RCSL No: 991-10081

Edition: March 1985

Author: Finn Gustav Strøbech

Title:

RC8000 Utility Programs, Version 2
Load, Incload
User's Manual.

RCSL Nr. 46-F 0087

A/S REGNECENTRALEN
af 1979

Keywords:

RC8000, utility, load, incload, backup, incremental backup, backup level, magnetic tape, streaming.

Abstract:

The program load relocates and reloads single files or file systems from backup tapes made by save or incsave.
The program incload relocates and reloads single files or file systems from incremental backup tapes made by incsave.

(70 printed pages)

RCSL Nr. 46-F 0087

Printed by A/S Regnecentralen af 1979, Copenhagen

Forword

In System Utility Package, SW8010/2, Release 1.0, 1984.10.01, four new programs load, incload, save and incsave have replaced the two programs save and load, issued in the package, through, with the names save13 and load13.
On the following pages the new manual for load and incload 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 incload.

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

TABLE OF CONTENTS **PAGE**

TABLE OF CONTENTS **PAGE**

TABLE OF CONTENTS **PAGE**

1. 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.

2. EXAMPLES.

2.1 Example 1

All catalog entries and backing storage entries saved in Example 1 of ref. (3), 2.1 are relocated and restored by the call:

```
load mtdp0001.2
```

In case

```
t=set mtlh 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
```

2.2 Example 2

All files saved in example 2 of ref (3), 2.2, 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 mdtp0001.2 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.

2.3 Example 3

Among the files saved in example 3 of ref (3), 2.3, the ones belonging to the discs 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 other discs than the ones known at the time of load will be loaded too, changing to the disc 'disc3', known to the job:

```
load mdtp0001.last disc.all.any changedisc.any.disc3
```

The ones belonging to discs not known will be loaded to discs chosen by the monitor by the call:

```
load mtdp0001.last disc.any changedisc.any.0
```

2.4 Example 4

The system backup in example 4, ref. (3), 2.4, contain all permanent files from the discs 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 mthh 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 the 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, mtlh (=mt0), 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 file, 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).

3. CALL.

$$\left\{\text{<outfile>}\right\} = \begin{matrix}1\\0\end{matrix}$$

load

$$\left\{\left[\text{<mount param>}\right]\begin{matrix}1\\0\end{matrix}\quad\text{<tape param>}\right\}\begin{matrix}2\\1\end{matrix}$$

$$\left\{\text{<special param>}\right\}\begin{matrix}1\\0\end{matrix}$$

$$\left\{\text{<load specifier>}\right\}\begin{matrix}1\\0\end{matrix}$$

3.1 Outfile

<outfile> ::= name of any filedescriptor

3.2 Mount Parameter

$$\text{<mount param>} ::= \left\{\begin{array}{l}\text{mountspec. <device no>}\\ \text{release.}\left\{\text{yes/no}\right\}\\ \text{m<modekind>}\end{array}\right\}\begin{matrix}*\\1\end{matrix}$$

<device no> ::= <integer>

$$\text{<modekind>} ::= \left\{\begin{array}{l}\text{mthh}\\ \text{mthl}\\ \text{mtll}\\ \text{mtlh}\end{array}\right\}$$

3.3 Tape Parameter

```
<tape param>    ::=  <tape param>.<file no> [.<next volume>]  31
                                                              0

                     [.label.<fpname>]  1
                                        0

<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

```
<special param>::= [ vol.<integer>                ]*
                   [ copy.<integer>               ]
                   [ list.[<yest/no/names>]       ]
                   [ load. [yes/no]               ]
                   [ survey. [yes/no]             ]
                   [ connect. [yes/no]            ]1
```

3.5 Load Specifier

```
<load specifier>::= [ <modifier>       ]*
                    [ <disc specifier> ]
                    [ <entry specifier>]1
```

3.5.1 Modifier

```
<modifier>      ::= [ changedisc [.<from disc>.<to disc>]  *
                                                           1 ]*
                    [ newscope.<new scope>                   ]1

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

3.5.2 Disc Specifier

<disc specifier>::= disc [.<from disc>]$_1^*$

3.5.3 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]

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 syntactially equivalent to <s>.

<file> ::= name of any filedescriptor.

4. FUNCTION.

The program will either
- relocate and reload the catalog entries and backing storage areas specified by <disc 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 disc 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, succeeding 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 exhausted, 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.

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: 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: mtlh (= mto)

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

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> ::= $\left[\text{<name>/<file descriptor>}\right]$
The name of the tape. If <file descriptor> is used, the name and modekind are taken from the descriptor.

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

4.4 Function, Special Parameter.

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

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 disc.
Default: load.yes

survey.{yes/no}
If survey.yes, only the save catalog transferred to disc 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 disc and renamed, thus
- spending extra resssources 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

4.5 Function, Load Specifier

<load specifier::= { <modifier> | <disc specifier> | <entry specifier> }* 1

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

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

Entries picked out which belongs to a disc specified in current disc 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 disc specifier or no load specifier is found, a default entry specifier will be used.

4.5.1 Modifier

<modifier> ::= { changedisc [.<from disc>. <to disc>]*1 | newscope. <newscope> }* 1

A modifier is valid until changed by another modifier.

4.5.1.1 Changedisc

changedisc $\left\{.\text{<from disc>}.\text{<to disc>}\right\}_{1}^{*}$

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

<from disc>	::=	{all/maincatdisc/<disc name>/any}
all		means all discs known in the system at time of reload
maincatdisc		means the disc with the main catalog
<disc name>		means the disc with that name
any		means any disc not known in the system at time of reload

<to disc>	::=	{no/maincatdisc/<disc name>/0/1}
no		means changedisc. <from disc>.<from disc>
o/1		means that the monitor will select the disc at reload
others		as for <from disc> exept 'any'.

Default: changedisc.all.no.any.no

4.5.1.2 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

4.5.2 Disc Specifier

<disc specifier>::= disc [.<from disc>]*1

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

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

<from disc> ::= [all/maincatdisc/<disc name>/any]
The parameters have the same meaning as for 'change-disc'.
Default: disc.all.any

4.5.3 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

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.
Default: any name

4.5.3.2 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.
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 of 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.

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 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 'nl' is equivalent to the character 'sp'.

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.

4.7 Alternative and Dummy Parameter Names.

For compatibility reasons, some alternative parameters are allowed.

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 disc> = main is still allowed and equivalent to <from disc> = all.

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

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

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

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

5. 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 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 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 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 received by installations with earlier release of load only, must be produced by earlier 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.

6. 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.

6.1 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 multi-level, 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 discs 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 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 ((3), 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 ((3), 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 disc.

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 uddated.

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. ref. (3), 6.3)
- a working name area entry of the scope wanted could not be created on the target disc (connect.no) or
- an existing area entry of the name, scope and disc 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.

6.2 Handling of Magnetic Tapes.

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, mountspec and release, as for save cf. ref. (3), 6.2.1, except that all modes will be tried sucessively in case of mode error.

6.2.2 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 contains a version or continue dumplabel record, 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 a neither version nor continue dumplabel record 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 tranferred to a working name temporary disc area (6.1), 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 ((3), 7.2.1).

The tape or set of tapes recorded in the save catalog head (cf. ref (3), 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.

6.3 Handling of Backing Storage Areas.

Whenever an area entry from the load catalog is refound 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 disc, 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 disc specified (if new disc name = 0 or 1, the monitor chooses the disc). 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 disc 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 sucess, 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.

6.4 The input/output Strategy.

The transfer of backing storage areas from tape to disc takes place as described in ref. (3), 6.4.2, Trans Memory Transfer.

7. EXACT FORMATS.

Described in ref. (3), Chapter 7.

8. REQUIREMENTS.

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

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

8.1 Memory.

As described in ref. (3), 8.1.

8.2 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 buffer
else it will need only
    4 message buffers.

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

8.3 Disc Ressources.

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 disc in the reload and for each permanent key to be used on that disc, it will need one entry and as many segments as the largest file to be reloaded on that disc for that permanent key.
If connect.yes is specified, no extra resources will be needed for the areas to be reloaded.

9. 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

9.1 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)

9.2 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.

9.3 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.

9.4 Monitor Alarm.

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

*** 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.

9.5 Other Alarms.

All other alarms from the program concern the reading and contents of dumplabels on tape and the transfer/by-passing 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>

Following an explanation for each possible <text>.

no dumplabel found on volume tape, file 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 <value>.

not all segments of 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.
The name of the tape and the size of the save catalog recorded are shown in <name> and <value>.
The reason is i/o troubles and the alarm should not occur.

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

dumplabel incompatible on volume tape, file 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>.

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>.

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

not all segments of area transferred from tape:
Not the number of segments recorded in the save catalog concerning the area are transferred from tape.
The name of the area and the number of segments transferred are shown in <name> and <value>.
Reason as above.

not all segments of 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.
Reason as above.

9.6 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 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.
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 specifier 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.

9.7 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 reserved or the area was inaccessible at save for the reason stated in <cause>.

The entry and the area have not been saved.

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

Cause:	Reason:
area claims exceeded	create area process failed
catalog i/o error, state of document does not permit call	create area process failed
entry not found	create area process failed
entry not an area entry	create area process failed
name format illegal	create area process failed
reserved by another process	reserve failed
not user, cannot be reserved	reserve failed
does not exist	reserve failed

9.8 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

disc: <disc 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.

9.9 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.

9.10 Parameter Input Syntax Error Message.

This message is caused by a syntax eror 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 one exception that the character 'NL' is equivalent to the character 'SP'.

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

The parameter reading is continued in the fp command stack.

The modebits are left unchanged.

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

9.11 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 (mtlh, mte, mtll, nrze, 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'.

10. FURTHER EXAMPLES.

10.1 Example 1.

Continuing the example in 2.4, where the volume tapes mt841201, mt841202 and mt841203 had been used for a backup of all permanent files in the system (cf. ref. (3), 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 mth1 mt841201.last connect.yes scope.user
```

The job will mount the tape mt841201, and if the operator mounts the tape at a station where 'low density' is 1600 bpi and sets the density at 'low', the job will search along the tapes until mt841203, from where the save catalog is taken.

Since the user files wanted may be scattered all over tapes, the job may now require the first tape mounted (if not still mounted), position from user file to user file and load them, require the second tape mounted a.s.o. 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.

Should the operator mount one of the tapes at a station where 1600 bpi is 'high density' and set the density 'high', the program would just display a mode shift message for each mode tried until 'high density' and continue the job in 'high density'.

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

10.2 Example 2.

Consider the example in ref (3), 10.3, where all files in the system, temporary or permanent, belonging to the discs 'disc', 'disc1' and 'disc2', are backed up on two copies of tapes, each specified to hold 5 volume tapes, all written in 'high density'.

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.
If it does, 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.

11. 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.

12. EXAMPLES.

12.1 Example 1.

In the example in ref. (3), 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.

12.2 Example 2.

In the example in (3), 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.

12.3 Example 3.

In the example in ref. (3), 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' - a.s.o.
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
a.s.o.
```

If the level 0 dump is the latest backup, stop here,
If the level 1 dump is the latest backup, stop here,
a.s.o.

A simple lookup of the files 'level0', 'level1', 'level2' a.s.o. 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.

13. CALL.

The program is called exactly as the program load.

14. 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 (created by a better name), the save catalog is taken from the tape, exactly as for the program load.

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.

Described in Chapter 6.

17. EXACT FORMATS.

Described in ref. 3 , Chapter 7.

18. REQUIREMENTS.

Described in Chapter 8.

19. ERROR MESSAGES.

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

20. FURTHER EXAMPLES.

In the example in ref.(3), 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 if becomes a level 1 backup and maybe even later if becomes a level 2 backup, a.s.o.

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 performs 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.

A REFERENCES.

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

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

(3) RCSL No 991-10080
RC8000 Utility Programs, Version 2
Save, Incsave
Users Manual

RETURN LETTER

Title: RC8000 Utility Program, Version 2
Load, Incload Users Manual.

RCSL No.: 991-10081

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