Raabo.

A/S REGNECENTRALEN
SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

SYSTEM LIBRARY

RCSL NO:	55-D22
TYPE :	Report
AUTHOR :	Peter Kraft
EDITION:	June 1969

RC 4000
EDITOR I

Content

The conventions for the editor has been designed by Donald Wagner and the author.

Peter Kraft
A/S Regnecentralen, June 1969

Introduction

The editor is a utility program of the RC 4000 system. It is designed for correcting text documents such as source programs or data files.

The editor can be used both on-line and off-line. In on-line mode the user is sitting at his console typing commands and receiving typed messages. In off-line mode the editing is performed as a job in a batch, whith all the corrections prepared beforehand.

The editing is performed as follows: The user specifies a 'source document', i.e. the device or the backing store area where the original text is to be found, and an 'object document', i.e. the device or the backing store area where the finished edited text is to be placed. The editor, in response to commands from the user, copies lines from the source document to the object document, modifying the text as it copies. A pointer keeps track of the number of lines copied to the object document.

The user may also give commands which move the pointer backward. This is performed by copying the remaining part of the source document to the object document; turning the object document into the new source document; assigning a new backing storage area as the object document; and copying lines until the specified line is reached.

To optimize the back spacing time a core store buffer will admit backspacing of a limited number of lines without activating the above described turning mechanism. If the object document is not a work area on the backing store, backspacing is only allowed for this limited number of lines.

There are several ways a user may specify a line in the text. He may specify the first or the last line in the text; or he may specify the line which is some number of lines before or after the current position of the pointer; or he may specify the next line in the text after the pointer which contains a certain string of characters.

The set of commands includes, beside the command for moving the line pointer, commands for printing of a number of lines in the vicinity of the pointer, for deletion and insertion of lines, and for replacing a text string within a line with another text string. The latter can be done repeatedly over a number of lines.

Some auxiliary commands make it possible to handle non-printable characters, and others make typing of commands easier.

The user normally types commands one-per-line, so that he must wait after each command for a response from the editor. Commands may also be typed in sequences which the editor stores and executes with no intervention from the user.

After execution of a command sequence the editor can print out the line at which the pointer currently points as a verification of what was done. If this becomes inconvenient the user may turn off the verification.

The editor interprets the characters as they are defined by the ISO-alphabet. The backspace character and the carriage return character will cause that the graphical picture of a line does not correspond to a unique sequence of input characters. The editor compensates for this by rearranging the characters of a line to a unique representation of its graphical picture.

An example

Source text:	Object text:
In Boston, the wheater	In Boston, the weather
is often better than	is seldom worse than
the radio forecast	the Weltanschauung.
to us.	

Console communication: (lines marked with equal sign (=) are typed by the user).

```
  edit begin
= line 1
  is often better than
= replace c often better c seldom worse c
  is seldom worse than
= line 1
  the radio forecast
= delete 1
  3 line, end document
= insert c
= the Weltanschauung
= c
  the Weltanschauung
= line top
  In Boston, the weather
= print bottom
  In Boston, the weather
  is seldom worse than
  the Weltanschauung
      4 line, end document
= finish
  edit end
```

Using a short form for the command names and sequencing the commands the console communication would look like:

```
  edit begin
= l1, r c often better c seldom worse c,
= l1, d1, i c
= the Weltanschauung
= c, f
  edit end
```

1. Call of the editor and the use of documents

The editor is executed as a utility program under the file processor. It is called by the file processor command:

```
     <object document> = edit <parameter list>
or   edit <parameter list>
```

where:

```
<object document> ::= <catalog name>|<note>
<parameter> ::= <source document>|<object area size>|
                <correction area size>
<source document> ::= <catalog name>|<note>
<object area size> ::= o. <integer>
<correction area size> ::= c. <integer>
<parameter list> ::= <empty>|<spaces><parameter>
                     <parameter list><spaces><parameter>
```

Object document:

When the object document is described in an empty note, a working storage area of size 2 k words is created, and the description of the area is transferred to the note. If the area is not large enough a new area 2 k words greater is created and so on until the area is large enough. When exit is made from the editor the working storage area is decreased to the necessary size.

