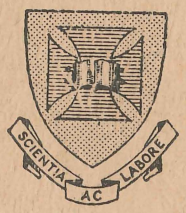

PRENTICE COMPUTER CENTRE

University of Queensland

TECHNICAL MANUAL No. 6

EDIT—A LINE EDITOR FOR THE PDP-10

This manual has been authorized by the
Director of the Prentice Computer Centre

TECHNICAL MANUAL NUMBER 6

EDIT - A LINE EDITOR FOR THE PDP-1Ø

MNT-6/ed6
16 March 1982

ACKNOWLEDGMENTS

The author wishes to thank Mr R.D. Nilsson for his comments and suggestions made during the preparation of this manual. Mr Nilsson, Senior Lecturer in the Department of Civil Engineering, is the original author of the QEDIT program.

Version 4 resulted mainly from contributions from La Trobe and Flinders Universities. Version 5, a major revision, is the work of Mr I.S. Burgess of the Prentice Computer Centre, University of Queensland. Due acknowledgment must also be given to Mr R.A. Cook of La Trobe University for the part that IOLIB contributes to Version 5.

The sixth edition of this manual describes the functions of QEDIT version 6D(54), dated 30-Oct-1981.

CONTENTS

chapter page

1 INTRODUCTION

1.1 USE OF THIS MANUAL 1-1
1.2 LOGGING INTO THE SYSTEM 1-2
1.3 GENERAL FEATURES OF THE EDITOR 1-2
1.4 USE OF TAB CHARACTERS 1-3
1.5 CONTROL CHARACTERS 1-3
1.6 CONVENTIONS ON EXAMPLES 1-3

2 RUNNING THE EDITOR

2.1 CREATING A FILE 2-1
2.2 EDITING A FILE 2-2

3 COMMAND FORMATS

3.1 LOCATING LINES
3.1.1 TOP and BOTTOM 3-2
3.1.2 UP and NEXT 3-3
3.1.3 LOCATE 3-3
3.1.4 SEARCH 3-4
3.1.5 LABEL and STATEMENT 3-5
3.1.6 LINE 3-6
3.1.7 BACK 3-7
3.2 PRINTING LINES
3.2.1 ASCII printing 3-7
3.2.2 Octal Printing 3-8
3.3 INSERTING LINES
3.3.1 Inserting One Line 3-9
3.3.2 Inserting One or More Lines 3-9
3.4 DELETING AND REPLACING LINES
3.4.1 Deleting Lines 3-10
3.4.2 Replacing Lines 3-11
3.4.3 Deleting and Inserting Lines 3-12
3.5 ALTERING LINES
3.5.1 Changing Character Strings 3-12
3.5.2 Appending and Prepending Character Strings 3-13
3.5.3 Splitting a Line into Two Lines 3-14
3.5.4 Concatenating Two Lines 3-14

4 FILES

4.1 SAVE 4-1

4.2 FILE 4-2
4.3 QUIT 4-2
4.4 EXECUTE 4-2
4.5 RUN 4-3
4.6 OUTPUT OF A FILE TO DISK 4-3
4.7 INSERTING FILES 4-4
4.8 SHIFTING PARTS OF A FILE 4-5

5 MODES OF OPERATION

5.1 EXTENT OF TYPEOUT
5.1.1 SET VERIFY and SET BRIEF 5-1
5.1.2 Temporary Reversal of Typeout Mode 5-1
5.1.3 SET VERBOSITY 5-2
5.2 UPPER/LOWER CASE IN SEARCHES 5-3
5.3 PRINT-OUT OF SPECIAL CHARACTERS 5-3

6 AUTOMATED EDITING AND RECOVERY TECHNIQUES

6.1 RECOVERY AFTER EDITING MISHAPS 6-1
6.2 RECORDING THE EDIT COMMANDS AND TEXT 6-1
6.3 PERFORMING A FILE OF COMMANDS 6-2
6.4 MAKING A FILE OF COMMANDS 6-3
6.5 REPEATING COMMANDS 6-3

7 JOB INFORMATION COMMANDS

7.1 JOB COST BALANCE 7-1
7.2 JOB TIME 7-1

8 SPECIALIZED COMMANDS AND OTHER FEATURES

8.1 SPECIAL TERMINAL COMMANDS
8.1.1 Lower Case Terminals 8-1
8.1.2 Display Terminals 8-1
8.2 TRANSLATION FROM Ø26 TO Ø29 CHARACTERS 8-2
8.3 FORTRAN DIRECT ACCESS CAPABILITY 8-2
8.4 BASIC PROGRAMS AND LINE-NUMBERED FILES 8-2
8.5 PROTECTION AND PRIVACY OF FILES 8-3
8.6 EDITING OTHER PROGRAMMERS' FILES 8-3
8.7 CONTROLLINC EDITOR SIZE FOR EFFICIENCY 8-4
8.8 CHANGING THE DEFAULTS FOR THE SET COMMANDS 8-5

9 ILLUSTRATED FORTRAN EXAMPLE 9-1

APPENDIX

A LISTING OF PROGRAM 'SAMPLE.FOR'

B SUMMARY OF COMMANDS AND SWITCHES

B.1 COMMANDS B-1
B.2 SET COMMANDS AND EDIT SWITCHES B-5
B.3 FILE SWITCHES B-6

C ASCII CHARACTERS AND THEIR OCTAL CODES

D INFORMATIVE MESSAGES

D.1 EDITING MESSAGES D-1
D.2 FILE SPECIFICATION SYNTAX ERRORS D-3
D.3 FILE INPUT/OUTPUT ERRORS D-3

E GENERAL COMMENTS

E.1 CONTROL-O E-1
E.2 CONTROL-U AND RUBOUT E-1
E.3 CONTROL-R E-1
E.4 CONTROL-S AND CONTROL-Q E-2
E.5 TREATMENT OF TAB CHARACTERS BY COMPILERS
E.5.1 FORTRAN Compiler E-2
E.5.2 COBOL Compiler E-2

CHAPTER 1

INTRODUCTION

A program written in a programming language (such as FORTRAN) is called a source program. Source programs must be input to the computer before they can be compiled (i.e. translated into the computer's machine language) and run by the computer. A program rarely runs correctly the first time and may need many corrections and modifications before it is thoroughly tested.

In a batch system a program is punched on to punch cards and these cards are input to the computer through the card reader. Program modifications are made by punching up new cards to replace those that are incorrect. The program must be input again to obtain a new compilation. The system is slightly different with terminals, however. In this case, a source program is stored as a file on the disk, the file having been created either by typing the program in via the terminal, or reading it in on punched cards. (A file is a collection of words comprising computer instructions and/or data). Corrections to the program file are carried out quickly and easily, by an editing program with which the file on disk can be altered, and there is no need to re-input the complete program from the terminal for subsequent compilations.

Data files can also be kept on the disk and edited if necessary. Files can be kept or deleted as the user wishes.

This manual is intended as a basic teaching manual for the program QEDIT. QEDIT is an editing program with a simple command structure designed to edit disk files. QEDIT is designed for use via the remote terminals on the PDP-10 and it is assumed that users are familiar with the procedures and commands necessary to operate the PDP-10 system from terminals. For the sake of clarity, the QEDIT program will be referred to as the Editor throughout this manual.

1.1 USE OF THIS MANUAL

This is a teaching manual designed to make users familiar with the use of the Editor via a terminal. It is therefore recommended that the user reads this manual while seated at a terminal and that he experiments with the facilities of the Editor as they are outlined.

The majority of the examples given in the text will refer to the sample program created in Section 2.1, an extra copy of which is appended to this manual, Appendix A.

Appendix B gives a summary of all commands, and Appendix D gives a list of error messages. Appendix E should be studied first by those unfamiliar with the terminals. The information detailed there is useful to know. Chapter 9 shows a fully documented FORTRAN example.

1.2 LOGGING INTO THE SYSTEM

It will be assumed that the user is familiar with the technique of logging into the system. If this is not so, instructions on how to log in can be found in the User's Handbook. Once the user has successfully logged in, communication is established with the monitor which signals its readiness to accept commands by responding with a period '.'.

1.3 GENERAL FEATURES OF THE EDITOR

There are two functions of the Editor; CREATE to create a file and EDIT which enables files to be updated. Files may be data files or source program files. They must be ASCII files (i.e. the data they contain is stored as characters). The maximum size of a file which can be edited is limited by the amount of disk space available and an absolute limit of 1.3 million characters, about 2000 blocks.

The Editor creates files with variable length lines. Lines may be of any length. On input, very long lines are broken into 280 character fragments and rejoined on output. During editing, lines may be increased to a maximum of 300 characters (including the carriage return/line feed characters appended to the end of the line). To enter lines longer than the terminal width, simply continue typing, giving <cr> only at the end of the line. The carriage will return automatically to the left margin after a certain number of characters has been typed. (See "SET TTY WIDTH n" Monitor command.)

Files in the PDP-10 system are provided with a standard nomenclature. A file specification, referred to as a filespec, consists of the device or disk structure, filename and directory path. Often only the filename need be given. A filename is made up of two parts - a name and an extension.

The name consists of from 1 to 6 alphanumeric characters. The extension consists of from 1 to 3 alphanumeric characters, preceded by a period (.). The extension is usually used to denote the type of file, the most common being:

.FOR	Fortran IV source program
.REL	file for processing by relocating loader (not editable)
.DAT	Fortran data file

Additionally, switches where appropriate may be included in the file specification as /switch:value.

examples:
(i) TEST.FOR
(ii) MAIL.MAC[160,100]
(iii) ANSA:STATS.SPS[174,104]
(iv) DACCES.DAT/DIRECT:120

1.4 USE OF TAB CHARACTERS

The tab character is used to space several positions on the terminal print out. It is similar to the tab on a typewriter where a tab or 'stop' can be made at certain positions on the print line. At any point on the print line, typing the tab character will space all intervening positions until a tab is encountered. On the terminal these stops are set automatically in multiples of 8 character positions i.e. the tabs are set at positions 9, 17, 25, ... etc. (see Section E.5 on the treatment of the tab character by the Fortran compiler).

The tab character is stored on the file as a single character and is not converted to the equivalent number of blanks. For this reason, a line containing tab characters can be effectively longer than the number of characters in the line.

Some terminals have a TAB or HT key. On others, the tab character is typed by depressing the CTRL key whilst depressing I. Tab characters are denoted in this manual by <tab>.

1.5 CONTROL CHARACTERS

A carriage return character is used to indicate that this is the end of a line in the file. This character is typed on most terminals by striking the RETURN key. It is echoed by a carriage return and a line feed character. It is denoted in this manual by <cr>.

Other break characters are allowed within search or insert strings of characters, but terminate the command otherwise. <lf>, <bel>, <vt>, <ff>, <esc> and ^Z are defined as break characters. ";" is not a break character but may be used to terminate a command. The Editor can begin processing a command only after a break character is typed.

1.6 CONVENTIONS ON EXAMPLES

For the sake of clarity it is necessary to differentiate between the characters typed in by the user and the characters typed back by the Editor or the monitor. Throughout this manual, therefore, the underlined portions of examples will refer to all the output that has been typed back by the system.

CHAPTER 2

RUNNING THE EDITOR

There are two major functions of the Editor: input and edit. Input state (signalled when the Editor types back 'Input:') means that consecutive lines of input text can now be inserted in the file. Edit state is signalled by the response '*' indicating that the Editor is ready to accept editing commands.

The input state is invoked with the system command CREATE and the edit state with the system command EDIT.

2.1 CREATING A FILE

CREATE filespec /switch1/switch2....

This command is used to create a new file. A file is said to be created if no file by the same name existed when the CREATE command was given. It invokes the Editor, setting it to its input state, and storing the file name given for later use when the file created is ready to be stored on disk. If a file of the specified name already exists, the warning '%EDISEF Superseding existing file' is given. The user may then either QUIT editing immediately or at the appropriate time FILE with another filename.

Switches may include the switch form of any of the SET commands, HELP, DELSEQ and NODELSEQ, in the form /switch:value
e.g. CREATE SUMSQ.FOR/HELP:TEXT/PAGE:20/ULC
See Appendix B for a brief description of the switches available. The switch form of the SET commands may be used on a CREATE or EDIT command as described here to change the default settings. Section 8.8 shows how to use SWITCH.INI to make more permanent changes to these default settings.

Lines can be typed in when input state is signalled by the response 'Input:'. The edit state can be entered at any time by typing a null line. This is a carriage return as the first character of the line. Blank lines can be inserted by typing at least one space before typing carriage return. The FILE command (see Section 4.2) can be used to keep this created file.

example:

```
.CREATE SAMPLE.FOR<cr>
Input:                                  [input state]
C<tab  >THIS IS A SAMPLE PROGRAM<cr>
C<tab  >PROGRAM CALCULATES THE SQUARES OF INTEGERS 1 TO 5Ø<cr>
C<tab  >PRINT HEADINGS<cr>
<tab   >LP=6<cr>
<tab   >WRITE (LP, 1Ø)<cr>
 1Ø<tab>FORMAT ('1   NUMBER     SQUARE'/)<cr>
C<tab  >CALCULATE SQUARE, PRINT NUMBER AND SQUARE<cr>
<tab   >DO 3Ø I=1, 5Ø<cr>
<tab   >J=I*I<cr>
 3Ø<tab>WRITE (LP, 2Ø) I, J<cr>
 2Ø<tab>FORMAT (I8,I12)<cr>
<tab   >STOP<cr>
<tab   >END<cr>
<cr>                                    [null line]
*FILE<cr>                               [* indicates edit state]
[EDIFIL Filed : SAMPLE.FOR]             [the FILE command will be
                                         described in Section 4.2 but
                                         is used here to save the
.                                        file for later editing]
```

CREATE may be abbreviated to CREA

2.2 EDITING A FILE

EDIT filespec /switch1/switch2....

This command is used to edit an existing file. It runs the Editor and reads the file into core setting the pointer to, and typing, the first line of the file. Edit state is signalled by the response '*'.

example:

```
.EDIT SAMPLE.FOR<cr>
C       THIS IS A SAMPLE PROGRAM
*FILE<cr>

[EDIFIL Filed : SAMPLE.FOR]             [to save file and return
                                         control to the monitor]
.
```

EDIT may be abbreviated to ED

If the Editor has been used since the user logged in, the filespec does not have to be specified for either the CREATE or EDIT commands, as the Editor will assume the filespec and switches of the previous CREATE or EDIT command. If no filespec is specified on the first use of the Editor, the fatal error message "?CMLNPC No previous command" is generated and the user returns to Monitor mode. Otherwise, if the file specified does not exist, the fatal error message "?EDIFNF LOOKUP, No file: filespec" is generated and the user returns

to Monitor mode. If no filespec is specified on a CREATE command but the Editor has been previously used, the warning "%EDISEF Superseding existing file" is given but the existing file is not overwritten until a SAVE, FILE or RUN command is given.

CHAPTER 3

COMMAND FORMATS

The Editor provides a dummy top line ***TOP and a dummy end line ***END to indicate the top and bottom of the file. These are not real entities: it is not possible, for example, to insert a line above the ***TOP line. It maintains a pointer which indicates the current line. Commands are provided to move this pointer up or down any number of lines (the farthest positions being the dummy top and the dummy end respectively); or locate it at a line containing any particular string of characters. Further commands are available for inserting, deleting or amending lines. Files may also be inserted at any point in the file.

A summary of commands is given in Appendix B. The command HELP gives a brief description of the Editor's function and commands, while H gives a list of commands with no explanation.

Many commands can be made to do an operation on a succession of lines. This is indicated by a positive or negative integer n following the command; the sign usually indicates the direction desired, i.e. (with the exception of the UP command) positive performs searches etc., down the file, negative back up the file. If n is omitted, or has the value Ø or 1, the command is performed upon one line, usually the current line. An '*' implies continuation of the specified operation to the limits of the file, in the direction indicated. Wherever ±n is given in the command descriptions, legitimate entries could be: 6 +6 -6 +* -*.

The notation /string/ following the command is used to indicate a string of characters. Delimiters are necessary because blanks may be included in the character string. The string consists of all characters between the leading and trailing delimiters. A '/' or any other non blank character (except '*','.','#','-','+' or ';') can be used as a delimiter, provided these delimiters are in matched pairs. The end of a command (when signified by a carriage return) will cause any necessary terminating delimiters to be supplied. Numeric delimiters are not allowed when using any deleting command.

For example, the string of characters (6X, could be indicated by one of the following:

```
/(6X,/
```

```
:(6X,:
A(6X,A
```

Note however that

```
(6X,    indicates the string    6X,
,(6X,   indicates the string    (6X
X(6X,   indicates the string    (6
```

Commands are normally terminated by a carriage return, but more than one command may be typed on one line by using ';' or one of the other break characters. This has special significance when using the REPEAT command. Note however that break characters may be included within strings, and as such the closing delimiter must be supplied if the command is not terminated by a carriage return.

When a filespec may be included in the command, it is indicated by the notation 'filespec' in the description. This notation includes the full specification of dev:filename.ext[path]/switches.

3.1 LOCATING LINES

It is necessary to be able to move the line pointer in order to indicate where new lines are to be inserted, or which lines need deleting or amending. Moving the pointer can be done by the following instructions.

3.1.1 TOP and BOTTOM

```
TOP
BOTTOM
```

TOP locates the dummy top line and types it out.

example:

```
.EDIT SAMPLE.FOR<cr>
C       THIS IS A SAMPLE PROGRAM
*TOP<cr>
***TOP
*
```

BOTTOM locates the last line of the file (not the dummy bottom line) and types it out.

example:

```
*BOTTOM<cr>
        END
*
```

TOP and BOTTOM may be abbreviated to T and B respectively.

3.1.2 UP and NEXT

```
UP +n
NEXT +n
```

UP and NEXT move the line pointer +n lines up or down respectively. The located line is typed out.

UP -n is equivalent to NEXT n.

example:

```
*T<cr>
***TOP
*NEXT 4<cr>
        LP=6
*UP<cr>
C       PRINT HEADINGS
*NEXT 6<cr>
        J=I*I
*UP 2<cr>
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
*
```

UP and NEXT may be abbreviated to U and N respectively. Since UP 1 and NEXT 1 are so widely used, especially short forms of these commands are provided. These are <esc> meaning UP 1 and <cr> meaning NEXT 1.

3.1.3 LOCATE

```
LOCATE +n /string/
```

LOCATE starts searching from the next line either up or down the file for a line containing the indicated string. If the current line is the bottom line or dummy end line the search starts at the beginning of the file. If the integer is negative, searching will proceed up the file.

If the current line is at the extreme of the file in the direction of searching, the search begins at the other extreme of the file.

example:

```
*B<cr>
        END
*LOCATE -/HEADING/<cr>
C       PRINT HEADINGS
*
```

For this command n does not refer to the number of lines to be searched. The command attempts to locate the string n times, typing out and leaving the pointer at the nth line containing any occurrence of this character string.

examples:

(i) To locate the second time 'WRITE' occurs in the sample program:

```
*T<cr>
***TOP
*LOCATE 2 /WRITE/<cr>
 30     WRITE (LP, 20) I, J
*
```

(ii) To locate the first and fourth comment lines:

```
*T<cr>
***TOP
*LOCATE /C<tab>/<cr>
C       THIS IS A SAMPLE PROGRAM
*LOCATE 3 /C<tab>/<cr>
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
*
```

The search for a character string is terminated by the dummy end statement. Thus, if this is typed, the string was invalidly written, or does not exist, or the number of occurrences requested exceeded the true number of lines on which it occurred.

example:

```
*T<cr>
***TOP
*LOCATE /WRITE (10)/<cr>          [this is a nonexistent
***END                             character string]
*
```

Since LOCATE is a search for the nth occurrence of a character string, a command such as LOCATE * /string/ is invalid.

example:

```
*T<cr>
***TOP
*LOCATE * /C<tab>/<cr>
%EDIIVS Illegal value: *
*
```

If the string to be searched for is omitted, the previous SEARCH or LOCATE string will be used. This feature saves typing if the occurrence found was earlier than the one wanted.

LOCATE may be abbreviated to L

3.1.4 SEARCH

```
SEARCH +n /string/
```

SEARCH, like LOCATE, starts searching from the next line either up or down the file for a line containing the character string. Here n

refers to the number of lines to be searched. If the next line is at the end of the file, searching begins at the other end of the file.

If the character string can be found the command will leave the pointer at the required line and will type that line out.

example:

```
*T<cr>
***TOP
*SEARCH * /LP=6/<cr>
        LP=6
*
```

If the character string cannot be found the message '%EDISNF Not found' will be typed out unless the dummy end line terminates the search. The pointer will be left at the last line checked by the command and this line will be typed out.

example:

```
*T<cr>
***TOP
*SEARCH 8 /TYPE 60/<cr>
[EDISNF  Not Found]
        DO 30 I=1, 50
*SEARCH * /TYPE 60/<cr>
***END
*
```

As with LOCATE, the search string may be omitted, in which case the previous SEARCH or LOCATE string will be assumed.

SEARCH may be abbreviated to S

3.1.5 LABEL and STATEMENT

LABEL ±label±n

A more efficient method of locating lines is with the LABEL command. The label required is any alphanumeric string commencing in column 1. This is more efficient because only column 1 is checked for many lines.

As in LOCATE, if the current line is at the extreme of the file in the direction of searching, the search begins at the other extreme of the file.

The n in this command signifies the number of lines either before (-n) or after (+n) the labelled line. The label is not delimited.

example:

```
*B<cr>
        END
```

```
*LABEL -C+2<cr>                    is equivalent to
      J=I*I                         *LOCATE -/J=I*I/
*
```

The LABEL command is particularly suited to straight data files, or Cobol and Macro source files, although it can be used with Fortran source programs if the statement label begins in column 1. Of special use for Fortran source programs is the STATEMENT command.

STATEMENT +statement+n

'statement' is the statement label of the Fortran line. It may be anywhere in the first five columns with trailing or leading blanks. The label is terminated by a tab or a blank. The Editor will not allow embedded blanks even though Fortran does. Again, the use of +n or -n allows lines to be located relative to a statement.

As in LOCATE, if the current line is at the extreme of the file in the direction of searching, the search begins at the other extreme of the file.

Unlike LABEL, which will check only on the characters given, a statement number must match completely; i.e. STATEMENT 1Ø will not match 1ØØ. Statements are not restricted to integer ones, and any string to fit the five column leading and trailing blank specification may be located.

example:

```
*STATEMENT -1Ø-3<cr>
C       PRINT HEADINGS
*
```

LABEL and STATEMENT may be abbreviated to LA and ST respectively.

3.1.6 LINE

LINE n

LINE is another command which will locate lines very quickly. Each line in the file is numbered sequentially. LINE with an argument will type out the indicated line as well as updating the line pointer to point to that line. LINE with no argument gives the current line number and the length of the line in characters, including the <cr><lf>.

example:

```
*LINE<cr>
Line 3, length = 18
*LINE 12<cr>
        STOP
*LINE
Line 12, length = 7              [<tab>STOP<cr><lf> = 7 chars]
```

LINE can be abbreviated to LI

3.1.7 BACK

BACK ±n

After a command which moves the line pointer, the BACK command may be given to move it back from whence it came and down (or up) n lines from there. Because deletion always begins with the current line, BACK after a DELETE command does not move the pointer. It is particularly useful after GET, INSERT, and PRINT commands but may be used after any command including BACK. BACK is a non-REPEATable command, allowing it to be used between a line of commands and a REPEAT command referring to it (see Section 6.5 for REPEAT).

example:

```
*LINE 4<cr>
        LP=6
*LOCATE /DO3Ø/<cr>
***END
*BACK<cr>
        LP=6
*LOCATE /DO 3Ø/<cr>
        DO 3Ø I=1, 5Ø
*BACK<cr>
        LP=6
*BACK<cr>
        DO 3Ø I=1, 5Ø
*
```

BACK may be abbreviated to BA

3.2 PRINTING LINES

3.2.1 ASCII printing

PRINT ±/string/

PRINT ±n

The PRINT command types out n lines of the file (in ASCII format) starting at the current line up or down the file and moving the pointer to the last line typed. PRINT * will type the complete file from the current line, typing being terminated by the dummy end line ***END.

If the SET FLAG command (Section 5.3) has been used, all control characters and lower case characters will be flagged as such on output. This setting may be temporarily reversed by appending a

period "." to the PRINT command.

example:

```
*T<cr>
***TOP
*L 4 /C<tab>/<cr>
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
*PRINT 5<cr>
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
        DO 30 I=1, 50
        J=J*I
 30     WRITE (LP, 20) I, J
 20     FORMAT (I8,I12)
*P.<cr>
 20<TAB>FORMAT (I8,I12)
*
```

If the current line is the last line or the dummy end line, PRINT * does not restore control to the dummy top and type the entire file. PRINT -n causes the printing to be done up the file, in reverse order.

PRINT, PRINT 0, PRINT 1 are all equivalent and will type out the current line.

PRINT may be abbreviated to P

3.2.2 Octal Printing

OPRINT +n

This command is available should the user wish to inspect the octal representation of certain characters, especially some of the nontyping control characters. See also the SET FLAG and PRINT. commands, which show tabs and other control characters.

This command prints n lines up or down the file starting from the current line, with each character output in its three character octal representation.

Appendix C lists the octal codes for ASCII characters in the PDP-10 system.

example:

```
*P<cr>
 20     FORMAT (I8,I12)
*OPRINT<cr>
040 062 060 011 106 117 122 115 101 124 040 050 066 130 054
040 111 062 054 040 070 130 054 040 111 064 051 015 012
*
```

OPRINT may be abbreviated to O

3.3 INSERTING LINES

3.3.1 Inserting One Line

INSERT line

Lines will always be inserted immediately after the current line and the pointer will be updated to point to the line inserted. The use of the dummy top and dummy bottom lines can be explained here as facilitating the insertion of lines at the beginning and end of the file. The line to be inserted will start one blank after the command. However, to facilitate the entering of Fortran lines the inserted line will start immediately after the command if the line starts with a <tab>. The insertion is performed without typing out the inserted line.

The text to be inserted may contain ';' or any break character, being terminated by <cr> (or more specificly, the <lf> of the <cr><lf> pair). Several identical lines may be inserted by following this command by a REPEAT command, described in Section 6.5.

example:

```
*L /STOP/<cr>
        STOP
*INSERT C<tab  >THIS IS THE END OF THE PROGRAM<cr>
*P 3<cr>
C       THIS IS THE END OF THE PROGRAM
        END
***END
*
```

3.3.2 Inserting One or More Lines

INSERT

This can be performed by typing the carriage return character immediately after the command INSERT. Input state is signalled when the Editor responds with 'Input:'. Any number of lines may be inserted after the current line and the line pointer is updated to point to the last line inserted. Edit state may be re-entered at any time by typing the null line. This is a carriage return as the first character of the line.

Lines longer than the nominal limit of 300 characters may be inserted, but they are broken into fragments of 280 characters or less for editing and reassembled when the file is output.

example:

```
*T<cr>
***TOP
*INSERT<cr>
```

```
Input:
C<tab  >FORTRAN PROGRAM<cr>
C<tab  >WRITTEN BY J. SMITH FEB 1970<cr>
C<tab  >UNIVERSITY OF QLD. ST. LUCIA<cr>
<cr>
*T<cr>
***TOP
*P *<cr>
***TOP
C       FORTRAN PROGRAM
C       WRITTEN BY J. SMITH FEB 1970
C       UNIVERSITY OF QLD. ST LUCIA
C       THIS IS A SAMPLE PROGRAM
C       PROGRAM CALCULATES THE SQUARES OF INTEGERS 1 TO 50
C       PRINT HEADINGS
        LP=6
        WRITE (LP, 10)
 10     FORMAT ('1   NUMBER     SQUARE'/)
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
        DO 30 I=1, 50
        J=I*I
 30     WRITE (LP, 20) I, J
 20     FORMAT (I8,I12)
        STOP
C       THIS IS THE END OF THE PROGRAM
        END
***END
*
```

If edit state is not re-entered (by typing an extra <cr>) before typing editing commands the Editor will treat these commands as further data to be inserted into the file.

INSERT may be abbreviated to I

3.4 DELETING AND REPLACING LINES

3.4.1 Deleting Lines

DELETE ±n

This command deletes n lines, either up or down the file starting at the current line.

DELETE ±/string/

This command starts deleting from the current line either up or down the file and continues deleting until the indicated string is located; the line containing the string is not deleted.

Both versions delete at least one line and the pointer is left at the

first line following the deleted section, this line being typed out. ***TOP and ***END are only indicators and so they cannot be deleted.

An attempt to delete ***TOP will result in the first line of the file being deleted instead. ***END will terminate a deletion.

example:

```
*T<cr>
***TOP
*DELETE<cr>
C       WRITTEN BY J. SMITH FEB 1970
*
```

It should be noted that if the string typed in is incorrect, or is only contained in the current line, then the command will delete the rest of the file, from, and including the current line. For this reason, numeric delimiters are not allowed in the DELETE command. When a string is used, the number of lines deleted is given by '[EDILDL Lines Deleted: n]'

For example, some of the comment lines inserted before are to be deleted from the program.

```
*T<cr>
***TOP
*DELETE /SAMPLE/<cr>
[EDILDL  Lines Deleted: 2]
C       THIS IS A SAMPLE PROGRAM
*T<cr>
***TOP
*P 2<cr>
***TOP
C       THIS IS A SAMPLE PROGRAM
*
```

If the command given had been DELETE /SAMLE/ then the entire file would have been deleted since SAMLE is a nonexistent character string.

DELETE may be abbreviated to DEL

3.4.2 Replacing Lines

REPLACE line

The current line is replaced by the line starting one space after the command, as in the INSERT command. The line pointer is maintained at its current position. Replacement is performed by the program without typing out the replaced line.

example:

```
*L / 10<tab>/<cr>
 10     FORMAT ('1   NUMBER     SQUARE'/)
*REPLACE  10<tab>FORMAT ('1', 3X, 'NO', 5X, 'SQ'/)<cr>
*
```

REPLACE may be abbreviated to R

3.4.3 Deleting and Inserting Lines

Quite often the user is interested in deleting a portion of the file and replacing or inserting extra text in its place. This can be done directly with the DINSERT command. DINSERT behaves in the same manner as DELETE, only it places the Editor back into input state, ready to receive new data.

DINSERT +n

Will delete n lines from the current line, move back to the line before deletion and print it out before entering input state.

DINSERT +/string/

Will delete all lines up to the line containing the string, move back to the line before deletion and print it and a count of the lines deleted before entering input state.

DINSERT may be abbreviated to DIN

3.5 ALTERING LINES

3.5.1 Changing Character Strings

CHANGE +n /string-1/string-2/

ALLCHANGE +n /string-1/string-2/

CHANGE substitutes the first occurrence of string-1 encountered in the line by string-2, expanding or contracting the line as necessary. ALLCHANGE substitutes every occurrence of string-1 in the line (and not just the first one) by string-2.

The changed line is typed out. If the data string, string-1, is not found '[EDINOC No Change]' is typed back. If a change would expand the line to more than 300 characters, the message '%EDILTL No change - line too long' is given.

example:

```
*P<cr>
 10     FORMAT ('1', 3X, 'NO', 5X, 'SQ'/)
```

```
*CHANGE /NUMBER/NO/<cr>
[EDINOC  No Change]
*CHANGE /NO/NUMBER/<cr>
 10     FORMAT ('1   NUMBER     SQ'/)
*CHANGE /SQ/SQUARE/<cr>
 10     FORMAT ('1   NUMBER     SQUARE'/)
*
```

These commands can be performed upon n lines either up or down the file starting at the current line. The line pointer is left at the last line checked, not the last line changed. Each changed line is typed out but the 'EDINOC' message is not given for unchanged lines.

example:

```
*T<cr>
***TOP
*ALLCHANGE * /J/II<cr>
        II=I*I
 30     WRITE (LP, 20), I, II
***END
*
```

If string-1 does not occur in any of the lines examined by the command then the dummy end line is the only line typed back.

example:

```
*T<cr>
***TOP
*CHANGE 5 /IVAL/I/<cr>
*CHANGE * /IVAL/I/<cr>
***END
*
```

CHANGE and ALLCHANGE may be abbreviated to C and A respectively.

3.5.2 Appending and Prepending Character Strings

APPEND ±n /string/

PREPEND ±n /string/

These commands allow a user to attach the specified character string onto the end or beginning of the line. The action occurs on n lines of the file down or up the file, and can be used, for instance, to place trailing sequence numbers into a file.

example:

```
*U<cr>
        END
*PREPEND / 999<cr>
 999    END
*APPEND /<tab>!COMMENT
```

```
 999     END      !COMMENT
*R<tab>END<cr>
*
```

APPEND and PREPEND may be abbreviated to AP and PRE respectively.

3.5.3 Splitting a Line into Two Lines

```
SPLIT +/string/
SPLIT +n
```

This splits the current line at the point following the specified string or the n'th character and inserts the rest of the line after the current line (before the current line for SPLIT -/string/ or SPLIT -n). Notice that n refers to a number of characters, not a number of lines as in other commands and that tab counts as one character only.

example:

```
*STATEMENT 10<cr>
 10      FORMAT('1   NUMBER      SQUARE'/)
*CHANGE /BER/BER','/<cr>
 10      FORMAT('1   NUMBER','      SQUARE'/)
*SPLIT /BER',/<cr>
 10      FORMAT('1   NUMBER',
'      SQUARE'/)
*PRE /<tab>1  /<cr>
        1   '      SQUARE'/)
*STA 30<cr>
 30      WRITE (LP, 20) I, J
*SPLIT -3<cr>                          [split before the <tab>]
 30
        WRITE (LP, 20) I, J
*NEXT<cr>                              [2nd part is inserted above]
 30
*AP /<tab>CONTINUE/<cr>
 30      CONTINUE
*
```

SPLIT may be abbreviated to SP

3.5.4 Concatenating Two Lines

```
CONCAT
CONCAT -
```

This command allows a user to join two adjacent lines of a file. If no sign is specified, the line below the current line is moved to the end of the current line. If a minus sign is used, the line above the current line is moved to the end of the current line. Concatenation can not create lines longer than 300 characters. The message '%EDILTL

No change - line too long' indicates this.

example:

```
*STATEMENT -10+1<cr>                    [to line after statement 10]
        1   '     SQUARE'/)
*CHANGE /<tab>1  //<cr>                 [remove the continuation code]
'       SQUARE'/)
*UP<cr>                                 [back to first line of pair]
 10     FORMAT('1   NUMBER',
*CONCAT<cr>                             [append the next line]
 10     FORMAT('1   NUMBER     SQUARE'/)
*
```

CONCAT may be abbreviated to CO

CHAPTER 4

FILES

4.1 SAVE

SAVE filespec

This is a similar command to FILE. SAVE will save the complete file using the filespec given. If no filespec is supplied the name specified in the original CREATE or EDIT is used. The previous version of the file is renamed with a .BAK extension, and the present incore file will get the specified name. If a LOG file (see Section 6.2) is open it will be closed and restarted.

However, SAVE does not return control to the monitor. Instead, it returns the line pointer to the line that was being signified before the SAVE command and types this line out. Editing can then continue.

Frequent SAVE commands can be executed during editing. Then, if a system malfunction should occur, or the current file is inadvertently destroyed, editing can recommence from the point of the last SAVE command. Using the LOG facility and the PERFORM command, this recovery may be automated making frequent SAVEs less important. Since SAVE closes off the current log file and restarts it, the current log file contains only the commands since the last SAVE.

Use of a log file is recommended in preference to frequent SAVEs as being less expensive and more flexible.

example:

```
*L /(LP, 1Ø)/<cr>
        WRITE (LP, 1Ø)
*SAVE<cr>                              [will be saved as SAMPLE.FOR -
[EDISAV Saved : DSKD:SAMPLE.FOR]        the filespec in the original
        WRITE (LP, 1Ø)                  EDIT command]
*
```

SAVE may be abbreviated to S

4.2 FILE

FILE filespec

This command will save the file using the name given. If the filespec is not present in this command then the file is output with the name included in the original EDIT or CREATE command.

The dummy top line and dummy bottom line are not written with the rest of the file. These are simply used during the edit procedure. FILE returns control to the monitor after the file has been written, and any current LOG file is closed. If control is transferred to the monitor by any means other than through the FILE command the created or edited file may be lost if editing is not CONTINUEd; any current LOG file will be lost unless it is CLOSEd or editing is continued.

example:

```
*FILE SAMPLE.FOR<cr>
[EDIFIL Filed : SAMPLE.FOR]

.
```

FILE may be abbreviated to F

4.3 QUIT

This command will return control to the monitor without outputting the file. As with the FILE command it clears the scratch file used by the Editor from the user's disk area and closes any log file. Exit to the monitor can be obtained by ^C but this will not clear this scratch file, and editing may be resumed by giving a CONTINUE or REENTER command.

example:

```
.EDIT SAMPLE.FOR/LOG:SAM.EDI<cr>
C       THIS IS A SAMPLE PROGRAM
*QUIT<cr>
[EDICLF Closed log file: SAM.EDI]
.
```

QUIT may be abbreviated to Q

4.4 EXECUTE

EXECUTE filespec

This command is exactly equivalent to FILE except that instead of returning to monitor mode, the last compile class command is performed again. Compile class commands comprise EXECUTE, COMPILE, LOAD and DEBUG. This feature is useful for rapidly re-editing and recompiling a program when an error is detected.

example:

```
*EXECUTE<cr>
[EDIFIL Filed: TEST.FOR]
FORTRAN:  TEST.FOR
LINK:    Loading
[LNKDEB DDT Execution]
```

EXECUTE may be abbreviated to E

4.5 RUN

```
RUN  filespec

SET RUN filespec                    /RUN:filespec
SET RUNOFFSET n                     /RUNOFFSET:n
```

The RUN command has the function of EXECUTE except that if a SET RUN command or /RUN switch has been given the program so specified is run after the edited file is saved.

The SET commands store the name of a program and a starting address offset respectively.

This facility is provided for those editing data files or RUNOFF files to provide the convenience EXECUTE gives for program testing.

example:

```
*SET RUN SYS:RUNOFF<cr>
*RUN TEST.RNO<cr>                   [file as TEST.RNO]
[EDIFIL Filed : TEST.RNO]
*                                   [RUNOFF waiting for commands]
```

RUN may be abbreviated to RU

4.6 OUTPUT OF A FILE TO DISK

```
PUT +n filespec
PUT +/string/ filespec

PUTDEL +n filespec
PUTDEL +/string/ filespec
```

These instructions output part of the file and name it as indicated. If PUTDEL is used instead of PUT the portion output is deleted from the incore file as well. Output (and deletion) starts from the current line and is terminated after n lines or when a line containing the character string is located (this line being neither output nor deleted).

The pointer is left at the first line after the section output and this line is typed out. The PUT command works in the same way as the

FILE command except that only a portion of the file is saved and no EXIT to the monitor is taken.

example:

```
.ED<cr>
C       THIS IS A SAMPLE PROGRAM
*L /DO 3Ø/<cr>
        DO 3Ø I=1, 5Ø
*PUTDEL 4 RTN<cr>
        STOP
*
```

```
*PUTDEL /STOP/ RTN<cr>
[EDILDL Lines deleted: 4]
        STOP
*
```

The file RTN will thus consist of:

```
        DO 3Ø I=1, 5Ø
        II=I*I
 3Ø     WRITE (LP, 2Ø) I, II
 2Ø     FORMAT (I8,I12)
```

while the program SAMPLE will have had these four lines deleted from the file:

```
*T<cr>
***TOP
*P *<cr>
***TOP
C       THIS IS A SAMPLE PROGRAM
C       PROGRAM CALCULATES THE SQUARES OF INTEGERS 1 TO 5Ø
C       PRINT HEADINGS
        LP=6
        WRITE (LP, 1Ø)
 1Ø     FORMAT ('1   NUMBER     SQUARE'/)
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
        STOP
C       THIS IS THE END OF THE PROGRAM
        END
***END
*
```

Reverse PUTting follows from the generalization of the n in command formats. It is doubtful whether it is a useful feature for these two commands.

PUT may not be abbreviated; PUTDEL may be abbreviated to PUTD.

4.7 INSERTING FILES

GET filespec

This command is used to insert a complete file if it exists. Insertion takes place after the current line and the pointer is left

at the last line inserted, this line being typed out.

For example, the file RTN is to be reinserted into its old position in the program.

```
*L 4 /C<tab>/<cr>
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
*GET RTN <cr>
 20     FORMAT (I8,I12)
*
```

If the filespec specified in the GET command does not exist '%EDIFNF LOOKUP, No file: filespec' is typed back by the program.

example:

```
*GET FILE15<cr>
%EDIFNF LOOKUP, No file: FILE15
*
```

GET may be abbreviated to G

4.8 SHIFTING PARTS OF A FILE

The PUT, PUTDEL and GET instructions provide a useful and convenient way of shifting parts of the incore file to other positions. This procedure can be simplified by leaving out the names in the commands, the Editor then provides its own name for naming the temporary files created. The temporary file should be deleted before logout from the system.

CHAPTER 5

MODES OF OPERATION

5.1 EXTENT OF TYPEOUT

5.1.1 SET VERIFY and SET BRIEF

The normal mode of operation under the edit state is VERIFY. This causes a typeout of located and altered lines.

SET BRIEF or SET NOVERIFY

Restoration of the normal mode of operation is achieved by:

SET VERIFY or SET NOBRIEF

The PRINT commands are not affected by these commands.

example:

```
*SET VERIFY<cr>
*T<cr>
***TOP
*L 2 /LP/<cr>
        WRITE (LP, 1Ø)
*SET BRIEF<cr>                    [this will suppress usual
*L /LP, 2Ø/<cr>                    Editor responses]
*PRINT<cr>
        WRITE (LP, 2Ø) I, II
*
```

The switch form, /BRIEF, may be given after the filename in the EDIT or CREATE command to set the initial mode.

SET VERIFY and SET BRIEF may be abbreviated to SET V and SET B respectively.

5.1.2 Temporary Reversal of Typeout Mode

If the current mode of typeout is VERIFY, suppression of the usual typeout may be achieved by appending a period '.' to a particular

command. The reverse applies if the current mode is BRIEF. Obviously this has no effect for commands which produce no typeout, such as INSERT OR REPLACE. A period appended to the PRINT and DISPLAY commands temporarily reverses the FLAG or NOFLAG mode of printing as described in Section 5.3.

example:

```
*SET VERIFY<cr>
*T.;L /LP/<cr>
        LP=6
*L. /WRITE/;PRINT 2<cr>
        WRITE (LP, 10)
 10     FORMAT ('1   NUMBER     SQUARE'/)
*SET BRIEF<cr>
*T.;L /LP/<cr>
***TOP
*L. /WRITE/<cr>
        WRITE (LP, 10)
*
```

5.1.3 SET VERBOSITY

Messages have a 6 character message identifier as a prefix. Fatal error messages are preceeded by "?", warnings by "%" and informative messages are enclosed in square brackets.

example:

```
?EDIDLT Direct access line too big - FILEing discontinued
%EDIAMB Ambiguous command
[EDIFIL Filed : filespec]
```

Several levels of verbosity are available. The lower levels should be used with discretion.

SET VERBOSITY HIGH	full messages
SET VERBOSITY STANDARD	full messages
SET VERBOSITY FIRST	message identifier missing
SET VERBOSITY LOW	only the message identifier
SET VERBOSITY NONE	only "?","%" or "[]" given

Other verbosity levels (CONTIN, FIRCON and PRECON) refer to messages continuing over several lines. At this time, there are no messages that continue over several lines.

example:

```
.EDIT SAMPLE.FOR/VERBOSITY:FIRST<cr>
C       THIS IS A SAMPLE PROGRAM
*LINE 100<cr>
%No such line: 100                  [the prefix EDINSL omitted]
*SET VERB H<cr>
*GET DSKD:ANZAS.TXT<cr>
%EDIIPP LOOKUP, No directory: DSKD:ANZAS.TXT
```

```
*SET VERB FIRST<cr>
*CHANGE /DECEMBER/JANUARY/<cr>
[No change]
*SET VERB LOW<cr>
*DELETE - 10<cr>                    [a space between "-" and "10"]
%EDIIND
*SET VERB NONE<cr>
*C/D/J/<cr>
[]
*
```

The initial setting for verbosity is STANDARD unless the /MESSAGE switch is used at LOGIN or /VERBOSITY switch is used in a QEDIT line in SWITCH.INI (see Section 8.8).

5.2 UPPER/LOWER CASE IN SEARCHES

SET ULCASE	/ULCASE
SET NOULC	/NOULC

Normally search strings must match exactly before success is declared, i.e. "Now" with some lower case characters does not match "NOW" in upper case. SET ULCASE relaxes this rule and allows upper/lower case to match lower/upper case in all commands where /string/ is an argument. SET NOULC reverts to the exact match requirement. See Section 8.8 for an example of setting ULCASE as the default.

example:

```
*LOCATE /lp/<cr>
***END                              [not found, end reached]
*SET ULCASE;BACK<cr>
        J=I*I
*LOCATE<cr>
 30     WRITE(LP, 20), I, J
*
```

SET ULCASE and SET NOULC may be abbreviated to SET U and SET NOU respectively.

5.3 PRINT-OUT OF SPECIAL CHARACTERS

SET FLAG	/FLAG
SET NOFLAG	/NOFLAG

SET FLAG causes lower case characters and control characters to be flagged as such in all subsequent print-out. 'X is printed for x, ^B for Control-B, <FF> for form-feed, <ESC> for escape. This mode remains in effect until cancelled by SET NOFLAG. Either mode of typeout may be reversed temporarily with "PRINT." or "DISPLAY." commands.

example:

```
*I C<tab  >This is a TEST<esc><cr>
*SET FLAG;PRINT<cr>                    [both commands on one line]
C<TAB>T'H'I'S 'I'S 'A TEST<ESC>
*P.<cr>                                [period reverses FLAG mode]
C       This is a TEST$
*SET NOFLAG<cr>
*
```

SET FLAG and SET NOFLAG may be abbreviated to SET F and SET NOF respectively.

CHAPTER 6

AUTOMATED EDITING AND RECOVERY TECHNIQUES

6.1 RECOVERY AFTER EDITING MISHAPS

Minor editing mistakes may be corrected with the DEL or RUBOUT key or by ^U (Control-U) if noticed before the end of the line (see Appendix E.2). Otherwise the line may be edited to correct a wrong word or replace the whole line (see CHANGE and REPLACE). More catastrophic errors sometimes occur. For example if the string in "DELETE /string/" is mistyped, many lines may be deleted. The present editing may then be aborted - using the QUIT command. If the user had the foresight to SAVE recently, re-entering the commands typed since then would not be onerous. The following sections show how commands may be recorded so that recovery can be automated.

6.2 RECORDING THE EDIT COMMANDS AND TEXT

```
SET LOG filespec                    /LOG:filespec
SET NOLOG
```

The LOG switch may be given on the EDIT or CREATE command, or SET LOG used when in edit state. This command closes any open log file and opens the specified file to receive all further characters, as commands or text, typed on the terminal.

The log file is check-pointed every 640 characters, so all but about ten lines should be available in the log after a system failure. Fatal errors close the log, as do QUIT, FILE, RUN and EXECUTE. SAVE closes and reopens the log file so that the current log refers to the disk copy of the file being edited.

If a command has damaged the in-core file and editing is aborted using the QUIT command, a log file will help in restoration. First edit the log file to remove the offending command and the QUIT command. Then EDIT the original file and PERFORM the log file.

The default filespec is nnnEDI.EDI, where nnn is the job number. It is useful in preventing conflicts where more than one person may use

your project-programmer number, to use the default filespec, especially if the LOG switch is included in SWITCH.INI (see Section 8.8). Notice that the PERFORM command uses the same default.

example:

```
.EDIT SAMPLE.FOR/LOG:SAM<cr>
C       THIS IS A SAMPLE PROGRAM
*SET LOG S<cr>
[EDICLF Closed log file: SAM]
*SET NOLOG<cr>
[EDICLF Closed log file: S]
*
```

SET LOG and SET NOLOG may be abbreviated to SET L and SET NOL respectively.

6.3 PERFORMING A FILE OF COMMANDS

Where a sequence of editing commands is to be performed more than once, it may be convenient to create a file of those commands and PERFORM them.

PERFORM n filespec

PERFORM takes commands from the specified file, reusing the file n times. "PERFORM *" is not allowed. PERFORM finds useful application in recovering from editing errors and system failures, where a log file provides the commands to be performed.

The progress of the PERFORM can be followed since each command executed within the PERFORM is output to the terminal preceded by a plus (+) sign, and followed by any output generated. If textual output is not required, the PERFORM command may be preceded by a SET BRIEF command. For safety, any warnings given while performing, abort the PERFORM. The command may also be safely aborted by the user typing ^C^C REENTER.

PERFORM is not REPEATable (see Section 6.5), but the file performed may contain both REPEAT and PERFORM commands.

The default filespec is nnnEDI.EDI, where nnn is the job number.

example:

```
*TOP;SET LOG DOTS<cr>
***TOP
*LOCATE. /C<tab>/<cr>
*APPEND /./<cr>
C       THIS IS A SAMPLE PROGRAM.
*SET NOLOG<cr>
[EDICLF Closed log file: DOTS]
*PERFORM 3 DOTS<cr>
+LOCATE. /C<tab>/
```

```
+APPEND /./C    PROGRAM CALCULATES THE SQUARES OF INTEGERS 1 TO 50.
+SET NOLOG                                  [NOTE: this command exists
+LOCATE. /C<tab>/                            in the log file]
+APPEND /./C    PRINT HEADINGS.
+SET NOLOG
+LOCATE. /C<tab>/
+APPEND /./C    CALCULATE SQUARE, PRINT NUMBER AND SQUARE.
+SET NOLOG
+
*
-
```

PERFORM may be abbreviated to PER

6.4 MAKING A FILE OF COMMANDS

MACRO filespec

If during an editing session a file of commands is required, this may be created in one of several ways. One of the most convenient is by use of the MACRO command. The Editor goes into MACRO input state, creating the specified file from the lines which follow. Any text may be inserted into the file, the syntax being checked when the file is PERFORMed. Return to edit state is accomplished as usual by typing an extra <cr>. The macro file is then closed as signified by the message "[EDICMF Closed macro command file: filespec]". No change is made to the file being edited by this command.

The default filespec is nnnEDI.EDI, where nnn is the job number.

example:

```
*MACRO DEDOT<cr>
MACRO input:
LOCATE. /C<tab >/<cr>
CHANGE /.//<cr>
<cr>
[EDICMF Closed macro command file: DEDOT]
*TOP;SET BRIEF<cr>
***TOP
*PERFORM 4 DEDOT<cr>                    [output not shown here]
```

MACRO may be abbreviated to M

6.5 REPEATING COMMANDS

REPEAT n RR n

This command repeats the previous command or line of commands n times. RR is simply an alternative to REPEAT which is quicker to type. This function is achieved by the Editor's remembering each line of commands in case a REPEAT is requested. Such a line is terminated by <cr> or <lf> or REPEAT. Note that <esc> and ";" do not terminate the "line of

commands" and either may be used as command separators to build up a group of commands. Certain commands are not repeatable and are not remembered in this way. BACK, REPEAT, SAVE and the SET commands are non-repeatable commands which may be used between the command line and the REPEAT command referring to it. Commands which invoke the input state - INSERT (multi-line), DINSERT and MACRO - leave nothing to be repeated, and neither does PERFORM.

There is a limit to the number of commands which may be stored for repeating. This limit is the number of commands that can be stored in one 300 character line using the shortest allowable abbreviations. Further commands may still be typed and are acted on as usual, but the bell is sounded to warn that the REPEAT command would act on only the first 300 characters. The MACRO and PERFORM commands have essentially no limit, so these should be used where the command line length may be restrictive.

example:

```
     *TOP<cr>
     ***TOP
     *L /<tab>/;SP /<tab>/;U;C /<tab>/      /;SP 6;DEL -1;CO.<cr>
     C       THIS IS A SAMPLE PROGRAM [L /<tab>/;]
     C                                [SP /<tab>/;]
     THIS IS A SAMPLE PROGRAM
     C                                [U;]
     C                                [C /<tab>/      /;]
     C                                [SP 6;]

     C                                [delete any excess spaces]
     *SET BRIEF<cr>                   [suppress typeout during REPEAT]
     *REPEAT *<cr>
     C     PROGRAM CALCULATES THE SQUARES OF INTEGERS 1 TO 50
     C     PRINT HEADINGS
           LP=6
           WRITE (LP, 10)
      10   FORMAT ('1   NUMBER     SQUARE'/)
     C     CALCULATE SQUARE, PRINT NUMBER AND SQUARE
           STOP
     C     THIS IS THE END OF THE PROGRAM
           END
     ***END
     *
     -
```

REPEAT may be abbreviated to REPE, or the equivalent command, RR may be used.

CHAPTER 7

JOB INFORMATION COMMANDS

7.1 JOB COST BALANCE (University of Queensland version only)

JOBBALANCE

The JOBBALANCE command types out the current job balance in dollars and cents.

At the start of a job the user imposes a cost limit on the job by means of the COST argument in the LOGIN command. He can change this limit by means of the SET COST command. However, if at any stage during a job the limit is exceeded the current task is terminated and output data could be lost. JOBBALANCE permits a user to enquire into the state of his job balance while still running the Editor. If the balance is nearing its limit he can Control-C out of his editing, issue a SET COST command and then CONTINUE.

example:

```
*JOBBAL<cr>
Balance of cost limit = $9.98
*
```

JOBBALANCE may be abbreviated to JOBB

7.2 JOB TIME

JOBTIME

The JOBTIME command types out the total central processor time used since the user logged in. The result is in hh:mm:ss.s

example:

```
*JOBTIM<cr>
Processor time = 00:00:03.40
*
```

JOBTIME may be abbreviated to JOBT

CHAPTER 8

SPECIALIZED COMMANDS AND OTHER FEATURES

There are a number of other commands available for performing specializing functions, e.g. depending on the type of terminal being used, or the requirements for the file being created.

8.1 SPECIAL TERMINAL COMMANDS AND OTHER FEATURES

8.1.1 Lower Case Terminals

The Editor will accept commands typed in lower case characters. The SET ULCASE command or /ULCASE switch facilitates editing mixed case files by allowing searches to succeed for either case. See Section 5.2.

The SET FLAG command or /FLAG switch could be of assistance in editing these files on a terminal which does not have lower case.

8.1.2 Display Terminals

DISPLAY +n
DISPLAY $\overline{+}$/string/

SET PAGE n	/PAGE:n
SET NOPAGE	/NOPAGE

The command DISPLAY is designed for output of sections of files on display terminals. The command is the same as PRINT except that the program waits when the bottom of a page is reached and erases the screen and starts a new page when signalled. The page size is set at a default of 19, but can be set by the command SET PAGE n or /PAGE:n on the CREATE or EDIT command. As with PRINT., DISPLAY. may be used to temporarily override the SET FLAG/SET NOFLAG condition (Section 5.3).

The DISPLAY command action is:

(a) outputs a hardware form feed to erase the screen

(b) outputs lines until

(i) the n'th line or the line containing 'string' is displayed,
(ii) the page 'n' is satisfied, or
(iii) a form feed is reached.

If it is (i) then a new command can be entered to the Editor. If it is (ii) or (iii) the Editor suspends output until <xon> (^Q) is typed.

In earlier versions of the Editor, it was possible to abort the display by giving an edit command. Control-C (^C) and REENTER will achieve this effect safely and return the Editor to edit state.

DISPLAY may be abbreviated to DIS

8.2 TRANSLATION FROM Ø26 TO Ø29 CHARACTERS

TRANSLATE ±n

TRANSLATE n means translate n lines converting the five Ø26 characters that fall within the FORTRAN subset of characters to their Ø29 equivalents.

i.e.	@ becomes	'	punch	8-4
	% becomes	(		Ø-8-4
	< becomes	)		12-8-4
	& becomes	+		12
	# becomes	=		8-3

This is used if a FORTRAN program has been punched with a mixture of Ø29 and Ø26 codes and the deck has then been read with a MODE.ASCII switch.

TRANSLATE may be abbreviated to TR

8.3 FORTRAN DIRECT ACCESS COMPATIBILITY

/DIRECT:n

This switch on an output filespec allows the Editor to create data files that will be compatible with Fortran direct access filing specifications. The 'n' is the RECORD SIZE used in the Fortran OPEN statement. Two characters are added for the <cr><lf> and lines are padded out with nulls to the exact word boundary. For example, 5-word lines would be created if 'n' were set at 2Ø. If a user's line is too long then an error message is given, the long line is printed and filing is stopped. The user may edit it if possible and try again, or use a larger value of n.

/DIRECT:n applies only to the output file to which it was specified, e.g... FILE filename/DIRECT:72 or PUT n filename/DIRECT:126

/DIRECT may be abbreviated to /DI

8.4 BASIC PROGRAMS AND LINE-NUMBERED FILES

BASIC and certain line editors create files with special line sequence numbers on each line. Such files may be edited leaving the line numbers intact for BASIC programs, but removing them for all other files. Two switches are available in case it is necessary to override the default cases.

/DELSEQ
This is the default for all files except BASIC programs. Line numbers are deleted.

/NODELSEQ
The default for BASIC files (extension .BAS), this switch leaves line sequence numbers on any lines that have them. The structure of the line is not preserved, relying on the ability of BASIC to restructure the sequence number.

/DELSEQ and /NODELSEQ may be abbreviated to /DE and /NOD respectively.

8.5 PROTECTION AND PRIVACY OF FILES

Files created by the Editor are given protection from access by other users depending on
(a) the protection of a pre-existing file of that name, or
(b) the protection of the file from which it was derived, or
(c) the installation default protection.
In addition, no .BAK file is created with so much protection that the owner cannot delete it without changing its protection first. The owner may edit files he is allowed to supersede or update. Other users will not be successful unless the owner has allowed renaming of the file.

8.6 EDITING OTHER PROGRAMMERS' FILES

Successful editing depends on the protection of the file concerned or of the directory if the file does not exist.

Ersatz devices are directories referred to by device-like names e.g. SYS:, NEW:, HLP: and LIB:. A library is a directory declared as such at LOGIN or set by program (e.g. SETSRC). A library is regarded as a read-only device when accessed implicitly. SYS: and NEW: may be given the status of libraries at LOGIN. The rule followed by the Editor in determining library status is, "if the path taken to find the file differs from the specified path or the default path for the device, the file is treated as if it was found in a library". This is extended to cover sub-file directories also.

The action of the Editor is best illustrated by examples. Suppose a person logged in on [100,100], specifying [100,5] as a library and requesting that NEW: and SYS: be searched for files not found on

[100,100] or [100,5].

```
     LOGIN 100,100 /LIB:[100,5] /NEW /SYS
```

Suppose that the following files existed with a low enough protection; A[100,100], B[100,5], SYS.C and NEW:D.

Referring to the files directly - "CREATE A", "CREATE LIB:B", "CREATE B[100,5]", "CREATE SYS:C", "CREATE C[1,4]", "CREATE NEW:D" and "CREATE D[1,5]" all supersede the existing file where protection allows it. "EDIT" in each of these cases would create a new file on the specified directory with the original becoming ?.BAK.

Referring to the files implicitly - "CREATE B" "CREATE C" and "CREATE D" all give a warning that such a file was found but create the new version in the default path [100,100]. "CREATE SYS.D" warns that NEW:D exists but creates SYS:D. Corresponding "EDIT" commands warn where the file was found, read it, and create the new file leaving the original. No .BAK file is created.

8.7 CONTROLLING EDITOR SIZE FOR EFFICIENCY

SET MAXCOR n /MAXCOR:n

The size of the Editor's data segment may be restricted by use of the /MAXCOR switch on the EDIT command or by the SET MAXCOR command. This value may be specified as n words, nP or nK. Any value outside the prevailing system limits e.g., "*" is adjusted to conform. The default value allows five pages of the file being edited to reside in core. This has been chosen to be efficient for most editing, but the optimum setting varies with the type of editing activity. A value as low as 5K may be the most suitable if only a few lines are accessed at any time, the other extreme reducing disk accesses for jobs which commonly search over the whole file. The Editor has a limit of 50 in-core pages and will not use more memory than necessary.

SET MAXCOR with no argument gives some details of the current setting and core usage.

example:

```
     .EDIT SAMPLE.FOR/MAXCOR:1<cr>     [will use the minimum allowed]
     C       THIS IS A SAMPLE PROGRAM
     *SET MAXCOR<cr>
     [EDIMCR 1 in-core page, 6 MAXCOR with 2K low core allows 3 pages]
     *SET MAX 20<cr>                   [change MAXCOR]
     *SET M<cr>      [see what changed]
     [EDIMCR 1 in-core pages, 20 MAXCOR with 2K low core allows 50
pages]
     *
     -
```

SET MAXCOR may be abbreviated to SET M

8.8 CHANGING THE DEFAULTS FOR THE SET COMMANDS

The default values for the various SET commands were originally chosen so as not to change the behaviour established by earlier versions of QEDIT. It is often convenient to take advantage of some of these features without having to give the SET command. For instance, if you prefer either case matching in searches, the SET ULCASE command provides this. Alternatively "EDIT filespec/ULCASE" may be used with the advantage that for subsequent EDIT commands within the same LOGIN session you need not give either the filename or the switches on an EDIT command.

For more permanent changing of the default settings, you may make use of SWITCH.INI. SWITCH.INI is a file that you can create with a line for any of the system programs that set their defaults in this way.

example:

```
.CREATE SWITCH.INI<cr>
Input:
LOGIN   /SETTTY/GAG:Ø/COST:$2Ø./DEFPRO.Ø17<cr>
RUNOFF  /CRETURN/AUTOP<cr>
QUEUE   /PRIORITY:4/NOHEAD<cr>
QEDIT   /LOG/ULCASE/MAXCOR:2Ø/RUN:SYS:RUNOFF/VERB:FIRST<cr>
FUR     /DECIDE<cr>
<cr>
*FILE<cr>
[EDIFIL Filed : SWITCH.INI]

.
```

CHAPTER 9

ILLUSTRATED EXAMPLE

A complete example will now be given which illustrates a number of the facilities outlined in this manual. A FORTRAN program will be written to calculate the area of a circle given the value of its diameter.

```
.CREA FRTRN.FOR/LOG:SAFETY.EDI<cr>
Input:                                  [input state signalled]
C<tab  >PROGRAM TO CALCULATE AREA OF CIRCLE<cr>
C<tab  >WRITTEN BY J. SMITH FEB 1970<cr>
 10<tab>READ (5, 20) DIAM<cr>
 20<tab>FORMAT (F10.3)<cr>
C<tab  >TEST FOR END OF JOB (DIAM=0)<cr>
<tab   >IF (DIAM.EQ.0.0) STOP<cr>
C<tab  >CALCULATE AREA AND PRINT RESULT<cr>
<tab   >AREA=3.1416*(DIAM*.5)**2<cr>
<tab   >WRITE (6, 30) DIAM, AREA<cr>
 30<tab>FORMAT (' DIAM', F10.3, ' AREA', F10.3//)<cr>
C<tab  >RETURN TO READ MORE DATA<cr>
<tab   >GO TO 10<cr>
<tab   >END<cr>
<cr>                                    [null line]
*FILE<cr>                               [edit state signalled]
                                        [this file has now been kept
[EDIFIL Filed : FRTRN.FOR]               with the same name given
[EDICLF Closed log file: SAFETY.EDI]        in the CREATE command]

.                                       [return control to the Monitor]
```

The program can now be compiled and run in the following manner:

```
.EXECUTE FRTRN.FOR<cr>
FORTRAN:  FRTRN.FOR
LINK:   Loading
[LNKXCT FRTRN Execution]
10.0<cr>
DIAM    10.000 AREA    78.540


5.0<cr>
DIAM     5.000 AREA    19.635
```

```
     Ø.Ø<cr>

     END OF EXECUTION
     CPU TIME: Ø.12  ELAPSED TIME: 22.44
     EXIT

     .
```

The program will now be modified in order that it may produce both the perimeter and the area for any given diameter. This could be performed in the following way. (The edit will be performed in verify mode so that the example can be followed easily).

```
.ED<cr>                                        [the filespec and switches
                                                from the create command
                                                will have been remembered]
C       PROGRAM TO CALCULATE AREA OF CIRCLE
                                               [first line of file printed]
*C /AREA/PERIMETER AND AREA/<cr>
C       PROGRAM TO CALCULATE PERIMETER AND AREA OF CIRCLE
*L 3 /C<tab>/<cr>
                                               [starts locating from the next
                                                line]
C       CALCULATE AREA AND PRINT RESULT
                                               [locates the third comment
                                                line]
*C. /AREA/PERIMETER AND AREA/<cr>
*C /RESULT/RESULTS/<cr>
C       CALCULATE PERIMETER AND AREA AND PRINT RESULTS
*SAVE<cr>
[EDICLF Closed log file: SAFETY.EDI]           [save file as precautionary
[EDISAV Saved : DSKD:FRTRN.FOR]                 measure - FRTRN.FOR assumed]
[EDILFR Log file restarted]
C       CALCULATE PERIMETER AND AREA AND PRINT RESULTS
*I<cr>                                         [insert two lines]
Input:                                         [input state]
<tab   >RAD=(DIAM*.5)<cr>
<tab   >PERM=2*3.1416*RAD<cr>
<cr>                                           [null line]
*N<cr>                                         [transfer pointer to next
        AREA=3.1416*(DIAM*.5)**2                line]
*C /(DIAM*.5)/RAD/<cr>
        AREA=3.1416*RAD**2
*N.<cr>                                        [transfer pointer to next
*C /AREA/PERM, AREA/<cr>                        line suppressing printing]
        WRITE(6, 3Ø) DIAM, PERM, AREA
*N<cr>
 3Ø     FORMAT (' DIAM', F1Ø.3, ' AREA', F1Ø.3//)
*R 3Ø<tab>FORMAT (' DIAM', F1Ø.3, ' PERM', F1Ø.3, ' AREA',<cr>
*I<tab>1F1Ø.3//)<cr>                           [replace line]
```

The entire file will now look like this:

```
*T.;N.;P *<cr>                          [multiple commands on a line]
C       PROGRAM TO CALCULATE PERIMETER AND AREA OF CIRCLE
C       WRITTEN BY J. SMITH FEB 1970
 10     READ (5, 20) DIAM
 20     FORMAT (F10.3)
C       TEST FOR END OF JOB (DIAM=0)
        IF (DIAM.EQ.0.0) STOP
C       CALCULATE PERIMETER AND AREA AND PRINT RESULTS
        RAD=(DIAM*.5)
        PERM=3.1416*RAD*2
        AREA=3.1416*RAD**2
        WRITE(6, 30) DIAM, PERM, AREA
 30     FORMAT (' DIAM', F10.3, ' PERM', F10.3, ' AREA',
        1F10.3//)
C       RETURN TO READ MORE DATA
        GO TO 10
        END
***END
*EXEC<cr>
[EDIFIL Filed : DSKD:FRTRN.FOR]
[EDICLF Closed log file: SAFETY.EDI]
```

Compilation and execution of the program now produces:

```
FORTRAN:  FRTRN.FOR
LINK:   Loading
[LNKXCT FRTRN Execution]
10.0<cr>
DIAM    10.000 PERM    31.416 AREA    78.540


5.0<cr>
DIAM     5.000 PERM    15.708 AREA    19.635


0.0<cr>

END OF EXECUTION
CPU TIME: 0.18  ELAPSED TIME  9.46
EXIT

.
```

APPENDIX A

LISTING OF PROGRAM 'SAMPLE.FOR'

This is a listing of the program SAMPLE.FOR:

```
C       THIS IS A SAMPLE PROGRAM
C       PROGRAM CALCULATES THE SQUARES OF INTEGERS 1 TO 50
C       PRINT HEADINGS
        LP=6
        WRITE (LP, 10)
 10     FORMAT ('1   NUMBER     SQUARE'/)
C       CALCULATE SQUARE, PRINT NUMBER AND SQUARE
        DO 30 I=1, 50
        J=I*I
 30     WRITE (LP, 20), I, J
 20     FORMAT (I8,I12)
        STOP
        END
```

APPENDIX B

SUMMARY OF COMMANDS AND SWITCHES

B.1 COMMANDS

command	abbr	general form of command
ALLCHANGE	A	ALLCHANGE ±n /string1/string2/ Changes every occurrence of string1 in a line to string2. Starts at current line for ±n lines.
APPEND	AP	APPEND ±n /string/ Inserts the string enclosed in any delimiters at the end of the line. Starts at current line for ±n lines.
BACK	BA	BACK ±n Move line pointer back to its position before the last command plus or minus n lines. BACK does not affect REPEATing.
BOTTOM	B	BOTTOM Moves line pointer to last line in file.
CHANGE	C	CHANGE ±n /string1/string2/ Changes the first occurrence of string1 in a line to string2. Starts at current line for ±n lines.
CONCAT	CO	CONCAT CONCAT - Concatenates the current line and the next line in either direction.
DELETE	DEL	DELETE ±n DELETE ±/string/ Deletes ±n lines starting at current line. or Deletes lines up to (but excluding) line

containing the string. Starts at current line.

DINSERT DIN DINSERT +n
DINSERT ±/string/
Behaves as with Delete, but moves back up to the line before the deletion and places the Editor into input state to accept further data.

DISPLAY DIS DISPLAY +n
DISPLAY ±/string/
Used to display data upon display terminals. Same as PRINT except that Editor waits when the bottom of a page is reached and erases the screen and starts a new page when signalled. Type ^Q <XON> to continue displaying.

EXECUTE E EXECUTE filespec
Saves the current file as filespec, then executes the last COMPILE/LOAD/EXEC/DEBUG command.

FILE F FILE filespec
Saves the file and returns control to the monitor.

GET G GET filespec
Inserts file into the incore file. Insertion takes place after the current line.

H H H
Lists the commands and switches with no explanation.

HELP HE HELP
Types a full help text.

INSERT I INSERT line
INSERT
Inserts one or more lines. Insertion takes place after the current line.

JOBBALANCE JOBB JOBBALANCE
Types out the current job cost balance.

JOBTIME JOBT JOBTIME
Types out the total CPU time since login.

LABEL LA LABEL +label+n
Searches for a label (the label must start in column 1) up or down the file plus or minus a number of lines. No delimiters are used in the LABEL command.

LINE LI LINE n
Locates the line number n and moves the pointer to that line. If no argument is given, the current line number and length including <cr><lf> will be typed out.

LOCATE L LOCATE ±n /string/
Locates the line containing the nth occurrence of the string. Search begins at the next line, up or down the file.

MACRO M MACRO filespec
Create a file of commands or any text without disturbing the current file.

NEXT N NEXT ±n
Moves the line pointer down ±n lines.

OPRINT O OPRINT ±n
Types out ±n lines starting at current line, in octal format.

PERFORM PER PERFORM n filespec
Perform the commands from the specified file n times.

PREPEND PRE PREPEND ±n /string/
Inserts the string enclosed in any delimiters at the start of the line. Starts on current line for ±n lines.

PRINT P PRINT ±n
PRINT ±/string/
Types out ±n lines starting at current line.
or
Types out all lines up to (and including) the line containing the string.

PUT PUT PUT ±n filespec
PUT ±/string/ filespec
Puts all lines up to (but excluding) line containing string, into a file. Starts at current line.

PUTDEL PUTD PUTDEL ±n filespec
PUTDEL ±/string/ filespec
Same as PUT except that it deletes the corresponding section from the incore file.

QUIT Q QUIT
Returns control to the monitor without keeping the current file. Closes the log file.

REPEAT RR REPEAT n
Repeat the last command line n times (REPEAT is not repeatable).

REPLACE R REPLACE line
Replaces current line with the one typed in.

RUN RU RUN filespec
Saves the current file as filespec, then runs the program specified in the last SET RUN command or /RUN switch.

SAVE SA SAVE filespec
Saves the file without returning control to the monitor. Pointer remains at the current line. Closes and reopens the log file.

SEARCH S SEARCH ±n /string/
Searches for line containing string. Starts searching at next line for ±n lines.

SPLIT SP SPLIT ±/string/
SPLIT ±n
Split the line after the string or the n'th character placing the rest of the line as a new line after (before) the current line.

STATEMENT ST STATEMENT ±label±n
Searches for Fortran type statements (labels anywhere in the first 5 columns of the line with leading or trailing blanks) plus or minus n lines. No delimiters are required.

TOP T TOP
Moves pointer to dummy top line.

TRANSLATE TR TRANSLATE ±n
Translates n lines from Ø26 to Ø29 characters.

UP U UP ±n
Moves pointer up ±n lines.

B.2 SET COMMANDS AND EDIT SWITCHES

EDIT switches and file switches are given after the CREATE or EDIT filespec. EDIT switches may be included in SWITCH.INI in a QEDIT line if you wish to change the default settings permanently. See Section 8.8 for a SWITCH.INI example.

command	abbr	command form	switch form
SET BRIEF	SET B	SET BRIEF	/BRIEF
		suppress the usual typeout unless temporarily overridden by "." after the command.	
SET VERIFY	SET V	SET VERIFY	/VERIFY
		typeout current line unless temporarily suppressed by "." after the command (default).	
SET NOBRIEF	SET NOB	SET NOBRIEF	/NOBRIEF
		same as SET VERIFY.	
SET NOVERIFY	SET NOV	SET NOVERIFY	/NOVERIFY
		same as SET BRIEF.	
SET FLAG	SET F	SET FLAG	/FLAG
		cause flagging of control characters and lower case characters printed. "PRINT." overrides "SET FLAG".	
SET NOFLAG	SET NOF	SET NOFLAG	/NOFLAG
		turn off flagging (default). "PRINT." overrides "SET NOFLAG".	
SET LOG	SET L	SET LOG filespec	/LOG:filespec
		write all terminal input to a log file. Used to recover after a system crash or user error.	
SET NOLOG	SET NOL	SET NOLOG	/NOLOG
		close any open log file. The file may later be PERFORMed.	
SET MAXCOR	SET M	SET MAXCOR n	/MAXCOR:n
		expand the low segment limit to n Kwords. If n=Ø give some core statistics.	
SET PAGE	SET P	SET PAGE n	/PAGE:n
		set page mode and page size to n lines for display terminals. By default page mode is unchanged.	

SET NOPAGE SET NOP SET NOPAGE /NOPAGE
clear page mode. By default page mode is unchanged.

SET ULCASE SET U SET ULCASE /ULCASE
allow upper/lower case to match lower/upper case in search strings.

SET NOULC SET NOU SET NOULC /NOULC
enforce exact case matching in searches (default).

SET VERBOSITY SET VERB SET VERBOSITY x /VERBOSITY:x
where x is one of NONE, LOW, FIRST, STANDARD, CONTIN, PRECON, FIRCON, HIGH - applies to most error and informative messages.

SET RUN SET R SET RUN filespec /RUN:filespec
set a program to run on a RUN command.

SET RUNOFFSET SET RUNO SET RUNOFFSET n /RUNOFFSET:n
offset from the normal start address of the program to be run.

B.3 FILE SWITCHES

switch	abbr	general form of switch
/DIRECT	/DI	/DIRECT:n Output as Fortran direct access, record size n.
/HELP	/H	/HELP:TEXT /HELP:SWITCH full help text or switches and commands.
/DELSEQ	/DE	/DELSEQ delete line sequence numbers from input file.
/NODELSEQ	/NOD	/NODELSEQ leave line numbers on input file, default for BASIC.

APPENDIX C

ASCII CHARACTERS AND THEIR OCTAL CODES

code	char	code	char	code	char	code	char
000	NUL	040	space	100	@	140	`
001	SOH	041	!	101	A	141	a
002	STX	042	"	102	B	142	b
003	ETX	043	#	103	C	143	c
004	EOT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	046	&	106	F	146	f
007	BEL	047	'	107	G	147	g
010	BS	050	(	110	H	150	h
011	TAB	051	)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FF	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	[	173	{
034	FS	074	<	134	\	174	|
035	GS	075	=	135	]	175	}
036	RS	076	>	136	^	176	~
037	US	077	?	137	_	177	DEL

APPENDIX D

INFORMATIVE MESSAGES

D.1 EDITING MESSAGES

Messages enclosed in brackets are informative, those preceded by % are warnings, and those preceded by ? indicate serious errors.

%EDIAMB Ambiguous command
abbreviation was non-unique. Type H for list of commands.

[EDICLF Closed log file: filespec]
response from SET NOLOG, SET LOG, QUIT, FILE, etc. or a fatal error.

[EDICMF Closed macro command file: filespec]
end of MACRO command input state, back to edit state.

?EDIDLT Direct access line too big - FILEing discontinued
edit the line or FILE with a greater /DIRECT:value.

%EDIDPF Error deleting paging file
the temporary work file could not be deleted due to some system fault.

%EDIFFI Found file in [path]
[CREATE] a file of this name exists in the library path or
[EDIT] the file was read from a library or ersatz device. No backup will be provided.

[EDIFIL Filed : filespec]
filed successfully.

%EDIIND Invalid delimiter: c
illegal delimiters include "+", "-", ".", ";", "#", "*" and digits.

%EDIIVS Illegal value: *
[LOCATE * or PERFORM *] see Section 3.1.3 or Section 6.3.

[EDILDL Lines deleted: n]
for a deletion limited by the occurrence of a string.

[EDILFR Log file restarted]
after a SAVE the log file is restarted at the begining.

[EDILND Line numbers deleted]
input file had special line sequence numbers used by BASIC and some editors. /NODELSEQ prevents such deletion.

%EDILTL No change - line too long
lines may not be changed to exceed the limit of 300

characters.

[EDIMCR n in-core pages, n MAXCOR with nK low core allows n pages]
information given in response to SET MAXCOR Ø.

%EDINAC Not a command - type "H" or "HELP"
"H" gives a list of commands, "HELP" gives an explanation.

%EDINAD Not a digit: c
LABEL nnn+d, or STATEMENT nnn+d, where d is not a digit.

?EDINFS No file specified
FILE, SAVE etc. commands with no file specified and no default available.

%EDINJT No job cost tables in monitor
only the University of Queensland monitor has the accounting features required.

%EDINLC No label in command
[LABEL or STATEMENT] an undelimited label is required.

[EDINOC No change]
string to be changed is not in the current line. See also the SET ULCASE command.

?EDINPU No PATH UUO
needs a modern version of the monitor.

%EDINSC No string in command
CHANGE, ALLCHANGE, SEARCH and LOCATE require a delimited string.

%EDINSL No such line: n
[LINE n] n is negative or greater than the number of lines in the file.

%EDINTF Nothing to file
[PUT or PUTDEL] The current line is the dummy end line.

%EDINTR Nothing to repeat
[REPEAT] No preceding repeatable commands. SET, SAVE, BACK and REPEAT are not repeatable. INSERT DINSERT, PERFORM and MACRO leave nothing to repeat.

%EDIPAB Perform aborted
For safety even warnings abort PERFORMs.

?EDIRPF Error reading paging file
a file system error. Recoverable only by performing the log file, if any.

[EDISAV Saved : filespec]
successful SAVE.

%EDISEF Superseding existing file
[CREATE] the existing file will be lost if no file name is given on the FILE command.

%EDISNF Not found
SEARCH over n lines did not find the specified string.

%EDITMP Too many nested PERFORM commands
PERFORMs may be nested to a depth of 5. A PERFORM which performs itself will give this.

?EDIWPF Error writing paging file
a file system error. Recoverable only by performing the log file.

%EDIWNF Writing a null file
FILE, SAVE, RUN or EXECUTE with no lines to write creates an empty file.

% Minimum abbreviation for DELETE is DEL
a safety feature to prevent accidental deletion of lines.

D.2 FILE SPECIFICATION SYNTAX ERRORS

Error Id	Description
EDIASW	ambiguous switch name
EDIILC	illegal character in input string
EDIIPG	programmer number zero or too big
EDIIPJ	project number zero or too big
EDINDV	Null device name in device specification
EDINOR	number out of range
EDIUKW	unknown keyword in switch specification
EDIUSW	unknown switch name
EDIVIL	value illegal in switch specification
EDIWDV	wild device name in device specification
EDIWFN	wild file name in file specification
EDI2DV	two device names in one file specification
EDI2EX	two extensions in one file specification
EDI2NM	two file names in one file specification
EDI2PT	two paths specified in file specification

D.3 FILE INPUT/OUTPUT ERRORS

Error Id	Description
EDIAEF	File already exists
EDIBKT	block too large, quota exceeded or disk full
EDIDER	device error

EDIDNA	device not available
EDIDTE	data error, e.g. checksum or parity error
EDIEOF	end of file
EDIFBM	file being modified
EDIFNF	file not found
EDIIPP	no directory
EDIIMP	improper IO mode for this device
EDILVL	too many levels of SFDs
EDINCE	can't create in directory
EDINEC	not enough core
EDINET	not enough table space in monitor
EDINFC	no free IO channels
EDINRM	no room on structure
EDINSD	no such device
EDIPRT	protection failure
EDIQTA	quota exceeded
EDIRSD	restricted device
EDISLE	search list empty
EDISNF	no subfile directory
EDITRN	transmission error
EDIWLK	write-locked device

APPENDIX E

GENERAL COMMENTS

E.1 CONTROL-O

Typeout from the program can be stopped at any time by typing ^O (depress CTRL key while striking O). This suppresses all printout from the program but will not suppress the '*' which indicates that the program is still in edit state. It is therefore possible to ascertain when the Editor is ready to receive additional data.

E.2 CONTROL-U AND RUBOUT

Typing ^U (depress CTRL key while striking U) can be used to erase any characters the user has typed in - back to the last break character. Break characters are <cr>, <lf>, <bel>, <vt>, <ff>, <esc> and ^Z.

example:

```
*RPLACE ^U
REPLACE <tab>LP=7<cr>
*
```

If only a few characters are incorrect and need to be retyped, these can be erased by typing the RUBOUT or DEL key until the appropriate number of characters has been removed. Those characters that have been rubbed out are bounded by '\'.

example:

```
*REPACE\ECA\LACE <tab>LP=7<cr>
*
```

E.3 CONTROL-R

After a line has been corrected using rubout, typing Control-R will retype the corrected line.

example:

```
*REPACE\ECA\LACE^R
REPLACE<tab>LP=7<cr>
*
```

E.4 CONTROL-S AND CONTROL-Q

When page mode is set either by the monitor command SET TTY PAGE or by the Editor command SET PAGE, typeout can be suspended by typing ^S <XOFF>. Output is resumed when ^Q <XON> is typed.

E.5 TREATMENT OF TAB CHARACTER BY COMPILERS

E.5.1 FORTRAN Compiler

It is worth noting the special treatment by the FORTRAN compiler of the tab character. The first tab character of a FORTRAN source statement, if it occurs before the 6th character of the line, will advance to column 7 if followed by a non-numeric character, or to column 6 if followed by a numeric character. Subsequent tab characters will index to the next multiple of 8 characters as normal. This feature enables one to create normal FORTRAN statements by preceding the body of the statement by a tab. A continuation of the statement is likewise created by a tab followed by a numeric continuation character. The feature is designed to simplify the creation of FORTRAN source programs and it must be emphasized that it is a property of the FORTRAN compiler, not the Editor which does not in any way differentiate between the first and subsequent tabs, or in fact between any other logical characters.

E.5.2 COBOL Compiler

The format of a COBOL program is organized so that division names, paragraph names etc. begin at column 1 (if there are no program sequence numbers) or column 8 (if the program has sequence numbers). This is referred to as the A margin.

The body of the program appears in the B margin, 4 spaces after the A margin. However, if a <tab> is typed anywhere within the A margin area, the compiler will treat the remaining data on the line as starting from the B margin, saving the programmer the labourious necessity of typing the individual spaces required to format the program correctly.

INDEX

Allchange (command) 3-12, B-1
Append (command) 3-13, B-1
Ascii 1-2, 3-7, 3-8, 8-2

Back (command) 3-7, 6-4, B-1
Basic language 8-3
Bottom (command) 3-2, B-1
Break character 1-3, 3-2, 3-9, E-1
Brief (set command or switch) . . 5-1, 6-2, B-5

Change (command) 3-12, 6-1, B-1
Check-point 6-1
Cobol compiler E-2
Command 3-1
Concat (command) 3-14, B-1
Concatenating lines 3-14
Continue (monitor command) 4-2, 7-1
Control character 1-3, 3-7, 5-3
Control-c 4-2, 6-2, 7-1, 8-2
Control-o E-1
Control-q 8-2, E-2
Control-r E-1
Control-s E-2
Control-u 6-1, E-1
Create (monitor command) 1-2, 2-1, 4-2, 5-1, 6-1, 8-1
Creating files 2-1

Data files 4-3
Delete (command) 3-10, B-1
Delimiter 3-1, 3-11, D-1
Delseq (switch) 2-1, 8-3, B-6
Dinsert (command) 3-12, 6-4, B-2
Direct (switch) 8-2, B-6
Direct access 8-2, B-6
Display (command) 5-2, 8-1, B-2

Edit (monitor command) 1-2, 2-2, 4-2, 5-1, 6-1, 8-1
Ersatz device 8-3, D-1
Execute (command) 4-2, 6-1, B-2
Extension 1-2

File (command) 2-1, 4-2, 6-1, B-2
Filename 1-2
Filespec 1-2, 2-2, 3-2, 4-1
Flag (set command or switch) . . . 3-8, 5-2, 8-1, B-5
Fortran compiler E-2

Get (command) 4-4, B-2

Help (command) 3-1, B-2
Help (switch) 2-1, B-6

Insert (command) 3-9, 3-12, 6-4, B-2

Jobbalance (command) 7-1, B-2
Jobtime (command) 7-1, B-2

Label (command) 3-5, B-2
Library 8-3, D-1
Line (command) 3-6, B-3
Line length (see also maximum) . . 3-6
Locate (command) 3-3, 3-5, B-3
Log (set command or switch) . . . 4-1, 4-2, 6-1, B-5
Login (monitor command) 5-3, 7-1, 8-3
Lower case 3-7, 5-3, 8-1

Macro (command) 6-3, 6-4, B-3
Maxcor (set command or switch) . . 8-4, B-5
Maximum command line 6-4
Maximum file size 1-2
Maximum line length 1-2, 3-9, 3-12, 3-15

Next (command) 3-3, B-3
Nobrief (set command or switch) . B-5
Nodelseq (switch) 2-1, 8-3
Noflag (set command or switch) . . 5-2, B-5
Nolog (set command or switch) . . 6-1, B-5
Nopage (set command or switch) . . 8-1, B-6
Noulc (set command or switch) . . 5-3, B-6
Noverify (set command or switch) . B-5

Octal 3-8
Oprint (command) 3-8, B-3

Page (set command or switch) . . . 8-1, B-5
Page size 8-1
Perform (command) 4-1, 6-1, 6-2, 6-4, B-3
Prepend (command) 3-13, B-3
Print (command) 3-7, 5-2, 5-3, 8-1, B-3
Privacy of files 8-3
Protection 8-3
Put (command) 4-3, B-3
Putdel (command) 4-3, B-3

Quit (command) 2-1, 4-2, 6-1, B-3

Reenter (monitor command) 4-2, 6-2, 8-2
Repeat (command) . . . . . 3-7, 3-9, 6-2 to 6-4, B-4
Replace (command) 3-11, 6-1, B-4
Return 1-3, 3-2, 3-9
Rr (command) 6-3, B-4
Rubout 6-1, E-1

Run (command) 4-3, 6-1, B-4
Run (set command or switch) . . . 4-3, B-6
Runoffset (set command or switch) 4-3, B-6

Save (command) 4-1, 6-1, 6-4, B-4
Search (command) 3-4, B-4
Sequence number 8-3
Set (command) 2-1, 6-4
Set flag (command) 5-2, 5-3, 8-1
Set log (command) 6-1
Set maxcor (command) 8-4
Set noflag (command) 5-2, 5-3
Set nolog (command) 6-1
Set nopage (command) 8-1
Set noulc (command) 5-3
Set page (command) 8-1
Set run (command) 4-3
Set runoffset (command) 4-3
Set ulcase (command) 5-3, 8-1
Set verbosity (command) 5-2
Split (command) 3-14, B-4
Statement (command) 3-5, B-4
String 3-1, 3-3, 3-5, 3-12
Switch 1-3, 2-1, 2-2, 3-2
Switch.ini 2-1, 5-3, 6-2, 8-5, B-5

Tab 1-3, 3-9, 3-14, E-2
Top (command) 3-2, B-4
Translate (command) 8-2, B-4
Typeout 5-1, 5-3, E-1, E-2

Ulcase (set command or switch) . . 5-3, 8-1, B-6
Up (command) 3-3, B-4
Upper case 5-3

Verbosity (set command or switch) 5-2, B-6
Verify (set command or switch) . . 5-1, B-5

MNT-6/ed6
16Mar82

COMMENTS SHEET

Title of Publication : EDIT - A LINE EDITOR FOR THE PDP-1Ø

The Computer Centre welcomes any suggestions that will assist in improving their publications. Please comment on the usefulness and readability of this document. Suggest additions and deletions and indicate any specific errors and omissions. Please provide page or section references where relevant.

Name :

Position :

Address :

ERRORS

COMMENTS

Please return to The Librarian,
Prentice Computer Centre,
University of Queensland.