AGARAD
PLATANED

introduction to

3600

musil text editor

INTRODUCTION TO MUSIL TEXT EDITOR

B. J. Rosenstein

INTRODUCTION

The MUSIL Text Editor can be used to modify, up-date, or even create MUSIL programs while sitting at the operator's console of the RC 3600. The editing procedure can be carried out on individual characters, on strings, or on program segments.

This manual is designed as a companion volume to Introduction to MUSIL. As such, it is directed at the beginning MUSIL programmer. For this reason it describes Text Editor when used by a single user, employing paper tape input and output, and working on a stand-alone system. Text Editor is, however, not itself limited by any of these restraints.

The handiest device to use with Text Editor is the F 13 Alphanumeric Display/Keyboard, because it is fast and quiet, and deletions and changes are easiest to observe when using this device, but the F 12 KSR Teletype is also preferred by many because of the permanent record it gives of the editing process. The F 14 Silent Printer/Keyboard combines quiet operation with a permanent record.

The user can also take advantage of whatever peripherals are attached to his system. For information on this, and on the specific features of operation of the operator device, please consult the Operators Guide.

General Characteristics of Text Editor

The Text Editor commands described in this booklet will allow you to produce punched paper tape versions of MUSIL programs. You may start with a paper tape that contains a MUSIL program that is to be updated, corrected, or expanded, or you may simply create a new MUSIL program by inputting your text to the console device's keyboard. Most commonly, Text Editor is used to prepare programs already on paper tape for a new compilation.

Text Editor can modify a program text either at the line or the character level. It does this by searching either for a string of characters or for an implicit line number. Text Editor provides these implicit line numbers, and also an implicit character pointer to the current character.

There are three types of Text Editor commands: input commands that put the user program – or a part of the user program – into the edit buffer, commands that modify the contents of the edit buffer, and commands that output the modified contents of the edit buffer onto punched paper tape.

At each point of the editing procedure there is a character pointer (CP) that points to the position that currently is available for operating on.

Each line that is read into the edit buffer is assigned an implicit sequential line number, beginning with 1, that is updated as the editing procedure progresses. Text Editor defines a line as text ending with a carriage return. When the user requests a line number, Text Editor calculates the line number by counting carriage returns from the beginning of the buffer.

All text is assumed to be in ASCII, and in even paritry.

The Logic of Text Editor

Text Editor views all input to it as a continuous stream of characters. This stream of characters is considered to be segmented into pages. A "page" is defined as a stream of characters up to a form feed character or to the end of input. Each page is segmented into lines. A "line" is defined as a stream of characters up to *and including* a carriage return.

The editing process takes place in three steps:

1. read a page into the edit buffer,
2. edit this page,
3. output this page, as modified.

When Text Editor has been loaded, it prints an * (asterix), and the user may begin editing. If at any point the capacity of the edit buffer is exceeded, then the appropriate error message will be printed.

The character pointer (CP) should be thought of as placed between two characters. Inserted characters will, then, be placed between these characters. Operations on the current character will occur on the character to the right of the CP.

The Use of Special Keys

The Escape Key, ESC, is used for two purposes:

1. Striking ESC twice initiates processing of the command(s) just typed.
2. Striking ESC once after the argument to a command code delimits that argument from whatever may be typed after it.

When ESC is struck, a $ (dollar sign) is displayed.

Each command consists of one or two letters or a special graphic. This is the "command code". Some commands allow you to place a number in front of them. These numbers must be decimal integers between zero and ± 2047. Some command codes may be followed by a string argument.

Each command, then, has the form

n code string $,

where $ represents the ESC key, and n and/or string may be absent, depending on the specific case.

To initiate processing of this command, the ESC key is pressed once more, if the command ends with a dollar sign, or twice if it does not.

Several commands may be written one after the other before processing is initiated. In this case we have a string of commands, such as

n code string$n code string$.....$

To execute this command string, the ESC key must be struck once more. Remember, that processing begins upon the reception of a sequence of two ESCs. An accidental third ESC will have no ill effects.

Carriage Return, CR, will have no effect if struck within a command string, as long as it struck only between commands, and not in the middle of a command. If CR is struck within a string of characters, then Text Editor will assume that CR is part of this string.

RUBOUT is used to delete the last character input. On a teletype it is represented by a back arrow. See the Operators Guide for its effect on other devices. Repeated RUBOUTs delete as many characters as there are RUBOUTs, from right to left, until a "terminator" is reached. At that point Text Editor will issue an *. A "terminator" is any character whose ASCII code is below octal 32, such as ESC, new line, tabulation, form feed, etc. In practice the deletion process usually goes to the CR that signals the end of the preceeding line.

Example:

start: FIRSTTT↑
action: two RUBOUTs
result: FIRST↑

The arrows indicate the position of the CP.

TABs are simulated with spaces, that is, they appear as spaces on the operator device. On the output tape they appear as the TAB character followed by the RUBOUT character. Predefined TAB positions occur at columns 1, 9, 17, 25, etc. TAB positions cannot be redefined by the user. The judicious use of TABs makes your program attractive and easy for you and others to read.

SHIFT key use will not always yield the expected result of inputting the character on the upper half of the key. Please refer to the Operator's Guide to learn the proper use of this key.

NULL and LINE FEED will be ignored on input, but a line feed will be provided for every CR sent to the output device.

When the CTRL, Control Key, is pressed together with a character, then the ASCII result of the combination is input. In some cases CTRL has the effect of inputting the character displayed on the upper half of a key. Please consult the Operator's Guide for information on this situation. CTRL used with a character, will result in the display of an up-arrow followed by the character:

CTRL A is displayed as ↑A

Parity Errors

Parity is always checked on input and even parity is assumed. If an input character contains a parity error, then the following message will be delivered

PARITY ERROR IN LINE n

If the user then examines that line, the character in error will appear as a / .

Beginning the Editing Procedure

Before editing can take place the machine must of course be in operation and the necessary programs will have to be loaded. Consult the Operators Guide for loading procedures.

Specifically, before beginning editing

The operating system must have been autoloaded,
All necessary driver programs must have been loaded,
The MUSIL interpreter must be loaded,
Text Editor must be loaded, and
your MUSIL program (if any) must be ready for input.

After program loading Text Editor takes command, clears its buffers, and is in control. This is indicated to you by the printing of an *.

Input Commands

Y read a page

The next page of the user program is read into the edit buffer. The form feed at the end of the page will be read, but not stored in the edit buffer. The CP will be positioned to just before the first character in the edit buffer. If the edit buffer is exceeded, the following message will be printed:

BUFFER IS FULL-Y OR A INPUT IS TERMINATED

If this message is received, then a part of the current page is in the edit buffer. To continue the editing process, the user must either read out the buffer or delete something from it. The rest of the page can then be read in.

A append a page

The present contents of the edit buffer are augmented with the next page. The CP will be positioned just before the first character of this new page. If the buffer is full before the command is given, then the following message will be printed:

BUFFER IS FULL-CANNOT DO A

If the edit buffer fills up while the command is being executed, then the message will be

BUFFER IS FULL-Y OR A INPUT IS TERMINATED

The situation is the same as for the Y command. When the next page is to be input from the keyboard, then the insert command is used instead of the A command.

T display contents of edit buffer on console device

The CP is not affected. The entire contents of the edit buffer will apear on the display device. If the user wants to see only a part of the contents of the edit buffer, then

nT display n lines of edit buffer

will display n lines of the edit buffer, starting from the current position of the CP, and CP will *not* be moved.

Commands to the Character Pointer

B place CP at beginning of edit buffer

The CP is moved to just before the first character in the edit buffer. If an argument is used with this command, then the argument will be ignored.

nJ jump CP to line n

The CP will be moved to just before the first character of the n-th line of the edit buffer.

L move CP to beginning of line

Moves the CP to before the first character of the current line.

nL move CP n lines

The CP will move n lines from its current position and then place itself just before the first character of the new line:

for $n > 0$,	move forward past n carriage returns
for $n < 0$,	move backwards past $\|n\| + 1$ carriage returns and forward one character
for $n = 0$,	place CP to just before the first character of the current line (equivalent to L)

If n is so large that it is attempted to move the CP past the limits of the edit buffer contents, then the command will result in the CP being placed after the last character of the buffer (if n is positive), or before the first character of the buffer (if n is negative).

nM move CP in characters

This moves the CP with respect to individual characters.

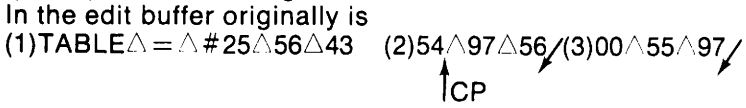
for $n > 0$,	CP is moved n character positions forward
for $n < 0$,	CP is moved $\|n\|$ character positions backwards
for $n = 0$,	there is no effect on CP

Z move CP to end of edit buffer

CP ends up just after the last character of the edit buffer.

Example of Commands to the CP

△ represents a blank space
(n) represents the implicit line number
↙ represents Carriage Return
In the edit buffer originally is



Subsequently:

Command	Result
B$$	(1)TABLE△=△#25^56△43↙ (2)54^97^56↙(3)00^55^97↙ CP
3J$$	(1)TABLE△=△#25△56△43↙ (2)54^97^56↙(3)00^55^97↙ CP
-1L$$	(1)TABLE△=△#25^56△43↙ (2)54^97^56↙(3)00^55^97↙ CP
Z$$	(1)TABLE△=△#25^56△43↙ (2)54^97^56↙(3)00^55^97↙ CP
-15M$$	(1)TABLE△=△#25^56△43↙ (2)54^97△56↙(3)00^55^97↙ CP
L$$	(1)TABLE△=△#25^56△43↙ (2)54^97△56↙(3)00^55^97↙ CP
2J$$	(1)TABLE△=△#25△56△43↙ (2)54^97^56↙(3)00^55^97↙ CP
50M$$	(1)TABLE△=△#25△56△43↙ (2)54^97△56↙(3)00^55^97↙ CP

The same effect can be obtained by writing the commands as a single command string, thus:

B$3J$ – 1LZ – 15ML2J$50M$$

Text Modification Commands

These commands often contain arguments. Each argument is delimited by $. Thus, commands with arguments end with one $, as shown below, and to initiate execution only one more $ is needed, for a sum of $$.

Cstring$_1$$string$_2$$ search for and replace string

Will cause Text Editor to search forward (only) for (only) the first occurence of string$_1$ and replace it with string$_2$. If the end of the edit buffer is reached before the string is found, then

STR NOT FOUND

will be displayed and the CP will be placed before the first character of the edit buffer, allowing the user to search the rest of the buffer for the string by repeating the command. If the string is found, then the CP is placed just after the last character of string$_2$.

To simply delete a string, the C command can be used with an empty string$_2$, that is, we type

Cstring$ delete string

The CP will be placed at the point of the deletion.

Istring$ insert string

The string is inserted at the current position of the CP. The CP is then moved to just after the inserted string.

nD delete characters

This command has the following effect

for n $>$ 0, delete n characters to the right of CP
for n $<$ 0, delete |n| characters to the left of CP
for n = 0, ignore command.

The CP is not moved.

nK delete n lines

n lines are deleted from the edit buffer, thus:

for n $>$ 0. delete n lines forward from the CP
for n $<$ 0, delete |n|+1 lines backwards from the CP
for n = 0. delete everything before the CP, until a carriage return is reached

After command execution, the CP will be placed just after the last deleted character.

Sstring$ search for string

The editor searches forward until the *first* occurence of the string, and positions CP to just after this first occurence. If the end of the edit buffer is reached before the string is found, then

STR NOT FOUND

is displayed and CP will be placed just before the first character of the edit buffer, enabling the rest of the buffer to be searched by a repetition of the command.

Nstring$ search, punch, and read

This command causes Text Editor to search forward in the edit buffer for the string. If the end of the buffer is rearched before the string is found, then Text Editor reads in the next page of the input until the buffer is filled again, having punched out the first buffer contents. This process continues until the first occurence of the string is found. If the end of the input is reached before then, then

STR NOT FOUND

is displayed. CP is placed just before an empty edit buffer, and all input is transferred to an output file.

Qstring$ search and read

The Q command is identical to the N command, except that no output punching takes place.

Example of Text Modification Commands

△ represents a blank
(n) represents the implicit line number
↙ represents a Carriage Return

In the edit buffer originally is

```
(1)BEGIN↙(2)△△△△IF△IN.ZMODE= 0△THEM△OPEN(IN,1);↙
  ↑CP
```

Subsequently:

Command	Result
CTHEM$THEN$$	(1)BEGIN↙(2)△△△△IF△IN.ZMODE=0△THEN△OPEN(IN,1);↙ ↑CP
CIN,1$$	(1)BEGIN↙(2)△△△△IF△IN.ZMODE=0△THEN△OPEN( );↙ ↑CP
IINN.1$$	(1)BEGIN↙(2)△△△△IF△IN.ZMODE=0△THEN△OPEN(INN.1);↙ ↑CP
-3M$$	(1)BEGIN↙(2)△△△△IF△IN.ZMODE=0△THEN△OPEN(INN.1);↙ ↑CP
3D$$	(1)BEGIN↙(2)△△△△IF△IN.ZMODE=0△THEN△OPEN(IN);↙ ↑CP
B2M$$	(1)BEGIN↙(2)△△△△IF△IN.ZMODE=0△THEN△OPEN(IN);↙ ↑CP
1K$$	(1)BE△△△△IF△IN.ZMODE= 0△THEN△OPEN(IN);↙ ↑CP
0K$$	(1)△△△△IF△IN.ZMODE=0△THEN△OPEN(IN);↙ ↑CP
2M$$	(1)△△△△IF△IN.ZMODE=0△THEN△OPEN(IN);↙ ↑CP
-1K$$	(1)△△IF△IN.ZMODE=0△THEN△OPEN(IN);↙ ↑CP
IBEGIN$$	(1)BEGIN△△IF△IN.ZMODE=0△THEN△OPEN(IN);↙

Output Commands

F form feed

This command outputs a form feed character to the output tape, along with 10 zeroes before and after the form feed character, so as to make it visible. These zeroes will be ignored when the tape is later read. When the output tape is later read, and if it is to be used with a printer, then the occurence of the form feed character will cause the printer to begin a new page.

nF n inches of leader output

The Editor will output n inches of leader, up to 100 inches. If n is greater than 100, then 100 inches will be output. In the case of paper tape this means that n inches of blank tape will be punched. 0F$$ causes the same effect as F$$. Neither the F nor the nF command has any effect on the CP.

P punch buffer

The entire edit buffer is punched out, followed by a form feed preceded and followed by ten zeroes.

nP punch n lines

Starting from the CP, n lines are punched, plus a form feed, as above. If the end of the text in the buffer is reached before n lines are punched, then punchting is terminated, and with a form feed.

PW punch buffer, no form feed

The entire edit buffer is punched and no form feed is inserted.

nPW punch n lines, no form feed

Starting from the CP, n lines are punched, but no form feed. If n is too big, punching stops when the end of text in the edit buffer is reached.
The punch commands have no effect on the CP. In counting the number of lines punched, the part of the current line after the CP is counted as the first line.

E punch buffer and remaining input

The current contents of the edit buffer plus the remaining input contained in the input device will be punched out.

R output and read page

One page of the edit buffer is output and the next page is read in. This command is equivalent to writing PY.

nR output and read

The equivalent of a combination of P and Y commands, written n(PY), n pages are output and n pages are read into the edit buffer. 0R and 1R are equivalent.

Special Commands

: print number of lines

The number of lines that are in the edit buffer will be displayed.

. CP line number

The number of the line the CP is pointing to will be displayed.

= print number of characters

The number of characters that are in the edit buffer will be dispayed.

CTRL insert tabulation

Insert a tabulation character in the output tape, allowing the tape to be shorter by compressing spaces. The tape will print as though the spaces were present, giving a printout with an attractive appearance.

Error Messages

BUFFER CAPACITY EXCEEDED DURING COMMAND INPUT, COMMAND IS TERMINATED AND BEING EXECUTED	Command string exceeds the capacity of edit buffer.
BUFFER IS FULL – CANNOT DO A	Attempting to append a page when the buffer is full.
BUFFER IS FULL – Y OR A INPUT IS TERMINATED	During a read, buffer capacity is exceeded. Part of a page has been read in.
PARITY ERROR IN LINE n	During a read, a parity error occurred in line n. When examined, the character in error will be replaced by a / .
STR NOT FOUND	unsuccessful string search.
?? comand string	Editor cannot understand the command. It displays the command it cannot understand plus the commands that follow it in the command string.

Reference List of Musil Text Editor Commands

Command	Format	Meaning
A	A	Append a page
B	B	Place CP at beginning of edit buffer
C	C$string_1$\$$string_2$\$	Search for $string_1$ and replace it with $string_2$ (also used for string deletions)
D	nD	Delete n characters
E	E	Output buffer and remainder of input tape
F	F	Punch a form feed
	nF	Punch n inches of leader
I	Istring$	Insert string
↑I	CTRL I	Insert tabulation
J	nJ	Jump CP to line n
K	nK	Delete n lines
L	L	Move CP to beginning of current line
	nL	Move CP n lines from current position
M	nM	Move CP n character positions
N	Nstring$	Search for string, if necessary through input and punch
P	P	Output edit buffer plus form feed
	nP	Output n lines plus form feed
PW	PW	Output edit buffer
	nPW	Output n lines
Q	Qstring$	Search for string, if necessary through input
R	R	Output edit buffer and read in next page
	nR	Output n pages and read in n pages
S	Sstring$	Search for string
T	T	Display contents of edit buffer
	nT	Display n lines of edit buffer
Y	Y	Read a page
Z	Z	Place CP at end of edit buffer
=	=	Display number of characters in edit buffer
:	:	Display number of lines in edit buffer
.	.	Display the current line number