If no object document is specified the object text ends up in the empty air.

If the initial size of the working storage area of 2 k words is not suitable the user may specify the object area size parameter. This parameter gives size in tracks. [1track = 4 segm. = 1024 ord].

When a working storage area is used for the object document, backspacing is unlimited.

An object document specified in a catalog name can be on any kind of text output medium. Backspacing will only be alowed for a limited number of lines.

Source documents:

In the parameter list the user may specify several source documents either by catalog names or by notes. These documents are numbered from left to right starting with one. Using the source-command (see chapter 2.10), the user change from one of these source documents to another.

Initial current source will be source number one. If no source document is specified it will be empty. [Ex.: skrivning af nyt program fra consol].

Editing commands:

The editor reads commands from current input and produces messages on current output. When current input and current output are on the same typewriter the editor is said to be in on-line mode with direct communication between the user and the editor. Otherwise the editor is in off-line mode.

The main difference between the modes is the reaction to command errors. In on-line mode editing can continue, while in off-line mode the editor makes an error return to the file processor.

Device errors are handled by the file processor in the standard way. Hard errors will thus always cause error return to the file processor.

Correction area:

Commands are stored up before they are executed. When the commands are sequenced they may take up some space. Also the insert-command by means of which lines are inserted may require some space. For this reason a working backing storage area of 4 k words is reserved during the editing. If this area is not great enough a greater area can be specified by the correction area size parameter in number of tracks.

2. Definition of commands

The editing is controlled by means of commands, some of which provide aids for typing of the commands and for checking that the correction has been executed as intended. The rest of the commands perform the actual editing.

```
<auxiliary command> ::=
     mark <mark specification>|
     verify <yes or no>|
     where

<editing command> ::=
     line <position>|
     print <position>|
     delete <position>|
     insert <delim><NL><text string><delim>
     replace <delim><id-string><delim><text string><delim>|
     global <number position>|
            <delim><id-string><delim><text string><delim>|
     source <integer>|
     finishes

<on-line command> ::=
     correct <NL><columnar corrections>
```

The commands can be stored in a sequence. When this sequence of commands is terminated they can be executed in the given order. A syntax error will cause that no commands of the sequence is executed.

```
     <terminator> ::= <NL>
     <separator> ::= , | , <NL>

<sequenceable commands> ::=
     <auxilary command>|<editing command>

<command sequence> ::=
     <sequenceable commands><terminator>|
     <on-line command><terminator>|
     <sequenceable commands><separator><command sequence>
```

Note that the correct command, which is intended only for on-line use, must always be the last in a sequence of commands.

For definition of <position>, <number position> <delim>, <id-string>, and <text string> see page 15

Auxiliary commands:

The auxiliary commands provide aids for the operator during typing of the commands and for controling the progress of the correction.

2.1. Mark-command:

Some characters can be selected to have special meanings when they are used in the commands.

```
<mark specification> ::=
     numeric <simple graphic>|
     character <simple graphic>|
     line <simple graphic>|
     standard|
     empty
```

Numeric delimiter mark:

This mark is used for specifying a character by its numerical code. It is intended mainly for use when non-graphical characters must be used. A numerically specified character is an integer between 0 and 127 surrounded by the numeric delimiter mark. They are significant only in the id-strings and in the text strings of the commands and will if used otherwise cause a syntax error.

Character-replace-mark:

This mark can be used to correct mistyping of one or more characters in a command line without changing the line format (see page 11, preparation of lines).

Line-erase-mark:

This mark is used for erasing the characters already typed on a line. (See page 11, preparation of lines).

Standard marks:

By the standard option the characters æ, ø and å are selected as numerical delimiter mark, character replace mark, and line erase mark respectively.

Empty option:

All selected marks are canceled.

Initially mark characters are standard.

Examples:

Mark command:	Typing	Content of line:
mark num æ	aæ32ææ12æb	a<SP><FF>b
mark char ø	agc<BS><BS>ø<BS>$\overset{c}{g}$<SP>d	abcd
mark line å	p b c å ↑back space.	a b c
	a b c	

2.2. Verify-command:

After execution of a command sequence the value of the line given by the line pointer can be printed as a message thus verifying the position in the text.

