RCSL No: 31-D601

Edition: June 1980

Author: Inge Borch
Edith Rosenberg

Title:

Extensions to the RC8000
Indexed Sequential Files System (ISQ)

A/S REGNECENTRALEN
af 1979

Keywords:

RC8000, Algol, Backing Storage Package, Indexed Sequential Files, Manual.

Abstract:

Describes new features in RC8000 Indexed Sequential Files System (RCSL No 31-D600), which e.g. include checking and recovering of broken files and 2 new procedures for fast updating of blocks.

(40 printed pages)

Printed by A/S Regnecentralen af 1979, Copenhagen

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without prior notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and shall not be responsible for any damages caused by reliance on any of the materials presented.

FOREWORD

First edition: RCSL No 31-D 559

The indexed Sequential Files System is part of the Backing Storage Package available for RC4000, RC6000, and RC8000 models. No feature in the system has been changed from the first release in 1971 until the release in April 1979, which holds a subset of the facilities mentioned in this manual.

RCSL No 31-D 558, RC8000 Indexed Sequential Files, is updated for the changes the new features have claimed on the 'old' system, while this manual describes the pure extensions. It replaces a preliminary manual, RCSL No 31-D 514 (September 1978). The definition of the update mark facility is slightly changed, and two more procedures, extendi and priorreci, are included. Procedure updotheri has been renamed putdirecti.

Inge Borch
A/S Regnecentralen, April 1979

Second edition: RCSL No 31-D 601

This edition contains a few corrections in appendix C and E, which are marked with correction lines.

Edith Rosenberg
A/S Regnecentralen af 1979, June 1979

TABLE OF CONTENTS PAGE

1. INTRODUCTION 1.

This manual describes new features for the Indexed Sequential Files System (ref. 1). Below in this section they are summarized, in chapter 2 explained for potential users, and in chapter 3 exact programming interfaces are given.

The Indexed Sequential Files System is designed to be very efficient in both sequential and direct access mode. The three types of blocks, the buckettable, the blocktabel and the one holding records are kept in the primary storage and are only transported, when it is absolutely necessary. E.g. as the file holds only one buckettable this is read when the file is started and only written when it has been updated and the user claims a mode shift (e.g. at closing). This may have the sideeffect that if the program breaks (e.g. caused by a power lack, or programming bug) a rewriting of an updated block may be pending, in other words, the file holds tables and records of different versions.

In the version of the system introduced here, a touch of redundancy has entered the record blocks. This is used by a check and recover program, recoveri, to examine, if the tables matches the occuring records. Further an 'update mark' guards that a file, which has not been closed correctly, is not reused accidentially.

A call of one of the mode changing procedures ensures that all updated blocks are rewritten and they are often called with that purpose, solely. The new procedure, putblocki, will ensure that the recordblock is rewritten immediately, which may save two rewritings compared to the mode procedures. In a break situation only rewriting of table blocks may be pending, and though the file is inconsistent, the check and recover program is able to repair the file without loss of records.

Another new updating procedure, putdirecti, updates immediately a previously fetched record without changing the current record situation. E.g. it may be used to copy information form the current record to another record.

A sequential scanning of the file is performed by repeated calls of the procedure nextreci. Now it is also possible to fetch records in the reverse order, as the new procedure priorreci makes the record prior to the current record available as current record.

Extension of an isq-file means adding new buckets to the file. The procedure startfilei will automatically include new buckets, if the area holding the file and the zone have room for it. The new procedure extendi makes it possible to extend the file during record processing if only the zone has room for it, and the user has the necessary backing storage claims for changing of the catalog entry. Extendi can also cut empty buckets in the end of the file.

For a long time users have asked for a procedure, which can make the users file parameters available. The procedure headparamsi is designed to be the inverse procedure of headfilei, and may thus solve the problem.

2. EXTENSIONS AND MAJOR CHANGES 2.

2.1 Update Mark 2.1

The update mark is actually an integer stored in connection with the buckettable. Bits are used as flags to indicate some file states:

- the file is during intialization or recovering
- the file has entered an updating mode

The check and recover program (see section 3.7) cannot handle a file, which has not got through the initialization, i.e. passed a call of one of the mode procedures after the init-procedures, or a former recovering.

If a file has been in updating mode, it must return to reading before it is closed. If it does not, the case is signalled as a result from startfilei (ref. 1). The file may then be read, but if it enters updating, the system will cause a run time alarm, and the file must be reestablished by a backup, or by the check and recover program.

2.2 Standard Integer blocki 2.2

Besides the standard integer resulti, the isq-system now supports an integer, blocki, which will hold the segment number of the record block in use. It's purpose is to supply the user with the segment number, which may be used as call value for the new procedure putdirecti. See example 2.

2.3 Recovery Program 2.3

2.3.1 Errors in isq-files 2.3.1

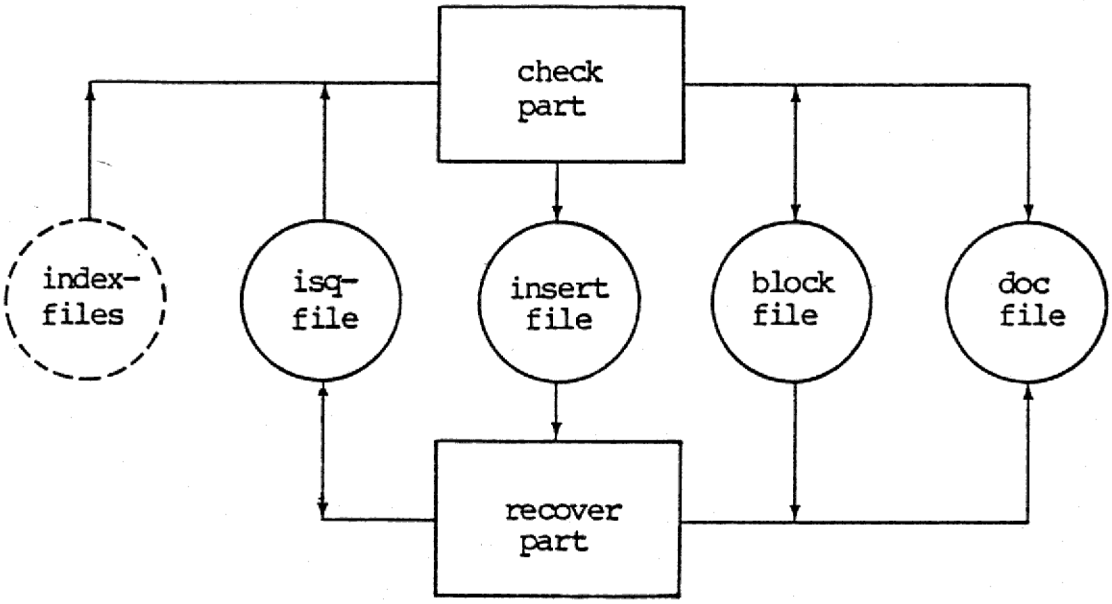
Inconsistencies in isq-files may origin from breaks between the writing of a record block and the corresponding table block, or between writings of two record blocks in a complicated insertion. Both cases will be signalled by the update mark. The check and recover program, recoveri, is able to recover the file without loss of records in the first case, which is the most probable. In the second one a number of halfwords (approx. an average record) may be lost or will exist in duplicate in the broken file, and recoveri cannot compensate the loss, while it will select a winner among the duplicates of records.

It is hard to predict, which other types of errors that may be reflected in a file, as some could occur when it is not reserved by the isq-system and are thus not signalled by the update mark. It is recommended to check the file regulary by recoveri.

2.3.2 Program Functions 2.3.2

recoveri is designed to check that a file is correct and to repair an incorrect file, so that usage can be resumed as soon as possible.

It consists of two parts, which may be executed separately or as a whole. Fig. 1 shows the two parts and the files they have in use.



Figur 1: The recovery program.

In the check part the isq-file is scanned sequentially, and it is only repaired in the recover part, which gives the user the possibility to assess the damages before the recovery. Errors are listed at the text file 'doc file', while the two files 'insert file' and 'block file', together named the 'recovery files', hold records, which act as transactions to the recovery process. See the survey in app. D. 'Indexfiles' are generated in the primary storage during the scanning of 'isq-file' to check the logical structure of 'isq-file'.

Reading of the remaining part of this section claims some knowledge of the isq-file structure (see ref. 1, sec. 1).

In the check part the record blocks are checked for 1) legal record length values, 2) correct key sequence, and 3) fillers (zeroes) in the free words of the blocks. Errors found in those tests cannot origin from the isq-system nor from break situations, so the whole block will be dismissed by a 'delete block'-record written at 'block-file'.

For correct record blocks in a bucket an index is created with elements holding the first and the last key and the number of used bytes in the record block. The index is sorted to check that the keys don't overlap each other, and the sorted index is matched with the blocktable in 'isq-file' to find any deviations. Overlapping blocks will be dismissed by 'delete block'-records like above, but the records are extracted and written at 'insert file'. Deviations from the blocktable are recorded at 'block file' as 'block table element'-records. Errors found as overlapping keys or blocktable deviations may very well origin from breaks like those mentioned in the beginning of section 2.2.1.

For all the buckets in the file an index is created with elements holding the first and the last key and the size of the block table for the bucket. Analogously with the blockindex, this index is sorted, checked for overlaps, and matched with the bucket table. If overlapping buckets exist, all blocks in the involved buckets will be dismissed by 'delete block'-records and the records extracted and written at 'insert file'. Deviations from the bucket table are recorded at 'block file' as 'bucket table element'- records.

Overlapping buckets should not be possible seen from the isq-system, while bucket table deviations may possibly origin from a break during updating of tables, and this gives no loss of records.

At the end of the check part the recovery files are sorted, the 'block file' primarily according to segment numbers in 'isq-file', and insert file accordng to the key fields of 'isq-file' plus a field (may be user defined), which may queue duplicates of records.

In the recover part the records of 'block file' are interpreted sequentially and 'isq file' repaired blockwise. The tables are adjusted and record blocks cleared, and the 'isq file' is ready for isq-processing. Then the 'insert file' is read and the records inserted in 'isq file' by the isq-procedure insertreci. Duplicates of records are dismissed with a diagnostic at 'doc file'.

'recoveri' uses current output for run time alarm messages and warning messages, see app. B, while error diagnostics are written at 'doc file', which may be handled as a normal text file. An error diagnostic consists of an explanatory text and various fields, which identify the error. A survey is given in app. C.

2.3.3 Program Requirements 2.3.3

The core requirements for 'recoveri' may be as low as 23000 hw for files with minimal block lengths, but as the program reads 'isq file' with super-buffering and have internal sortings the processing time will decrease much with a greater core area. A sensible lower limit will be about 50000 hw for small files to which may be added the size of 'isq-file' to get the upper limit for profitable core utilization. If more than one error is found 'recoveri' may need some working area at the backing storage as mentioned in ref. 3.

Files which have been broken during initialization or recovering cannot be handled by recoveri.

2.4 New Record Processing Procedures 2.4

The system is extended with two procedures which may speed up the updating of an isq-file, putblocki and putdirecti, and one procedure, which reads the record prior to the current record, namely priorreci.

Putblocki will ensure that the currently available record block is immediately written at the backing storage, while other writings take place when a block change is needed or when a mode procedure is called.

Example 1, putblocki:

```
setputi(z);
.
.
.
getreci(z, key);
if resulti = 1 then
begin
    z.cash:= z.cash + money;
    putblocki(z);
    comment now money is in cash, if the system breaks.
       as the file is in put mode, this record block
       will not automatically be rewritten at a later
       block change.
    ;
end;
z.a:= nothing;
z.b:= something;
comment nothing and something will be remembered,
if the system does not break.
;
setreadi(z);
comment the record block and the updated table
blocks are rewritten now.
;
close(z, true);
```

Putdirecti makes it possible to update a record without changing the current record situation. The procedure uses the same work area in the zone buffer as insertreci for complicated insertions (ref. 1) and accesses the file directly by blockaddressing, not by indexing as the other procedures. The user must yield the block address, but this can easily be obtained from the standard integer blocki.

Example 2, putdirecti:

```
comment while traversing a file the greatest value of a field is
  immediately copied to a distinct record in the same file.
;
startfilei(z);
if resulti > 1 then system(9, resulti, <:<10>bad start:>;
firstrec.key:= z.key;
maxrec.key:= 8000000;
getreci(z, maxrec);
if resulti = 1 then
  maxaddr:= blocki
else
  systemi(9, maxrec.key, <:<10>lost rec:>);
getreci(z, firstrec);
setupdatei(z);
while resulti = 1 do
if z.n > maxrec.n then
begin
  maxrec.n:= z.n;
rep:
  putdirecti(z, maxrec, maxaddr);
  case resulti of
  begin
    ; <*resulti = 1, ok*>
    begin <*resulti = 2, maxrec lost*>
      getreci(z, maxrec);
      if resulti = 1 then
      begin
        maxaddr:= blocki;
        getreci(zm, key);
        goto rep;
      end
      else
        system(9, maxrec.key. <:<10>lost rec:>);
    end;
    system(9, maxrec.l, <:<10>length:>);
    system(9,        0, <:<10>no buf:>)
  end;
  nextreci(z);
end while if;
```

Priorreci makes the record prior to the current record available as current record and may thus be used to scan the file from the final record to the first one, but it is faster to use the reverse order with nextreci (ref. 1).

Example 3, priorreci:

```
startfilei(z); <*get the first record*>
priorreci(z);  <*get the last record, resulti = 2*>
comment count the records;
i:= 0;
repeat
  priorreci(z);
  i:= i+1;
until resulti <> 1;
```

2.5 New Service Procedures 2.5

Two new file handling procedures are introduced, headparamsi, which reads the file definition parameters from a file head, and extendi, which may extend a file with more buckets or cut unused buckets. See the procedure definitions in section 3.3 and 3.2 respectively.

headparamsi may be perceived as the reverse procedure of headfilei (ref.1), which creates an isq-file head from the user's parameters. headparamsi reads a file head and supplies the user with the original file head parameters.

Example 4, headparamsi:

```
comment create a file head for file b, which is equal
     to that of file a, except that it has an extra key
     field.
;
open(za, 4, <:a:>, 0);
open(zb, 4, <:b:>, 0);
```

```
headparamsi(za, recdescr, nkey, maxreclength, maxlength,
          segsperbuck, segsperblock);
nkey:= nkey +1;
comment move the length definition:;
recdescr(nkey +1, 1):= recdescr(nkey, 1);
recdescr(nkey +1, 2):= recdescr(nkey, 2);
comment the extra field is of type integer and placed two
   hw after the previous field:
;
recdescr(nkey, 1):= 2;
recdescr(nkey, 2):= recdescr(nkey -1, 2) +2;
```

extendi may be used when the file is in an updating mode or during initialization. It includes new buckets in the file or excludes unused buckets and segments in the end of the file by changing the catalog entry and the buckettable. New buckets will hold an empty blocktable and cleared record blocks. For the extension extendi uses the work area in the zone, so this must be declared for full insert besides some extra double words for addition of new bucket table entries. The size of an entry is given in ref. 1 section 1.3.

Example 5, extendi, include new buckets:

```
zone zi(buflengthi(<:i:>, true) +10, 3, stderror);
comment the zone is declared to hold five extra bucket
   table entries of the size 1 + keypartsize = 1 + 1 =
   2.
;
.
.
.
insertreci(zi, record);
case resulti of
begin
   ; <*insert ok*>
   result2;
```

```
 result3;
 begin <* resulti = 4, file is full*>
     extendi(zi, 1);
     if resulti <> 1 then
        system(9, resulti, <:<10> extend:>)
     else
     begin
        insertreci(zi, record);
        if resulti <> 1 then
           system(9, resulti, <:<10>2nd insr:>);
     end
  end
  result 5;
  result 6;
end
```

Example 6, extendi, exclude unused segments:

```
comment the program reorganizes an isq-file by
sequential copying from one file to another.
;
initfilei(zo, 0.8, 0.8);
startfilei(zi);
while resulti = 1 do
begin
  initreci(zo, zi);
  if resulti <> 1 then
  begin
      .
      .
      .
      .
  end;
  nextreci(zi);
end;
extendi(zo, -1);
if resulti <> 1 then
  system(9, resulti, <:<10>no cut:>);
```

3. PROGRAMMING INTERFACES 3.

In this chapter the new entries to the isq-system are described in alphabetic order.

3.1 Standard Integer blocki 3.1

Function: After the call of an isq record processing procedure, this integer holds the segment number of the available record block.

3.2 Procedure extendi 3.2

Call: extendi(z, segments)
z(call and return value, zone).
Specifies the file.

segments (call value, integer).
If segments > 0, the file will be extended with as many buckets as needed to include 'segments'.
If segments < 0, unused segments in the end of the file area are released.
If segments = 0, the catalog entry is updated with shortclock.

Function: The procedure changes the catalog entry of the file, so that it may either be extended or cut. If extension is wanted, the zone should be declared for full insertion and with some extra room for extension of the buck table.

Requirements: zonestate = initializei, puti, or updatei.

Results: zonestate unchanged.
procnoi: 17
Available record: unchanged

resulti:

1	Done
2	Not done. Only room for simple insertion.
3	Not done. The length of z can not accomodate the new buckets.
4	Not done. Maxbucks exceeded.
> 10000	Not done. Error at a call of a monitor function:

resulti = monitor result *10000 + monitor
function no.

Probable results:

40044 changeentry, protected

60044 changeentry, claims exeeded

See ref. 5 for monitor functions.

3.3 Procedure headparamsi 3.3

Call: headparamsi(z, recdescr, nkey, maxreclength, maxbucks, segsperbucks, segsperblock)

The parameters are similar to those of headfilei (ref. 1), but they are all used for return values.

Function: Extracts from the head of an indexed sequential file connected to the zone z, the call values of the original call of headfilei. The zone should be able to hold at least nkey* 10 + 45 double words. Zonestate is 0 after the call.

Errors: The run may be terminated with an alarm, if the parameters z and recdescr cannot hold the return values or if one of the following rare causes coincides:

head i <i>
relative position in the filehead exceeds limits. <i> displays the position.

comp ins <i>

the compare code in the filehead is erroneous. <i> displays the value for an instruction.

gets ins <i>

the getsize code in the filehead is erroneous. <i> displays the value for an instruction.

The three error causes above will normally indicate that the filehead has been violated and it will not be possible to initialize, start or recover the file. Otherwise the cause should be reported as a basic program error.

3.4 Procedure priorreci 3.4

Call: priorreci(z)
z (call and return value, zone)
Specifies the file.

Function: Makes the prior record available. The function is the inverse of that of nextreci, but is more time consuming.

Requirements: zonestate = readonlyi, readnexti, updatei, or puti.

Results: zonestate:= if readnexti then readonlyi else unchanged.
procnoi: 18

resulti:		Available record:
1	Found	The predecessor to the available.
2	Found, start of file	The last in the file.

3.5 Procedure putblocki 3.5

Call: putblocki(z)

z (call and return value, zone). Specifies the file.

Function: The current block, i.e. the block containing the currently available record, is immediately rewritten to the backing storage.

Requirements: zonestate = updatei or puti.

Results:

zonestate: unchanged

procnoi : 15

resulti:		Available record:
1	Done	Unchanged

Note: Further updatings in the current block will be handled in the usual way. See example 1 in section 2.4.

3.6 Procedure putdirecti 3.6

Call: putdirecti(z, record, blockaddr)

z (call and return value, zone). Specifies the file. Should be declared as for full insertion.

record (call value, real array). Holds a record from lexicographical index 1 and on.

blockaddr (call value, integer). Segment no. of a block which holds a record with the same key and length as the one in the former parameter.

Function: The block addressed by blockaddr is read and the elements of 'record' are copied to the matching record in the block.

Requirements: zonestate = updatei or puti.
blockaddr must specify a block used for records.

Results: zonestate: unchanged
procnoi : 16

resulti:		Available record:
1	Done,	Unchanged
2	Not done, the key is not present in the block addressed.	Unchanged
3	Not done, improper length.	Unchanged
4	No buffer, the zone should be declared for full insertion.	Unchanged

3.7 Program recoveri 3.7

Call:

```
                                           1
recoveri <isq file> <recover files>  <doc>
                                           0
                    1            1             1
        <runtype>  <dupspec>  <maxerror>
                    0            0             0
```

<isq file> (area entry). The name of the isq-file.

<recover files>:: = <insert file> <block file>
(area entries). If a pure checking is wanted the two file names may be omitted. If errors a the two files are used for the recovering information.

<doc> (key word parameter)
doc.<doc file>
<doc file> is the name of an area to be used for error listing and run summary.

<runtype> (key word parameter)

$$\text{run.}\left\{\begin{array}{l}\text{check}\\ \text{recover}\\ \underline{\text{all}}\end{array}\right\}$$

If 'check' is specified only the file checking will be run.
If 'recover' is specified only the file recovering will be run, which requires that a 'check'-run has filled <recover files>.
If 'all' is specified both checking and recovering will be run.
run.all is default.

<dupspec> (key word parameter)

$$\text{dup.<duptype>.<dupaddr>}\left\{\begin{array}{l}.\ \underline{\text{asc}}\\ \\ .\ \text{des}\end{array}\right\}_{0}^{1}$$

This parameter is the specification of the winner record in the case of duplicates in the file. See section 2.3.

$$\text{<duptype>:: =}\qquad\left\{\begin{array}{l}\text{half}\\ \text{integer}\\ \text{long}\\ \text{real}\end{array}\right\}$$

<dupaddr>:: = <fieldaddr>

asc means ascending order (default).
des means descending order.
dup.integer.<last +2> is default, <last +2> meaning the word after the last defined key field.

<maxerror> (key word parameter)

max.<unsigned integer>
if the number of erroneous blocks exceeds the specified value, the run stops after the checking.
Default is no stop.

Function: The program holds two phases, the checking and the recovering, which may be called separately. During the checking errors are reported at <doc file> and records for recovering stored at <recover files>. The recovering part will read <recover files> and repair <isq file>. The user may modify <recover files>, see the file formats in app. D.

Requirements: Any trouble caused by parameters will cause a run time alarm. A list is given in app. B.1.

Results: After the recovering <isq file> may be accessed the normal way. If the program breaks during the recovering phase it cannot be guaranteed that <isq file> may be accessed again by this program or the isq-procedures. (Therefore it is recommended to take a backup copy before a complex recovering, see app. F).

After run.all or run.check with errors in <isq file> the fp-bit warning is true. After run.recover the same bit means that some of the records of <insert file> cannot be inserted.

A. REFERENCES

1. RCSL No 31-D 600, June 1980, Inge Borch
RC8000 Indexed Sequential Files.

2. RCSL No 31-D 602, June 1980, Inge Borch
RC8000 SQ-SYSTEM.

3. RCSL No 31-D 562, April 1979, Jørgen Winther
RC8000 Backing Storage Area Sorting.

4. RCSL No 42-i 1278
ALGOL7, User's Manual, Part 2.
If this is not yet available, please use
RCSL No 31-D 322
ALGOL6, User's Manual.

5. RCSL No 31-D 477, January 1978, Tove Ann Aris
RC8000 MONITOR, PART 2, Reference Manual.

B. MESSAGES FROM THE RECOVERY PROGRAM B.

B.1 Run Time Alarm Messages B.1

In the case that a requirement of an isq-procedure is violated, a message is printed at current output and a run time alarm is invoked. A list of the messages is given in ref. 1.

The recover-program uses the algol i/o-System, the SQ-System, mdsortproc, as well as the isq-system itself, and therefore some of the messages listed in ref. 1, 2, 3, and 4 may appear as a run time alarm from this program, but often the alarm is supplied with one of the messages listed below:

freecore <i> the program needs more 'core' to handle the isq-file. <i> is the number of halfwords left for a single share and should be increased to hold as many segments as possible to decrease processing time.

initmark <i> an initmark was found in the isq-file, meaning that an initialization run was not completed and the file cannot be recovered. <i> is irrelevant.

lookup <i> lookup of the isq-file without success. See ref. 4 and 5, monitor function 42. <i> is the monitor result. (<i> = 3 means file does not exist).

progcall <i> error in the program call at parameter no. <i>.

sortdisc <i> too few backing storage claims for sorting of recover files. <i> <0 means lacking entries, <i> > 0 means lacking segments.

sortout <i> the backing storage device for the recoverfiles does not exist or the user has too few claims on it. <i> = -1 means does not exist, otherwise the segment claim.

sortsize <i> the program needs more core to sort the recoverfiles. <i> is the extra core claim in halfwords.

The three messages above are preceded by an indication of which recover file is to be sorted. The insertfile is sorted primarily.

B.2 Warning Messages B.2

If errors are found by the recover-program, the messages below are printed at current output,

```
***isq-check warning: <i> errors,
```

or

```
***isq-recover warning: <i> errors.
```

The first means that <i> blocks or tables in the isq-file are interpreted as being erroneous, and the second that <i> recover-records are rejected in the recovery-phase. The types of errors are explained in the documentation file.

C. ISQ-SYSTEM RECOVERY DOCUMENTATION C.

The recovery documentation is stored in the file named at the keywordparameter 'doc', see sec. 3.7.

The types of recovery informations are identified by a unique number, which is printed in the beginning of each information line prefixed by two asterixes, e.g. **09. After the number follows an explaning text and depending on the type, some leading texts and data from the isq-file that may help the user to get a survey of the needed recovery actions.

The recovery informations are listed in the orders in which they are discovered, which may differ from the sequential scanning of the file, and from the key-order.

Below the possible types of recovery informations are listed and commented. The underlining of some terms means that the quantity is printed. Key fields are printed with the algol standard layout for their types.

This appendix is enclosed by an example.

**01 RECORDLENGTH

The recordblock starting at segm.no has a record at field-address which has an illegal length. The id. of the first record of the block is printed if possible. The type may origin from some rubbish in the block so length, record, and first rec. may not be informative.

The block will be scratched.

**02 CLEARED BLOCK

Not used.

**03 KEY SEQUENCE

The recordblock starting at segmno has a record at field-address which is out of sequence order. The id. of the

first record of the block is printed.
The block will be scratched.

**04 OVERLAP BLOCK

The recordblock starting at segmno has records with keys that overlap the keys of another block in the same bucket. The id. of the first rec and the last rec are printed. The block will be scratched and the records written at the insertfile.

**05 NEW BLOCKTABLE

Because of errors in the recordblocks of the bucket starting at segm.no, a new blocktable will be created and written.

**06 OVERLAP BUCKET

The bucket starting at segmno holds records with keys which overlap the keys of another bucket. The id. of the first rec and the lastrec in the bucket are printed.
All the blocks in the bucket and the blocktable at segmno will be scratched and the records written at insertfile.

**07 NEW BUCKETTABLE

Because of errors which cannot be repaired within bucketlimits, a new bucket table will be created and written.

**08 BUCKETHEAD

The file needs correction for internal purposes. A new buckettable will be written.

**09 BLOCKTAB.ERROR

The recordblocks of the bucket starting at segmno are accepted, but the corresponding blocktable is not correct. The key of the firstrec in the bucket is printed.
The blocktable will be corrected.

**10 BUCKET DELETE

A whole bucket starting at segmno is lost because of errors in the record blocks.

**11 BUCK.TAB. ERROR

All the buckets of the file are accepted, but the buckettable is not correct.
The buckettable will be corrected.

**12 FILE DELETE

The file is so damaged that no record is accepted. After revocering the file will contain only records from the insert file.

**13 FILE DESCRIPT

The number of records or of halfwords used for records stored in the file differs from the number counted during the checking. The deviation is printed (number counted-number stored). The number in the file does not need to be exact for processing of the file.
This recovery information type may include the text 'update mark found', in which case the file must be recovered before updating by the file-i procedures.

**14 INSERT RESULT

During recovering of the file records may be dismissed by the file-i procedure insertreci, see ref.1. The reasons are indicated in clear text and the key of the record is printed.

An example of recovery documentation is shown next page.

```
ISQ-SYSTEM RECOVERY DOCUMENTATION 79 03 02   FILE: ISQFILE          PAGE 1

**04 OVERLAP BLOCK   SEGM NO:   11 FIRST REC : 110010 LAST REC :  190020

**04 OVERLAP BLOCK   SEGM NO:    7 FIRST REC : 180000 LAST REC :  180000

**01 REC  LENGTH     SEGM NO:   19 FIELD: 504 LENGTH: 0
                     RECORD:  0 FIRST REC :  201230

**01 REC  LENGTH     SEGM NO:   20 FIELD: 16 LENGTH: -1
                     RECORD:  -1 FIRST REC :  190430

**03 KEY SEQUENCE    SEGM NO:   21 FIELD: 16 RECORD:  210421
                     FIRST REC :  210420

**03 KEY SEQUENCE    SEGM NO:   22 FIELD: 16 RECORD:  221230
                     FIRST REC :  221210

**05 NEW BLOCKTABLE  SEGM NO:   18

**04 OVERLAP BLOCK   SEGM NO:   25 FIRST REC :  250011 LAST REC :  250422

**04 OVERLAP BLOCK   SEGM NO:   26 FIRST REC :  250420 LAST REC :  261250

**09 BLOCKTAB  ERROR SEGM NO:   24 FIRST REC :  250011

**09 BLOCKTAB  ERROR SEGM NO:   30 FIRST REC :  300000

**07 NEW BUCKETTABLE SEGM NO:    1
```

E. Zonestate Legality and Changing E.

procedure \ zonestate	ref.	4 after declaration	0 after open	10 readonlyi	11 readnexti	12 puti	13 updatei	14 initializei
close	4		4	4	4			
deletereci	1					12	13	
extendi	–					12	13	14
getparamsi	1			10	11	12	13	14
getreci	1			10	11	12	13	
headfilei	1		0					
headparamsi	–		0					
initfilei	1		14					
initreci	1							14
insertreci	1					12	13	
nextreci	1			11	11	12	13	
open	4	0						
priorreci	–			10	10	12	13	
putblocki	–					12	13	
putdirecti	–					12	13	
putreci	1					12	13	
setparamsi	1			10	11	12	13	14
setputi	1			12	12	12	12	12
setreadi	1			10	10	10	10	10
settesti	1			10	11	12	13	14
setupdatei	1			13	13	13	13	13
startfilei	1		10					

(non filled field means illegal relation)

F. Example of a Recovery Run F.

This example shows a job with a separate checking and a backup before the recovery:

```
recoveri isqfile recover insert doc.pap run.check
; check if parameter problems:
if ok.no
( message stop caused by recoveri call
finis
)
; check if isqfile ok
if warning.yes
( convert pap
  message recoveri dok converted
  ; take backup
  claimtest perm.discn.1000.1
  if ok.no
  finis
  oldisq = entry isqfile discn isqfile isqfile isqfile isqfile isqfile
  scope user oldisq
  oldisq = move isqfile
  recoveri isqfile recover insert doc.pap run.recover
  if warning.yes
  convert pap
)
```

After a recovery run it may happen that the quantities 'recbytes' and 'noofrecs', ref. 1 sec. 1.3, are not exact, but this has no consequences for the further processing. They may be repaired by an extra recovery, e.g.:

```
  recoveri isqfile a b doc.pap
  if warning.yes
    convert pap
```

RETURN LETTER

Title: Extensions to RC8000 Indexed Sequential Files System (ISQ) RCSL No.: 31-D601

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

. . . . Fold here . . . .

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

Affix
postage
here

A/S REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark