 }1 ,

{ <special param> }1
{                 }0 ,

{ <save specifier> }1
{                  }0
```

3.1 Outfile

<outfile> ::= name of any filedescriptor

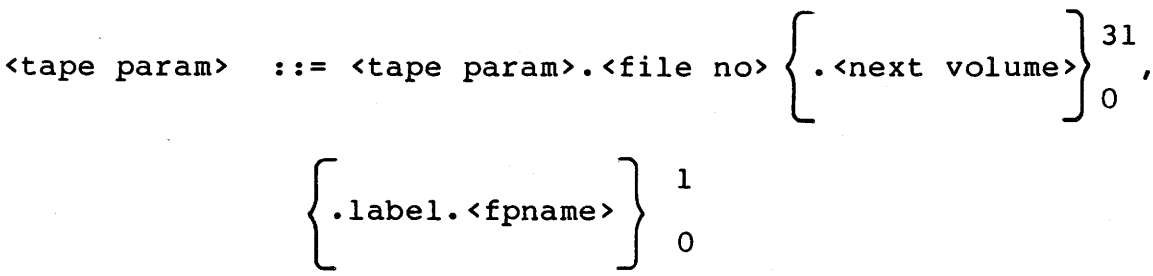
3.2 Mount Parameter

```
<mount param> ::= [ mountspec. <device no>          ]*
                  [ release. {yes/no}               ]
                  [ <modekind>                      ]1
```

<device no> ::= <integer>

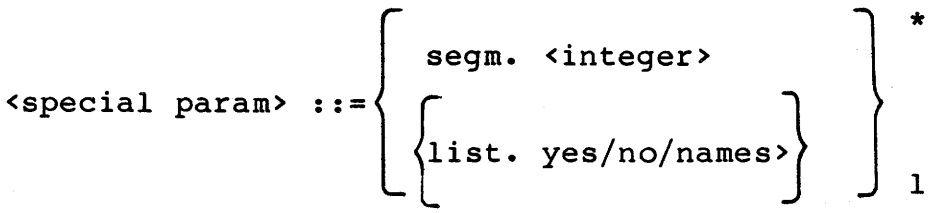
```
<modekind> ::= [mthh]
               [mtlh]
               [mthl]
               [mtll]
```

3.3 Tape Parameter

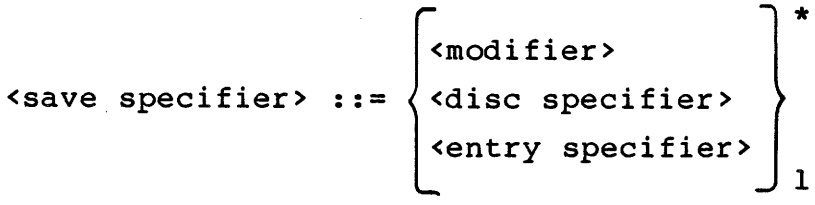


<tape name> ::=
<next volume> ::= <name>/<filedescriptor>
<name> ::= name of magnetic tape
<file descriptor> ::= name of magnetic tape file descriptor
<file no> ::= <integer>/last
<fpname> ::= name obeying fp syntax

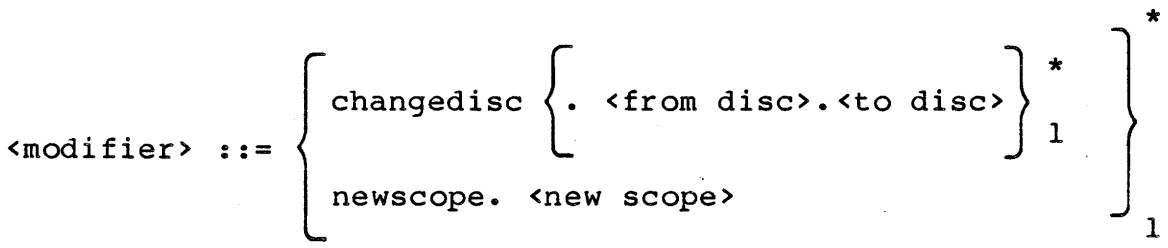
3.4 Special Parameter



3.5 Save Specifier

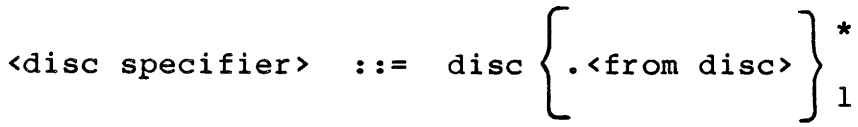


3.5.1 Modifier

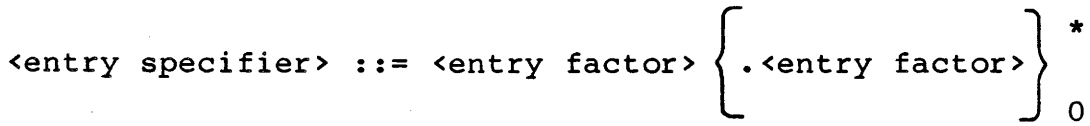


<from disc> ::= all/maincatdisc/<disc name>
<to disc> ::= no/maincatdisc/<disc name>/0/1
<new scope> ::= no/temp/login/user/project

3.5.2 Disc Specifier



3.5.3 Entry Specifier



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

3.6 Infile Parameter

Everywhere the delimiter <s> is allowed in the parameter list according to syntax for FP commands, (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 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.

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

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

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

<mount param ::= [mount spec .<device no> | <modekind> | release. { yes/no }]* 1

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: mtlh (=mto)

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> $\left\{ .\text{<next volume>} \right\}_{0}^{31}$,

$$\left\{ .\text{label.<fpname>} \right\}_{0}^{1}$$

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 file, which does not start with a version or a continuation dump label record, 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

<special param> ::= [segm.<integer> / list.[yes/names/no]]* 1

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.

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). 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 = changenetry 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

$$\text{<save specifier>} ::= \left\{ \begin{array}{l} \text{<modifier>} \\ \text{<disc specifier>} \\ \text{<entry specifier>} \end{array} \right\}_{1}^{*}$$

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

An entry specifier will start a main catalog scan, picking out entries specified belonging to discs currently specified and record them in the save catalog with current modifiers.

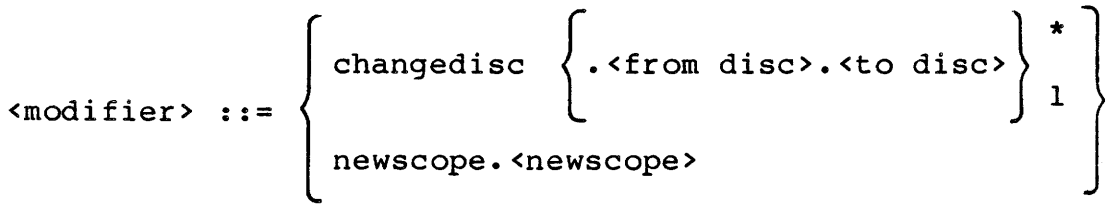
Entries picked out which belongs to a disc specified in current disc specifiers will be saved in a temporary save catalog together with its specification (disc, scope) and its modification (changedisc, newscope).

If no entry specifier is found in the parameter list after a modifier or a disc 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.

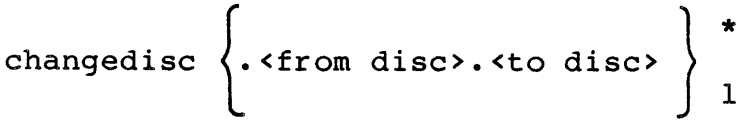
Concluding the backup, a tape mark is output and a file of one block containing an empty dump label record, listed on current output too, is written.

4.5.1 Modifier



A modifier is valid until changed by another modifier.

4.5.1.1 Changedisc



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 disc with the main catalog
<disc name>	means the disc with that name
<to disc>::=	no/maincatdisc/<disc name>
no	means changedisc. <from disc>.<from disc>

0/1	means that the disc is selected by the monitor at time of reload (the first disc with sufficient temporary resources).
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

<disc specifier> ::= disc $\left\{ \text{.<from disc>} \right\}_{1}^{*}$

A disc specifier specifies one or more discs from where the entry specifier will specify entries.

A disc specifier is valid until the next disc 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

<entry specifier> ::= <entry factor> $\left\{ .\text{<entry factor>} \right\}_{0}^{*}$