The user specifies whether he wants verification or not using this command.

<yes or no> ::= yes|no

Initially verification corresponds to: verify yes.

2.3. Where-command:

By this command the operator can ask at what line the linepointer is pointing. The message:

<integer> line.

is printed back.

Executive commands:

The remaining commands perform the actual corrections of the source string.

2.4. Line-command:

This command moves the line pointer to the specified position in the text. The next command starts its execution from the beginning of the thus selected line. The position can specify the top or the bottom of the text, a line an integer number of lines away, or a line forward in the text identified by a substring.

2.5. Print-command:

The command facilitates a way to look upon the text under correction, by printing a selected number of lines as a message.

The lines selected for printing is the line determined by the position and the line, given by the line pointer, and all lines in between. The line pointer ends up by pointing at the beginning of the last printed line.

2.6. Delete-command:

The delete command works almost as the print command, only the lines are deleted from the text in stead of printed.

After deletion the line pointer points at the beginning of the line following the last deleted line.

2.7. Insert-command:

The text string is inserted in the source text at the beginning of the line given by the line-pointer. The text string can be any number of lines.

The line pointer ends by pointing at the last inserted line.

Note: If the last inserted line is not terminated with a <NL>* the characters of the last inserted line are added directly to the line in the text following just after the insertion.

* also at the beg-! (s.6).

2.8. Replace-command:

The replace command works only on the current line. The first appearance of the character sequence specified in the id-string is replaced by the characters of the text string. The id-string must be a substring within the current line. The inserted text can be any number of lines.

The line pointer ends by pointing at the beginning of the line which contain the last characters of the original line.

2.9. Global-command:

The command is a repetitive version of the replace command. It replaces each occurrence of the id-string with the text string in a number of lines. The number position can identify either the top or the bottom or a line relative to the current line. The replacement is performed on the identified line, on the current line and on all lines in between. The pointer ends by pointing at the beginning of the last line worked upon. ← (!)

↳ works forward (s.17).

2.10. Source-command:

This command is used where a text has to be composed of parts from different source documents. The source documents must be specified as parameters to the call of the editor. Corresponding to the integer in the source-command a new source document is selected from the parameter list.

h

2.11. Finis-command:

The remaining of the source text is copied unto the object document. The object document is terminated and exit is made from the editor.

2.12. Correct-command:

This command is intended for use in on-line mode only.

The current line is printed on the console terminated with a <NL>. It is then simulated, that this line did not originate from the source text, but instead was under preparation on the console. The last character is not interpreted as a <NL> but as a <CR> which allows the operator to continue the preparation, e.g. by making columnar corrections to the line by means of the character replace mark. When the operator types <NL> the line is copied back to the source text.

The line pointer remains pointing at the beginning of the line.

3. Shorthands for command names

The command names and the names used in the mark and verify commands are identified by their first letter only. Each name can consist of any number of letters.

```
<command name> ::= <identifying letter>|<command name><letter>
```

Example:

line	l	linie
no	n	nej

4. Preparation of lines

The editor interprets the characters in accordance with the ISO alphabet. In the appendix is given a table specifying the correspondance between the numeric value and the ISO characters. From the editors point of view the characters are classified as simple graphic, non-graphic, skipped, or special.

The need for comparing substrings given in the commands with the source text in a unique manner makes it convenient to preanalyse both the command input and the source input line by line and collect the characters which occupy the same graphical position.

