UNIVERSITY OF CAMBRIDGE
COMPUTING SERVICE

ZED REFERENCE MANUAL

University of Cambridge
Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
England
Telephone Cambridge (0223) 334600

—dvj—

UNIVERSITY OF CAMBRIDGE COMPUTING SERVICE

ZED
REFERENCE
MANUAL

P. HAZEL

November 1984
(Chapter 4 updated March 1988)

CONTENTS

1. INTRODUCTION

ZED is a text editor that processes sequential files line by line under the control of *editing commands*. As ZED moves through the input, or *source* file, lines which have been passed (and possibly altered) are written out to a sequential output file, the *destination*. The effect of a ZED run is therefore to make a copy of the source, with changes as requested by the editing commands.

Although the source is normally processed in a forward sequential manner, there is a capability for moving backwards a limited number of lines. This is possible because lines which have been passed are not written to the destination immediately, but are held instead in an output queue whose maximum length depends on the amount of available store. Normally several hundred lines can be held in store at once.

If the destination is a disc file, or if the source file is sufficiently small that it can be held entirely in main store, it is possible to make more than one pass through the text.

ZED commands are available for moving parts of the source about (using in-store buffers), for outputting parts of the source to alternative destinations, and for inserting material from alternative sources.

Most users will find that the UPDATE command provides the most convenient way of using the editor.

New users of the ZED editor should consult Chapter 9 of the Newcomers' Guide and Chapter 11 of the Users' Guide.

1.1 CALLING ZED FROM PHOENIX

The Phoenix macro command ZED is used to call the ZED program. The syntax for use with Phoenix is

```
ZED [[FROM] dsd] [TO dsd] [WITH dsd] [VER dsd] [LIST dsd]
   [EPRINT dsd] [WORK dsd] [OPT text] [DD text]
```

The syntax of the UPDATE command is:

```
UPDATE dsd [WITH dsd] [VER dsd] [LIST dsd] [EPRINT dsd]
   [OPT text] [NOPROMPT] [DD text]
```

There are also STORE and TIME keywords with their usual meanings. Many of the keywords are common to both commands.

The FROM dataset defines the file to be edited (ZED command only). If the FROM argument is omitted, the current dataset (%C) is used. Note that FROM is the only item whose key may be omitted.

The TO dataset defines the file to which the output of ZED (that is, the result of editing FROM) is to be sent. If the TO argument is omitted, the output dataset (%O) is used. This will become the new current dataset on exit from ZED.

The WITH dataset defines the file containing the editing commands. If the WITH argument is omitted ZED will read from the Phoenix terminal in a session; in a job no editing will take place (unless OPT is set, see below).

The VER dataset defines the file to which error messages and line verifications are to be sent. If the VER argument is omitted, the Phoenix terminal is used in a session and the line printer in a job.

The LIST dataset defines the file to which a copy of every output line is sent. Each edited line is preceded by its original (source) line number, inserted lines are preceded by four asterisks, and changes and deletions are indicated (see section 3.3.9 for further details). If the LIST argument is omitted, no listing takes place.

The EPRINT dataset defines the file to which a copy of every line of editing commands is sent. If the EPRINT argument is omitted no record of editing command lines is kept.

The OPT item can be used to provide a line of ZED commands to be executed before those in the WITH file. If omitted it defaults to null.

The WORK dataset is used by ZED when multiple passes are made through a source text which is too big to hold entirely in main store. The default is %N (i.e. no allocation), as ZED allocates work datasets dynamically when required if they are not provided.

The DD argument can be used to add additional information to the environment in which ZED is to run. The most usual use is to set up (arbitrary) ddnames for additional datasets in more sophisticated editing applications. The NOPROMPT keyword (UPDATE only) may be used to stop the system asking the user whether he wants the edited version of the dataset filed back to the original dataset.

Examples of UPDATE calls:

```
UPDATE .FILE
```

This updates the dataset .FILE. When the editing session is concluded (by the user typing Q or W) the system prompts with a message:

```
Refile as .FILE (yes/no)?
```

If the user types no at this point, the system issues a message:

```
Type in a new file name, or QUIT to discard edited version:
```

Examples of ZED calls:

```
ZED
```

This uses all the standard files, and edits the current dataset to make a new current dataset. Editing commands are read from the terminal if in a session; in a job no editing is done.

```
ZED SPQR.FILE
```

The file SPQR.FILE is edited to form a new current dataset. Again, editing commands are taken from the terminal if in a session. Note that this command does not cause any changes to be made to SPQR.FILE; the result of editing is left as the new current dataset. If updating of SPQR.FILE is required then an additional Phoenix command

```
FILE TO SPQR.FILE
```

is needed after exiting from ZED.

```
ZED ABCD.PDS:A WITH %H! TO ABCD.NEW
<editing commands>
!
```

This edits ABCD.PDS:A non-interactively, using the editing commands supplied as an in-stream dataset. The output from ZED is written to the file ABCD.NEW.

1.2 CALLING ZED FROM JCL

The catalogued procedure ZED is used to call the ZED program. It contains the following parameters:

name	**default**	**use**
OPT	''	initial command line
VER	SUBSYS=#PRC	verification output
STORE	100K	region size

The following DD statements must be supplied by the caller:

FROM	input file
TO	output file
EDITS	editing commands file

The following DD statements may be supplied if either of the listing facilities is required:

LIST	output listing
EPRINT	edits listing

Additional ddnames (of the user's choice) may be required for sophisticated applications. Example of a ZED call:

```
//     EXEC ZED
//FROM  DD   DSN=SPQR.FILE,DISP=SHR
//TO    DD   DSN=&TEMP,DCB=FB,DISP=(NEW,PASS),
//           UNIT=DISC,SPACE=(TRK,(1,1))
//EDITS DD  *
<editing commands>
/*
```

2. BASIC ZED COMMANDS

This section introduces some of the basic ZED commands, omitting many of the advanced features. A complete description of the command syntax and of all commands appears in section 3.

2.1 The Current Line

As ZED reads lines from the source and writes them to the destination, the line which it has "in its hand" at any time is called the *current line.* This is the line to which all textual changes are made. New lines are always inserted before the current line. When ZED is initialized, the current line is the first line of the source.

2.2 Line Numbers

Each line in the source is identified by a unique line number. This is not part of the information stored in the file, but is computed by counting the lines as they are read. (Note that the LIST utility program is available to give printouts of text files with line numbers.) A line that has been read retains its original line number all the time it is in main storage, even when lines before or after it are deleted.

2.3 Selecting a Current Line

2.3.1 By Line Counting

The simplest way to select a new current line is to specify its line number. The command

```
M45
```

means *move to line 45.* If line 45 has already been passed this command moves back to it, provided that it is still in main storage (normally several hundred lines are accessible). For moving on a known number of lines, the command

```
N
```

is used. It means *move to the next line.* It can be preceded by a count to indicate repetition. Thus

```
4N
```

means *move on four lines.* The sequence

```
M12; 3N
```

would have the effect of making line 15 the current line. Note that a semicolon is used to separate consecutive commands on the same line.

The command

```
P
```

moves to the previous line and, like N, may be used with a repeat count to move back a given number of lines. It is only possible to go back to lines which have been passed but which have not yet been written to the output.

2.3.2 By Context

The F (find) command is used to select a current line by context.

```
F/Jabberwocky/
```

means *find the line containing* 'Jabberwocky'. The search starts at the current line and moves forward through the source until the required line is found. If the end of the source is reached without finding a matching line, the error message `SOURCE EXHAUSTED` is given.

It is also possible to search backwards by using the BF command.

```
BF/gyre and gimble/
```

This also starts with the current line, but moves backwards through lines already passed until the required line is found. If the head of the output queue is reached without finding a matching line, the error message `NO MORE PREVIOUS LINES` is given.

The character string being searched for is enclosed in *delimiter characters*; in the example above / was used as the delimiter. A number of special characters such as : / . , and ' are available for use as delimiters; naturally the string itself must not contain the delimiter character. Spaces between the command name and the first delimiter are ignored, but spaces within the string are of course significant, since the context is matched exactly.

```
F /tum tum tree/
```

will not find "tum-tum tree" or "tum tum   tree".

A find command with no argument causes a repeat of the previous find. Thus

```
F/jubjub bird/; N; F
```

finds the second occurrence of a line containing "jubjub bird". The N command between the two F commands is necessary because an F command always starts by searching the current line. If N were omitted the second F would find the same line as the first.

The basic form of the F command described above will find a line which contains the given string anywhere in its length. The search can be restricted to the beginning or the end of lines by placing one of the letters B or E in front of the string. In this case, one or more spaces must follow the command name. Thus

```
F B/slithy toves/
```

means *find the line Beginning with* 'slithy toves', while

```
F E/bandersnatch/
```

means *find the line Ending with* 'bandersnatch'. As well as putting further conditions on the context required, the use of B or E speeds up the search, as only part of each line needs to be considered.

B and E as used above are called *qualifiers*, and the whole argument is called a *qualified string*. A number of other qualifiers are also available:

```
F 3/dormouse/
```

means *find the line containing at least three occurrences of the string* 'dormouse'. Any unsigned number other than zero may be used, and is termed a *count qualifier*.

```
F P/a-sitting on a gate./
```

means *find the line which contains Precisely the text* 'a-sitting on a gate.'. The required line must contain no other characters, either before or after the given string.

An empty string can be used with the P qualifier to find an empty line:

```
F P//
```

The qualifier N changes the sense of the command so that it finds a line *not* containing the given context. For example

```
F NP//
```

finds a non-blank line, while

```
F NB/The sun was shining/
```

finds a line that does *not* begin "The sun was shining". Note that this is not the same thing as a line containing "The sun was shining" not at the beginning.

As the examples above show, more than one qualifier may be given. They may be in any order, with optional spaces between them. Certain incompatible combinations are however forbidden, and in particular only one of B, E and P may appear.

The qualifier W specifies a *Word search*, which means that the found string must neither be preceded nor followed by a letter or digit, while the qualifier U specifies that the search is to take place as though both the string and each line scanned were *Uppercased.* Thus

```
F UW/king/
```

will find lines containing the words "king", "KING" or even "KiNg", but not "kingpin" or "thinking".

The qualifier S specifies that lines are to be considered as starting at the first *Significant* (i.e. non-space) character. This qualifier is usually used in conjunction with B or P. For example

```
F SB/grassy/
```

finds any of the lines

```
grassy
   grassy knolls
      grassy
```

whereas

```
F SP/grassy/
```

finds only the first and the last of them.

2.4 Making Changes to the Current Line

The E command *Exchanges* one string of characters in the line for another:

```
E/Wonderland/Looking Glass/
```

removes the string "Wonderland" from the current line, and replaces it by "Looking Glass". Note that a single central delimiter is used to separate the two strings. A null second string can be used to delete parts of the line:

```
E/monstrous crow//
```

New material can be added to the line by the A or B commands, which insert the second of their arguments *After* or *Before* an occurrence of the first. If the current line contained

```
If seven maids with seven mops
```

then the command sequence

```
A/seven/ty/; B W/seven/sixty-/
```

would turn it into

```
If seventy maids with sixty-seven mops
```

If the W qualifier were omitted from the B command above, the result would be

```
If sixty-seventy maids with seven mops
```

because the search for a string normally proceeds from left to right, and the first occurrence that is found is used. There is however a qualifier L, which specifies that the search proceed *Leftwards*, thus causing the command to act on the *Last* occurrence of its first argument. The B command in the above example could have been written

```
B L/seven/sixty-/
```

to achieve the same effect.

If the first string in an A, B or E command is empty, the second string is inserted at the beginning or the end of the line, according as the L or E qualifiers are absent or present.

A count qualifier may be used with the A, B and E commands to change the *n*th occurrence of a string in the current line. If the L qualifier is also present then the *n*th last occurrence is changed. For example, the command line used in the above example could be written

```
A 2L/seven/ty/;  B 2/seven/sixty-/
```

to achieve the same effect.

If an A, B or E command is obeyed on a line which does not match the qualified string given as the first argument, the error message

```
**No match
```

is output on the verification dataset.

2.5 Deleting Whole Lines

A range of lines may be deleted by specifying their line numbers in a D command.

```
D97 104
```

deletes lines 97 to 104 inclusive, leaving line 105 as the new current line. The second line number may be omitted if only one line is being deleted, and both numbers may be omitted if deletion of the current line is required.

```
F/plum cake/; D
```

deletes the line containing "plum cake", the line following it becoming the new current line, while

```
F BW/The/; 4D
```

deletes four lines, the first of which is the line beginning with the word "The".

The characters . and * may be used in place of line numbers to refer to *the current line* and *the end of the source* respectively.

```
D. *
```

deletes the rest of the source, including the current line.

2.6 Inserting New Lines

A single new line can be inserted *before* the current line by means of the IS (insert string) command. It takes a single string argument, which becomes the inserted line. For example, the commands

```
M17; IS/They would not listen to advice./
```

insert a new line containing the text "They would not listen to advice." *before* line 17, while

```
F B/The Queen of Hearts/; IS//
```

insert a new blank line before the line beginning with "The Queen of Hearts".

For longer inserts, the I command can be used to insert any number of lines of new material *before* a given line, specified by line number:

```
I468
The little fishes of the sea,
They sent an answer back to me.
Z
```

The command itself must be the last on a line, the insertion material following on successive lines, terminated by Z on a line by itself. If the line number is omitted from the command, the new material is inserted before the current line.

```
F/corkscrew/; I
He said, "I'll go and wake them, if..."
Z
```

inserts a new line before the line containing "corkscrew".

After an I command containing a line number, the current line is the line of that number; otherwise the current line is unchanged.

I may be followed by * instead of a line number to insert material at the end of the source.

There is also a *Replace* command, R, which is exactly equivalent to D followed by I, and is provided purely for convenience:

```
R19 26
In winter when the fields are white
Z
```

deletes lines 19 to 26 inclusive, then inserts the new material before line 27, which becomes the current line.

2.7 Command Repetition

Individual repeat counts as shown in the examples for N and D above can be used with many ZED commands. In addition, a collection of commands can be repeated by forming them into a *command group* using brackets:

```
6(F P//; D)
```

deletes the next six blank lines in the source. Command groups may extend over more than one line of command input:

```
3(F B/    /; I

Z
N)
```

inserts a blank line before the next three indented lines. When a command group such as the above is typed in interactively, the normal ZED prompting character (which is a colon) is replaced by a plus for the second and subsequent lines of the group. Command groups may be nested to any depth.

2.8 Leaving ZED

The command W (for *Windup*) is used to end a ZED run. It causes ZED to *wind through* to the end of the source, copying it to the destination, and then to exit.

The command Q causes ZED to stop obeying commands from the current command dataset and to revert to the previous one. If there is no previous dataset (i.e. Q is encountered on the WITH dataset) its effect is the same as W.

Reaching the end of the WITH dataset has the same effect as Q. Note, however, that a line containing /* does not terminate a WITH dataset connected to a terminal. In this case, Q or W must be used.

The command STOP causes ZED to terminate without any further processing. In particular, any pending lines of output still in store will not be written out, with the result that the destination file will not be complete. STOP should only be used if the output from the ZED run is not required.

2.9 A Combined Example

Most simple editing requirements can be met with the commands already described. Before going on to describe further facilities, an example which uses several commands is presented. Text following a backslash character in the editing commands is comment; this is allowed in actual command lines, though not in insert material or its terminator.

The source text (with line numbers):

```
1  Tweedledee and Tweedledum
2  agreed to a battle,
3  For Tweedledum said Tweedledee
4  ad spoiled his nice new rattle.
5
6  As black as a tar barrel
7  Which frightened both the heroes so
8  They quite forgot their quorell
```

The ZED commands:

```
M1; E/dum/dee/; E/dee/dum/               \the order of the
                                         \E commands matters!
N; E/a/A/; B W/a/have /                  \now at line 2
F B/ad/; B//H/                           \H at line start
F P//; F NP//; I                         \before non-blank line
Just then flew down a monstrous crow,
Z
M6; 2(A L//,/; N)                        \commas at end of lines
F/quore/; E/quorell/quarrel./            \F is in fact redundant
Q                                        \quit
```

The resulting destination text (with new line numbers):

```
1  Tweedledum and Tweedledee
2  Agreed to have a battle,
3  For Tweedledum said Tweedledee
4  Had spoiled his nice new rattle.
5
6  Just then flew down a monstrous crow,
7  As black as a tar barrel,
8  Which frightened both the heroes so,
9  They quite forgot their quarrel.
```

3. COMPLETE SPECIFICATION OF ZED

3.1 INTERFACE TO THE OPERATING SYSTEM

ZED supports the Cambridge Standard Utility Interface, which is documented elsewhere. When called directly from JCL or Phoenix this involves only a normal PARM field, which may contain a line of commands to be obeyed before the command file.

ZED is capable of processing any sequential file supported by MVS with the exception of VSAM files. It is hoped to remove this restriction in the future. Normally each record in the file is treated as one line for editing purposes. However, if a record is longer than 400 data characters it is split by ZED into two or more lines as necessary. This limit is a default which can be changed by use of the MXLL command (see section 3.4.20).

Source input from Phoenix (TSO) terminals is terminated by a line containing only the two characters /*. For command input however, such a line is treated as a genuine input line. The end of a ZED run is specified by one of the commands Q, W or STOP when commands are from a terminal.

The following ddnames are used by ZED:

VER

This ddname is used for the output of error messages, reflections, and line verifications. If it is absent in a Phoenix session, the output will be sent to the terminal. VER is not opened if it is not required.

EDITS

ZED commands are initially read from this dataset. WITH is an alternate keyword for this dataset in the Phoenix command ZED. If EDITS (or WITH) is not specified in the Phoenix command, the terminal is used online (i.e. EDITS=* is set up). If EDITS is absent it is treated as dummy.

FROM

This is the ddname for the main source file.

TO

This is the ddname for the main destination file. If, when opened, it has no DCB characteristics associated with it, those from the source will be copied.

WORK

This is the ddname for the work file, used when more than one pass is made through a source that does not fit in the available store. If it is absent when needed, ZED will allocate an anonymous file.

LIST

This is the ddname for the optional listing file, to which a listing of the output is sent (see section 3.3.9).

EPRINT

This is the ddname for the optional edits printing file, to which a copy of editing lines is sent (see section 3.3.9).

Other ddnames, chosen by the user, may be required in applications involving multiple datasets. When datasets are referred to by dsname, ZED allocated ddnames in the series DD1, DD2, ... as necessary.

3.2 COMMAND SYNTAX

ZED commands consist of a command name followed by zero or more arguments. One or more space characters may optionally appear between a command name and the first argument, between non-string arguments, and between commands. A space character is only necessary in these places to separate successive items which would otherwise be treated as one (e.g. two numbers).

A command is terminated by end of line (except in a few special cases), by reaching the end of its arguments, or by one of the characters semicolon, closing round bracket (parenthesis), backslash or cent sign. The MVS codes X'E0' and X'4A' are the standard EBCDIC codes for the backslash and cent sign characters. In export versions of ZED different characters or code values may be used.

Round brackets (parentheses) are used to delimit command groups (see section 3.2.3).

Semicolon is used to separate commands that appear on the same line of input, but it is only strictly necessary in cases of ambiguity where a command has a variable number of arguments. ZED always tries to read the longest possible command.

Backslash or cent sign introduce comment when they appear in places where newline may appear, that is, at the end of a command or adjacent to an operator in a search expression (see section 3.2.2.4). The comment is terminated by the end of the command input line.

Except where they appear as part of a character string, upper and lower case letters are synonymous in ZED commands.

3.2.1 Command Names

A command name is either a sequence of letters or a single special character (e.g. #). An alphabetic command name is terminated by any non-letter; only the first four letters of the name are significant. One or more spaces may appear between command names and their arguments; at least one space is required when an alphabetic name is followed by an argument starting with a letter.

3.2.2 Arguments

Seven different types of argument are used with ZED commands: strings, qualified strings, search expressions, names, numbers, switch values and command groups.

3.2.2.1 Strings

A string is a sequence of up to 256 characters, enclosed in *delimiters*. An empty (null) string is allowed. The character chosen to delimit a particular string may not appear in the string. The terminating delimiter may be omitted if it is immediately followed by the end of the command line.

The following characters are available for use as delimiters:

```
/ ! . + - , ? ' " : *
```

that is, common English punctuation characters (except ;) and the four arithmetic operators.

Some examples of strings:

```
/A/
!Menai Bridge!
??
+String with final delimiter omitted
```

In hexadecimal mode (see section 3.4.18) non-null strings must contains an even number of hexadecimal digits together with an arbitrary number of space characters (which are ignored).

3.2.2.2 Multiple Strings

Commands which take two string arguments use the same delimiter for both, and do not double it between the arguments. An example is the A command:

```
A /King/The Red /
```

For all such commands the second string specifies replacement text; if it is omitted the null string is used.

3.2.2.3 Qualified Strings

Commands which search for contexts, either in the current line or scanning through the source, specify them by means of *qualified strings*. A qualified string is a string preceded by zero or more *qualifiers*. The qualifiers are single letters, an unsigned number, or a pair of numbers enclosed in square brackets or underline characters and separated by a comma. They may appear in any order. For example:

```
BN[4,10]/Abc/
L2R_5,40_U/xyz/
```

Spaces may appear between the qualifiers, but are not necessary. No individual qualifier may appear more than once in a qualified string. The list of qualifiers is terminated by any delimiter character. The available qualifiers are described in section 3.3.6.

Square brackets are interpreted as the EBCDIC codes X'AD' and X'BD' at Cambridge. In export versions of ZED only the syntax using underline characters is normally used.

3.2.2.4 Search Expressions

Some conditional commands and commands which search for a particular line in the source take a *search expression* as an argument. A search expression is either a single qualified string, or a number of qualified strings separated by the AND and OR operators (which are & and |), and enclosed in round brackets. Nested brackets may be used to alter the normal precedence rule (& is more binding than |). Examples of search expressions:

```
B/The/
(B/The/ & N[2,99]/The/)
```

Spaces may be used anywhere to improve the layout, except of course in the strings. Search expressions may extend over more than one line of input, provided the line breaks occur immediately before or after an operator. Comment may occur on the intermediate lines. For example

```
(NB/*/ &                                \not a comment and
([10,15]B/LA/ | [10,15]B/L /)           \LA or L order
)
```

3.2.2.5 Names

A name is a sequence of letters terminated by any non-letter. Only the first four letters are significant.

3.2.2.6 Re-use of Last Search Expression

In commands which take a search expression or a qualified string as an argument, the character & may be given instead of that argument to specify re-use of the last obeyed search expression. In the case of a qualified string certain restrictions apply. Full details appear in section 3.5. The character & may only appear in place of a complete argument; it may not appear as part of a search expression. Some examples of the use of & are

```
F/dum/; N; F&
F/dee/; E&/dum/
IF (/cat/|/dog/) A&/fish/
```

3.2.2.7 Numbers

A number is a sequence of decimal digits. For some commands which take numerical arguments, the character + is permitted instead of a number. Its meaning depends on the context.

Line numbers are a special form of number and must always be greater than zero. Wherever a line number may appear, the characters . and * may appear instead. These are conventional ways of representing *the current line* and *the end of the source* respectively.

For the M (move) command only, there are two further conventional line number arguments, the characters + and -. These mean *the last line in store* and *the first line in store* respectively. Further details are given in section 3.4.2.

3.2.2.8 Switch Values

Commands which alter ZED switches take a single character as an argument. The character must be either + or -. For example

```
V+
```

The character + indicates that the switch is to be turned on, while - indicates that it is to be turned off.

3.2.3 Command Groups

Round brackets (parentheses) can be used to make a number of individual ZED commands into a *command group*. Command groups may span more than one line of input, and may be nested to any depth, subject only to available store. The conditional commands can take a command group as their final argument, for example

```
IF /hatta/ (E/walrus/carpenter/;
        E/oyster//)
```

3.2.4 Command Repetition

Many commands may be preceded by an unsigned decimal number to indicate repetition, for example

```
24N
```

If a value of zero is given the command will not be obeyed at all. Repeat counts are not allowed for commands for which repetition is pointless.

Repeat counts may be specified for command groups in the same way as for individual commands:

```
12(F/handsome/; E/handsome/hansom/; 3N)
```

Command repetition may also be achieved by means of the RPT command (see section 3.4.9.2).

3.3 ZED PROCESSING

3.3.1 Prompts

When ZED is being run interactively, that is, with both the command and the verification datasets connected to a terminal, it outputs a prompt when it is ready to read a new line of commands unless the last command of the previous line caused verification output.

If the current line has not been verified, or has been changed since it was last verified, or if the position of the operational window (see section 3.4.4.1) has changed, the current line is verified in place of a prompt if the V switch and either or both of the VN and VT switches are on (see section 3.4.16). Otherwise a colon character (:) is output to indicate that a new line of commands is required.

When a command group or search expression spanning more than one line is read from the terminal, a plus character (+) is used as a prompt for the second and subsequent lines.

However, prompts are never given for lines of insertion material, whether inside a command group or not.

In export versions of ZED the prompt characters may be different, and prompts may be given for inline insertion material.

3.3.2 Attention Handling

An *attention* can be used to interrupt ZED processing from an interactive session. The means of generating an attention varies from terminal to terminal; most commonly there is a key marked "BREAK" or "INTRPT". Certain types of network connection may require interaction with a network processor in order to send an attention signal to the host.

The system responds to an attention with the prompt

```
***ATTN (ZED):
```

The following strings are recognized by ZED when typed in response to the prompt after an attention:

V+ or V-

The verification switch is set or unset as appropriate, and processing continues. Pending lines of verification output may be lost.

VJ+ or VJ-

The justification verify switch is set or unset as appropriate and processing continues. Pending lines of verification output may be lost.

VG+ or VG-

The global change verify switch is set or unset as appropriate and processing continues. Pending lines of verification output may be lost.

Following V- VG- or VJ- one further line of verification may be produced before ZED detects the changed switch value.

<null>

This causes the current command line to be abandoned. If ZED's current command file is attached to an interactive terminal, a new line of commands is read. Otherwise ZED successively abandons command files until either an interactive one is found, or there are none left. In this case the ZED run is abandoned with a non-zero return code.

In non-interactive sequences of edits, the effect of an attention can be delayed by use of the BRK command (see section 3.4.20).

When abandoning a command line because of an attention, execution is stopped either immediately following a complete command or, for commands which move about the source, immediately after a new line has been made current. In particular, an attention will not be taken during forward movement until after all global operations (see section 3.4.13) have been applied to a new current line. In exceptional circumstances an erroneous global command may enter an unending loop; a second attention followed by a <null> response can be used to interrupt this condition. This may leave the current line with only some of the globals applied.

3.3.3 Command Decoding

ZED reads and decodes a complete line of commands before obeying any of them. In the case of command groups or search expressions spanning line boundaries, more than one line is read, and the phrase *line of commands* is extended to mean *sequence of commands terminated by end of line not within a command group or search expression.* The length of a command group is limited only by the available store. Syntax errors are reported during decoding, and before any of the commands have been obeyed. Commands following the one in error are not parsed.

3.3.4 The Current Line

As ZED reads lines from the source and writes them to the destination, the line which it has "in its hand" at any time is called the *current line.* This is the line to which all textual changes are made. New lines are always inserted before the current line. When ZED is initialized, the current line is the first line of the source. However, this line is not in fact read until a command which requires a current line is obeyed. This ensures that commands which affect the way in which lines are read (e.g. MXLL, TR) apply to all lines of the source if they appear at the start of the editing commands. A list of commands which do not cause the first line to be read is given in section 3.4.21.

There is an upper limit, MXLL, to the length of the current line. Any longer lines which ZED encounters are split into two or more lines. A warning message is generated. The value of MXLL defaults to 400, but it can be changed by the MXLL command (see section 3.4.20).

3.3.5 Line Numbers

Each line in the source is identified by a unique line number. This is not part of the information stored in the file, but is computed by counting the lines as they are read. Lines which are moved to a different place in the source retain their line numbers for verification purposes (the numbers are verified in parentheses), but are no longer accessible by explicit line number. New lines which are inserted into the source do not have line numbers.

ZED distinguishes between *original* and *non-original* lines. The former are source lines which have not been split or moved from their original order; the latter are split lines, moved lines, and inserted lines. Commands which take line numbers as arguments may only refer to original lines. As such lines are by definition in order of line number, the direction of motion implied by the command is obvious. Non-original lines are passed over or deleted (as appropriate) in searches for a given original line.

When . is used as a line number, it always refers to the current line, whether original or non-original.

The = command (see section 3.4.20) can be used to renumber lines in main storage. This ensures that all lines which follow the current line are original.

3.3.6 Semantics of Qualified Strings

Qualified strings are used to specify contexts for which ZED searches. The null string is permitted, and always matches at the initial search position, which is the beginning of the line except as specified below. In the absence of any qualifiers, the given string may be found anywhere in a line. Qualifiers specify additional conditions for the context, and the following are defined:

B

The string must be at the *Beginning* of the line. This qualifier may not appear with E, L or P.

E

The string must be at the *End* of the line. This qualifier may not appear with B, L or P. If E appears with the null string it matches at the end of the line.

C

The string must match the control character in the line. This qualifier requires that the string be exactly one character long, and may not appear with B, E, P, W, L, U, S, the window qualifier or R.

L

The search for the string is to take place *Leftwards* from the end of the line instead of rightwards from the beginning. If there is more than one occurrence of the string in a line, this qualifier ensures that the *Last* one is found instead of the first. L may not appear with B, E or P. If L appears with the null string it matches at the end of the line.

P

The line must match the string *Precisely* and contain no other characters. P must not appear with B, E or L. If P appears with a null string it matches an empty line.

W

The string consists of one or more *Words*, and must be preceded and followed (in the line) by a character which is not a letter or a digit, or by the start or end of line. It is possible to specify which characters are to be treated as "letters or digits" for this purpose by means of the WORD command (see section 3.4.20). The default set is A–Z, a–z and 0–9.

U

The string match is to take place as though both the string and the line were *Uppercased* before comparison.

S

The line is to be considered as starting at the first *Significant* (i.e. non-space) character, and any trailing spaces are to be ignored. This qualifier may not appear with C. Note that S does not imply B, as in some other text editors. Both SB and SP are useful combinations. If S appears with the null string it matches immediately after the last leading space character.

N

This qualifier *negates* the result of the string match after all the other context conditions have been considered. This effectively inserts the word *not* at the beginning of the description of the context. Thus, for example,

NB/unicorn/

means *not beginning with* 'unicorn' rather than *containing* 'unicorn' *not at the beginning.*

[n,m]   or   _n,m_

This qualifier specifies a *window*, that is, it gives character positions in a line within which the search for the string is to take place. It can also be thought of as specifying a column to be searched. The specification is inclusive. The first character position in a line is numbered 1. If *m* is greater than the number of characters in a line, then the line length is used instead. If *n* is greater than the number of characters in a line then the window contains the null text.

If *n* is omitted, a default of 1 is used. If *m* is omitted but the comma is present, a default of 32767 is used. If both *m* and the comma are omitted the window consists of a single character position.

If B, E, P or S are used in conjunction with a window, then they apply to the characters in the window rather than the whole line. For example,

B[10,16]/L/

matches a line containing L in column 10, while

```
P[36,]//
```

matches a line with no text after column 35.

If the W qualifier is used in conjunction with a window the edges of the window behave as *non letters or digits*.

A window specified for a qualified string overrides any operational or search window that may be in force (see section 3.4.4).

Square brackets are interpreted as the EBCDIC codes X'AD' and X'BD' at Cambridge. In export versions of ZED different characters or code values may be used.

```
n
```

This qualifier, called the *count* qualifier, specifies that the *n*th occurrence of the given string in the line is required. *n* must be greater than or equal to 1. Occurrences are counted from the left unless the L qualifier is also present, in which case they are counted from the right and the nth last occurrence is matched. If the line contains fewer than *n* occurrences of the string the qualified string will not match. Thus

```
3/river/
```

and

```
3L/river/
```

will both match a line containing at least three occurrences of the string "river"; however, unless the line contains exactly five occurrences they will refer to different positions in the line.

If there are overlapping occurrences of the string in a line, the count qualifier counts each occurrence. For example, if the current line is

```
ababa
```

then the qualified string

```
2/aba/
```

matches the last three characters.

If the N qualifier is used with the count qualifier, the qualified string will match lines containing fewer than *n* occurrences.

The count qualifier is not permitted with the B, E, C or P qualifiers.

```
R
```

The string being matched is not taken as a literal string, but is interpreted as a Regular Expression, as described below. This provides a facility for very complex string matching. The R qualifier may occur in conjunction with all other qualifiers except C, and qualified strings containing R may appear wherever a qualified string is permitted, for example

```
F N2R/0-9#0-9/
F (R/(A|B)~Z/ & NB/**/)
E LR/cat#a-z/!!/
SA [10,60]WUR/elephant#s/
```

The use of a Regular Expression involves a lot more processing than a simple string match, since the string must be interpreted character by character, and multiple possibilities must be checked when searching a line. Regular Expressions should therefore only be used when other facilities are inadequate.

The syntax and semantics of Regular Expressions are now described. All characters in a Regular Expression stand for themselves except for

"	double quote
?	question mark
|	vertical bar
(	opening round bracket
)	closing round bracket
-	minus sign
#	sharp (hash) sign
~	twiddles

which have special meanings. Note that the Regular Expression consists of the contents of the qualified string; the delimiters are not part of the expression, and follow the normal ZED rules.

The character " is a quoting character which is used to remove the special meaning from any of the above. It is used immediately before the character whose special meaning is to be cancelled. For example, the qualified string

```
R/long"-eared"?/
```

is equivalent to

```
/long-eared?/
```

(but should not be used, since it requires more cpu time to process). Note that the two occurrences of " in this example are totally unrelated. The first removes the special meaning from the character - and the second from the character ?.

The character ? is a *wild character* which matches any character at all. It does not however match non-existent characters at the end of a line. Thus

```
WR/c?t/
```

matches any of “cat”, “cot” or “cut” (and also “ctt”, “cxt” and so on).

The character | specifies alternation, that is, a choice of alternatives.

```
R/cat|dog/
```

matches either “cat” or “dog”. Unlike the search expression

```
(/cat/ | /dog/)
```

the Regular Expression is defined to match the first of “cat” or “dog” in a line which contains both.

Round brackets (parentheses) are used for grouping, for example to specify more complicated forms of alternation such as

```
R/c(a|o|u)t/
```

which matches “cat”, “cot” or “cut”. Without the brackets,

```
R/ca|o|ut/
```

matches “ca”, “o” or “ut”. Brackets may be nested to any depth.

Null alternatives are permitted whether in brackets or not; for example:

```
R/cat(astrophe|)/
```

matches “cat” or “catastrophe”.

The character - is used between pairs of upper case (capital) letters, lower case (small) letters, or digits, to specify a range of characters. The two characters must be in conventional collating order. For example

```
R/0-9/
```

matches any digit. Only one character is matched. If - appears other than as defined above it is an error. Thus, for example, P-A, A-z, a-Z, a-9 and .-@ are all illegal. The alphabetic ranges include letters only; they do NOT include other characters whose

EBCDIC code lies between the EBCDIC codes for the start and end of the range.

The character # specifies arbitrary repetition for the character, character range or bracketed item which follows it. Zero occurrences are permitted. Thus

```
R/#*/
```

matches a sequence of zero or more asterisks, while

```
R/*#*/
```

matches a sequence of one or more asterisks. The qualified string

```
R/#a-z/
```

matches any sequence of lower case letters, while

```
R/A-Z#(A-Z|0-9)/
```

matches a traditional *identifier*. Note that

```
R/#?/
```

matches an arbitrary number of any characters, that is, it matches the whole line (within the operational window, as always).

Finally, the character ~ is used before a single character or character range to specify a mis-match. Thus

```
R/~A-Z/
```

matches any single character that is not an upper case letter, while

```
R/~*~!~'?/
```

matches any three characters such that the first is not *, the second is not !, and the third is not ?. If ~ appears other than immediately before a (possibly quoted) character or character range, it is an error. Thus expressions such as

```
R/~(cat|panther)/
```

are illegal.

When attempting to match a *Regular Expression*, ZED scans the line character by character from left to right, or, if the L or E qualifier is present, from right to left. It keeps track of all current possible matches simultaneously. If more than one match is possible, the longest matched string is chosen. For example

```
R/cat|catapult/
```

matches "catapult" if it is present.

In the above example, the letters c, a and t are each tested twice, as they appear independently in each match path. It is more efficient to combine part strings where possible in order to avoid this, for example

```
R/cat(|apult)/
```

A formal definition of the syntax of Regular Expressions is given below. Each right hand side is an alternative for the entity named on the left hand side. Square brackets denote optional items, while an asterisk denotes arbitrary repetition of the preceding item.

```
<exp>::= [<prim>*] [| [<prim>*]]*            but including at least one <prim>

<prim> ::= (<exp>)
           ?
           #<prim>
           <ch>
           ~<ch>
           <range>
           ~<range>

<ch>    ::= '<any character>                          ) in
            <any non-special character>               ) non

<range> ::= <upper case letter>-<upper case letter>   ) hexadecimal
            <lower case letter>-<lower case letter>   ) mode
            <digit>-<digit>

<ch> ::= <a pair of hex digits>                       ) in hexadecimal
<range> ::= <ch>-<ch>                                 ) mode
```

In each alternative for <range>, the second character must appear later than the first in the conventional collating sequence.

3.3.7 Semantics of Search Expressions

Search expressions are used as arguments for the find commands F, DF and BF, and also for the conditional commands such as IF. The & and | operators have their usual meanings (AND and OR). Sufficient comparisons are done to determine whether the result of the search on a given line is true or false; redundant comparisons are omitted. The order in which the comparisons is performed is not defined.

Qualified strings with the R qualifier (i.e. Regular Expressions) may appear as part of search expressions.

3.3.8 Output Processing

Lines which are passed in a forward direction are not written to the destination immediately, but instead are added to an output queue in main store. When the store available for such lines is used up, lines at the head of the queue are written out as necessary. Until a line is actually written it is possible to go backwards and make it the current line again. Normally several hundred lines can be held in main store.

There is a command for directing output lines to one of 16 in-store buffers instead of the destination output queue. Each buffer can contain a queue of an arbitrary number of lines, subject only to the available store. The contents of these buffers can be inserted into the source at any point. By this means blocks of lines can be moved around in the source.

Portions of the output may also be sent to destinations other than TO. Selection of an alternative destination dataset causes the queue of lines for the current output dataset to be written out (see section 3.4.8.3).

3.3.9 Output and Edits Listing

If the ddname LIST exists, ZED writes to the associated dataset a copy of every line that is output to any destination dataset. At the start of each listing line the original line number is inserted if appropriate; for inserted lines **** is used, while for moved lines the number is enclosed in brackets. Between the line number and the first data character there are two character positions which are used to mark deleted and changed lines. A - is printed in the first position for any original line whose predecessor has been deleted or moved. A + is printed in the second position for any line whose contents have been changed. The total size of the added information is seven characters, the same amount as used by the LIST program. (This overflows if it is necessary to output a line number greater than 99999 or one greater than 9999 enclosed in brackets.)

If a line contains the control character + the line number and flag characters are replaced by spaces.

ZED does not distinguish between TO and alternative outputs in the listing. All output lines are listed. If a rewind occurs which necessitates writing to the destination, multiple listing will occur.

If the ddname EPRINT exists, ZED writes to the associated dataset a copy of each line of editing commands including the OPT (PARM) line. Provided the blocksize of the dataset is not too large, this can provide a useful means of recovery from system crashes in the middle of long editing sessions. Editing command lines can also be copied to the verification dataset by means of the VE command (see section 3.4.17).

ZED makes no attempt at pagination, nor does it provide headings for listing datasets. If these are required the listing should be sent to a temporary file and post processed with the LIST program.

3.3.10 End of File Handling

When ZED reaches the end of a source file, a dummy *end-of-file* line becomes current. This has a line number one greater than the number of lines in the file. Verification of this line consists of the line number followed by an asterisk.

Commands which make changes to the current line, and commands which move forward through the source provoke an error if obeyed when the end-of-file line is current, unless they are contained within an UTEOF group (see section 3.4.9.2).

If a string match is attempted on the end-of-file line (e.g. via the IF command) the match fails or succeeds according as the N qualifier is absent or present in the qualified string.

3.4 ZED COMMANDS

This section contains descriptions of all ZED commands, split up by function. A summary and an alphabetic list of commands appear in section 5.

3.4.1 Notation for Command Descriptions

In the descriptions which follow, the character / is used to indicate a delimiter character. Command names are given in upper case; lower case is used to indicate argument types as follows:

a, b	line numbers (or . or *)
cg	command group
m, n	numbers
q	qualifier list (possibly empty)
se	search expression
s, t	strings of arbitrary characters
sw	switch value (+ or -)
name	name (up to four letters)

3.4.2 Selection of a Current Line Only

These commands have no function other than to select a new current line. Lines that are passed in a forward direction are added to the destination output queue or current in-store buffer as appropriate. Lines which are passed in a backward direction are stacked up for subsequent re-processing in a forward direction.

Ma

Move forward or backward to line a in the source. Only original lines are accessible by line number (see section 3.3.5). If the argument a is equal to the current line number, or is specified as '.', this command causes the operational window to be reset (see section 3.4.4.1).

A special facility which applies only to the M command, and not to other line number commands, is the ability to specify the line number as one of the characters - or +. These mean the first and the last lines currently in main store respectively. Thus M- moves backward as far as possible in the current output, while M+ moves forward to the line most recently read from the current source.

```
        N
```

Move forward to the next line in the source. When the current line is the last line of the source, executing an N command does not cause an error. The line number is incremented and a special end-of-file line is created. This can be tested for by conditional commands (see section 3.4.9). Executing an N command when already at the end of the source causes an error.

```
        P
```

Move back to the previous line. The number of previous lines available depends on the amount of main store in the region in which ZED is running.

```
        F se
```

Find the line specified by the search expression. The search starts by default at the current line and moves forward through the source. This is to cover the case where the current line has been reached as a side effect of previous commands such as line deletion. However, the command FN+ (see section 3.4.20) can be used to set a mode in which F starts its search on the line following the current line.

An F command with no argument searches using the last obeyed search expression. In the case of lines longer than MXLL data characters, F will treat each part line as a separate line.

```
        BF se
```

Backward find: this behaves like F, except that it moves backward until a matching line is found. It starts its search by default at the current line, but the command FN+ can be used to set a mode in which it starts at the previous line.

3.4.3 Line Insertion and Deletion

A single new line may be inserted before the current line by the command

```
        IS/s/      \insert string
```

The inserted line consists of the given string with the default control character (space) added. The command

```
        IS//
```

inserts a blank line before the current line.

The current line itself may be replicated by the command

```
        IC         \insert current
```

which inserts a copy of the current line before the current line.

The remaining commands in this section may select a new current line as a side effect of their main function. Those that are followed by in-line insertion material must be the last command on a line (only comment may follow). The insertion material is on successive lines, terminated by Z on a line by itself. The Z command (see section 3.4.20) can be used to change the terminator, which is recognized in either upper or lower case.

```
Ia
<insertion material, as many
lines as necessary>
Z
```

Insert the insertion material before line a. If a is omitted the material is inserted before the current line; otherwise line a becomes the current line.

```
Ia  BUFFn
```

Insert the lines contained in in-store buffer n (0<=n<=15) before line a. This command leaves buffer n empty.

```
Ia  COPYn
```

Insert a copy of the lines contained in in-store buffer n before line a. This command leaves the contents of buffer n intact.

```
Ia  /s/
```

Insert the contents of the file whose ddname or dsname is s before line a.

```
Ra b
<replacement material>
Z

Ra b BUFFn
Ra b COPYn
Ra b /s/
```

The R command is equivalent to D (see below) followed by I. The second line number must be greater than or equal to the first and may be omitted if only one line is being replaced (i.e. if b=a). Both may be omitted if it is the current line which is being replaced. The line following line b becomes the new current line.

```
Da b
```

Delete lines a to b inclusive. The second line number may be omitted if only one line is being deleted (i.e. if b=a) and both may be omitted if it is the current line which is being deleted. The line following line b becomes the new current line.

For both the D and R commands, if the second argument is specified as the character . it still refers to the line which is current at the time the command is obeyed. Thus for example

```
M18; D9.
```

causes lines 9-18 inclusive to be deleted. The commands

```
M18; D.9
```

cause an error because lines to be deleted must always be specified in a forward direction.

```
DF se
```

Delete Find: delete successive lines from the source until the line specified by the search expression is found. This becomes the new current line. A DF command with no argument searches using the last obeyed search expression. Like F, the DF command starts its search by default at the current line, but the command FN+ can be used to set a mode in which it starts at the next line. The current line is, however, still deleted.

```
DREST
```

Delete the current line and the rest of the source which follows. This command has the same effect D.*, except that it does not actually read through the source. As a result it is more efficient; its disadvantages are that the end-of-file line which is current as a result of DREST may not have a line number, and any halt that has been set up will be ignored (see the H command in section 3.4.20).

3.4.4 Line Windowing

3.4.4.1 The Operational Window

ZED normally scans all the characters in a line when looking for a given string unless a range of characters positions is specified in the command (as a string qualifier). A default range of character positions can be specified by the RV (restrict view) command:

```
RVn m
```

This command sets a *window* on all lines within which all context operations take place unless explicitly requested otherwise by the use of the window qualifier. Context operations include string searches and character insertion and deletion. The window consists of columns *n* to *m* inclusive.

The window for the current line, which may have been set by a previous RV command or changed dynamically (see below), is reset when an RV command is obeyed.

Characters outside the window are omitted from context operations, and those to the right of it are kept in the same character positions when a line is changed, by adding spaces or deleting characters at the right hand window edge. An error occurs if a character other than a space is deleted. As an example, the command

```
RV1 72
```

used when editing a file of card images, ensures that the sequence numbers remain unchanged and in the correct position.

In all the descriptions of ZED context commands *the beginning of the line* always means *the beginning of the operational window*, and *the end of the line* always means *the end of the line or window, whichever comes first.*

If the left hand edge of the window is past the end of the line, the window contains the null string. If characters are inserted in such a case, spaces are added to the line to bring it up to the required length.

The characters . and * may be used instead of numbers as arguments to the RV command. The former means *the current value* and the latter means *the default value.* If any argument is omitted it defaults to *. Hence the command RV with no arguments resets the window to its initial value, which is positions 1 to 32767. It is equivalent to RV * *.

Whenever a current line is verified, the position of the operational window is normally shown by the use of the characters > and < underneath the line, for example

```
26.
This is line 26 this is.
        >       <
```

The operational window contains the characters "line 26" in this case. The left hand indicator is omitted if it is at the start of the line, and the right hand indicator is omitted if it is past the end of the line. The VWL, VWR and VW switches can be used to disable verification of the operational window edges (see section 3.4.16).

For a current line, the edges of the operational window are dynamically adjustable, the RV command specifying the initial positions when the line becomes current. The left hand edge of the window is also called the *character pointer* in this context, and the following commands are available for moving it:

```
>
```

Move the pointer one character to the right; an error will occur if an attempt is made to move it past the right hand edge of the operational window or the maximum line length (MXLL).

<

Move the pointer one character to the left; an error will occur if an attempt is made to move it past the initial lefthand window specified in the most recent RV command (defaulting to start of line).

PR

Pointer reset: set the pointer to its initial value (as specified in the most recent RV command).

PA q/s/

Point after: the pointer is set so that the first character in the window is the first character following the string. The C and N qualifiers are not permitted. If q does not contain a window qualifier, the search for the string takes place within the current operational window. If the argument is omitted, the last obeyed search expression is used. A window qualifier and a null string can be used to set the pointer to an absolute position. For example

PA[36]//

moves the pointer from any current position to immediately before column 36, while

PA L//

moves the pointer to the end of the line.

PB q/s/

Point before: as PA, but the string itself is also included in the window.

The right hand edge of the operational window is dynamically moved by the following commands:

EWR

End window right: the right hand edge of the window is moved one position to the right.

EWL

End window left: the right hand edge of the window is moved one position to the left. It cannot pass the left hand edge.

EWA q/s/

End window after: the right hand edge of the window is set immediately after the last character in the string. If q does not contain a window qualifier, the search for the string takes place within the current operational window. If the argument is omitted, the last obeyed search expression is used.

EWB q/s/

End window before: the right hand edge of the window is set immediately before the first character in the string.

3.4.4.2 Single Character Operations

The following two commands move the character pointer one place to the right after forcing the case of the first character in the window, if it is a letter. (If not, they are equivalent to >.)

$

Force lower case (Dollar for Down).

%

Force upper case (Percent for uP).

The command

```
_                          \(an underline character)
```

changes the first character in the window into a space character, then moves the character pointer one place to the right.

The command

```
#
```

deletes the first character in the window. Normally the remainder of the window moves one character to the left, thus leaving the character pointer pointing at the next character in the line. The command is exactly equivalent to

```
E/s//
```

(see section 3.4.5.1) where s is the first character in the window. A repeat count on the # command is treated specially. If its value is n say, then the repeated command is equivalent to the single command

```
E/s//
```

where s is the first n characters in the window, or the whole of the contents of the window, whichever is the shorter. A sequence of # commands is treated in the same way as a single, repeated # command. This means that such a sequence is able to cross field boundaries (see section 3.4.11).

A combination of > % $ _ and # commands can be used to edit a line character by character, the commands appearing under the characters they affect. The following line of text and commands illustrates this:

```
o Oysters,, Come ANDDWALK with    us
%>$$$$$$$#>>$$$$$$$$_$$$$$$$$$$###
```

The resulting line would be

```
O oysters, come and walk with us
```

and the character pointer would be left immediately before the word "us".

3.4.4.3 The Find Window

Normally the find commands F, BF and DF operate within the operational window described in the preceding section unless their arguments contain overriding window qualifiers. However, there are sometimes requirements to select lines by context in one field and to operate in another. An example of this is the editing of card image files, where line selection is to be done by sequence number. The command

```
RFn m
```

specifies a window for all strings in find commands that do not have explicit window qualifiers. When set, it acts independently of the operational window. RF with no arguments unsets the find window, and the find command then reverts to using the operational window.

The characters . and * may be used as arguments for the RF command. The character . means *the edge of the operational window* (left or right hand as appropriate) while the character * means *the default value*. The second argument may be omitted if it is *, and both arguments may be omitted if they are both * (as described above).

The availability of . as an argument means that the find window may be set according to the contents of a particular line.

Use of a window with the DF command does not affect the deleting operation; it still deletes whole lines, not just the part that is windowed.

3.4.5 String Operations on the Current Line

3.4.5.1 Basic String Alterations

Three similar commands are available for altering parts of the current line. The A, B and E commands insert their second (string) argument after, before or in exchange for their first argument respectively. The first argument may be qualified, but the C and N qualifiers are forbidden. If the current line were

```
The Carpenter beseech
```

then the commands

```
E U/carpenter/Walrus/         \exchange
B/bese/did /                  \insert string before
A L//;/                       \insert string after
```

would change it to

```
The Walrus did beseech;
```

A null second string in an E command causes removal of the first string from the line; an A or B command with a null second string does nothing (provided the first string is found - an error will occur if it is not.) A null first string in any of the three commands will match at the initial search position. This is the beginning of the line unless either of the E or L qualifiers is present, in which case it is the end of the line.

If a window qualifier is present in an A, B or E command, the whole operation takes place within the given window, and characters outside it are not affected. The operational window can be overridden by this means.

Regular Expressions (see section 3.3.6) may be used in A, B or E commands to specify more complicated contexts for string alterations. For example

```
E R/(Walrus|Carpenter)/Oyster/
```

replaces whichever of the strings "Walrus" or "Carpenter" occurs first on the line by "Oyster". The replacement string is treated as a literal string and is not interpreted in any way.

3.4.5.2 Pointing Variant

The AP, BP and EP commands have two arguments and act exactly like A, B and E, with the additional feature that, when the operation is complete, the character pointer is left pointing to the first character following any text which participated in the alteration. Thus

```
AP/s/t/
```

is equivalent to

```
A/s/t/; PA/st/
```

while

```
BP/s/t/
```

is equivalent to

```
B/s/t/; PA/ts/
```

and

```
2EP U/tweadle/Tweedle/
```

would change

```
tweadledum and TWEADLEdee
```

into

```
          Tweedledum and Tweedledee
```

leaving the character pointer just before "dee".

3.4.5.3 Deleting Parts of the Current Line

The commands DTA (delete till after) and DTB (delete till before) are used to delete from the beginning of the line (or character pointer) until a given context. The commands DFA (delete from after) and DFB (delete from before) are used to delete from a given context until the end of the line (or window). If the current line were

```
          All the King's horses and all the King's men
```

then

```
          DTB L/King's/
```

would change it to

```
          King's men
```

while

```
          DTA/horses /
```

would change it to

```
          and all the King's men
```

If the argument for any of these commands is omitted, the last obeyed search expression is used, provided that it is suitable.

3.4.5.4 Case Changing

The case of alphabetic characters in a line may be changed character by character as described in section 3.4.4.2. There are also two commands for forcing the case of complete strings:

```
          LC q/s/              \force lower case
          UC q/s/              \force upper case
```

The N and C qualifiers are not permitted. Alphabetic characters in the matched string are forced to the relevant case while non-alphabetic characters are unaltered. The character pointer is not moved.

The case of the entire current line (within the operational window) may be forced by the commands

```
          LCL                  \lower case entire line
          UCL                  \upper case entire line
```

These commands do not cause motion of the character pointer. See also UCE and LCE in section 3.4.8.1.

3.4.5.5 Setting the Control Character

The CC command, which takes a string of length one as its argument, is used to set the (carriage) control character for the current line. The character should be one of the standard ANSI set. Section 3.7 describes how ZED handles datasets with the various different kinds of control character. When a current line is verified, the control character is given following the line number.

```
          CC/1/
```

Set the current line's control character to '1'.

3.4.5.6 Miscellaneous Current Line Commands

Whenever one of the string alteration commands A, AP, B, BP, DFA, DFB, DTA, DTB, E, EP, LC or UC is obeyed it becomes the current string alteration command and can be repeated by obeying the ' command which has no arguments. Anomalous effects occur if sequences such as

```
E/castle/knight/; 4('; E/pawn/queen/)
```

are used; the second and subsequent executions of the ' command refer to a different command than the first.

Alterations that have been made to a current line, other than those caused by margins or space compression (see sections 3.4.10 and 3.4.20), can be cancelled by the command

```
UNDO
```

The current line is reset to what it was when it last became current. Margin, space compression and global operations are not undone, and splitting a line is equivalent to establishing a new current line. UNDO also causes the operational window to be reset.

The precise layout of a line may be ascertained by means of the show column command

```
SHC q/s/
```

This causes the column number of the first character of the string to be written to the verification file. The N and C qualifiers are forbidden.

3.4.6 Splitting and Joining Lines

ZED is primarily a line editor. Normal ZED editing commands do not operate over line boundaries, but there are facilities for splitting a line into more than one line, and for joining together two or more successive lines.

```
SB q/s/
```

Split the current line *Before* the given context. The first part of the line is sent to the output, and the remainder becomes a new non-original current line. If any of the remaining characters lie to the right of the operational window in the original line, they are kept in the same character positions by the insertion of spaces. This also applies to any remaining characters to the right of any field boundaries (see section 3.4.11). The effect is precisely as though an exchange command with a null replacement string were executed.

If the argument for SB is omitted, the last obeyed search expression is used, provided that it is suitable.

```
SA q/s/
```

Split the current line *After* the given context.

```
CL/s/
```

Concatenate line: a new current line is formed by concatenating the current line, the given string, and the next line from the source, in that order. The string may be omitted from the command if it is null. The position of the operational window is retained.

As an example of splitting and joining lines, consider the text

```
Humpty Dumpty sat on a wall; Humpty
Dumpty had a
great fall.
```

Assuming that the first line is the current line, the commands

```
SA /; /; 2CL/ /
```

cause the line

```
Humpty Dumpty sat on a wall;
```

to be output, and leave as the new current line

    Humpty Dumpty had a great fall.

3.4.7 Inspecting Parts of the Source

The following commands all cause ZED to advance through the source, outputting the lines it passes to the verification dataset as well as to the normal output (where relevant). As these commands are most frequently used interactively, the verb *type* is used for them. This does not however mean that they cannot be used for outputting to, say, a line printer. After one of these commands has been obeyed the last line typed becomes the new current line.

    Tn

Type *n* lines. If *n* is omitted, typing continues till the end of the source. The first line typed is the current line, so that, for example

    F /It's my own invention/; T6

types six lines starting with the one beginning "It's my own invention".

    TLn

Type *n* lines as for T, but with line numbers added in the LIST file format (see section 3.3.9).

Typing up to a given context can be done by using the T command with one of the conditional looping commands (see section 3.4.9.2).

To send a copy of the current line to the verification dataset, the command T1 (or TL1) can be used. This form of the command (with argument 1) is permitted in ON groups. The verification differs from that produced by the ? command in that the line number is omitted (or appears on the same line) instead of being output on the line above.

The character + is permitted as an argument for the T and TL commands. It causes typing to continue up to the line most recently read from the source (c.f. M+ in section 3.4.2).

If VI+ is set (see section 3.4.16) then output from T and TL is in the same format as that from the ! command (see section 3.4.16).

3.4.8 Control of Command, Input and Output Files

Files used by ZED are divided into four types: command, input, output and verification. There are no commands for changing the verification file.

ZED contains 16 in-store buffers which may be used for small command, input and output files as described below. Any one buffer may only be used for one thing at a time; for example, it cannot simultaneously be used for both input and output.

A list of the non-empty buffers may be written to the verification file by obeying the command

    SHBUFF

The list contains pairs of numbers in the form n/m, where *n* is a buffer number and *m* is the number of lines in that buffer.

The contents of any of the 16 in-store buffers may be examined by the command

    TBUFF n   \type buffer n

If the argument is omitted the contents of all non-empty buffers are typed, each preceded by a line containing "Buffer <n>". The command

    DBUFF n

deletes the lines contained in buffer *n*, leaving the buffer empty. DBUFF with no argument deletes all lines in all buffers.

When ZED is ending normally, a message is output if any of the in-store buffers is not empty. For non-interactive use this is treated as a warning, but for interactive use it is an error. This allows the user to retrieve the material if it was left in a buffer accidentally. This check (and others) may be suppressed by the use of the WARN command (see section 3.4.20).

3.4.8.1 Command Files

When ZED is entered, commands are read from the PARM field, followed by EDITS if present. The command

```
C /s/
```

where s is a ddname or dsname, can be used to read commands from another file. When the file is exhausted (or a Q command obeyed) it is closed and control reverts to the command following the C command. Thus

```
C /DD1/
```

reads and obeys commands from the file allocated to ddname DD1, while

```
C /SPQR.EDS(A)/
```

reads commands from the named disc file.

The commands

```
LCE sw          \force lower case for edits
UCE sw          \force upper case for edits
```

can be used to force the case of any letters in lines of editing commands. Lines of insertion material are also included in this operation, which is applied to lines of commands read from in-store buffers as well as to those read from datasets. UCE+ implies LCE- and vice versa.

Commands can be read from any one of the 16 in-store buffers by the commands

```
C BUFF n
C COPY n
```

where 0<=n<=15. The first of these leaves the buffer empty, while the second leaves it intact. This facility allows commands to be constructed whose content depends on the source data.

3.4.8.2 Input Files

The I and R commands, which can be used to insert the entire contents of a file or in-store buffer at a specific point in the source, are described in section 3.4.3 above.

The FROM command may be used to associate the source file with a dataset other than that with ddname FROM by using the form

```
FROM /s/
```

where s is a ddname or dsname. In the latter case ZED allocates and verifies a ddname which must be used in subsequent references to the file. A FROM command with no argument re-selects the original source file.

When a FROM command is obeyed, the current line remains current; however the next line will come from the new source. (An exception to this occurs if the current line is the end-of-file line. In this case the first line of the new source becomes current, but is not read until a command requiring a current line is obeyed.) Any backed up lines will be held with the previous source and can be subsequently accessed after obeying another FROM command.

Each dataset has its own sequence of line numbers. Output lines in store at the time of a switch may become non-original lines in order to maintain a unique ascending sequence. An M, D, I or R command which immediately follows a FROM command referring to a dataset and which contains an explicit line number always moves forwards (i.e. into the new source).

A source file is not closed when it ceases to be current, so further lines can be read from it by later re-selection (by ddname). The CF command must be used to close files that are no longer required.

```
        CF /ddname/
```

CF with no arguments closes all closeable files (both input and output). When ZED is interactive and V+ is set, the ddnames of the files being closed are output to the verification dataset.

An example of the use of the FROM command to merge lines from two datasets follows:

```
        M10             \pass lines 1-9 from FROM
        FROM /DD1/      \select new input,
                        \line 10 remains current
        M6              \pass line 10 from FROM,
                        \lines 1-5 from DD1
        FROM            \re-select FROM
        M14             \pass line 6 from DD1,
                        \lines 11-13 from FROM
        FROM /DD1/      \re-select DD1
        M*              \pass line 14 from FROM,
                        \the rest of DD1
        FROM            \re-select FROM
        CF /DD1/        \close DD1
        M*              \pass the rest of FROM
```

Input may also be read a line at a time from any one of the 16 in-store buffers by means of the command

```
        FROM BUFF n
```

where 0<=n<=15. In this case all the lines are treated as moved lines and have no line numbers. The M, D, I and R commands may not be used with explicit line numbers while input is being read from a buffer. Commands such as I., M*, M+, M- and D.* are however allowed.

Margins and space compression are not applied to lines "read" from buffers. A pseudo *end-of-file* is generated when the buffer becomes empty. Moving backwards when the current source is a buffer has the effect of adding lines onto the front of the buffer.

3.4.8.3 Output Files

Output from ZED is normally sent to the dataset with ddname TO. Output is not however immediate; lines which have been passed are kept in a queue in main store as long as possible in order to provide a capability for moving backwards in the source (see section 3.4.2).

Output lines may be sent to one of 16 in-store buffers instead of to the output queue. This facility is useful for shuffling blocks of lines within the source.

```
        TO BUFF n
```

Select buffer n for subsequent output lines (0<=n<=15). Output is redirected to the main output queue by the command

```
        TO
```

The following example illustrates the line shuffling facility:

```
M45           \lines 1-44 to output queue
TO BUFF3      \switch to buffer 3
M50           \lines 45-49 now in buffer 3
TO BUFF7      \switch to buffer 7
M52           \lines 50-51 now in buffer 7
TO            \revert to normal output
M40           \back to line 40
I BUFF3       \lines 45-49 before line 40
I75 BUFF7     \lines 50-51 before line 75
```

The TO command can also be used to associate the output queue with a dataset other than that with ddname TO by using the form

```
TO /s/
```

where s is a ddname or dsname. In the latter case ZED allocates and verifies a ddname which must be used in subsequent references to the file.

When a TO command that does not specify an in-store buffer is obeyed, the existing queue of output lines is written out if the output dataset is actually switched.

An output file is not closed when it ceases to be current, so further lines can be added to it by later re-selection (by ddname). The following example splits up the source between the main destination TO and an alternate destination DD1.

```
M11           \lines 1-10 to TO
TO/DD1/       \switch output dataset
M21           \lines 11-20 to DD1
TO
M31           \lines 21-30 to TO
TO/DD1/
M41           \lines 31-40 to DD1
TO
```

If a file is required for re-use it must be explicitly closed. The command

```
CF /ddname/
```

closes the file of the given ddname. As an example of its use is for moving part of the source file which is too big to hold in an in-store buffer to a later place in the output:

```
TO/DD1/       \output to DD1
1000N         \advance through source
TO            \revert to TO
CF /DD1/      \close output file DD1
I2000/DD1/    \re-use as input file
```

The use of CF on files that are completely finished with minimizes the amount of store needed. CF with no argument closes all closeable files (both input and output). When ZED is interactive and V+ is set, the ddnames of the files being closed are output to the verification dataset.

3.4.9 Conditional Commands and Loops

3.4.9.1 Conditional Commands

The IF command can be used to obey a command or command group only if a given search expression matches the current line. The syntax for the two forms is

```
IF se THEN command
IF se THEN cg
```

where the command or command group may have a repeat count if required. The word THEN is optional, and may be omitted.
For example

```
IF B/*/ THEN 99$

IF (NB/*/ & [10,13]/MVCL/) (I
      BAL   LINK,SIM360
Z
)
```

The IF command may also optionally in addition specify a command or command group to be obeyed when the search expression does not match the current line:

```
IF se THEN cg ELSE cg
```

For example

```
IF /vorpal sword/ THEN e/sword/dagger/ ELSE (
  COMMENT /no vorpal sword/)
```

Note the use of command brackets to continue the command over more than one line. An end of line not in parentheses always terminates a ZED command (for the benefit of interactive users). The only exception is when an I or R command is followed by in-line insertion material. However if such a command is required before ELSE, brackets must be used:

```
IF /uffish/ THEN (I
some text
Z
) ELSE 3N
```

If the command brackets were not used, ZED would treat the end of line following the insertion terminator as terminating the IF command.
The UL (unless) command is the complement of IF. The THEN command or command group is obeyed if the search expression does NOT match the current line; the ELSE command or command group is obeyed if it does.
A number of successive tests using IF and UL can be joined by using the connectors ELIF (else if) and ELUL (else unless). This reduces the number of brackets needed, and makes the commands easier to read. For example

```
IF /walrus/ THEN (
 COMM /animal/) ELIF /carpenter/ THEN (
 COMM /human/)  ELUL /oyster/ THEN (
 COMM /eh?/)   ELSE (
 COMM /bivalve/)
```

Two special commands are provided for detecting the end of the source file, IFEOF and ULEOF. In their simplest form they take only one argument, a command or command group:

```
ULEOF B//  /
```

An alternate command group introduced by ELSE, ELIF or ELUL is permitted:

```
IFEOF THEN COMM /reached end/ ELIF /cabs/ E/cabs/taxis/
```

3.4.9.2 Loops

Many commands may be preceded by an unsigned decimal number to indicate repetition, for example

```
24N
```

Repeat counts may also be specified for command groups in the same way as for individual commands:

```
12(F/handsome/; E/handsome/hansom/; 3N)
```

A repeat count of zero causes the command or command group to be skipped.

Indefinite repetition can be specified by the RPT command, which takes as its argument a command or command group. For example:

```
RPT (F/queen/; IF/red/ E/queen/king/; N)
```

Two complementary commands, WH (while) and UT (until), can be used to cause repetition while, or until, the current line matches a given search expression.

```
WH se cg
UT se cg
```

For example

```
WH /  / E/  / /
```

converts all sequences of spaces in the current line to a single space.

```
UT P// (B//  /; N)
```

indents lines until a blank line is reached.

The command UTEOF can be used to repeat a command or command group until the end of the source is reached, for example

```
UTEOF (N; B//*/)
```

The group of commands is repeatedly obeyed until a command which is illegal at end-of-file is encountered when the end-of-file line is current. In the above example, assuming that a real line is current at the start, it is the B command which will eventually cause the loop to exit. For the command

```
UTEOF (N;?)
```

the N command will eventually fail following the execution of the ? command for the end-of-file line.

3.4.9.3 Exceptional Flow of Control in Loops

The execution of any command group can be terminated by the AGP (abandon group) command. If the group is an argument of another command such as WH, that command is also terminated. For example

```
WH B/!/ (A/!/ /; N; IFEOF AGP)
```

inserts a space after the initial ! of successive lines until either a line not beginning with ! or the end of the source is reached.

A repeat count for the AGP command specifies the number of textual levels of brackets to abandon. At the highest level, AGP abandons the current line of commands.

If AGP appears in a procedure (see section 3.4.19) its effects are limited to groups within that procedure.

3.4.10 Margins

Margins may be specified for source and verification files. Control characters do not participate in margin operations.

```
MAn m
```

Set margins for the source at positions *n* and *m*. Characters before position *n* and after position *m* are removed from each source line when it is read. Further processing, including the application of windows, works on the transformed lines. The application of margins does not cause line verification, and it is not reversible via the UNDO command (see section 3.4.5). Margins are automatically reset when a REWIND command is obeyed.

```
MAVn m
```

Set margins for the verification file at positions *n* and *m*. This applies only to the verification of data lines, and its effect is that only those characters appearing between positions *n* and *m* (inclusive) are output.

The characters . and * may be used as arguments for the MA and MAV commands. The character . means *the edge of the operational window* (left or right as appropriate) while the character * means *the default value*. The second argument may be omitted if it is *, and both arguments may be omitted if they are both *. The default values are 1 and 32767 for the first and second arguments respectively.

The availability of . as an argument means that margins may be set according to the contents of a particular line.

3.4.11 Fixed Format Fields

The MF (mark fields) command is used to designate certain character positions in source lines as the boundaries of fixed format fields. Up to 255 such fields are permitted, specified numerically:

```
MF n1 n2 n3 ...
```

where the *n*'s are in ascending order. The boundaries are immediately before the character positions specified, and start of line is also considered to be a field boundary. When changes are made to the current line, the number of characters in each field is kept constant, if possible, by the addition or removal of spaces at the right hand side of fields during each alteration. Note that this means that a command which removes spaces at the right hand side of a field has no effect. If a field overflows, a warning message is given, but characters are not lost: the next field is shifted to the right. This contrasts with the action when the operational window overflows (see section 3.4.4.1).

A sequence of # commands or a # command with a repeat count will act across a field boundary, since both are executed as a single command (see section 3.4.4.2).

MF with no arguments cancels all field settings; the K command temporarily suspends field settings during the processing of the remaining commands on the same command line. It may appear with an ordered list of field settings as arguments, or with no arguments, in which case all fields are suspended.

A common example of the use of MF is for keeping Assembler source code aligned during editing. This is usually in columns beginning at positions 10, 16 and 36, with a continuation marker in position 72. Because it is desirable to retain one space character between the fields even if they overflow, the command

```
MF 9 15 35 72
```

should be used. Assuming this setting, if the current line were

```
LABEL    LA    R4,DATA             comment
```

then the commands

```
8#; E/LA/L/; A/DATA/WORD/
```

would change it to

```
         L     R4,DATAWORD         comment
```

while

```
E L/LA/LONGNAME/
```

would change it to

```
        LABEL    LONGNAME R4,DATA         comment
```

and give the *field overflow* warning message. The sequence

```
        K15; E L/LA/LONGNAME/
```

would result in

```
        LABEL    LONGNAME    R4,DATA      comment
```

3.4.12 Text Justification

ZED contains a facility for very basic text justification. The result of justifying a sequence of lines is as though all the lines had been concatenated, with one space between each, and the result chopped up again into the largest sized pieces that would fit into the output records, the new breaks always being made immediately after a space character (which will subsequently be removed for U and V format output). No attempt is made to right justify, nor are spaces inserted or removed in the body of the text.

A blank line, a line beginning with a space character (indent), or a line containing a control character other than space is taken as the start of a new paragraph. A line containing the control character + is never joined to the previous or following line, nor is it ever split. It is copied without change, and a new paragraph is started with the following line.

Justification takes place concurrently with other editing, whenever a line is added to the output queue or an in-store buffer during forward motion through the source. Note that this means that inserted lines are also processed. Justification is enabled/disabled by use of the J command.

```
        J+                  \enable justification
        J-                  \disable justification
```

J- is automatically set when the REWIND command is obeyed. The length of output line produced can be controlled by the JL (justification length) command. If this is not given the effective LRECL of the main destination file (TO) is used.

```
        JL 58
```

specifies that the text is to be justified into lines of maximum length 58 characters.

Indented text can be justified by use of the RJ (restrict justification) command.

```
        RJn m
```

This sets a *justification window*. Provided no characters in the output lie outside the window, the lines are justified normally, the *n*-space indent being retained. The value specified for the output line length (via the JL command) must include the indent.

The characters . and * may be used as arguments for the RJ command. The character . means *the edge of the operational window* (left or right as appropriate) while the character * means *the default value*. The second argument may be omitted if it is *, and both may be omitted if they are both *. The default values are 1 and 32767 for the first and second arguments respectively.

The availability of . as an argument means that the justification window may be set according to the contents of a particular line.

Lines with characters to the left of the window are treated as start of paragraph. Lines with characters to the right of the window are output unchanged. The operational window does not take part in justification operations.

If backward movement through the source occurs when justification is on, the justified lines are retrieved. Those that have been split become non-original lines, and are not accessible by line number.

Justified lines may be verified as justification occurs by use of the VJ command.

```
VJ+                     \enable justification verification
VJ-                     \disable justification verification
```

This switch is subject to the overall verification switch V (see section 3.4.16), but not to any other verification switches such as VT, etc.

To illustrate the effects of the justification commands, the following piece of text has been justified in three different ways. (All four samples are indented with respect to this text, in keeping with other examples in this document and to prevent their re-justification when it is edited.)

```
This is a sample of text
and so gobbledegook
is
quite in order while the quick brown fox
jumps continuously and in ever-decreasing
circles over the
lazy dog.

   Indentation is frequently used
   to highlight particular
   parts of a text.
   However, the correct ZED commands are required
   in order to make it work
   as
   expected.
*   The left hand margin
   can be used to
   indicate a new
   paragraph.
```

Using the commands JL40; J+ only:

```
This is a sample of text and so
gobbledegook is quite in order while the
quick brown fox jumps continuously and
in ever-decreasing circles over the lazy
dog.

   Indentation is frequently used
   to highlight particular
   parts of a text.
   However, the correct ZED commands
are required
   in order to make it work
   as
   expected. *   The left hand margin
   can be used to
   indicate a new
   paragraph.
```

Using (on the original text) the commands

```
JL40; J+
F BSU/indent/
RJ5 40
```

```
This is a sample of text and so
gobbledegook is quite in order while the
quick brown fox jumps continuously and
in ever-decreasing circles over the lazy
dog.

   Indentation is frequently used to
   highlight particular parts of a
   text.
   However, the correct ZED commands are required
   in order to make it work as
   expected.
*   The left hand margin can be used to
   indicate a new paragraph.
```

Using (again on the original text) the commands

```
JL40; J+
F BSU/indent/
PB SU/indent/; RJ.
```

```
This is a sample of text and so
gobbledegook is quite in order while the
quick brown fox jumps continuously and
in ever-decreasing circles over the lazy
dog.

   Indentation is frequently used to
   highlight particular parts of a
   text. However, the correct ZED
   commands are required in order to
   make it work as expected.
*   The left hand margin can be used to
   indicate a new paragraph.
```

3.4.13 Global Operations

Global operations are operations which take place automatically as the source is scanned in a forward direction. They are initiated and stopped by special commands. Care should be exercised if global operations are enabled when backward movement through the source occurs. Margining, space compression and justification may be viewed as pseudo global operations.

3.4.13.1 Simple Character Changes

Three commands, GA, GB and GE are provided for simple string changes on each line.

```
GA q/s/t/
GB q/s/t/
GE q/s/t/
```

The C, N and count qualifiers are not allowed. When the B, E or P qualifiers are NOT present, the effect of GA, GB or GE is to apply an A, B or E command, as appropriate, to any occurrence of the qualified string q/s/ in a new current line after the operational window is initialized. The G commands are also applied to the line which is current at the time the command is obeyed.

The exact effect of, say, GA on any line can be defined in terms of other commands thus:

```
WH q/s/ AP q/s/t/; PR
```

provided q does not contain B, E or P. This definition ensures that G commands do not re-scan their replacement text; for example

```
GE/Tiger Lily/Tiger Lily/
```

would not loop forever, but would have no visible effect on any line. However, as a result of the "change", certain lines would be verified.

When the B, E or P qualifiers are present in a G command, the appropriate A, B or E command is applied once only to each line which matches the qualified string. Thus, for example,

```
GE BU/Begin/Start/
```

is equivalent to

```
IF BU/Begin/ THEN E BU/Begin/Start/
```

applied to the current line and to every subsequent line as it becomes current.

Global changes are applied to each new current line in the order in which the global commands which set them up were obeyed.

3.4.13.2 Complicated Cases

The ON command is used to set up a sequence of commands to be obeyed when a line which matches a given search expression becomes current:

```
ON se cg
```

The command group may not contain any command which moves to a new current line, nor any command which alters the current globals. However, DG and EG (see section 3.4.13.4) are permitted. The T and TL commands are permitted if their argument has the value 1, but not otherwise. The command

```
ON /king/ (WH/king/ EP/king/queen/; PR)
```

is exactly equivalent to

```
GE/king/queen/
```

The WH command is necessary because ON is not a looping command; it only tests the line once. ON groups are applied to each new current line in the order in which the ON commands were obeyed.

An ON command may also optionally specify a command or command group to be obeyed when a line which does not match a given search expression becomes current:

```
ON se cg ELSE cg
```

GA, GB and GE commands are treated as shorthand notations for ON groups, and if both appear they are ordered together. For example, the commands

```
GA W/kin/g/        \command 1
ON /knight/ 3#     \command 2
GE /king/King/     \command 3
```

will be applied to each new current line in the order 1, 2, 3.

If a ' command (see section 3.4.5.5) occurs in an ON group it is local to the group and must therefore follow a suitable string alteration command. The operational window is not local however, and changes made in an ON group persist after the group has been obeyed. Thus for example a command such as

```
ON /crown/ EP/crown/coronet/
```

alters the position of the character pointer whenever it matches.

3.4.13.3 Cancelling Global Operations

All global operations are automatically cancelled when a rewind (see section 3.4.20) occurs. The CG command can be used to cancel individual commands at any time.

When a global operation is set up by one of the commands GA, GB, GE or ON, it is allocated an identification number which is output to the verification file if ZED is being run interactively and V+ is set. The argument for CG is the number of the global operation to be cancelled. If CG is obeyed with no argument it cancels all globals and resets the global counter so that the next global created will be numbered 1. CG may also be specified with the character + as an argument, in which case the most recently created global is cancelled.

3.4.13.4 Suspending Global Operations

Individual global operations can be suspended and later resumed by use of the DG (disable global) and EG (enable global) commands which take the global identification number or the character + as their argument. If the argument is omitted, all globals are disabled or enabled as appropriate.

DG and EG are the only global commands permitted inside an ON group. Globals are applied to each new current line (during forward motion) in the order of their identification numbers. Hence it can be determined whether the disabled/enabled global will or will not be applied to the current line which provoked the ON group containing DG or EG.

If EG is obeyed outside an ON group the global(s) thereby enabled are applied to the current line; inside an ON group this action is not taken (to prevent recursion).

3.4.13.5 Looping Globals

It is not possible to detect looping globals in advance and fault them when ZED commands are decoded, hence loops may occur when globals are being obeyed. Because global operations are not normally interrupted by an attention until all the current globals have been applied to a line, it is necessary to use two attentions (followed by <null>) to interrupt a looping global. This may leave a line with only some globals applied.

As a check against one form of looping, ZED counts the number of times a global created by the GA, GB or GE commands is applied to each line, and generates an error if the count exceeds the number of characters in the line. A common example of this form of loop is a command such as

```
GE[10,12]/ //
```

where the spaces that are removed are replaced by others to keep columns 13 onwards aligned.

3.4.14 Displaying the Program State

Several commands beginning with SH (for *show*) output information about the state of ZED to the verification file.

```
SHD
```

Show Data: saved data values, such as the last search expression, are displayed.

```
SHF
```

Show files: the ddnames currently open are displayed.

```
SHG
```

Show Globals: the current global commands are displayed, together with their identification numbers. The number of times each global search expression has matched is also given. An individual global can be displayed by giving its number as an argument to SHG.

SHS

Show switches: the state of all ZED switches is displayed.

3.4.15 Terminating a ZED Run

The W command (windup) causes ZED to *wind through* the rest of the source. It is illegal if output is not currently directed to TO. When the end is reached, all files are closed, the store is relinquished, and ZED exits. Reaching the end of the highest level command file has the same effect as W.

The in-store buffers should be empty before W is obeyed. If any of them are not, an error occurs. This is a soft error for non-interactive runs (return code 4), but a hard error when interactive. It may be disabled by the command WARN- (see section 3.4.20).

The STOP command causes ZED to stop immediately. No further input or output is attempted. The return code is set to 8.

The Q command causes ZED to stop obeying the current command file and to revert to the previous one. If encountered at the outermost level it is synonymous with W.

3.4.16 Current Line Verification

The following circumstances can cause automatic verification to occur:

(a) Reading a new line of commands with a current line which has not been verified since it became current, or has been changed since it was last verified.

(b) Moving past a line which has been changed, but has not been verified since the change.

(c) Outputting an error message.

In cases (a) and (b) the verification only occurs if the V switch is on. The command

V sw

is used to change the setting of the V switch. It is initialized ON (V+) if the initial state of ZED is interactive (commands and verifications both connected to a terminal), and to OFF (V-) otherwise.

When the V switch is in the ON state, further control of automatic verification is possible by using the following switches:

VG sw

Controls verification due solely to changes made by global operations. The default is VG+.

The format of verifications other than justified lines is also controlled by switches:

VN sw

Controls verification of the line number. The default is VN+. As control characters are verified after, and on the same line as, the line number, they will not appear when VN- is set.

VT sw

Controls verification of the text of lines. The default is VT+.

VI sw

Controls whether character indicators are to be given with verified lines. If VI+ is set, all verification, including that produced by the T and TL commands and as a side effect of global changes and justifications, appears in the format described below for the ! command. The default setting is VI-.

VX sw

This switch determines whether verification with character indicators (as described for the ! command below) uses hexadecimal or escape sequences for certain non-printing characters. The default is VX-, which specifies that escapes are to be used.

VWR sw
VWL sw

Control verification of the positions of the right hand and lefthand edges of the operational window respectively. The left hand edge is always omitted if it is at the start of the line, and the right hand edge is omitted if it is past the end of the line.

VW sw

VW sets both VWR and VWL at the same time. The default is VW+.

The justification verification switch VJ (see section 3.4.12) is also subject to the setting of V.

Verification of the current line may be explicitly requested by the command

?

This command verifies the positions of the fixed format fields using the character + as well as the window edges. The text of the current line is always output, even if the VT switch is off.

An alternate form of verification, useful for lines containing non-printing characters, is provided by the command

!

Verify the current line with character indicators. Two lines of verification are produced. The first is the current line with all non-graphic characters replaced by the first character of their hexadecimal value and certain characters with a Cambridge standard escape combination replaced by @, see below. As verification output normally appears on a terminal or line printer (eventually, if not immediately), characters that are available on all these devices are not escaped. If the VX switch is set, then escape combinations are not used, and all "unusual" characters are given in their hexadecimal representation.

The second line contains a minus sign in all positions corresponding to uppercase letters, the second hexadecimal digit in positions corresponding to non-graphic characters, and the escape characters below any @'s. All other positions contain space characters.

If the VN switch is on, the two lines are preceded by a line containing the line number, a full stop character, and two characters representing the line's control character as translated according to the algorithm described above.

The output of the ! command is not affected by the hexadecimal switch (see section 3.4.18) or the VT switch.

The escape combinations used are as follows:

Character	**EBCDIC code**	**Name**	**Escape combination**
¢	4A	centsign	@%
¦	6A	split vertical bar	@|
^	71	circumflex	@F
£	72	pound sign	@4
`	79	grave accent	@G
@	7C	at sign	@@
~ (EBCDIC)	A1	EBCDIC tilde	@¬
[	AD	open square bracket	@(
]	BD	closing square bracket	@)
{	C0	opening curly bracket	@<
}	D0	closing curly bracket	@>
\	E0	backlash	@/

N.B. The ASCII code for "~" translates to the EBCDIC code for logical not (¬) but is printed as "~" at Cambridge.

3.4.17 Edit Line Verification

```
VE sw
```

Verify Edits: lines of editing commands are reflected on the verification dataset as they are read when this option is switched on. The default is VE-.

3.4.18 Processing Lines in Hexadecimal

Lines can be processed as sequences of hexadecimal digits instead of ordinary characters. This option is switched on and off by means of the command

```
X sw
```

The default is X-. When hexadecimal is being used, lines are verified in 8-digit groups (except as a result of the ! command). Qualified strings and replacement strings used in commands must consist of an even number of hexadecimal digits, as must all lines of in-line insertion material. Other strings, such as the arguments for the COMM, TO and FROM commands, are not affected. Space characters may appear in hexadecimal strings to improve the layout. They are ignored when the string is decoded.

The R (regular expression) qualifier is permitted for qualified hexadecimal strings. For example

```
R/12?(AB)#00|)2B-4D/
```

The only legal characters in this case are the regular expression syntax characters, excluding ", and pairs of hexadecimal digits. Space characters are not permitted. See section 3.3.6 for further details of regular expressions.

Control characters other than space (X'40') are verified in hexadecimal. The argument for the CC command is treated as a replacement string and hence must consist of two hexadecimal digits.

It should be noted that, because ZED decodes a complete line of commands before obeying any of them, the effect of X on string decoding does not begin until the line of commands following the one containing X.

Qualified and replacement strings which are verified (as a result of an error, or as a result of the SHPROC or SHD commands for example) are verified in 8-digit hexadecimal groups if the X switch is on.

When processing lines in hexadecimal, care must be taken with the use of the character pointer and its related commands. The > and < commands still move the pointer one character (byte) position, but this occupies two printing positions in a line verified in hexadecimal. Similarly, the # command deletes a whole character.

The position of the operational window is correctly verified in hexadecimal mode, with the left hand marker immediately before the first digit of the first windowed character, and the right hand marker immediately after the last digit of the last windowed character.

Commands which are remembered by ZED (such as the string alteration commands, global commands, and procedures) and which are set up in hexadecimal mode can still be used on return to the normal character handling mode. For example, the sequence

```
x+
ge/42/ad/
ge/43/bd/
x-
```

sets up two global commands using hexadecimal strings and then reverts to non-hexadecimal mode for subsequent processing.

3.4.19 Naming Groups of Commands

A facility is provided for giving a name to a group of commands, and subsequently obeying the group by name. Such a group is called a procedure. There are at present no facilities for providing arguments to procedures.

```
PROC name cg
```

This defines a procedure called *name*, and fails if such a procedure already exists. For example

```
PROC castle (E/king/***/; E/rook/king/; E/***/rook/)
```

The commands of a given procedure can be examined by the command

```
SHPROC name
```

which writes to the verification file. If the name is omitted, all existing procedures are given. A procedure can be cancelled by the command

```
CPROC name
```

Again, omission of the argument causes all procedures to be cancelled. A procedure is obeyed by means of the command

```
DO name
```

Procedures may be called from within procedures. Recursive calls are permitted. Note that if the AGP command (see section 3.4.9.3) appears inside a procedure its effects are limited to that procedure.

3.4.20 Miscellaneous Commands

```
FN sw
```

If FN+ is set the F, DF and BF commands start their search on the next or previous lines, as appropriate (DF still deletes the current line). With the default setting, FN-, the searches all start on the current line. This is appropriate for non-interactive use when the line being searched for may already have become current as a side-effect of previous commands.

```
Z /s/
```

This command is used to change the terminator for inline insertion material. The string may be of any length up to 16 characters, including zero. It is recognized in upper or lower case; in effect the search for a terminator uses the qualified string PU/s/.

```
TR sw
```

ZED normally suppresses all trailing spaces. TR+ switches this default off, and allows trailing spaces to remain on both input and output lines.

```
SQ  sw
SQS sw
```

These switch on/off the placing of sequence numbers into character positions 73-80 of output lines. SQ uses output line numbers, SQS uses source line numbers, with asterisks for inserted lines. Lines longer than 72 characters are truncated. SQ- is synonymous with SQS-; SQ+ implies SQS- and SQS+ implies SQ-.

```
Ha
```

Halt at line a. ZED will refuse to move forwards past line a until the limit is reset. If line a has been deleted an error occurs. H* resets the limit to end-of-file. A limit by context can be obtained by the use of ON, for example

```
ON (/FUNCTION/ | /SUBROUTINE/) H.
```

as H. can apply to non-original lines. The halt limit is reset if a rewind occurs.

COMMENT /s/

The string is written to the verification dataset.

COLS
COLS n
COLS +

This command generates a line of output on the verification dataset of the form

123456789 123456789 123456789 12345...

If no argument is given, the number of characters is equal to the terminal width if verification is to a terminal, or 132 otherwise. If an explicit count is given, *n* characters are output; if + is specified the length of the current line determines the number of characters output.

CS sw

Compress spaces: when this option is on, initial spaces are removed and multiple spaces are converted to one space in each source line, when it becomes current for the first time, and in each insert line at the time of insertion. Recall that, when ZED is mpoving forwards or backwards through the source, each line becomes current *as it is passed*, even though a command may not be complete at that stage. This operation does not cause verification of changed lines and is not reversible via the UNDO command (c.f. margins). CS- is automatically set when the REWIND command is obeyed.

=n

Set the current line number to *n*. If backward movement through the source has occurred, stacked up lines are also re-numbered and become original lines. Previous lines still in the output queue are not affected.

BRK sw

When the BRK switch is on (the default) the break-in action for a null response to the prompt generated by an attention is taken immediately following the current command or immediately after a new line has been made current (see section 3.3.2). BRK- can be used to delay the break-in action over a series of related commands. Any attention which occurs while the BRK switch is off is taken immediately following the next BRK+ command. (This does not apply to an attention with other keys such as V+; such attentions do not interrupt ZED's normal processing and they are always obeyed immediately.) If a second interrupting attention occurs while the BRK switch is off it is taken immediately (thus providing a way of escaping from infinite loops). The facility for delaying break-in is useful when writing non-interactive sequences of commands (or ZED procedures) to be called from an interactive command stream and is it desired to limit the places at which an attention may be taken.

REWIND

This command is only permitted when output is directed to the main destination dataset. The rest of the source is scanned (for globals etc) and all globals are then cancelled. If the source is sufficiently small that no output lines have actually been written out, backward movement to the start of the source takes place, followed by a renumbering of all the lines. Otherwise, the lines are all written to the destination, which is then closed and re-opened as a new source. This is only possible when the destination is a disc file. The original source is closed, and the WORK file is used as a temporary destination if necessary after a rewind. Any source margins previously set up by the MA command are cancelled when REWIND is obeyed and the CS and J switches are turned off.

MXLL n

Set the maximum line length accepted by ZED to *n*. Lines longer than MXLL are split up into two or more lines when they are read. The default value for MXLL is 400 and it cannot be set less than 40. Note that MXLL applies to lines of editing commands as well as to lines of source and insert datasets.

WORD /s/

The contents of the string specify which characters appear in words, from the point of view of the W string qualifier (see section 3.3.6). The character - is treated specially in the string: it may be used between two upper case letters, two lower case letters or two digits in order to specify a range of characters. Thus the default state is given by

WORD/A-Za-z0-9/

It is an error to specify - other than as described above. For example, A-z, a-9 and <-> are all illegal. In order to specify that - may itself appear in words, it must be preceded by the quoting character ", which must also be used when specifying " itself. In fact, any character may be preceded by ", but it is only necessary for " and -. This is the same rule as for Regular Expressions. Thus the command

WORD/0-9"-+.E/

defines "words" that are actually floating point numbers.

Although the WORD command defines which characters are INCLUDED in words, the W qualifier still works by checking that the characters preceding and following a matched string are NOT in this set. Thus it is not necessary to use the WORD command in order to obey

F W/well-formed/

for example, which would correctly not match "swell-formed". However, in order that

F W/well/

should not match the first part of "well-formed" it is necessary to obey

WORD/A-Za-z"-/

previously. Note that this implementation means that

WORD//

has the effect of disabling the W qualifier.

WARN sw

If WARN- is set, the warning and soft errors

```
Already selected
Buffer n is empty
Buffer n is not empty
Field overflow
```

are suppressed. If WARN- is set in a non-interactive run when one of the last three messages would have been given, no return code is set. (The first warning never sets a return code.) The default is WARN+.

ERRSTOP sw

When ERRSTOP+ is set, which is the default state, ZED abandons a non-interactive command sequence after any hard error. When ERRSTOP- is set, ZED's behaviour after a hard error does not depend on whether the current command sequence is interactive or not. Details of error handling are given in section 3.6 below.

SCC sw

Suppress control characters: SCC+ suppresses the error message which normally occurs when a line containing a significant control character is added to the output queue for a file without control characters (see section 3.7). The default is SCC-. The control characters themselves are not suppressed during ZED processing.

SO sw

Suppress overflow: if the SO switch is on, overflowing output lines are not faulted (see section 3.6.4). Instead they are split into as many lines as necessary when eventually written to the output file.

FLUSH

This command causes ZED to write the queue of output lines to the relevant output file.

STACK n

Set ZED's internal stack size to *n* bytes. The stack is an area of store set aside for saving linkage information when nested command groups or internal subroutines are called. The default size is 800 bytes, which is adequate for many applications. If the error message STACK OVERFLOW is encountered, this command may be used to increase the stack size. It may be obeyed at any time.

MXSS n

Set the maximum store size for ZED's working storage to the amount currently in use or *n* bytes, whichever is the greater. This command is intended primarily as a debugging aid, but may also be useful when ZED is being called from another program. The minimum amount of working store used by ZED is 16K bytes, and it increments this as necessary in multiples of 16K. The default maximum store is 512K.

STORE

This command is intended mainly as a debugging aid. It causes ZED to write details of its store usage to the verification dataset in the following format

```
T=tK  L=l  D=d  S=s  F=f
```

where

t =	number of kilobytes in use by ZED, excluding store for the program itself
l =	number of bytes in use for holding lines
d =	number of bytes in use for dynamic items, which include commands, strings, etc.
s =	number of bytes in use for static items, which include output buffers, NIOP bases and basic work areas
f =	number of bytes left on ZED's free queue

When ZED runs out of store it attempts to increase t by requesting more store from the operating system, and only if this fails does it start to write lines from the output queue. Hence the value of t may increase during the course of a run.

MS sw

Monitor store: this is another debugging command. When the switch is on ZED outputs details of its store usage between each line of commands. The format is the same as for the STORE command.

3.4.21 Reading the First Line

When ZED is initialized it does not read the first line of the source until a command which requires a current line is obeyed. This allows commands such as MXLL, which affect the reading of lines, to be obeyed before the first line is read. The same state obtains after a REWIND command and when a FROM command is obeyed when an end of file line is current.

The following commands, if encountered when ZED is in this state, do not cause a line to be read:

(a) All switch setting commands.
(b) All *show* commands except SHC (show column).
(c) Global commands: CG, DG, EG, GA, GB, GE, ON.
(d) Unconditional flow control commands: AGP, C, DO, RPT, STOP, Q (unless Q => W).
(e) Procedure commands: PROC, CPROC.
(f) Margin, window, justification and field setting commands:
  MA, MAV, MF, RF, RJ, JL, K.
(g) Store control commands: MXLL, MXSS, STORE, STACK, FLUSH.
(h) I/O control commands: CF, DBUFF, FROM, TBUFF, TO.
(i) Miscellaneous commands: COLS, COMM, WORD, Z.

3.4.22 Chinese ZED

The Cambridge version of ZED contains a special feature to assist in the use of the experimental Chinese terminal which has been constructed as part of a research project. The character 'circumflex', when followed by certain specific sequences of hexadecimal digits, becomes a valid ZED command. This enables certain Chinese characters to be used directly as ZED commands. Circumflex is not treated specially in any other circumstance, and if it is encountered as a ZED command, but NOT followed by a defined Chinese sequence, the usual error message, *Unknown command* is given.

3.5 THE LAST SEARCH EXPRESSION

The last obeyed search expression may be re-used in subsequent commands by writing the character & in place of a search expression or qualified string (see section 3.2.2.6). Inside an ON group & refers initially to the search expression of the group. Obeying other search expressions in an ON group does not affect the use of & outside the group.

3.5.1 & As a Search Expression

& is always valid as a search expression, provided that at least one other search expression has previously been obeyed. For example

```
F N/manxome foe/; N; F&
```

finds the second line not containing "manxome foe". For commands which take a single search expression argument, omission of the argument is equivalent to specifying &. Thus

```
DF
```

is equivalent to

```
DF &
```

Where more than one argument is required, & may not of course be omitted. For example

```
F B/ /; WH&#      \delete leading spaces
```

Because & refers to the last obeyed search expression, it need not appear in the same line of commands as the expression to which it refers. However, anomalous effects occur in sequences such as

```
F/dum/; 3(N; F&; <other commands>; F/dee/)
```

The second and subsequent executions of F& use a different search expression to the first (c.f. the ' command in section 3.4.5.5).

3.5.2 & As a Qualified String

Except in the case of the GA, GB and GE commands, & may be used in place of a qualified string argument, provided that the last execution of a search expression defined a single appropriate qualified string. This means that
(a) the execution ended in success, and
(b) one and only one qualified string not containing the qualifiers N or C was matched during the execution.

For example, the following search expressions are suitable for reference by & as a qualified string

```
/shun/
(/monstrous/ | /crow/)
(/king/ & N/red/)
```

In cases where the | operator is used, such as the second example above, it is not defined which of the alternatives will be chosen if both appear in the same line. Some search expressions may define a single qualified string by one of their alternatives and not by another, for example

```
(/Alice/ | (/Tweedledum/ & /Tweedledee/))
```

Again, it is not defined which alternative will be obeyed if both match in any one line.

For commands which take a single qualified string argument, omission of the argument is equivalent to specifying &. Thus

```
F/plum pudding/; DTA
```

is equivalent to

```
F/plum pudding/; DTA&
```

and both have the effect of

```
F/plum pudding/; DTA/plum pudding/
```

Similar anomalous effects can occur for & used as a qualified string to those described in section 3.5.1 above for & used as a search expression.

Note that & may only be used to specify a complete qualified string argument. It may not be used as a qualified string forming part of a search expression.

3.5.3 Examples of the Use of &

```
UTEOF (F/**/; WH& E&/*/)
```

This changes all occurrences of multiple asterisks into a single asterisk. This could also be done concurrently with other editing by

```
ON /**/ (WH& E&/*/)
```

A common interactive use of & is for *find and exchange* sequences, such as

```
F/resisted/
E&/rested/
```

while the use of the | operator is illustrated by

```
UTEOF (F(BS/FUNCTION/ | BS/SUBROUTINE/)
     PA&
     IF BS/XYZ/ THEN (F BS/END/;N
       ) ELSE (DF BS/END/;D)
     )
```

The above command selects from a file containing Fortran source all the routines whose names begin with the string "XYZ".

3.6 ERROR HANDLING

ZED distinguishes three sorts of error, soft errors, hard errors and disastrous errors. Certain errors are disastrous during initialization, but merely hard thereafter. Others are soft in non-interactive runs and hard in interactive runs.

All error messages are preceded by two asterisks. Some warning messages are also output by ZED, without asterisks. The command WARN- suppresses certain warnings and soft errors (see section 3.4.20).

A non-zero return code is given by ZED if any errors have been detected since the last time a line of ZED commands was read from an interactive command file, or if the STOP command is obeyed from any command file.

3.6.1 Soft Errors

Soft errors occur as a result of conditions which are well understood, and which do not prevent further processing. ZED writes a message to the verification file, sets a non-zero return code, and continues.

3.6.2 Hard Errors

Hard errors include all errors detected while analyzing lines of commands. A message is always written to the verification file and the current line of commands is abandoned. (Recall that the *current line of commands* may actually include several physical lines of input if brackets have been used.)

If the command file and the verification file are attached to an interactive terminal, or if ERRSTOP- is set (NOT the default), ZED reads the next line of commands from command file. Otherwise ZED successively abandons command files until an interactive state is found, or there are none left, in which case the ZED run is abandoned with a non-zero return code.

When ZED continues with a non-interactive command file after an error as a result of ERRSTOP-, a non-zero return code remains set. This is cleared subsequently if control ever reaches an interactive command file during the run.

3.6.3 Disastrous Errors

Disastrous errors include a number of internal consistency failures, and insufficient store and OPEN errors during initialization. A message is written to the verification file; the return code is set to 16 for internal consistency failures or 12 otherwise, and the ZED run is abandoned.

3.6.4 Overflows and Control Character Errors

During non-interactive processing, passing in a forward direction a line which is either too long for the output file for which it is being queued, or which has a significant control character when the output file has no control characters is treated as a soft error.

Reading an input line which is longer than the maximum line length is also a soft error when non-interactive (see MXLL command in section 3.4.20). The line is split up as necessary.

During interactive processing the above conditions are treated as hard errors. After giving a message ZED stops at the offending line if it is a source line, or after the current command if the error is in an insert line. Further lines of commands are then read from the terminal.

Output line overflow errors can be suppressed by means of the SO switch; significant control character errors can be suppressed by means of the SCC switch (see sections 3.4.20 and 3.7).

Overflow of a fixed format field is always treated as a soft error.

3.6.5 Store Errors

A number of hard errors related to the use of main store may occur, the most common being

```
        Insufficient store [for holding lines]
```

The parenthesized phrase is present when the store was requested for holding a line of text.

The message

```
        Stack overflow
```

means that ZED has used up that part of the store which it sets aside to hold linkage information when obeying nested command groups or internal subroutines. The default amount of space allocated to the stack is 800 bytes. This may be changed by the STACK command (see section 3.4.20).

3.7 CONTROL CHARACTER HANDLING

ZED works internally in terms of ANSI control characters, and converts lines on input and output if necessary. For files that do not contain control characters, the conversion consists of the addition of a space control character on input, and the removal of the control character on output.

The control character is handled separately, and is not treated as part of the data in a line. Normal string operations, marginning, windows, etc. operate only on the data portion. A special string qualifier C is provided to enable particular control characters to be searched for, and a special command CC is provided for setting control characters.

If a file containing control characters other than the default (space) is being edited to a destination file which does not contain control characters, the error message

```
        Control characters not present in output
```

is given when attempting to pass a line containing a significant (non-space) control character during interactive editing. ZED stops at the offending line and waits for further command input. During non-interactive editing the message

```
        Significant control character(s) lost
```

appears, but editing proceeds with the control characters being removed when the lines are eventually written to the output file. A non-zero return code is set. In both cases, use of the command

```
        SCC+
```

suppresses the error, and significant control characters are then removed without any indication when the lines are written out. Lines still in store on the output queue retain their control characters.

4. FULL-SCREEN SUPPORT

(Updated February 1987, also March 1988)

Full screen editing facilities have been added to ZED as an 'add-on' to the existing line editing features. Full screen editing is available from IBM 3270 and Cifer 2632 terminals, and also on BBC micros fitted with the SSMP (The Simple Screen Management Protocol) ROM. Most BBC micros on Computing Service premises are fitted with SSPM chips which may be purchased by Cambridge University users from the Microcomputer Support for £25. Further information in given in the files SPEC.ZEDFS.INTRO, SPEC.ZEDFS.FULL and INFO.SSMP.

In December 1987, a new full-screen editor called E was pre-released. The full-screen facilities of E are rather better than those provided by ZED, though the line-by-line facilities are currently not complete. Type 'HELP E' or see Leaflet L6 or PRINTOUT the file SPEC.E.FULL.DRAFT (c. 8,000 lines, 65 line-printer pages) for further information.

4.1 USING SSMP AT A BBC MICRO

The current version of the PHX terminal emulation program (version 1.*n*) requires the SSMP program to be explicitly entered before ZED's full-screen features can be used at a BBC micro (this will not be necessary with the next version):

```
{SHIFT/f4} (System Command)
(*)SSMP  n
```

where *n* is optional and specifies the line speed: 4=1200, 5=2400, 6=4800 (default), 7=9600.

Note that while using SSMP the scrollback and file transfer facilities are not available; to use these facilities it is necessary to re-enter the PHX program using {SHIFT/f4} and *PHX *n* .

4.2 USING FULL SCREEN ZED

To use ZED in full-screen mode use the instruction

```
FS+
```

and if required, FSW– (switch off line-wrapping) and FSN– (switch off line-numbering). This may either be done after entering ZED (using the UPDATE or ZED Phoenix commands) or on the command line with the OPT keyword e.g.

```
UPDATE  .x
FS+

UPDATE  .x  OPT=FS+

ZED  .fred  OPT=FS+FSN–FSW–
```

To exit from full-screen ZED, press the f3 function key (instead of usual Wind-up).

The key-strokes which may be used to perform full-screen ZED functions are listed on the next page.

4.2.1 Full Screen ZED Key-strokes

When using full-screen ZED under SSMP, many operations may be performed using the red function keys and various other key-strokes. A birdcage (slip of paper labelling the function keys) is available. Note that the COPY key deletes characters rather than copying them. Ordinary ZED instructions may be used by moving the cursor to the top line of the screen, typing the command and then pressing {CTRL/RETURN} (together).

action	**ISO**	**BBC micro**
suspend SSMP session	ESC >	SHIFT/CTRL/f7
restart SSMP session	ESC ?	SHIFT/CTRL/f8
interrupt (attn)	ESC !	SHIFT/CTRL/f9

The following key-strokes control the movement of the cursor:

action	**ISO**	**BBC micro**
left	CTRL/H	←
right	CTRL/L	→
up	CTRL/K	↑
down	CTRL/J	↓
next tab	CTRL/I	TAB
previous tab	CTRL/T	CTRL/TAB
command (top) line	CTRL/G	CTRL/f5
next line (or line number)	RETURN	CTRL/↓
last non-space on line + 1	CTRL/O	CTRL/→
previous line (or line number)	CTRL/P	CTRL/↑
first non-space on line	CTRL/U	CTRL/←
start line (or line number)	CTRL/Y	CTRL/f4

The following key-strokes perform character editing functions:

action	**ISO**	**BBC micro**
delete char	CTRL/C	copy
insert space	CTRL/S	CTRL/f8
delete/erase previous	DELETE	–
erase previous	–	CTRL/DELETE
erase right	CTRL/V	CTRL/f0
toggle insert mode	CTRL/A	CTRL/COPY

The following key-strokes request ZED actions:

action	**ISO**	**BBC micro**
enter (return)	CTRL/D	CTRL/RETURN
insert block	ESC 0	f0
expand line	ESC 1	f1
quit (equivalent to Wind-up)	ESC 3	f3
insert line	ESC 6	f6
delete line	ESC 7	f7
scroll left	ESC a	SHIFT/←
scroll right	ESC s	SHIFT/→
scroll up	ESC d	SHIFT/↑
scroll down	ESC f	SHIFT/↓

5. COMMAND LIST

Argument abbreviations:

a, b	line numbers (or . or *)
cg	command group
m, n	numbers
q	qualifier list (possibly empty)
se	search expression
s, t	strings of arbitrary characters
sw	switch value (+ or -)
name	sequence of letters (4 significant)

Qualifiers:

B	beginning
E	ending
C	control character
L	last
P	precisely
W	word
U	uppercase
S	significant
N	not
R	regular expression
n	count
[n,m]	window
_n,m_	window

Regular expression special characters:

"	quoting character
?	wild character
:	alternation
( )	grouping
-	range
#	repetition
~	negation

BY FUNCTION

Line Selection (3.4.2)

Ma	move to line a
M*	move to end-of-file
M-	move back as far as possible
M+	move forward to high water
N	move to next line
P	move to previous line
F se	find (forward)
BF se	backward find

Line Insertion and Deletion (3.4.3)

Insert material for the I and R commands may be any one of
<in-line text, any number of lines terminated by Z on its own line>

BUFF n	to insert an in-store buffer
COPY n	to insert a copy of an in-store buffer
/s/	to insert the file with ddname or dsname s

IC	insert current line
IS /s/	insert string
Ia	insert before line a
Ra b	replace lines a to b
Da b	delete lines a to b
DF se	delete find
DREST	delete rest of source
DBUFFn	delete contents of buffer n
DBUFF	delete contents of all buffers

Line Windowing (3.4.4)

RFn m	restrict find
RVn m	restrict view
>	move character pointer to right
<	move character pointer to left
PR	reset pointer to start
PA q/s/	point after
PB q/s/	point before
EWR	end window right
EWL	end window left
EWA q/s/	end window after
EWB q/s/	end window before

Single Character Operations (3.4.4.2)

$	force lower case
%	force upper case
_	change character to space
#	delete character

Splitting and Joining (3.4.6)

SA q/s/	split after
SB q/s/	split before
CL /s/	concatenate

Text Inspection (3.4.7)

Tn	type n lines
TLn	ditto, with line numbers
T+	type to high water
TL+	ditto, with line numbers
TBUFFn	type buffer n

String Operations (3.4.5)

A q/s/t/	after
AP q/s/t/	after and point
B q/s/t/	before
BP q/s/t/	before and point
E q/s/t/	exchange
EP q/s/t/	exchange and point
DFA q/s/	delete from after
DFB q/s/	delete from before
DTA q/s/	delete till after
DTB q/s/	delete till before
LC q/s/	force lower case
LCL	force entire line to lower case
UC q/s/	force upper case
UCL	force entire line to upper case
'	repeat last string operation
UNDO	undo current line
SHC q/s/	show column
CC/s/	set control character

File and Buffer Control (3.4.8)

C /s/	commands from s
C BUFF n	commands from buffer n (destructive)
C COPY n	commands from buffer n
LCE sw	force editing commands to lower case
UCE sw	force editing commands to upper case
FROM BUFF n	select buffer n as source
TO BUFF n	select buffer n for output
TBUFFn	type contents of buffer n
DBUFFn	delete contents of buffer n
DBUFF	delete contents of all buffers
SHBUFF	show non-empty buffer numbers
TO /s/	select dsname or ddname
TO	select TO
FROM /s/	select dsname or ddname
FROM	select FROM
CF	close all (closeable) files
CF /s/	close file with ddname s

Conditionals and Loops (3.4.9)

Wherever ELSE may appear, the forms

```
   ELIF se THEN cg ELSE cg
   ELUL se THEN cg ELSE cg
```

may also appear.

```
IF se THEN cg ELSE cg
UL se THEN cg ELSE cg
```

IFEOF cg ELSE cg	if end-of-file

ULEOF cg ELSE cg	unless end-of-file
WH se cg	while
UT se cg	until
UTEOF cg	until end-of-file
RPT cg	repeat indefinitely
AGP	abandon group

Margins (3.4.10)

MAn m	source margins
MAVn m	verification margins

Fixed Format Fields (3.4.11)

MFn1 n2 ...	mark fields
Kn1 n2 ...	temporarily suspend fields

Justification (3.4.12)

JLn	set justification length
J sw	enable/disable justification
RJn m	restrict justification
VJ sw	justification verification

Global Operations (3.4.13)

GE q/s/t/	global exchange
GA q/s/t/	global after
GB q/s/t/	global before
ON se cg ELSE cg	complicated global
CGn	cancel global n
DGn	disable global n
EGn	enable global n
SHGn	show global n
VG sw	global change verification

State Display (3.4.14)

SHD	show data
SHF	show files
SHG	show globals
SHS	show switches

Termination (3.4.15)

W	windup
Q	quit (exit command level)
STOP	sic (return code 8)

Chinese ZED (3.4.22)

^ (circumflex)	introduces Chinese command

Line Verification (3.4.16)

V sw	automatic verification
VI sw	verify with indicators
VJ sw	justification verification
VG sw	global change verification
VN sw	verify line numbers
VT sw	verify line texts
VWR sw	verify right window
VWL sw	verify left window
VW sw	verify both windows
VX sw	verify in hex for indicators
VE sw	verify edits lines
?	verify current line
!	ditto, with character indicators

Procedures (3.4.19)

PROC name cg	define
SHPROC name	show
CPROC name	cancel
DO name	obey

Hexadecimal Mode (3.4.18)

X sw	switch hex mode on/off

Miscellaneous (3.4.20)

FN sw	find next mode
Z /s/	set insert terminator string
TR sw	trailing spaces
SQ sw	sequence numbers in output
SQS sw	source sequence numbers
Ha	halt at line a
COMM /s/	comment to verification file
COLS	)
COLS n	)verify column numbers
COLS +	)
CS sw	compress spaces
=n	set line number
BRK sw	enable/disable break-in
REWIND	sic
MXLLn	max line length
WORD /s/	set word characters
WARN sw	warnings mode
ERRSTOP sw	non-interactive error action
SCC sw	suppress control characters
SO sw	suppress overflow errors
FLUSH	flush output queue
STACK n	set stack size to n bytes
MXSS n	set maximum store size to n bytes
STORE	show store usage
MS sw	monitor store usage

6. ALPHABETICALLY

<	move pointer left
!	verify with character indicators
$	force lower case
_	exchange char for space
^ (circumflex)	introduces Chinese command
%	force upper case
>	move pointer right
?	verify current line
#	delete first char
'	repeat string alteration
=n	set line number
A q/s/t/	after
AGP	abandon group
AP q/s/t	after and point
B q/s/t/	before
BF se	backwards find
BP q/s/t/	before and point
BRK sw	enable/disable break-in
C /s/	commands from s
C BUFF n	commands from buffer n (destructive)
C COPY n	commands from buffer n
CC /s/	set control char
CF	close all (closeable) files
CF /s/	close file s
CG n	cancel global
CL /s/	concatenate line
COLS	)
COLS n	)output column numbers
COLS +	)
COMMENT /s/	output string
CPROC name	cancel procedure
CS sw	compress spaces
Da b	delete
DBUFF	delete all buffers
DBUFFn	delete buffer n
DF se	delete find
DFA q/s/	delete from after
DFB q/s/	delete from before
DG n	disable global
DO name	obey procedure
DREST	delete rest of source
DTA q/s/	delete till after
DTB q/s/	delete till before

E q/s/t/	exchange
EG n	enable global
EP q/s/t/	exchange and point
ERRSTOP sw	non-interactive error action
EWA q/s/	end window after
EWB q/s/	end window before
EWL	end window left
EWR	end window right
F se	find
FLUSH	flush output queue
FN sw	find next mode
FROM	select FROM
FROM BUFF n	select buffer n
FROM /s/	select alternate source file
GA q/s/t/	global after
GB q/s/t/	global before
GE q/s/t/	global exchange
Ha	halt at line a
Ia	insert
Ia BUFF n	insert buffer n
Ia COPY n	insert copy of buffer n
Ia /s/	insert dsname or ddname s
IC	insert current
IF se THEN cg ELSE cg	if
IFEOF cg ELSE cg	if end-of-file
IS /s/	insert string
J sw	justify
JLn	justify length
K n1 n2 ...	suspend fields
LC q/s/	force lower case
LCE sw	force editing commands to lower case
LCL	force entire line to lower case
Ma	move to line a
MAn m	margins
MAVn m	margins for verification
MFn1 n2 n3 ...	mark fields
MS sw	monitor store usage
MXLLn	set max line length
MXSS n	set maximum store size
N	next line
ON se cg ELSE cg	complicated global

P	move to previous line
PA q/s/	point after
PB q/s/	point before
PR	pointer reset
PROC name cg	define procedure
Q	quit (exit command level)
Ra b	replace
Ra b BUFF n	replace with buffer n
Ra b COPY n	replace with copy of buffer n
Ra b /s/	replace with dataset
REWIND	rewind source
RFn m	restrict find
RJn m	restrict justification
RPT cg	repeat
RVn m	restrict view
SA q/s/	split after
SB q/s/	split before
SCC sw	suppress control chars
SHBUFF	show non-empty buffer numbers
SHC q/s/	show column
SHD	show data
SHF	show files
SHG	show globals
SHPROC name	show procedure
SHS	show switches
SO sw	suppress overflow errors
SQ sw	sequence
SQS sw	sequence with source line numbers
STACK n	set stack size
STOP	stop
STORE	show store usage
Tn	type
TBUFF n	type buffer
TLn	type with line numbers
TO	select main output (TO)
TO BUFF n	select in-store buffer n
TO /s/	select alternate output file
TR sw	trailing space control
UC q/s/	force upper case
UCE sw	force editing commands to upper case
UCL	force entire line to upper case
UL se THEN cg ELSE cg	unless
ULEOF cg ELSE cg	unless eof
UNDO	undo current line
UT se cg	until
UTEOF cg	until end-of-file

V sw	verify
VE sw	verify edits
VG sw	verify globals
VI sw	verify with indicators
VJ sw	verify justification
VN sw	verify line numbers
VT sw	verify text of lines
VW sw	verify window
VWL sw	verify left hand window
VWR sw	verify right hand window
VX sw	verify indicators in hex
W	windup
WARN sw	set warning mode
WH se cg	while
WORD /s/	specify word characters
X sw	hexadecimal
Z /s/	set insert terminator

INDEX