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, 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 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 program will be held up while the writing takes place, and then it will continue. No warning is given.

If the size of the area changes during the backup, the program will continue, but the file itself and maybe some neighbouring files will be unloadable.

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 by 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 any 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'.

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 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 changedisc parameter <to disc> = 0/1 is allowed and is equivalent to <to disc> = no.

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

The disc 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 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 produced are not compatible with the tapes produced by earlier releases of save.

It should be noted, then, that
- tapes produced by version 2 of save must be loaded by version 2 of load
- 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 disc 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 disc 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 (7.1) is written in the file and at this point zones to handle disc 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 disc 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) removing the area processes succeedingly.

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

Finnally the use of the magnetic tapes is finished, writing a one block file with an empty dump label block (cf. 7.2.1) and 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

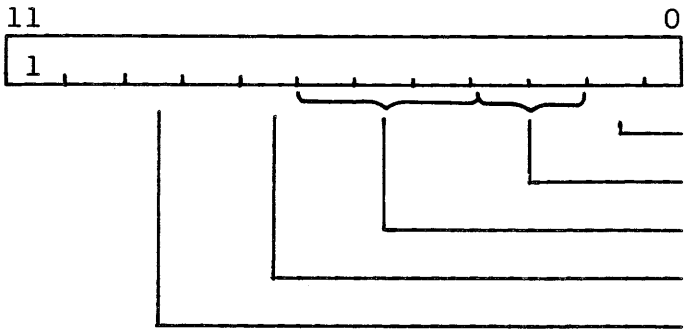
- the mount parameters
- the tape parameters

6.2.1 Mount Parameters

The magnetic tapes are prepared according to the mount parameters

- modekind abbrivation
- mountspec parameter

The modekind specified is converted into the mode word:



parity : 0 odd, 1 even
density : 0 high, 1 low
trail : 0-7
speed : 0 low, 1 high
block gap length: 0 std, 1 long

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

The fields of the mode word have different meanings for different tape stations:

		RC3715	RC8343	RC8344
parity	even	even	odd	odd
	odd	odd	odd	odd
density bpi	high	1600 pe	3200 pe	6250 gcr
	low	800 nrz	1600 pe	1600 pe
speed ips	high	45	100/50 stream	75 stream
	low	45	25/12.5 stream	25 stream
block gap inch	std	0.6	0.6-0.9/ 0.3-0.6	0.6-0.9/ 0.3-0.45
	long	0.6	0.6-1.2	0.6-1.2/ 0.3-0.6

Note: slash (/) means at low density/at high density.

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:

ring <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.

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 file which does not start with a version or continue dump label (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 covered by a better entry.

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 the area.

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 entire save 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 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 Strategi

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

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

- 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 both 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:

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 = 3, unintelligible, may be caused by the choice of blocklength and/or the slicelength of the disc 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 discs 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 discs and areas belonging to other discs, 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 specfied no matter which kind of discs 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 discs are involved in the backup

- any device status, disc 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 both 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 RC8000 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 core 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 discs 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 disc no. 1 (8 hwds)
+20 + 8 * 2	: name of disc no. 2 (8 hwds)
	.
	.
	.