After the characters have been rearranged a simple line can be defined as follows:

```
<composed graphic> ::= <simple graphic><BS><simple graphic>|
                       <composed graphic><BS><simple graphic>
<graphic> ::= <simple graphic>|<composed graphic>
<positional graphic> ::= <graphic>|<SP>
<positional character> ::= <positional graphic>|<non graphic>
<simple line> ::= <NL>|<EM>|<positional character><simple line>
```

The rules for preparation of a line can be given as follows:

a. The characters are sequenced in the order which the graphical picture dictates.

b. <SP> are removed from composed graphics.

c. In a composed graphic the same simple graphic can be stated only once. Duplicated characters are removed.

d. Non graphics cannot be mixed within the composed graphic; they will instead preceed the graphic.

e. Superfluous <BS> in the beginning or in the end of a simple line are removed.

f. <CR> is treated as backspacing up to the beginning of the line.

g. In composed graphics containing underline or bar the underline and then the bar will preceed the remainder of the composition.

h. A simple line is terminated either by <NL> or <EM>

Examples: (< symbolizes the <BS> and ␣ <SP>)

```
graphic picture:     begin a1
input sequence:      begin␣<<<<<<_____␣a1<NL>
simple line:         _<b_<e_<g_<i_<n␣a1<NL>

graphic picture:     ƀa
input sequence:      <<a<b<|<ba<<<NL>
simple line:         |<a<ba<NL>

graphic picture:     de
input sequence:      ab< <HT><de <EM>
simple line:         a<d<HT>b<e<EM>
```

graphic picture:	a:end b
input sequence:	␣␣end <CR> a:___␣b<NL>
simple line	a:_<e_<n_<d␣b<NL>

Line erase mark:

The line erase mark is active during preparation of lines input from the command document. When a line erase mark is input, all characters collected for the simple line are removed.

Example with line erase mark equal to å:

input sequence:	simple line:
a letter å <NL> a letter was <NL> }	a letter was <NL>

Character replace mark:

The character replace mark is active during preparation of line input from the command document.

When the mark is met in the input sequence it replaces the graphic on its position with a space. Afterwards a new graphic can be given for that position.

Example with character replace mark equal to ø
(< symbolizes <BS> and ␣ <SP>).

graphic picture:	a løtter was
input sequence:	a lat<<ø<e␣ter was <NL>
simple line:	a letter was <NL>

graphic picture:	than
input sequence:	than<<<<____<<ø<e<_<NL>
simple line	_<t_<h_<e_<n <NL>

Internal line representation and limitation:

The internal representation of the characters of a line does not contain the backspace character explicitly but as a mark on the character preceeding the back space.

Simple lines are processed in buffers of limited length. Lines longer than the maximum line length are terminated at the maximum length without error action, and the following characters will be processed as being the next simple line. Backspace or carriage return which are input after the max length has been exceeded may be misleading.

A line which is going to exceed the maximum line length while a composed character is inserted will cause the alarm: line too long.

Standard maximum line length corresponds to 126 characters, not counting backspaces and carriage returns.

5. Line counting

The editor keeps track of the lines counted for the produced object string. Only lines containing at least one graphic are counted. This give raise to a special line concept entirely independent of the simple lines prepared on input.

```
<empty set> ::= <empty>|<SP>|<non graphic>|
                <empty set><empty set>
<proper set> ::= <empty set><graphic>|<graphic><empty set>|
                 <proper set><proper set>
<empty lines> ::= <empty set><NL>|<empty lines><empty lines>
<proper line> ::= <proper set><NL>
<line> ::= <proper line>|<empty lines><proper line>
```

The editor contains a line pointer, which before execution of a command will point at the beginning of the line to be worked upon. If the line contains empty lines it will be on the beginning of its first empty line.

Note on correct command: If the line to be corrected contains empty lines they will be removed even if they are printed.

The lines are counted consecutively, the first line being line number one.

When the last line containing the <EM> is to be ~~printed~~ output the <EM> will cause printing of the message:

<line number> line, end document.

Examples on line counting: The line number is shown in parenthesis.

```
Ex 1:  (1)  abc <NL>            Ex 2:  (1)  <NL>
       (2)  <NL>                            abc
            pqr <NL>                   (2)  <NL>
       (3)  <HT><SP><NL>                    def <EM>
            abc <NL>
       (4)  <EM>
```

printing at last line:

```
      <NL>
        4 line, end document         2 line, end document
```

6. Definition of delim, position, id-string, and text string

<delim> ::= <graphic>

<number position> ::= <empty>|<integer>|top|bottom
<context position> ::= .<delim><id-string><delim>
<position> ::= <number position>|<context position>

<id-string> ::= <any string not containing current delim>

<text string> ::= <any string not containing current delim>

String delimitor:

Strings in commands are delimited by a graphic. may be composed... The value of the delimitor is determined by its first occurrence in the context.

Position:

An empty position or the integer zero selects the current line. A positive integer selects a line the integer number of lines forward in the text. Similar a negative integer identifies a line the integer number of lines backward in the text.

The position top selects the first line, i.e. line 1, and bottom the line following the last line, i.e. the position before the <EM>.

By the context position a line is identified by a substring. The position determined will be at the beginning of the (last)(jvf. næste side) identified line.

Note: In the global command a position can not be specified as a context position. A possible point will be taken as the string delimiter.

If a position can not be found, an alarm is given: position not found.

Strings:

Two kinds of strings are distinguished, the id-string which is used to identify a substring in the source text, and the text string which is inserted in the source text.

In both kinds of string a character can be specified by its numeric value.

7. Matching with id-string

The source text is searched forward until a sequence of characters which matches the id-string is found. The matching adheres to the following rules:

a. A composed graphic matches a composed graphic if they are composed by the same simple graphics. The matching is independent of the order in which the simple graphics of the composed graphic originate.

b The characters <SP>, <NL> and non-graphic characters are blind for identification, i.e. they are skipped by the matching procedure when met in the source string.

(c) The <SP>, <NL> and non-graphic characters will when specified in the id-string take part in the matching.

(d) <CR> and <BS> should not be specified in the id-string by numeric values as the source string will never contain this characters explicitly.

Examples:

id-string	source text	match
abc	a b c	yes
a bc	abc	no
a bc	a b c	yes

When an id-string is used to identify a line position it may consist of any number of lines. When the match is found, the position pointer points at the beginning of the last identified line.

Examples: (Line number are shown in parenthesis.)

source text:		commands	resulting pointer
(1)	aaa	line. - q -	(4)
(2)	abc	1 t , 1.-a	
(3)		a-	(2)
	ab c	1 t, 1.-	
(4)	p q r		
(5)	<EM>	a-	(3)
		1 t, 1.-b-	(3) ← (2)!
		1 t, 1.-h-	(5) with an alarm

In the replace command and the global command the id-string is used to identify those characters which have to be replaced. The string which has to be replaced must be within one line, i.e. the character <NL> can be used only in connection with empty lines.

Examples:

```
Command:                 Before replace:          After replace

replace -q-a-            pqr                      par
replace -qr-ab-          q  q  r                  p  ab
replace -                <NL>                     <NL>
--                       <NL>                     p q r
                         p q r
replace -p -p-           p  q r                   p q r
replace - q-q-           p  q r                   pq r
              +↑↓
```

In the last example space after p do match, the second space is blind for identification and q match. As the blind space is surrounded by matching characters it is removed as well.

8. Messages and alarms.

Messages and alarms are printed on current output document. The message:

edit begin

is printed when the editor is loaded successful and prepared for input of commands.

Before leaving the editor the message:

edit end

is printed.

The alarms fall into three groups. (1) Initial alarms which are caused during loading of the editor and after which no editing can be performed. (2) Alarms concerning communication with peripheral devices, and (3) alarms caused by erroneous commands.

1. Initial alarms:

xxxedit end: no core.

The size of core store is not large enough.

xxxedit end: param.

The parameters to the call of the editor are not syntactically correct.

xxxedit end: connect object.

The object document cannot be connected by the file processor. If a working area should be created for the document the alarm may indicate that there is no room on the backing store.

xxxedit end: work area.

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

2. Alarms concerning communication with peripheral devices:

xxxedit <command no.> connect source.

The source document can not be connected by the file processer. Note: When the source-command is used to select a source outside the sources given in the parameter list, the source document is defined as empty.

xxxedit <command no.> work area.

When the object document is assigned to an empty note a work area on the backing storage is created. The area may be copied into another area either during backspacing or during extension of the area. The alarm 'work area' tells that there is no room for these operations on the backing store.

xxxedit <command no.> character.

A character with a code greater than 127 has been input either from the source document or the command document.

Other errors in connection with the transfer of characters and blocks are handled by the file processer and treated as hard errors.

3. Alarm caused by erroneous commands:

xxxedit <command no.> syntax.

A syntax error in the command format is found. None of the commands in a command sequence are executed.

xxxedit <command no.> position not found.

A line position cannot be found, or no match with the id-string in the replace-command can be obtained.

xxxedit <command no.> backspace error.

If random access to the text is not allowed, i.e. when the object document is not specified by an empty note, backspacing is only allowed a limited number of lines. The alarm is given when backspacing is attempted beyond this number of lines.

xxxedit <command no.> line too long.

In preparation of a simple input line both from the source document and the command document, composed characters may cause that the maximum line length is exceeded. The input line will be lost.

Appendix: ISO alphabet

The editor handles any 7-bit character as interpreted by the ISO-alphabet. The characters are characterised by the editor as follows:

	numeric value	ISO	kind	numeric value	ISO	kind
	0	<NUL>	skipped	32	<SP>	blind
	1		non-graphic	33		simple-graphic
	2		-	34		-
	3		-	35		-
	4		-	36		-
	5		-	37		-
	6		-	38		-
	7		-	39		-
back space	8	<BS>	special	40	(	-
	9	<HT>	non-graphic	41	)	-
newline	10	<NL>	special, blind	42	×	-
	11	<VT>	non-graphic	43	+	-
form feed	12	<FF>	-	44	,	-
car. ret.	13	<CR>	special	45	-	-
	14		non-graphic	46	.	-
	15		-	47	/	-
	16		-	48	0	-
	17		-	49	1	-
	18		-	50	2	-
	19		-	51	3	-
	20		-	52	4	-
	21		-	53	5	-
	22		-	54	6	-
	23		-	55	7	-
	24		-	56	8	-
end mark	25	<EM>	special	57	9	-
	26		non-graphic	58	:	-
	27		-	59	;	-
	28		-	60	<	-
	29		-	61	=	-
	30		-	62	>	-
	31		-	63		-

numeric value	ISO	kind	numeric value	ISO	kind
64		simple-graphic	96		simple-graphic
65	A	-	97	a	-
66	B	-	98	b	-
67	C	-	99	c	-
68	D	-	100	d	-
69	E	-	101	e	-
70	F	-	102	f	-
71	G	-	103	g	-
72	H	-	104	h	-
73	I	-	105	i	-
74	J	-	106	j	-
75	K	-	107	k	-
76	L	-	108	l	-
77	M	-	109	m	-
78	N	-	110	n	-
79	O	-	111	o	-
80	P	-	112	p	-
81	Q	-	113	q	-
82	R	-	114	r	-
83	S	-	115	s	-
84	T	-	116	t	-
85	U	-	117	u	-
86	V	-	118	v	-
87	W	-	119	w	-
88	X	-	120	x	-
89	Y	-	121	y	-
90	Z	-	122	z	-
91	Æ	-	123	æ	-
92	Ø	-	124	ø	-
93	Å	-	125	å	-
94	∧	-	126	-	-
95	-	-	127	<DEL>	skipped

Appendix 2. SLANG-transformation of the editor.

The editor is programmed in SLANG. On text form it is saved on four rolles of paper tape in the ISO 7-bit character code.

SLANG-translation shall be performed together with the h-names of the file processes. During translation of the first tape some parameters for the editor can be redefined. The parameters have d-names and are used as follows:

```
d1  = 123 ; standard num mark = æ
d2  = 125 ; standard line mark = å
d3  = 124 ; standard char mark = ø
d15 =   1 ; mark is standard (1 = true, -1 = false)

d6  =   1 ; first line number in the line number
          ; counting.
d4  = 128 ; max line length in bytes.
d5  = 100 ; max line back up.

d7  =   8 ; initial object work area (no of tracks)
d8  =   8 ; increment object work area (no of tracks)
d9  =   8 ; correction work area (no of tracks)
```