T A N D B E R G

TDV-2100 series.

Buffered Editor

Specifications

TANDBERGS RADIOFABRIKK A/S
Data div.
P.O. Box 9, Korsvoll
Oslo 8, NORWAY
Tel.: (47-2) 23 20 80
Telex: 16441 tanra n

Part no.: 367594
Publ. no.: 5003
September 1978

CONTENTS

1. Introduction

The buffered editor for use with the TOS21 operating system, has taken into consideration that the display screen is an integrated part of the TDV-2114.

The screen is used as a window of the buffer containing the text lines.

The text lines may be altered directly when shown on the screen, or by a well defined set of text handling functions.

Different search and roll functions enables the user easily to find actual information in the text buffer.

2. Accessing the buffered editor

The buffered editor is invoked by the command

```
EDIT si=<input file>,so=<output file>,sl=<list file>,
     ai=<auxiliary input file>
```

The command is typed by the user on the console keyboard. The editor copies information from the input file to the output file under control of commands from CI, the console input.

The list file is identical to the output file, but in front of each line, the new line number is given.

If no input file is to be edited, the assignment of the standard input (SI) should be omitted.

Examples:

```
EDIT so=abc
EDIT si=abc,so=abcd,sl=abcd.l
EDIT si=abc,so=de,ai=aux
```

Requirements:

- Minimum amount of memory is 32K of RAM, but 48K is recommended to increase the capacity of the editor.

3. Text addressing

All text currently in the text buffer can be addressed directly Therefore it is an advantage to have as large buffer area as possible.

With 32k of memory, the buffer may contain approximate 50 lines of length 80 characters. With 48k the amount of lines are approximate 250.

Shorter lines causes the amount of lines in memory to increase.

3.1 Line address

The contents of the text buffer is divided into two categories,

- the text lines read from the standard input file
- the new or altered text lines.

More restricted addressing rules applies on the latter category than the first one. See below.

There are five different ways to address a line:

1 - line number (decimal number)
2 - text string
3 - current pointer ($ key)
4 - first line (: key)
5 - last line (. key)

To each line address a signed decimal constant may be added, i.e. $+5 or .-10 .

3.1.1 Line number

Each of the lines read from the standard input file can be addressed directly by its line number. The user only has to type the decimal number.

New or altered lines can not be addressed in this way.

3.1.2 Text string

A line can be directly addressed by a text string occurring in the line. The string must be typed enclosed by the character ".

A text string may contain one or more VERTICAL BAR characters. Characters in the position specified by the VERTICAL BAR characters are not tested.

VERTICAL BAR i.e. 7EH. On ECMA keyboards the symbol is represented by the ¬ character.

3.1.3 Current pointer

The editor has two current pointers,

- a line pointer
- a character position pointer.

The values of these pointers are dependend on the editing function performed.

These pointers are later referred to as the current pointer.

The line pointer can be addressed by typing the $ (dollar) key.

The character pointer is updated either by the user, by altering the position of the cursor, or by the different commands or functions.

3.1.4 First line

The first line in the text buffer can be addressed by typing the : (colon) key.

3.1.5 Last line

The last line in the text can be addressed by typing the character . (dot) .

3.2 Text interval

A text interval is an interval of one or more consecutive lines. The first and the last line of the interval must be specified, separated by comma. If the interval contains one single line, only this line must be specified.

Later in this manual a text interval will be referred to as I.

I = L

or I = Li,Lj,(if j < i then j = i)

If Li is blank then Li = first line
if Lj is blank then Lj = last line

The last specified text interval may be referred to by a '&'.

4. Screen layout

The screen is divided into 2 parts:

a) line 1 - command and message line
b) line 2-25 - data part of the screen

a) command and message line

column 1	contains an asterisk when the editor is ready to accept a new command or function
column 2 -72	command/function field
column 20-72	will contain (error) messages from the editor
column 73-77	gives current line number or "NEW"
column 78	separator "/"
column 79-80	gives current character position number (1-80)

The message field will be cleared when the HOME key is pressed.

On termination of a command, the cursor will either be located at the character pointed out by the current pointer or remain in the command field. This is dependend on the last command executed (see section 6 and 7).

When an error situation is detected, a relevant message is displayed. A new command should then be specified after having pushed the HOME key.

b) The screen will be used in two different ways, depending on the last executed command:

- as a window showing 24 lines of the text buffer (23 lines if automatic roll mode).
- as a display of lines containing certain information

In the first case, initially or when certain commands are executed, line 13 on the screen always will contain the line located by the current pointer. The cursor indicates the current character on this particular line.

The contents of the screen (line 2-25) may be altered, either by using the control function keys combined with the alpha-numeric keys or by pressing the HOME key followed by a new command.

In the latter case, the prompting asterisk is given and the previous specified command may be excuted once again, just by pressing the CR button.

If the first character entered is an alpha-numeric key, the command field is cleared and the character entered is displayed.

5. Operating the buffered editor

5.1 General

The text editor signals its readiness to accept commands by prompting with an asterisk (*) in first position of the first line.

Three types of commands may be given when operating the editor:

- general commands
- text handling functions
- control function keys

When specifying a command, a block cursor identifies the next available position of the command field.

Commands may be entered in upper or lower case lettes.

Commands may be abbreviated as specified in each individual description. This is indicated by enclosing optional characters in brackets.

Commands may be entered singly or in strings. Commands in a string must be separated by a semicolon (;).

Commands in a command string are executed in the sequence they are entered.

If one of the commands in a string could not be performed or gives an erroneous result, rest of the string is skipped.

Carriage return (CR) indicates that the specified command or command string should be executed. If the command string is empty, the editor returns to edit mode.

When a command is finished, the cursor will either be positioned in the command field of line one (command mode), or in the 13th line of the screen (edit mode). This is dependend on the last command executed.

If edit mode, the position of the cursor may be altered by using the control function keys (section 8).

A new command can only be given after the HOME key is pressed. This causes the last command string specified to be displayed. The block-cursor is located at the end of the command.

If the same command string is to be executed once again, the CR may be pressed.

If a new command is to be given, the command may be entered directly. The first printable character causes the command field to be cleared.

The editor allows the record length of the Standard Input file (SI) to be more than 80 characters long. Only the first 80 characters of a line will be shown on the screen and acted upon.
The maximum length is 128.

5.2 Deleting typographic errors (LEFT CURSOR, DEL)

Any typographic errors in entering a command can be removed by pressing the '<-' LEFT CURSOR key once for each character to be removed. (If the cursor is located in the first position of a line, the '<-' will have no effect).

Pressing the DEL key causes the characters in the command field to be deleted. The editor will again prompt an asterisk.

5.3 Interrupting commands (!).

After the command terminator is entered and the commands are being executed, the operation can be terminated by pressing the ! key.

The interrupt is acted on only during a time consuming operation, i.e. input/output operation, others are executed completely.

If a function string is specified, rest of the string is skipped.

5.4 Defining function iteration

A function or function string can be repeated any number of times by enclosing the string in angle brackets '<' and '>', preceeded by a decimal number, specifying how many times the iteration is to be performed.

The syntax of the iteration is:

n <'function or function string'>

Where n has to be in the range 1-255.

6. Commands

This section describes the commands for performing input/ output operations and tabulator handling.

The characters [ ] is used to identify optional information.

Examples:

E[XIT]	- may be written as: E, EX , EXI or EXIT
AI [n] [*L]	- may be written as: AI AI * L AI n AI n * L

6.1 Tabstop handling

The editor may read files containing tabulator characters (09H and 18H) and produce output containing the tabulator character 09H.

On input the lines will be expanded according to the current tabulator setting.

On output, if the TP command (below) has been given, the lines are packed according to current tabsetting.

NB! It is the responsibility of the user that the tabsetting used on output and later on input corresponds.

6.1.1 Setting tabstops

The command for setting tabstops is divided into 4 sub-commands:

Command:	T[n,m,....]	where n,m...=decimal numbers
	TC	COBOL tabsetting (1,8,12,16,.....)
	TA	ASSEMBLER tabsetting (1,9,16,26)
	TF	FORTRAN tabsetting (1,6,7,10,13,16,.)

The first sub-command (T[n,m..]) may be used when a special tabsetting is required. The tab stops are specified by decimal numbers separated by commas.

If no numbers are specified, the default values are assumed.

Max number of tabstops is 15.

Previous defined tabstops are automatically cleared when a command is given.

Default values are: 1,9,17,25,.....

The value of the current pointer is not altered and the editor remains in command mode.

6.1.2 Display current tabsetting

Command: TD

This command causes the current tabsetting to be displayed in the message field of line one.

The value of the current pointer is not altered.

6.1.3 Pack output according to tabstops

Command: TP

This commands signifies that lines to be written on standard output (SO) should be packed according to current tabulator setting.

The value of the current pointer is not altered, and the editor remains in command mode.

NB! This command should only be used when handling PROGRAM source. STRINGS OF SPACES IN QUOTES may cause surprise during compilation/assembly. In such cases the TP command should not be used.

6.1.4 Stop packing of output

Command: TU

This command cancels the effect of the TP command.

The value of the current pointer is not altered, and the editor remains in command mode.

6.2 Input/output commands

Before performing an input/output operation, an input/output file must be specified. This is done when selecting i/o device (see section 2 and 6.4).

If the standard listfile (SL) is assigned, all output written on the standard outputfile (SO) is also copied to the SL-file. In addition the corresponding line number is inserted in front of each line.

6.2.1 Reading lines from standard input file (SI) to textbuffer.

Command: R[EAD] [n] (n = decimal number)

This command will make the editor read n lines from standard input file to textbuffer. When n is omitted, the editor will read until the textbuffer is 3/4 full.

When a read is performed, the last line read will be shown on line 13 of the screen.

Current pointer, as well as the cursor, will be located at the beginning this line.

If a tabulator character is encountered, the input line will automatically be expanded according to the current tabsetting.

If this causes a line to be of more than 128 characters, an error message is given.

6.2.2 Writing textbuffer on output file (SO/SL).

Command: W[RITE] [*] [n] (n = decimal number)

This command will make the editor write n lines of the text-buffer on the specified output files. If n is omitted, the whole textbuffer will be written.

The written lines will be removed from the text buffer unless the write command is followed by the character *.

The current pointer will be located at the beginning of the text buffer.

Current pointer, as well as the cursor, will be located on the beginning of line 13.

If the pack tab command previously is given, all lines written on standard output (SO), will be packed according to the current tabulator setting.

6.2.3 Output close

Command: OC

This command causes the current output file on SO/SL to be closed.

If no output file was assigned, an error message is given.

The value of the current pointer is not altered, and the editor remains in command mode.

This command followed by the Output Assign command is specially useful when the mass storage media is full or not useable any more (see section 9).

6.2.4 Output Assign

Command: OA "string"

This command causes a new output file on SO to be assigned and opened. The string must satisfy the format of a standard TOS filename. Only single file is allowed, see TOS21 Users Guide Part 2.

If SO already was assigned or the TOS-filname specified already existed, an error message is given.

The value of the current pointer is not altered, and the editor remains in command mode.

6.2.5 Copy until

Command: CU <line number> !
CU "string"

This command will copy lines from SI to SO/SL until the specified line number or text string is found.

This line is placed as the first line in text buffer, and the editor will then read until end of file or text buffer is 3/4 full.

If specified line number is less than the first line in buffer or SI or SO not assigned, an error message is given.

If no match for the specified text string, the whole file will be copied.

The current pointer will be located at the beginning of the text buffer.

Current pointer, as well as the cursor, will be located at the beginning of line 13.

6.2.6 Skip until

Command: SU <line number> !
SU "string"

This command will read from SI until the specified line number or text string is found. This line will be placed as the first line in text buffer, and the editor will read until end of file or text buffer is 3/4 full.

If the specifed line number is less than the first line in the text buffer or SI not assigned, an error message is given.

If no match for the specified text string, rest of the input file is skipped.

The current pointer will be located at the beginning of the text buffer.

Current pointer, as well as the cursor, will be located on the beginning of line 13.

6.2.7 Normal exit

Command: E[XIT]

This command causes the editor to copy the rest of the input file, including the current textbuffer, to the output file directly.

The control is then returned to the operating system, TOS, which will close the assigned files.

For tabulator expand/pack, see the commands READ/WRITE.

6.3 Auxiliary input

The editor has the possibility of merging two input files. The following commands may be used.

6.3.1 Auxiliary Assign

Command: AA "string"

Where the string parameter is a TOS filename in standard format. Only single file is allowed. (For further information, see TOS21 Users Guide).

This command may be used when a merge file is desired.

The value of the current pointer is not altered, and the editor remains in command mode.

6.3.3 Auxiliary Include

Command: AI [n] * L !
AI "string" * L

(Where: n= decimal number, L= line address)

This command causes the editor to read n lines or until a match for the specified text string is found from the auxiliary input file and append the lines to the specified line address L.

If '* L' is omitted, current line is assumed.

If n is omitted, the editor will read until the text buffer is 3/4 full.

The file is automatically closed upon end of file.

The current pointer will be located at the last line read. Current pointer, as well as the cursor, will be located at the beginning of line 13.

6.3.4 Auxiliary Skip

Command: AS [n] !
AS "string"

(Where: n= decimal number)

This command causes n lines or all lines until a match for the specified text string to be skipped (inclusive the matching line).

If n is omitted or a match for the specified text string could not be found, rest of the auxiliary input file is skipped.

The file is automatically closed upon end of file.

The value of the current pointer is not altered, and the editor remains in the command mode.

6.4 Miscellaneous commands

The commands described in this section are not used during a normal editing operation. However, now and then these may be helpful to the user.

The value of the current pointer is not altered.

Cursor will be in the first position of the command field.

6.4.1 Clearing text buffer

Command: K[ILL]

This command causes the entire textbuffer to be cleared.

6.4.2 Return to TOS

Command: Q[UIT]

This command causes immediately return to TOS.
The assigned files will be closed.

7. Text handling functions

This section describes the functions for altering text in the text buffer.

General syntax of the functions:

<addr.ref> <func name> <func par>

Given a line address without specifying a function name, the current pointer will be updated to point to the beginning of the actual line.

7. Text handling functions

This section describes the functions for altering text in the text buffer.

General syntax of the functions:

<addr.ref> <func name> <func par>

Given a line address without specifying a function name, the current pointer will be updated to point to the beginning of the actual line.

7.1 Current pointer adjustment

These functions are used to alter the current character position pointer.

These will usually be used in complex function iterations.

7.1.1 Numerical adjustment

Syntax:
```
<addr.ref>    ::= <empty>
<func name>   ::= P[N]
<func par>    ::= <sign> <dec. const>
```

Current pointer is moved the number of characters specified by the decimal constant, either to the right (sign= empty or +) or to the left (sign= -).

The current pointer will not be moved outside the actual line.

Current value of the pointer is assumed.

The cursor will be located at the character corresponding to the new value of the current pointer.

Examples:

PN3	- causes the cursor to be moved 3 positions to the right.
pn-5	- causes the cursor to be moved 5 positions to the left.

7.1.2 Start of line

Syntax:
```
<addr.ref>    ::= <empty>
<func name>   ::= PS
<func par>    ::= <empty>
```

This function causes the current pointer to be moved to the beginning of current line.

7.1.3 Last character

Syntax:
```
<addr.ref>    ::= <empty>
<func name>   ::= PL
<func par>    ::= <empty>
```

This function causes the current pointer to be moved to the last non-blank character of current line.

7.2 Delete functions

7.2.1 Delete lines

Syntax:
```
<addr.ref>    ::= <text interval>
<func name>   ::= DL
<func par>    ::= <empty>
```

The lines included in the address reference specification are deleted.

Current pointer will be located to the first position of the line following the last deleted one. If the last line in the text buffer has been deleted, the current pointer will be located to the first position of the new last line.

The current line will be shown on line 13 of the screen.

Examples:

100dl	- line number 100 is removed from the text buffer.
:,"xxx"dl	- the first line and up to (including) the first line containing the string "xxx" will be deleted.

7.2.2 Delete character

Syntax:
```
<addr.ref>    ::= <empty>
<func name>   ::= D[C]
<func par>    ::= <empty> ! <dec. const> ! "string"
```

If the function parameter is either empty or a decimal constant, the number of characters specified, starting with the character located by the current pointer are removed.

If no value is specified, 1 is assumed.

If a text string is specified, all characters starting with the character located by the current pointer, up to (not including) the match of the specified text are removed.

No deletion is performed if a match on current line could not be found.

All characters to the right on the same line are shifted the actual number of positions to the left.

The last postions on the line is space filled.

Current pointer (as well as the cursor) indicates the character next to the deleted one.

Examples:

d	- delete one character
dc5	- delete 5 characters
d"tl"	- if match for the text "tl", all characters from current position up to "tl" are deleted

7.3 Insert line

Syntax: <addr.ref> ::= <empty> ! <line address>
<func name> ::= I[L] ! A[L]
<func par> ::= <empty> ! <dec. const.>

To be able to insert lines in front of the first line in the text buffer or append lines to the last, two functions may be used.

- insert in front of specified line
- append after specified line

If the address reference specification is empty, the line located by the current pointer is assumed.

Function parameter: <empty>

The editor enters the insert line mode which causes the line number 13 on the screen to be erased and cursor located in the first position.

Remaining lines of the screen are filled, if appropriate.

In case IL, the specified line will be found on line 14 of the screen, otherwise on line 12.

These two functions have effect on the CR and '->' control function keys (see section 8).

The insert line mode will be in effect until any control function keys causes the cursor to leave the current line (except for CR, LEFT CURSOR, RIGHT CURSOR, FORWARD TAB or when passing column 80 enterring alpha-numeric character).

These functions must be the last one in a function string, otherwise their appearence has no effect.

Function parameter: <dec. const>

The specified number of blank lines will be created and placed in front of or after the specified line address depending on the actual function name used.

7.4 Insert text string

Syntax:
```
<addr.ref>    ::= <empty>
<func name>   ::= IT ! AT
<func par>    ::= "string"
```

If the IT function name is used, the specified string is inserted into the text buffer in front of the character located by the current pointer. Otherwise the string will be appended this character.

If the length of the line exceeds 80, no insertion will be performed and a message is given.

Current pointer (as well as the cursor) will be located at the first character after those inserted.

Examples:
```
IT"xx"    - insert the string "xx" in
            front of the character located
            by cursor.
AT"yy"    - insert the string "yy" after
            the character located by
            cursor.
```

7.5 Copy Line

Syntax:
```
<addr.ref>    ::= <empty> ! <line address>
<func name>   ::= C[L]
<func par>    ::= <empty> ! <text interval>
```

This function causes the the specified text interval to be copied and placed in front of the specified address reference.

If the address reference is empty, the line located by the current pointer is assumed. If no function parameter is specified, current line is copied.

7.6 Move lines of text

Syntax:
```
<addr.ref>    ::= <empty> ! <line address>
<func name>   ::= M[L]
<func par>    ::= <empty> ! <text interval>
```

This function causes the specified text interval to be moved in front of the specified address reference.

If the address reference is empty, the line located by the current pointer is assumed.

If no function parameter is specified, current line is moved.

Examples:
```
100m20,30        - the lines in the interval of
                   20-30 are removed and placed
                   in front of line number 100.
```

7.7 Substitute functions

7.7.1 Substitute text strings

Syntax:
```
<addr.ref>    ::= <empty> ! <text interval>
<func name>   ::= S[T]
<func par>    ::= "string1" "string2"
```

Through use of this function a text string, string1, can be substituted by another string, string2, in the specified address reference. The string2 must not contain vertical bar characters.

If no address reference is specified, the current line is assumed.

The last line on which a substitution is performed, will be displayed on line 13 of the screen. Remaining lines are filled, if appropriate.

The current pointer (as well as the cursor) will be located at the character following the last substituted.

Number of substitutions will be shown in the message field of line one if the function appears as the last one in a function string.

Examples:
```
100s"xx""yy"        - the strings "xx", if any, on
                      line 100, will be changed to
                      "yy".

:,"xx"ST"y""zz"     - all occurences of the string
                      "y" in the specified inter-
                      val, i.e. from the beginning
                      of the textbuffer until the
                      first occurences of the
                      string "xx", will be altered
                      to "zz".
```

7.7.2 Substitute word

Syntax: <addr.ref> ::= <empty> ! <text interval>
<func name> ::= SW
<func par> ::= "string1""string2"

Through the use of this function, a text string representing a word, string1, can be substituted by another string, string2, in the specified address reference. The string2 must not contain vertical bar characters.

If no address reference is specified, the current line is assumed.

The last line on which a substitution is performed, will be displayed on line 13 of the screen. Remaining lines are filled, if appropriate.

The current pointer (as well as the cursor) will be located at the charcter following the last substituted.

Number of substitutions will be shown in the message field of line one if the function appears as the last one in a function string.

Examples: 100sw"xx""yy" - as for substitute text, except
:,"zz"SW"ABC""abc" if the strings "xx" or "ABC" are parts of a longer string. In this case no substitution is performed.

7.8 Find functions

7.8.1 Find text string Forward

Syntax: <addr.ref> ::= <empty> ! <text interval>
<func name> ::= F[F]
<func par> ::= "string"

If no address reference is specified, the function will search forward in the text buffer until the first match of the "string", starting from the character specified by the current pointer.

The line containing the match, will be displayed in line 13 and the rest of the screen is filled, if possible.

If the value of the current pointer is not altered after a match is found, the current pointer is moved one position ahead, and the search is performed once more.

If a text interval is specified, the function will search within the specified interval, starting from the beginning.

If no match, a message is given.

The value of the current pointer is in this case not altered.

7.8.2 Find text string Backwards

Syntax: <addr.ref> ::= <empty>
<func name> ::= FB
<func par> ::= "string"

The function will search backwards in the text buffer until the first match of the "string", starting from the character specified by the current pointer.

The line containing the match, will be displayed in line 13 and the rest of the screen is filled, if possible.

If no match, a message is given.
The value of the current pointer is not altered.

7.9 Line split

Syntax: <addr.ref> ::= <empty>
<func name> ::= LS ! LSL
<func par> ::= <empty>

This function causes current line to be divided in two.

A new line is created and appended current line.

The text, starting from the position indicated by the current pointer, is moved to the new line. If the second function name, split line left justify, is used, the text on the new line will be left justified.

The value of the current pointer is not altered.

7.10 Line concatenation

Syntax: <addr.ref> ::= <empty>
<func name> ::= L[C]
<func par> ::= <empty>

This function causes current line and the next to be concatenated. Leading spaces on the second line will be scanned off.

If the concatenation causes the length of current line to exceed 80 characters, this line is filled up to contain 80 characters. In this case the remaining contents of the next line is left justified. Otherwise the next line is deleted.

The value of the current pointer is not altered.

8. Control function keys

The following non destructive cursor positioning keys are available:

These keys causes the cursor to move:

LEFT CURSOR	to the previous character on the current line.
RIGHT CURSOR	a) if insert-line mode: to the next tabulator stop on current line b) otherwise to the next character on the current line.
UP CURSOR	to the same position on the previous line.
DOWN CURSOR	to the same position on the next line.
HOME CURSOR	to the first position of the first line on the screen.
CR	a) if insert-line mode: the lines in front of included the current line are rolled one line up. Current line is erased and cursor located in the first position. b) otherwise positions to the first character of the current line.
ROLL UP	contents of the screen is rolled one line up if more lines in the text buffer, otherwise no effect. Cursor position is not altered.
ROLL DOWN	contents of the screen is rolled one line down if not on the top of the text buffer. Otherwise no effect. Cursor position is not altered.
CTRL+I	to the next tabulator stop on the current line. If no more tabstops, this function key acts as the RIGHT CURSOR.
CTRL+G	to the previous tabulator stop on the current line.

The following editing function keys are available:

CTRL+P	INSERT CHARACTER (10H) (ONLY edit mode) Insert one space character in the position located by cursor. Rest of the text on the line is move one position to the right. If the line exceeds 80 characters, the last character on line is lost.
CTRL+A	DELETE CHARACTER (01H) (ONLY edit mode) The character located by cursor is deleted, rest Last position is space filled.
DEL	Erase contents of line

9. How to use the editor

This section contains examles of how to use the commands and functions of the editor.

9.1 Creation of a new element

To invoke the editor to create a new element, the call should be:

```
edit so=newelm
```

When loaded, the editor responses with the message

```
TANDBERG BUFFERED EDITOR VER. x.x
```

The editor is now ready to accept commands.

At this stage, the append line function may be used.

Type "al" followed by a push on the carriage return (CR) button.

The editor is now in the insert line mode, and the user may now enter all lines of information.

If another command should be given, press the HOME-button followed by the actual command.

The last command to be given is the "EXIT" command. This causes all information entered to be written onto the diskette.

The control is then passed on to the operating system.

9.2 Updating existing element

The previously generated element, newelm is to be updated.

To invoke the editor, the following call could be given:

```
edit si=newelm,so=newelm.new
```

In this case the first command to be given is the "READ" command.

This causes the information to be read from newelm and placed in memory.

The editing session should again be completed by giving the "EXIT" command.

9.3 What to do when no more space on media

If the error message

E29: No more space on diskette/cartridge

is displayed, rest of the information in memory must be written on another diskette/cartridge.

The following steps should be performed:

1. OC - close output file

2. READ1000 - ensure that as much input as possible is read.

3. remove the full diskette and insert a new

4. OA "scrtch" - assign a new output file named "scrtch".

   If the TDV 2114 has more than one diskette unit, the "scrtch" file should be assigned such that input- and output- files are on different units.

5. The next operations depends upon the number of diskette units available.

5a. If only one diskette unit.

   I. WRITE - output rest of information in memory

   II. If end of input
      OC - output close
      Goto IV.

   III. If not end of input
      Change diskette - insert the ordinary input/output diskette.
      READ1000 - read as much input as possible
      Change diskette - insert the scratch diskette
      Goto I.

   IV. Change diskette - insert the ordinary input/output diskette

   V. QUIT - return to the operating system

4b. If more than one diskette unit

   EXIT - output rest of information in memory and on input file

   The two output files may now easily be concatenated by using the utility routine MOVE.

9.4 How to find all occurences of a string in memory

The current pointer (cursor) should be placed at the beginning of the memory.

This may be done by typing ":" followed by the CR.

The actual find function may now be typed:

```
f"wanted string"
```

To locate the next occurences of the text, the HOME button followed by the CR button have to be pressed, and so on.

9.5 Remove all lines containing a specific text

All lines in memory containing the text "abc" should be deleted.

Command:

```
:; n<f"abc"; dl>
```

Where n is a decimal number, large enough to find all occurrences of the specified text.

9.6 Change strings through a whole file

The strings "abc" and "xyz" should be changed to "cba" and "zyx" respectively.

Command:

```
n<r; :,.s"abc""cba"; &s"xyz""zyx"; w>
```

Where n is a decimal number, large enough to read the whole file.

9.7 Insert characters at the end of all lines in memory

This example shows how to insert a missing final dot on each line of a COBOL program.

This could create lines containg double dots, such occurences will be substituted to one single dot.

Command:

```
:; n<pl; at"."; $+1>; :,.s".."."."
```

Where n is a decimal number equal or larger than the number of lines in memory.

9.8 Create copies of special parts of memory

Ten copies of the lines 20 to 25 (inclusive) of the memory should be created and placed in front of the last line in memory.

Command:

```
10<.c120,25>
```

9.9 Sideways move

Move the same part of the next five lines three positions to the right.

Before performing the command, the current pointer (cursor) should be located at the right position.

Command:

```
5<it"   "; p-3; $+1>
```

9.10 Convert assembly list file to source file

If all source files of an element are lost or destroyed, a list file may easily be converted.

The example runs on a list file generated with a page length of 48 lines. No $e directive must have been used.

All blank lines and the line containing heading text should be deleted.

From each line of the assembly program, the first 26 characters should be deleted.

Command:

```
:; n< 6<dl>; 38<dc26; $+1>; 4<dl>>
```

Where n is large enough to cover all pages in memory.

The 6<dl> removes the heading text on top of each new page.

The 38<dc26; $+1> deletes the 26 first characters of each line (38 lines) of the assembly code on each page.

The 4<dl> removes the blank lines at the bottom of each page.

9.11 Generation of EXEC files

A large program is usually divided into several modules (elements).

During the debugging phase, the following utility routines may be used several times on the elements:

RASM, XREF, LINK, DELETE, RENAME, MOVE

The name of the elements, which should be of equal length, should be entered and placed in an element.

Suppose the following element is previously generated:

```
!elemn1%elemn1&
!elemn2%elemn2&
!elemn3%elemn3&
.
.
.
!elemnQ%elemnQ&
```

and all elements should be assembled.

The command for this is:

```
:,.s"!""rasm si=:f1:"; &s"%"",so="; &s"&"".h,sl=:lp:"
```

The result of this operation will be:

```
rasm si=:f1:elemn1,so=elemn.h,sl=:lp:
rasm si=:f1:elemn2,so=elemn.h,sl=:lp:
rasm si=:f1:elemn3,so=elemn.h,sl=:lp:
.
.
.
rasm si=:f1:elemnQ,so=elemn.h,sl=:lp:
```

To perform a deletion of the elements with the extension "h", the command reads:

```
:,.s"!""delete$"; &s"%¦¦¦¦¦¦&"".h"
```

Where "¦" is the vertical bar character, see section 3.1.2.

The result of this operation will be:

```
delete$elemn1
delete$elemn2
delete$elemn3
.
.
.
delete$elemnQ
```

Appendix A - Command summary

Commands

AA	Auxiliary Assign
AI	Auxiliary Include
AS	Auxiliary Skip
CU	Copy Until
E[XIT]	Normal return to TOS
K[ILL]	Clear text buffer
O[A]	Output Assign
OC	Output Close
Q[UIT]	Escaping to TOS
R[EAD]	Read from standard input (SI)
SU	Skip Until
T[n,m...]	Setting tab stops
TA	Setting ASSEMBLY tabstops
TC	Setting COBOL tabstops
TD	Display current tabsetting
TF	Setting FORTRAN tabstops
TP	Pack output according to tabstops
W[RITE]	Write text buffer on output file (SO/SL)

Functions

A[L]	Append Line(s) (insert line mode)
AT	Append text
C[L]	Copy Line
D[C]	Delete Character
DL	Delete lines
FB	Find string Backwards
F[F]	Find string Forward
I[L]	Insert Line(s) (insert line mode)
IT	Insert text string
L[C]	Line Concatenation
LS	Line Split
LSL	Line Split Left justified
M[L]	Move lines of text
PL	Adjust current pointer: Last character
P[N]	: Numerical
PS	: Start of line
S[T]	Substitute Text strings
SW	Substitute Word

Control function keys

LEFT CURSOR	
RIGHT CURSOR	
UP CURSOR	
DOWN CURSOR	
HOME CURSOR	
CR	
ROLL UP	
ROLL DOWN	
CTRL+A	Delete character
CTRL+P	Insert character
CTRL+I	Forward tab
CTRL+G	Backward tab
DEL	Erase line

Appendix B - Error messages

The error messages are ment to be self-explanatory, however, some of them needs an additional explanation. This is attached to the specific error message.

E00: Internal error
Please, report on a TANDBERG Software Error Report to TANDBERG or to your local service representative.

E01: Text buffer overflow
The text buffer is full, delete or write to achieve new work space.

E02: Freelist area too small
The text buffer is full, delete or write to achieve new work space.

E03: Too many function iterations

E04: Illegal or misspelled command

E05: Illegal value of address ref

E06: Illegal syntax of address ref.

E07: Missing number after +/- in adr. ref

E08: Illegal start of text string

E09: Conflicting numbers of " in text string

E10: No match for specified text

E11: Specified line number not found

E12: No output device assigned

E13: System error code xxH
A system error code was returned during input/output operation, please find the corresponding message in The Tos21 Users Guide.

E14: Input device not assigned

E15: Illegal tabulator value

E16: Too many tabulator stops specified

E17: No line to be copied from

E18: No insertion is performed

E19: Illegal value of decimal constants

E20: Illegal syntax of command/function

E21: Concatenation not performed, missing second line

E22: Commands interrupted by operator

E23: Tab characters causes input line to exceed 128

E24: Text buffer empty

E25: No charcter to be deleted

E26: Attempt to read after end of file

E27: Vertical bars in string2 of substitute

E28: Commands must be separated by a ;

E29: No more space on diskette/cartridge

E30: Output file already on diskette/cartridge

E31: Illegal filename specified
An illegal file name was specified, please refer to the TOS21 Users Guide for the correct format.

E32: Output device already assigned

E33: Input device already assigned

E34: File could not be opened due to lack of memory
The necessary amount of memory space for a new file control block does not exist. This may, however, be solved by closing existing output (OC) and auxiliary input (AS) files. Try the auxiliary file first.

E35: Buffer conflict. Too many files assigned
More than 4 files must not be assigned to the SI, SO, SL or AI when invoking the editor.