```
+20 + 8 * no_of_discs   : name of disc no. no_of_discs (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 to 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             : disc number
+42             : new disc name (1)
+44             : new disc name (2)
+46             : new disc name (3)
+48             : new disc 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 disc number is the number of the disc to which the entry belongs, numbered 1, ..., no of discs in the system.

The field new disc name contains the name of the new disc 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 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).
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 tapemark is written and the next file on the tape will be a one block file containing an empty dump label block.

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
- empty   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.', 'cont.' or 'empty' 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 (halfwords)

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.

7.2.4 Partial Catalog Blocks

A partial catalog is a number of block 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 disc 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 practive 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 followid 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 disc specifier was unknown. Previous disc specifier is cancelled, all other discs specified in this disc 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 disc 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 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	create area process failed
catalog i/o error, state of document does not permit call	create area process failed
entry not found	create area process failed
entry not an area entry	create area process failed
name format illegal	create area process failed
reserved by another process	set write protect/reserve failed
current output file	the area is connected as current output file
process does not exist, process is not user of area process	set write protect/reserve failed
covered by a better entry	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 one exception that the character 'NL' is equivalent to the character 'SP'.

The current parameter is listed in the error message and the zone stack chain is emptied, listing the chain on current output the same way fp does in 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 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 1

The file pip of scope user and all files with documentname pip and scope user are saved on mtdp0001 file 1 by the call:

```
save mtdp0001.1 pip.scope.user docname.pip.scope.user
```

10.2 Example 2

All files of scope temp, login, user or project on the discs: disc, disc1 and disc2 are saved changing their disc 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 3

All files in the main catalog, except
- the main catalog itself
- the auxiliary catalogs
- files of name c or v with permkey = 0
- files of name primout with permkey = 2
- files belonging to other discs than disc, disc1 and disc2

are saved, disc by disc 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 4

In example 1.4, the permanent files on two logical ida discs 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. EXAMPELS

12.1 Example 1

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 2

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.

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.

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 3

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

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

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

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

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

```
mountspec.55
mthh
mtdp0201.last.mtdp01202.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 ammount 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, specially 7.1 where 'shortclock latest changed' is defined.

18. REQUIREMENTS

As for save, cf. 8.

19. ERROR MESSAGES

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

20 FURTHER EXAMPLES

20.1 Example 1

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.

A. REFERENCES

(1) RCSL No 991-9773
RC8000/IDA801 Main process

(2) RCSL No 31-D676
Utility Programs, Part One

(3) RCSL No 991-10081
RC8000 Utility Programs, Version 2
Load, Incload
Users Manual

RETURN LETTER

Title: RC8000 Utility Program, Version 2 RCSL No.: 991-10080
Save, Incsave, Users Manual

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

_____

_____

_____

_____

Do you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

How can this manual be improved?

_____

_____

_____

_____

Other comments?

_____

_____

_____

_____

_____

Name: _____ Title: _____

Company: _____

Address: _____

Date: _____

Thank you

RCSL Nr. 46-F 0093

..... Fold here .....

..... Do not tear - Fold here and staple .....

Affix
postage
here

REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark