CDL Z-TEL: Z80 Text Editing Language

User's Manual

June 30, 1977

Written by

Evelyn J. Tate and Randall B. Enger

Table of Contents

Chapter I

Introduction to Z-TEL

Z-TEL is a CDL utility program designed to provide a powerful set of techniques for editing text files. In addition to providing standard text editing features such as adding text, deleting text, searching, and changing text, Z-TEL offers ways for moving blocks of text, for command iteration, for expression arguments to commands, for forward and backward searching, and for invoking command macros.

Not only does it contain these powerful editing features, Z-TEL is easy to use because it includes detailed error detection and a number of options for tailoring the program to the user's individual needs.

This manual has four chapters and several appendices. This is Chapter One. Chapter Two is provided as a guide to the first-time user, and explains the concepts upon which Z-TEL is based. This is NOT a complete description of the entire editor. Chapter Three elaborates on Chapter Two by examining a typical Z-TEL editing session in detail. Chapter Four is a reference guide for the experienced user; it contains a detailed description of all Z-TEL commands and their effects.

Error messages are given in Appendix A, and some more advanced examples are given in Appendix B. Appendix C contains a short summary of all Z-TEL commands; it is meant to be a "quick-glance" guide to Z-TEL. Additional appendices will be included for special versions which become available.

Chapter II

Overview of Z-TEL Operation

The model for much of Z-TEL is paper tape: the file being edited is viewed as a string of individual characters. Conceptually, a pointer exists pointing somewhere into this string or at one of the ends, and almost all of the editor commands act relative to this pointer. The pointer is always before or after a character. The characters to the RIGHT of the pointer are thought of as being in the "positive" direction, and the characters to the LEFT of the pointer are thought of as being in the "negative" direction.

A. Deleting characters

For example, to delete the character immediately to the right of the pointer one enters "1D" (or just "D", since in the absence of an argument, the value "1" is generally assumed); to delete the character immediately to the left of the pointer, one enters "-1D" (or just "-D").

The commands "D" and "-D" are instances of the "delete characters" command, which takes a numeric argument and deletes that many characters from the pointer position. Again, negative numbers refer to characters before the pointer and positive numbers refer to characters after the pointer.

B. Moving the pointer

The "C" command behaves similarly, but instead of deleting characters, this command moves the pointer position. The "C" command takes one numeric argument - positive or negative - and moves the pointer to the RIGHT if the argument is positive and to the LEFT is the argument is negative. For example, the command "-5C" will move the pointer position to the LEFT 5 places.

Another way to move the pointer is with the "J" or "jump" command. "J", like "C", takes one argument, but instead of interpreting this argument as a relative offset from the current pointer, the argument is treated as an absolute position in the file. Thus, "0J" will put the pointer at the beginning of the file, and "100J" will put the pointer after the 100th character in the file.

The pointer will not go past the end of the buffer in either direction, so it doesn't hurt to use an argument too big. This is true in general for all commands in Z-TEL: the

two ends of the file behave as infinite sources of characters (for character oriented commands) and infinite sources of lines (for line oriented commands).

C. Inserting characters

Characters can be inserted at the current pointer position. The characters are inserted between the character to the left of the pointer and the character to the right of the pointer. For example, the command "ihi there" when the pointer is between the "well " and "George" of the string "well George" will result in the new string "well hi there George". By convention, the pointer is left after the last character inserted, so that the two commands "ihi " and "ithere" (in the order given) are equivalent to the one command "ihi there".

D. Searching

A character string can be searched for using the "s" command. The "s" is followed by a string, which is compared to the characters in the file starting at the current pointer position. "S" also takes a numeric argument, the sign of which is the direction to search. If this number is positive (remember that the absence of an argument is the same as "+1") the editor compares the string to the characters to the right of the pointer looking for the specified occurrence of the string, and if this number is negative, characters to the left of the pointer are searched. Again by convention, the pointer is positioned after the rightmost character of the matched string -- if the search was successful. If the string couldn't be found, then the pointer isn't moved.

E. Lines

Although the file is treated as a string of characters, the notion of a "line" has been added for convenience. A "line" is merely the string of characters between two linefeed characters, not including the left linefeed but including the right one.

Some editor commands are line oriented. For instance, the "L" command moves the pointer a number of lines, positioning it always at the start of a line -- just after the previous line's linefeed character. The command "3L" moves the pointer to the right past three linefeed characters, or "down 3 lines". The ends of the file are treated as linefeeds when a line oriented command is looking for more linefeeds than exist. The command "3L" given when only one line exists in a file will position the pointer at the end of the file. No matter how hard you try, you can't move the pointer further than just past the ends of the file.

F. Files and paging

A "file" to the editor is an arbitrarily long string of ASCII characters that is presented to it through the reader device. The beginning of a file is the first character presented to the editor through the reader; the end of the file is whenever a <control-z> is encountered in the input stream or when the monitor returns the carry flag set after a call on the reader device. The carry flag is returned by the monitor when the reader runs out of tape or otherwise stops for a fixed time period. A flag is set by the editor upon encountering one of these conditions; it can be reset by the user and more text can be read in if necessary, for example, in the case of spurious <control-z>'s in a file. (See the description of value register "E" in Chapter Four, Section D.)

The editing process generally consists of reading a file, making changes to it, and writing out a new file. All changes are made in the editor's file buffer kept in main memory. The <control-z> character is not considered part of the file itself. When one is read, the editor sets the "end of file" flag and stops reading. When the writing out of a file is finished, a <control-z> is appended to the new file by the editor so that end of file may be found when the file is read in again.

The editor can handle files much larger than the amount of main memory available to it for the file buffer. When this is the case, part of the file must be brought in, changed, and then written out so the next section can be read in. This process is loosely referred to as "paging" through the file. A "page" has no fixed length and is not delimited by any special character. Rather, what can fit in the file buffer, or what is currently in the file buffer at any given time is called a page. Paging through the file is a one-way proposition: once a page has been written out, it can be accessed again only by re-editing the file. (See the "EA" command described in Chapter Four.)

Some commands perform an action with respect to a "page", the same way some commands perform an action relative to "lines" (e.g. L, K) or relative to individual characters (e.g. C, D). The "yank" ("EY") command, for example, deletes the current page and reads in a new page. The looseness of the term "page" can cause problems if this command is used carelessly. The detailed descriptions in Chapter Four will describe which commands use "pages".

G. Basic syntax

The basic command structure is the same for all Z-TEL commands. In the following, square brackets denote an optional field, and angle brackets denote a class of similar items or numbers. All commands (with minor exceptions) fall into the following forms:

[<n>[,<m>]][:][@][f]<command>[<reg-spec>][<string>]

where <n> and <m> are expressions and <command>, <reg-spec>, and <string> denote the class of commands, registers (to be discussed immediately), and strings, respectively. No one command takes all of the optional arguments, especially since <reg-spec> and <string> are mutually exclusive. The following is an explanation of each part of the syntax:

<command> -- one of the many commands described in Chapter Four. Most commands are single letters; some consists of two letters, the first of which is always "E".

<n> or <m> -- an expression. Expressions consist of numbers, variables, parentheses, and the following five operators: plus (+), minus (-), times (*), divide (/), and unary minus (-). The expression is evaluated using the "standard" precedence rules: "*" and "/" take precedence over "+" and "-"; unary minus has higher precedence than "*" and "/". Operators of equal precedence are evaluated left to right. Parentheses can be used to alter the standard precedence, and nesting of parentheses is provided but is limited to about 10 due to stack requirements. Variables -- value registers -- can be used in expressions in the same way numbers are. See Chapter Three for examples of the use of expressions.

<string> -- a string that either starts immediately after the command and terminates with the first escape character encountered, or starts after the first character after the command and terminates with the second occurrence of the first character. The "@" option invokes the alternate string specification. For example, the "S" (search) command will find (if one's there) the first occurrence to the right of the pointer of the string "abc" in each of the following cases: (1) "sabc$", (2) "@s/abc/", and (3) "@seabce".

<reg-spec> -- a single letter or number denoting either a text register or a value register. Text registers hold text and are denoted by the numbers "0" through "9". Value registers hold numbers between -65,535 and +65,535 and are denoted by numbers or letters. Some of the value registers have predefined meanings, and some of these are "read only" variables. Chapter Four explains these in depth. See also sections I and J below.

The "f" option -- signals that this command has two string arguments following. For example, "fsabcdef" is the search command with the two-string option in effect. Most commands that take two strings search for the first and replace it with the second. The above command finds

"abc", deletes it, and inserts "def" in its place. There is no requirement that the two strings be of equal length.

The ":" option -- means something different for each command, but usually indicates that something is to be deleted. An example is the "P" command, which puts text into a text register: if the ":" option is present, the text is deleted from the file buffer after it is copied into the text register, whereas if the ":" is absent, the text exists in both places after the command. See the detailed descriptions in Chapter Four for the effect that the ":" option has on each command.

Any number of commands may be strung together to form a "command string" -- a list of commands that are to be executed in order until the end is reached or an error is encountered. A command string is not executed until two consecutive escapes are typed at the console, but the escapes are not considered part of the command string and are ignored during command execution.

H. Iteration

A command string can be executed repeatedly by enclosing it in angle brackets ("<" and ">"). The character "<" is a command that optionally takes a numeric argument (which must be positive) as the number of times the enclosed command string is to be executed. For example, the command string "20<I-$>" will insert the character "-" 20 times.

Angle brackets nest just as parentheses do, so command strings like the following are possible:

"20<I-$5<CI.$>0TT>"

which will (insert a dash followed by five (skipping one character and then inserting a dot) and then display the line without moving the pointer) all 20 times. Nesting depth is limited to five due to stack requirements.

With no argument to the "<", the editor assumes an argument of infinity (which really means 65,535). The command string "<IHI$>" will insert the string "HI" until the file buffer is full. Also, a minus argument or a zero argument is treated the same as no argument.

This is not as useless as it might seem at first glance, because certain situations terminate the iteration before the count goes to zero. A failed search, upon the execution of the closing ">", will cause the iteration to stop. For example, the command string "J<Sabc$0TT>" will print out every occurrence in the file buffer of the string "abc". When the ">" is executed after the search fails, the iteration will stop even though the command string has not executed 65,535 times. Notice that the "0TT" will be

executed whether the search succeeds or fails, producing a situation where the current line will always be printed once before the iteration terminates. The semicolon command is available to provide a more graceful stop.

The semicolon ";" command is closely related to angle brackets, and, in fact, it can't be used outside of an iteration. The semicolon command, when encountered in a command string, checks the result of the last search command in the iteration. If this search was successful, then the semicolon command does nothing. If the search failed, however, then the semicolon command terminates the iteration. Perhaps the most common use of this command is in a case like the following:

"<Sxxx$;CIyyy$OTT>"

which will find all occurrences of "xxx" and insert the string "yyy" one character after the last "x", and then print the line. If the semicolon were NOT included, the same action would take place UNTIL no more of the string "xxx" were found. Then, instead of stopping, an additional "yyy" would be inserted and the line printed. Then the iteration would stop, because - as mentioned above - iterations stop on failed searches. The semicolon command, then, acts as a kind of "guard" for the rest of the string.

I. Value registers

Value registers hold numeric values in the range -65,535 to +65,535. These registers can be set using the "W" command: for instance, "1W0" loads into value register 0 the value 1. Value registers can also be used as arguments to commands by using "V" to pull the value out. For instance, "55W5" followed by "V5*3<I*$>" will insert the character "*" 165 times.

Some value registers are used by the system for storing information about the file or the environment. The end-of-file status is always stored in value register "E" where 0 means no end-of-file and 1 means end-of-file (other values for "E" are undefined). Another example is the "X" value register, which contains the "eXtent" of the matched string after a successful search. Therefore, the command string "Sabc$-VXD" will delete the first occurrence of the string "abc". Section D of Chapter Four describes all the value registers.

J. Text registers

Text registers hold text. The amount of text they can hold is limited only by the size of available memory. The "P" command ("Put") is used to put text into a text register. It has two forms, one will put a number of lines into a text register (e.g., "5P3" will copy 5 lines starting at the current pointer position into text register 3), and

the other will put a number of characters into a text register (e.g., "0,9P2" will copy the first nine characters of the file into text register 2).

The "P" command, when the colon option is present, will delete from the file buffer those characters that have been "put" into the text register. For instance, "5P3" copies 5 lines into text register 3 and leaves them in the file buffer, whereas the command "5:P3" puts 5 lines into text register 3 and then deletes them from the file buffer.

The "G" command does the inverse of "P": "G" "gets" text from a text register. This command takes only the text register argument, which means only the whole text register can be "gotten". The text from the text register is copied into the file buffer at the current pointer position, and the pointer is moved to the right by the number of characters inserted, leaving it after the last character inserted.

The "G" command understands the colon, when present, to mean a request to delete the contents of the text register after the contents are copied into the file buffer.

K. Macros

The contents of a text register will be interpreted as a command string when the "M" command is used. "M" takes a text register and causes the text in that text register to be interpreted as editor commands. For example, if text register 1 contains the text "Sabc$20<I*$>0TTL" and the command "M1" is entered, then the above text string will be executed as if it had been entered from the console.

Macros are made even more useful by the "*" command which, when entered as the first character after the editor's prompt, copies the previous command into a text register. Using this command is an easy way of saving the retyping of a command. This sequence:

<command-string>
*<text-register>
M<text-register>

will cause the <command-string> to be executed again.

L. Input and output

Input and output are accomplished through the two commands "append lines" ("A") and "output lines" ("O"). Each takes a numeric argument, which is interpreted to be the number of lines to be input (for "A") or output (for "O").

For example, the command "50A" will append to the end of the file buffer fifty lines from the reader device, unless

either there's not room for fifty lines or there aren't fifty lines left in the file. The pointer position after the append command is unchanged.

For output, the "O" command will take the specified number of lines from the top of the file buffer and output them to the punch device. The pointer position is left at the beginning of the buffer after the output command.

Some commands will cause I/O to take place depending on the environment. Examples of these commands are "N" and "EN" (which are the "non-stop search" commands: these read in more text if the string is not in the current file buffer), the "EX" command (which terminates the editing session), the "Y" command (which "yanks" in a new page), among others.

M. Getting started

Z-TEL is a relocatable, "rom-able" program, which requires about 100-hex bytes of working storage at absolute 100-hex. Address 100-hex contains a jump instruction to the beginning of the program, so after the program is loaded, a "G100" to ZAPPLE will start the program. The program will ask for the low and high addresses that it's to use for its file buffer. If no addresses are given, then Z-TEL will use the larger of the space between its lower end and 200-hex or so (where the fixed working storage ends) and its upper bound and the highest address available (as ZAPPLE tells it). If addresses for the file buffer are given (and they must be in DECIMAL), then Z-TEL will use these as long as this buffer area doesn't overlap the program code. Also, it is the user's responsibility to ensure that the file buffer resides in write-able main memory. The editor will then display the size of the file buffer and issue a prompt character ("*"). At this time, the editor is ready to accept commands.

Chapter III

Some Basic Examples

This section shows how some of the basic Z-TEL commands might be used in an editing session. Our short source file contains poetry which needs some revision.

In the following examples, user-typed commands (in upper-case only) are preceded by the Z-TEL prompt character (an asterisk); the dollar-sign symbol represents an escape character.

We start the file edit process by loading Z-TEL and then loading the file with the "A" ("append") command. Z-TEL starts at address 100-hex, so the ZAPPLE command "G100" starts the editor. Z-TEL signs on and issues a prompt. The file can then be loaded using the "append" ("A") command.

When the first page of the file has been read in, Z-TEL will respond with another prompt. Here we might want to find out what's in the file buffer. The command "HT", which is equivalent to "0,ZT", prints out the entire buffer -- it types all characters from the first to the last.

```
*HT$$
To be or knot 2 B:
Whether 'tis nobler 'tis the mind to suffer
Or to take arms against a sea of troubles,
Alas! poor Yorick.  I knew him, Horatio;
A fellow of the bows and arrows of outraged fortune
etc.
etc.
etc.
```

The text pointer is initially at position zero -- that is, before the first character of the buffer. The first error which we will tackle is the "knot" in the first line (which is the current line).

```
*SK$-0OTT$$
```

This command string does the following: searches for the character "k" (the pointer is left after the "k"), deletes the preceding character (which is the "k"), and types the entire line without moving the pointer (the "OT" types the line up to the pointer, and the "T" types the part of the line which is after the pointer). At the end of the command string the pointer is left right before the string "not".

```
To be or not 2 B:
*S2$-02Dito be$0T$$
```

This command string finds the character "2" on the line, deletes it and the following two characters (which are " B"), inserts the string "to be", and leaves the pointer directly after the inserted string. Note that the "OT" types only that part of the text line which is before the pointer; since the line so far doesn't include a carriage return and line feed, the Z-TEL prompt is displayed on the same line as the typed text.

```
To be or not to be*CI  that is the question:$OLT$$
```

This command string (typed directly after the prompt character) moves the pointer forward one character to space over the colon in the text, and then inserts more text on the same line. The "OLT" moves the pointer to the beginning of the line and types out the entire line.

```
To be or not to be:  that is the question:
```

We now tackle the next line. The second occurrence of the word "'tis" is a mistake.

```
*L.T$$
Whether 'tis nobler 'tis the mind to suffer
*2FS'tis$in$OLT$$
```

The "2FS" command finds the second occurrence of the first argument string ("'tis") and replaces it with the second string ("in").

```
Whether 'tis nobler in the mind to suffer
```

The next line is in fine shape, but the ones after that have some problems.

```
*4T$$
Whether 'tis nobler in the mind to suffer
Or to take arms against a sea of troubles,
Alas! poor Yorick.  I knew him, Horatio;
A fellow of the bows and arrows of outraged fortune
*2L.$of the$V0,.-3KY$$
```

This command string moves the pointer down two lines, to point to the line beginning with "Alas". It searches for a string which marks the end of the text we wish to remove. The search leaves the pointer right after the word "the" (which we don't want to remove). Value register "0" contains the pointer position before the search (see chapter IV), and "." is the pointer position after the search. The command "V0,.-3K" deletes all characters from the old pointer position up to three characters before the new pointer position, thus sparing the word "the". The command "Y" is equivalent to "-TT", and types the preceding line and the now-current one.

```
Or to take arms against a sea of troubles,
the bows and arrows of outraged fortune
```

The "bows and arrows" line contains a few mistakes. Here we correct them and print out the results of what we've done so far.

```
*DIT$FSbows$slings$Soutraged$-DIous$0,.TT$$
```

This changes lower-case "t" to upper-case "T", "bows" to "slings" and "outraged" to "outrageous". The command "0,.T" is an example of the "T" command with two arguments; it means "type out everything between character position 0 (the beginning of the text buffer) and character position "." (the current pointer position)". The second "T" command, with a default argument of +1, types out the remainder of the current line.

```
To be or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
Or to take arms against a sea of troubles,
The slings and arrows of outrageous fortune,
```

The fourth line should be the third, and vice versa. Rather than deleting a line and retyping it, we can move it elsewhere by using a text register.

```
*0L:P2-LG2HT$$
```

This command string moves the pointer to the beginning of the current line (the one we want to move), "puts" one line (the default) into text register 2 (any other would do as well), moves the pointer back up a line, "gets" the text from the text register, and types out the entire buffer.

```
To be or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
etc.
etc.
etc.
```

We have yet to clean up the end of this.

```
*Setc.$0L.,ZKIAnd by opposing end them?
$-2T$$
```

This command string finds the first "etc.", deletes all text from the beginning of that line to the end of the text buffer (which is the three "etc."'s), inserts the correct phrase to complete the sentence (notice that we have included a carriage return and line feed in the inserted text string), and types out the last two lines of the file.

```
Or to take arms against a sea of troubles,
And by opposing end them?
```

One more change which we might want to make to our text is to double-space it rather than single-space it. This is easy to do with the iteration feature.

```
*OJ<S
$;I
$>HT$$
```

This command string does the following. The pointer is positioned at the start of the text buffer. The text is searched for a carriage return; when one is found, a second carriage return is inserted after it. When the search fails because there are no more carriage returns in the text, the iteration is ended and the entire text buffer is typed.

To be or not to be: that is the question:

Whether 'tis nobler in the mind to suffer

The slings and arrows of outrageous fortune,

Or to take arms against a sea of troubles,

And by opposing end them?

To save our corrected file, we use the "EX" command, which writes out the contents of the text buffer (our now-edited file) back to the punch device, and returns control to the monitor.

```
*EX$$
```

Chapter IV

Detailed Description of Commands

A. Basic commands

A [<n>] A

The "append" command inputs <n> lines from the reader device and stores them at the end of the file buffer. The pointer position is unchanged. If there isn't enough room in the buffer for <n> more lines, as many characters as will fit are read in. A <control-z> in the input marks an end-of-file, and appending will stop. The <control-z> is not stored in the file buffer.

If <n> is zero, then the "A" command will read text until either the buffer is three-quarters full or an end-of-file situation is encountered. Negative arguments are not meaningful to the "A" command. If <n> is omitted, then a value of 1 is assumed.

B B <string>

Upon execution of the "branch" command, control is transferred to the first label (defined by "!" -- see the "!" command) in the current command string that matches the <string> argument to "B".

The "current command string" is defined to be the buffer or text register from which commands are currently being executed. This means that branching into or out of a macro can't happen, since only the labels in the "current command string" are searched.

C [<n>] C

This command moves the pointer <n> characters from the current position. If <n> is negative the pointer is moved backwards. If <n> is zero the pointer position is unchanged. If no argument is specified, n=1 is assumed ("-C" means "-1C").

D [<n>] D

This command deletes characters from the file buffer. If <n> is positive, the <n> characters immediately following the pointer position are deleted; if n= is

negative, the <n> characters preceding the pointer position are deleted. If <n> is zero, nothing happens; if <n> is omitted, "1" is assumed ("-d" means "-1d").

E (not a command by itself)

This is used to create an "extended" command set; e.g., "ER", "EW". For specifics see the "extended commands" section.

F (not a command by itself)

This is used to modify the "S" (search) and "N" (non-stop search) commands. For specifics refer to those commands.

G [:] G <text-register>

This command gets the contents of the text register indicated by <text-register> (where <text-register> is a number from the set 0 - 9) and copies the contents into the file buffer immediately following the current pointer position. The pointer position is changed by the amount of text inserted, so the pointer points after the last "gotten" character. When modified by ":", the command causes the contents of the text register to be deleted after the get.

H (not a command)

This is not a command; it is the equivalent of "0,Z" and is used with commands which can take two arguments. E.g., "HT" is equivalent to "0,ZT" and will type out all characters between the first and the last character positions in the file buffer.

I [@] I<string>

This command inserts <string> into the file buffer at the current pointer position. Normally, the ESCape character signals the end of the string to be inserted. When "@" is used, an alternate string delimiter (other than escape) is used; this provides for the insertion of a single escape in the text. (Since a double escape always indicates the end of a command string, two "I" commands are required to insert two consecutive escapes.) The alternate string delimiter is the first character following the "i" and can be any character not used in <string> (including <space>). E.g., the command strings "ihello$", "@i/hello/", and "@iiheiloi" all do the same thing.

J [<n>] J

The "jump" command puts the pointer after character position <n>, for positive <n>. If <n> is zero, the pointer will be put before the first character in the buffer. If <n> is negative, it is treated as a positive number with value (0 - <n>); e.g., "-1J" is equivalent to "1J". Note that J without an argument means "0J", not "1J".

K [<n> [,<m>]] K

If only one argument is supplied, <n> lines from the current pointer position will be "killed". When <n> is positive, "<n>K" will delete all characters from the current pointer position up to and including the <n>th following line feed character. When <n> is negative, "<n>K" will delete all characters from the character preceding the pointer position back to the beginning of the current line, plus the <n> preceding entire lines. "0K" will delete all characters on the current line which precede the current pointer position. When two arguments <n> and <m> are supplied, all characters following position <n> and up to and including position <m> will be deleted; the pointer will be left following position <n>. For example, "1,2K" will delete one character (which was in character position 2) and leave the pointer at position 1. If arguments are omitted, "1K" is assumed.

L [<n>] L

This command moves the pointer position by <n> lines. If <n> is positive, the pointer will be positioned after the <n>th following line feed. If <n> is zero, the pointer is moved to the beginning of the current line. If <n> is negative, the pointer is moved to the beginning of the <n>th previous line. If <n> is omitted, "1L" is assumed.

M M <text-register>

This is a macro invocation. The text in text register <text-register> is treated as a command string and executed. The contents of the text register are not changed. The following restrictions apply: a) any iterations or other commands must be wholly contained within the macro; likewise, iterations cannot start outside and finish inside a macro; b) no command can be executed in a macro which affects the contents of the text register in which the macro is stored.

N [<n>][:][@] [/] N<string> [;]

This is a non-stop search. It is similar to the "S" command (q.v.) But if the <string> is not found in the current buffer, the contents of the buffer are written out and another page is read in and searched. "N" will fail only when the last page has been read in and the <string> still hasn't been found. When "N" fails, the pointer is left following the last character of the page currently in the buffer (which is the last page of the file).

When the "f" modifier is present, the "N" command takes a second string which will replace the string found - after a successful search. Thus, the command string "<FNhi$hello$>" will replace all occurrences of the string "hi" with the string "hello" throughout the entire file (starting at the current pointer position, of course.)

The "N" command takes one numeric argument which, like the numeric argument to the "S" command, is the occurrence for which to search. Unlike the "S" command, however, the "N" command does not handle negative arguments.

O [<n>] O

This command outputs <n> lines from the beginning of the file buffer to the punch device and deletes them. If the file buffer contains fewer than <n> lines, the entire file buffer is written out and deleted. If <n> is omitted, 1 is assumed. The pointer is left at the top of the file buffer after the deletion of the output lines has occurred.

P [<n> [,<m>]] [:] P <text-register>

This command puts text from the file buffer into the text register specified by <text-register>. If ":" is used, the text is deleted from the buffer after it has been copied into the text register. Use of arguments is the same as for the "K" command (q.v.); "<N>P" will put the number of lines specified by <n>, while "<n>, <m> P" will put characters as specified by <n> and <m>. If the text register contained text at the time of the put, its previous contents are deleted and the new text copied in. The command "0,0P<text-register>" will delete the contents of register <text-register> and leave it empty.

Q (not defined)

R (not implemented)

S [<n>] [@] [f] S<string> [;]

This command searches the file buffer, to the right if <n> is positive and to the left if <n> is negative, for the <n>th occurrence of <string>. <N> must be a positive number; if it is omitted, 1 is assumed. If the <n>th occurrence of <string> is found, the pointer is left immediately following the last character in the <n>th occurrence of <string>. If the <n>th occurrence isn't found, the pointer is left after the last character in the last occurrence which was found (if the absolute value of n > 1) or at the same position as at the beginning of the search (if |n| = 1, or if |n| > 1 but no occurrences were found). "@" can be used to specify an alternate delimiter (other than the normal escape) to be used for <string>; use of "@" is described in the "I" command (q.v.). A semicolon can follow the search command only in an iteration; it means, "If found, continue with the command string, else jump outside the scope of the current iteration". E.g., the command string

"<2Shello$;Igeorge$> HT"

would insert the string "george" after every other occurrence of "hello" and print out the entire buffer when done. (Note: the semicolon need not immediately follow the "s" command, and, indeed, is a command in its own right. See "special character commands" section.)

When the "f" modifier is present, the "S" command takes a second string. After a successful search, the editor will replace the first string (the one found) with the second string. For example, "FSbad$worse$" will change the first occurrence in the file buffer of the string "bad" to the string "worse".

T [<n> [,<m>]] T

This command types <n> lines if one argument is specified, or all characters between positions <n> and <m> if two arguments are given. If <n> is negative, the previous <n> lines and all characters on the current line which precede the pointer position are typed. If <n> is zero, all characters on the current line which precede the pointer position are typed. If <n> is omitted, "1T" is assumed.

U (not defined)

V V <value-register>

This command pulls the value out of the specified value register. Valid value register names are 0 - 9 and a - z. Certain registers are read-only, containing system variables; these can be interrogated, but not set by the "W" command. A list of predefined value registers is given in section D of this chapter.

W <n> W <value-register>

This command puts the value <n> into the specified value register if the specified value register is not a read-only register. An error occurs if it is a read-only register. (See Section D below for a complete list of the value registers.)

X [@] X <string>

This command displays <string> on the console device when executed in a command string. This is useful especially in macros. "@" can be used to specify an alternate string delimiter (other than the escape character).

Y [<n>] Y

This command is shorthand for "-<n>T<n>T". If <n> is negative the sign is discarded.

Z (not a command)

This represents the number of characters in the file buffer. It is a value, not a command. E.g., "ZJ-T" positions the pointer after the last character in the buffer and prints out the preceding full line and any partial line.

B. Extended Command Set

EA EA

The "edit again" command terminates the editing of the current file, just like "EX", but does not exit back to the monitor. Instead, after the file is completely written out another prompt is given. At this point, all text registers and value registers are intact. If, when editing a large file, it is desired to move a large piece of code back to a page that has been written out already, then use "EA" instead of "EX". Follow this with the appropriate sequence of "append" commands to load the first part of the file and then with the "get" command to include the new text.

EC [:] EC <value-register>

The "extract character" command puts the number corresponding to the ASCII value of the character to the right of the current pointer position into the value register specified. If the colon option is invoked, the character is deleted from the buffer. If the pointer is at the bottom of the file buffer (i.e., "." = "Z"), then an error occurs.

EF EF

The "end file" command closes the current output file by writing out a <control-z>, does any required name changing, and returns control to the editor for continued editing.

EI <n> EI

The "extended insert" command inserts the character whose ASCII representation is the number <n> into the file buffer at the current pointer position. The pointer is moved past the character inserted. <N> must be a number between 0 and 127 inclusive.

EK [:] EK <value-register>

The "keyboard" command provides a means for interactive command strings and macros. Upon execution of the "EK" command, Z-TEL will wait for one character from the console input device. When the character appears, its ASCII representation is put into the specified value register.

The optional colon controls the echoing of the character. If the colon is present, the echoing is inhibited; without the colon, the character is echoed.

EL [<n>[,<m>]] EL

The "print to list device" command functions exactly like the "T" command except that the output goes to the list device instead of to the console device.

EN [<n>] [@] [f] EN <string>

The "extended non-stop search" command is much like the "N" command, but instead of writing out the current page of text, the command deletes the current page before reading the next page. This can be useful for splitting a file into several pieces or for extracting a piece of a file.

A specific occurrence of the <string> can be sought by including a numeric argument <n>. This number must be positive.

EQ EQ

The "quit editing" command traps back to the monitor without writing out any more of the file.

ES [<n>] ES <text-register>

The "extended search" command performs the same search operation as the "S" command, except the string to search for is in the specified text register instead of the input string. The same value registers are used here as are in the "S" command (i.e., the O, S, X value registers).

ET <n> ET

The "extended typeout" command displays the character whose ASCII representation is the number <n>.

EV EV <text-register>

The "extended value of" command is much like "V" -- the value of the register is returned, except with the "EV" command the value is represented by the string (assumed to be decimal, optionally prefixed by a "-") found in the specified text register.

EX EX

The "exit" command is the normal method used to end an editing session. The input file (if any) is copied after the file buffer to the output file. When the file has been completely written out, control is passed

back to the monitor.

EY EY

The "yank" command brings a new page into the file buffer after deleting the one already there. As mentioned in Chapter Two, "a page" is loosely used throughout this text, and is roughly equivalent to whatever is in the file buffer (i.e. main memory) or -- when doing I/O -- about half the total capacity of the file buffer.

C. Special-character Commands

= [n>[,<m>][:]=

The "=" command displays the number <n> on the console device. If a second argument <m> is present, then it is interpreted as a field length in which the number <n> is to be displayed right justified. For example, "VF,6=" would display something like "^^243i". This feature is useful for outputting columns of numbers. If the number to be displayed (<n>) won't fit in the field specified, the number will be prefixed by the character "!" to indicate overflow. The colon is used to inhibit the output of the carriage return, linefeed sequence that normally follows the number. <m>, if present, must be between 0 and 256 exclusive.

\ [n>[,<m>][:]\

The "\" command is much like the "=" command, the main difference being that the number is inserted into the file buffer at the current pointer position instead of being displayed on the console device. As above, the second argument, <m>, denotes the size of the field in which the number will appear - again, right justified. If present, <m> must be between 0 and 256 exclusive. If the first number won't fit in the field as specified by the second number, then the first number will be prefixed by the character "!". The colon option for "\" is the inverse of the colon option for "=": if present with "\", the ":" causes a carriage return, linefeed sequence to be inserted after the number.

% [<n>]%<value-register>

The "%" command adds the number <n> to the contents of the specified value register and both stores the sum back into the value register and returns it to the next command. If <n> is missing, the value 1 is assumed (as with most commands). Negative values of <n> are permissible. For example, if value register one contains the number 55, then the command string "%1=" will add 1 (the default) to 55, storing the sum back into value register 1 and passing the sum (56) to the "=" command to be displayed on the console device.

? ?

The "?" command starts a command-by-command trace of the editor. (A further description of the trace feature will be forthcoming soon.)

# #

The "#" command resets the trace feature.

. .

The "." ("dot") command is equivalent to the number of characters to the left of the current pointer position -- that is, it is the numeric representation of the current pointer position. For example, after a successful search, the command "VS,.K" will delete the string that was just matched.

< [<n>]< .... >
> The "<" command signals the start of an iteration. The ">" command signals the close thereof. The command string enclosed by the "<", ">" pair is repeatedly executed -- <n> times if <n> is present, or "forever" if <n> is not. ("Forever" is defined to be 65,535.) The exception to this rule is when a search is included within the "<", ">" pair. For example, "35<%i\i. Hi there$>" will insert 35 numbered "Hi there"s into the file buffer. Searches, when failing, will cause the iteration to stop, when either a ";" or a ">" is reached. (See the discussion under the ";" command for more detail.) Iterations can be nested up to 5 levels. The pair "<", ">" must be wholly contained in a macro or outside a macro; splitting across a macro boundary is not recognized.

; ;

The ";" command changes the flow of control inside an iteration. This command checks the result of the most recent search command; if the search was successful, the ";" command does nothing. If the search wasn't successful, control is transferred to the matching ">" and the iteration is terminated. The ";" need not follow the search command immediately -- anytime later within the iteration will do.

* *<text-register>

When used as the first character of console input, the character "*" is interpreted as a command which puts the previous command string as text into the specified text register. The "*" character normally signifies the multiplication operator, and is interpreted as a command only when it is the first character in the input buffer when the input string is completed. If another character was typed, <rubout> or <control-u> or <control-e> can be used (see below) to erase it; this special use of "*" can still be invoked. In fact, up to nine characters can be typed and rubbed-out, as long

as the "*" is the first character in the buffer when the string completes. Commands after the "*" in this command string are ignored.

, ,

The "," command is used to separate two arguments, and causes no overt action to take place. "<n>,<m>T", for example, is the means to pass two arguments to the "T" command.

<linefeed>

A linefeed character as the first character typed on the console device after a prompt is interpreted as the string "LT<escape><escape>". It is echoed as a "LT$$", and is executed immediately. It is included as a shorthand notation for a commonly used command.

<backspace>

A backspace character (control-h) as the first character typed on the console device after the prompt is interpreted as the string "-LT<escape><escape>". It is echoed as the string "-LT$$", and is executed immediately.

! [@]!<string>

The "!" command does nothing, but it is used by the "B" command to find where in the command string to branch to. The "B" command will search the command string from the beginning looking for a matching string prefixed by a "!". When found, the next command executed will be the command following the "!" command. Note that the "!" command provides a convenient way to document long macros, since the <string> is not used in any way by the editor. As usual, the "@" option provides an alternate way to specify the <string>.

Note that the "B" command will not look outside the current command string. In other words, if a "B" command is executed inside a macro, then only labels (the "!" commands) INSIDE the macro body are searched.

" <n>"<branch condition>

The """ command is a conditional branch command. The argument <n> is compared according to the specified <branch condition> and, if satisfied, control is passed to the matching "'" character to the right. The different <branch condition> codes are as follows:

E - branch if <n> = 0
N - branch if <n> # 0
L - branch if <n> < 0
G - branch if <n> > 0
D - branch if <n> is (in ASCII) a digit
A - branch if <n> is (in ASCII) alphabetic
V - branch if <n> is (in ASCII) lower case alphabetic
W - branch if <n> is (in ASCII) upper case alphabetic

, '

The "'" command is the closing character for the "`" command. When a condition is satisfied for the "`" command, then the editor searches for the matching "'". When found, control is passed to the command past the "'". The "'" command does nothing by itself; it merely marks a place in the command string. Notice that "`" and "'" nest the same way "<" and ">" do, and matching requires being at the same level lexicographically.

<control-e>

The <control-e> character has two meanings: it can be used during the typing of a command string when it is desired to erase the entire string typed since the prompt, and it can be used when the editor is executing a command string, and it is desired to cause a break in the execution. This break occurs the next time a command is about to be executed. In either case, a prompt is given and the editor waits for the next input string from the console device. The current command string execution is terminated.

<control-f>

The <control-f> character causes the entire input string typed since the prompt to be retyped. The <control-f> is not inserted into the input buffer unless it is preceded by a <control-r> (see below). This is usful when many corrections have been made using <rubout> and <control-u> (again, see below).

<control-o>

The <control-o> character stops output to the console without stopping the execution of a command string. A second <control-o> removes this inhibition. The inhibition is always removed when the editor is waiting for console input. This character is useful for leapfrogging through a long typeout: by hitting pairs of <control-o>'s, one can skip over the displaying of groups of characters.

<control-r>

The <control-r> character causes the next character (with the exception of <control-x>) to be inserted into the command string input buffer without any special effects of the next character occurring (e.g. <control-f>). A <control-r> itself may be inserted into the command string by prefixing it with another <control-r>.

<control-s>

(not yet implemented)

<control-t>

The <control-t> character is much like the <control-f> character, but instead of causing the entire input string to be retyped, only the current line is retyped -- with any local corrections made, as above. The <control-t> is not inserted into the input buffer unless it is preceded by a <control-r>, in which case the <control-t> is treated as a normal character.

<control-u>

the <control-u> character is to the <control-e> character what the <control-t> character is to the <control-f> character. That is, <control-u> erases the current line of input. The <control-u> character is not inserted into the buffer unless it is preceded by a <control-r>, in which case the <control-u> is treated as a normal character.

<control-x>

The <control-x> character causes control to be passed back to the ZAPPLE monitor. As long as no registers and no memory locations are disturbed, editing can be resumed with a simple "G" command to ZAPPLE. This is useful when it is desired to assign a device for input or output (the ZAPPLE "A" command), for instance.

<rubout>

The <rubout> character causes the previous character in the input buffer to be echoed on the console device (unless the console device is a CRT -- covered next) and then deleted from the buffer. The <rubout> character is not itself inserted into the input string. If the console device is a CRT, then the cursor is moved back one space, unless it is at the leftmost edge. The "T" value register controls whether the

editor considers the console device to be a CRT or not. (See the section "Text and Value Registers", below.)

: (not a command by itself)

The ":" modifier is a general purpose command modifier, and is usually used to indicate that something should be deleted. For instance, "5P1" means "copy 5 lines into text register 1" and "5:P1" means "copy 5 lines to text register 1, and then delete them from the file buffer". See the individual command descriptions for the specific actions taken in each case.

@ (not a command by itself)

The "@" modifier can be used with commands that take strings to change the string delimiter from <escape> to anything else. For example, the command "@s/abc/0tt" will do the same thing as "sabc$0tt". The string delimiter in all cases is the first character after the command, and the string is closed by the second occurrence of that character.

D. Text and Value Registers

There are 36 value registers and 10 text registers. The value registers are denoted by "0" through "9" and "A" through "Z"; text registers are denoted by "0" through "9". Some of the value registers have predefined meanings -- values that are set by Z-TEL and sometimes by the user that affect the behavior of the editor or return interesting information. For example, the value register "L" is used by the system to control the number of characters displayed on a line. Thus "VL=" displays the current setting (it defaults to 72) and "80VL" sets it to 80.

Currently no text registers have predefined meanings.

The following describes all the predefined value registers.

A conversion inhibit flag

When value register "A" contains 0 (the default value), control characters encountered during output are converted by the editor into printable representations -- a "^" followed by a letter. When value register "A" has been set by the user to a non-zero value, this conversion is inhibited and characters are sent untranslated to the console device.

C carriage-return enable flag

When value register "C" contains 0 (the default value), a double escape ends a command string. When value register "C" has been set by the user to a non-zero value, a carriage return is equivalent to double escape as a command string terminator. This may be useful with a terminal on which use of the escape is inconvenient. When carriage return enable is on, a carriage return character may be inserted into the text by preceding it with <control-r> in the command string. The linefeed character must be added explicitly in this case. (Note: carriage return enable does not disable double escape as a command terminator. Double escape always acts as a command terminator.)

D duplex flag

When value register "D" contains 0 (the default value), the editor treats the console device as a full-duplex terminal. This can be changed by the user to a non-zero value, which indicates half- duplex. Note: this feature is not yet implemented.

E end-of-file flag

This is set to 1 by the editor when a <control-z> character has been encountered on reading characters from the reader device. This normally indicates end of the input file. When value register "E" is set to 1, the "A" (append) command does nothing. The user can reset this flag to 0 (for example, if he knows that the end of file condition was spurious, caused by garbage in the input file) so that further "A" commands can be done.

F free space

Value register "F" is a read-only value register which always contains the amount of free space left in the text buffer (in bytes); it is updated by the editor whenever characters are inserted or deleted. To inquire about the amount of empty file buffer space left, the command "VF=" can be used. This value, however, reflects a changing internal state, and it is consistently accurate for this purpose at the beginning of the execution of a command string only.

I iteration depth

Value register "I" is a read-only register which contains the current iteration depth. This will be zero outside an iteration and will be incremented by one for every "<" executed and decremented by one for every ">" executed.

L line length

Value register "L" is used to control the number of characters on a line; it defaults to 72. This can be changed by the user (for example, to accommodate a terminal with a shorter or longer line length) by using the "U" command.

N nulls required after a carriage return

Value register "N" is used to provide a delay for the physical carriage return when a carriage return character is sent to the console device. The default value is 3; increasing this will provide a longer delay for the carriage to return, and decreasing "N" will shorten this delay.

O old "dot" after a successful search

Value register "O" is a read-only value register which contains, after a successful search ("S", "N", etc.) command, the pointer position prior to the search. (Remember that the pointer is left after the matched string on a successful search.)

R lines read from the reader device

Value register "R" contains the number of lines read from the reader device from start of editing. This value register is a read-only register.

S string start position after a successful search

Value register "S" is a read-only register which contains, after a successful search, the pointer position of the start of the matched string. For example, "VS,.T" will print out the matched string if done immediately after a successful search.

T teletype/crt flag

Value register "T" contains the flag used by the editor to control the handling of <rubout>'s. The setting of this value register is done during initialization and depends on the monitor's setting for the console device: "T" is set to 0 for no assignment (which is assumed to be a teletype) or 1 for a CRT. In "teletype mode", the editor echoes the rubbed-out character. When set to 1, the editor will move the cursor back one position using the <backspace> character, erase the character just typed by printing a <space>, and then back up (since the <space> just moved the cursor again).

U upper case/upper-lower case flag

Value register "U" contains the flag used by the editor to control the handling of the output of lower case letters. If this register is 0 -- which it defaults to -- then no special provision is made for lower case letters. If this register contains a 1, then all lower case letters (the ASCII range 61H to 7AH) are converted to upper case for console output. Lower case letters inserted into the file are not affected by this flag. Also, the non-zero setting provides a way to use the "altmode" key in the same fashion as the "escape" key. The escape character -- for those terminals with only an "altmode" key -- can be entered by typing "control-shift-K". This provides the means to set value register "U".

W lines written to the punch device

Value register "W" contains the number of lines written to the punch device from the start of editing. The "W" value register is a read-only register.

X extent of matched string after a successful search

Value register "X" contains, after a successful search, the length of the string that was just matched. The two command strings ".-VS=" and "VX=" are equivalent, so this register is merely a convenience. Value register "X" is a read-only register.

Y character to display for "Y" command

Value register "Y" -- when not zero -- is interpreted as the character to display between the two halves of the "Y" command. The "Y" command is defined to "-<n>T<n>T", but if value register "Y" is non-zero, then the "Y" command is defined to be "-<n>TVYET<n>T". (See the "V" and "ET" commands for more detail.) This can be used to insert a visual "marker" into the output of the "Y" command at the current pointer position.

Z total size of workspace

Value register "Z" is a read-only value register which contains the total size of the "in-core" file buffer. This is set up at editor initialization time and remains unchanged during the editing session. Users of the standard version will note that the limits provided for the file buffer will not equal the total size as shown in value register "Z". This is because part of that area is used for the editor's stack.

Appendix A

Error Messages

Most error messages display only a number in order to save room. This appendix explains all these numbers.

Error #	meaning
19	No room for insert. This can happen on an insert ("I") command as well as an "FS", a "\", and others.
30	Expression

expression.

40 Missing operator. An operand was left "dangling" during the evaluation of an expression. For example, the command string "55V5=" will cause this error message.

41 Missing string delimiter. A character to match the first character after a string command used with the "@" option was not found.

42 Unknown "E" command. This command is undefined.

43 Unrecognized use of "F". Only a subset of the commands that take string arguments take an extra one. This command wasn't one of them.

51 Unknown command. This command is not in the editor's set.

52 Missing left angle bracket. A right angle bracket (">") was used without the editor seeing a matching left angle bracket ("<") first.

53 Iteration nested too deeply. Due to stack requirements of iteration, the nesting of angle brackets is limited to five.

54 Semicolon used outside an iteration. The semicolon (";") command has a meaning inside iterations only.

55 Macro calls nested too deeply. Due to the stack requirements of macro processing, the nesting of macro calls is limited to about five. This restriction includes recursive calls.

56 Missing right angle bracket. An iteration was started with a left angle bracket ("<") but the editor ran out of input without finding a matching right angle bracket (">"). Note that the command string will be executed once; the editor does not have a look-ahead feature.

57 Missing label. The editor tried to execute a branch command ("B") but couldn't find the appropriate label in the current environment.

58 Unknown conditional. The letter after the "conditional branch command" (the " command) was not in the set of known conditions.

59 Missing "'". A conditional branch was attempted, but no matching single quote could be found.

61 Negative argument to "O". The "output lines" command insists on positive arguments.

71 Bad value register specifier. The character given for the value register was not "0" through "9" nor "A" through "Z".

72 Protection error. This command attempted to change a read-only value register. Read-only registers are usually used internally by the editor and show up in the value register list because of the information they convey. Changing them is prevented because some untold side effects could occur.

101 Second argument too small. It turns out that almost all commands that take two arguments require the second one to be larger than -- or at least equal to -- the first. This error indicates that the command encountered a situation where the first was larger than the second.

115 Text registers in use by macros are "sacred". If a macro is executing from a particular text register, then that text register can't be deleted. This means a ":G" command and "F" command are disallowed for this text register.

116 No room for "put". The free space was smaller than the size of the text requested to be put into a text register.

117 No room for "put" or "get". This indicates a limitation of the current version in the handling of the case of ":F" and ":G".

118 No room for "get". There's not room enough in the work area for the size of text to be gotten.

122 Field length too big. The size field in which the number is to be positioned (right justified) is greater than 255. This is an arbitrary cut off point, but it was felt that a number larger than this is probably a typing mistake. This applies to the "=" and "\" commands.

123 Field length zero. The size field in which the number is to be positioned is zero. While provisions are made for handling overflow (the number is prefixed by a "!"), a

field length of zero is probably a mistake.

131 Bad text register specifier. The character used to specify the text register was not in the range "0" to "9".

137 Negative argument to a non-stop search. A backward search was requested of "N" or "EN". These two commands only work in the forward direction, mainly because paging itself works only in the forward direction.

139 Previous command not available. The special form of the "*" command was used when the previous command was overwritten prior to the entering of the "*" command. This can happen in several situations: more than nine characters were typed before the "*" was entered, or the space was needed by an insertion of text, or the buffer is so full that there's just no room.

171 Non-numeric text. The text in the specified text register can't be converted into a number because it's not all numbers. This error message only occurs when the "EV" command is used.

173 Non-ASCII character. The numeric argument to this command does not represent any ASCII character. This message only occurs when the "EI" command is used.

174 End of buffer reached. The "EC" command attempted to look at the next character in the buffer but found that the end of buffer had been reached, that there were no more characters.

Some error messages are textual and usually self-explanatory. They are included here for completeness.

"BUFFER ERROR"

The location specified during initialization for the file buffer overlaps the editor code space or the fixed location variables that start at location 100-hex.

"CAN'T FIND 'XXX' "

The specified string was not found.

"FOUND ONLY NN OF 'XXX' "

This message occurs when the string searched for was found, but not in the quantity requested. The command "4Sabc" when only 2 copies of the string are to the right or

the pointer will respond with "FOUND ONLY 2 OF 'abc' ". Note that the pointer will be left at the right of the last occurrence of the string that was found.

Appendix B

Advanced Examples

This appendix contains several examples of the lesser-used commands in Z-TEL. These examples, which use some of the more powerful commands, demonstrate why Z-TEL is indeed a "language" instead of just a text editor.

The first command string counts the lines in the buffer and to the current pointer position. Then, these values are displayed on the console.

```
0W10W2.W0           @!/zero counters, and save "dot"/
0J          @!/start at beginning of file buffer/
<@S/
/;%1>       @!/count all CR'LF strings/
V0J         @!/jump back to old "dot"/
<@S/
/;%2>       @!/count lines from "dot" to end/
V0J
            @!/display the counts:/
@X/LINE COUNT = /V1:=                @!/"V1" has total/
            @!/then the current if there are any lines/
V1"N@X/, CURRENT LINE = /v1-v2+1="
```

This next command string reverses the lines in the file buffer. The algorithm is as follows. The last line in the file buffer is put into a text register. This register is then unloaded at the top of the file, and the pointer position is recorded. Note that the pointer is left AFTER the last character gotten from the text register. At this point, if there are no more lines after the one we just got, we stop. Otherwise, we go get the (new) last line in the file buffer and continue.

```
0J          @!/start at beginning/
@!/LOOP/ @!/declare a label/
Z-."E@B/OUT/"
            @!/when "dot" = "Z", we branch out/
            @!/otherwise .../
.U0         @!/put pointer into valreg 0/
ZJ-L:P9     @!/put last line into textreg 9/
V0J         @!/go back to end of previously moved line/
G9          @!/get this line/
@B/LOOP/
@!/OUT/
0J          @!/and leave the pointer at 0/
```

This next command string converts all upper case characters in the file buffer to lower case letters. This takes advantage of the ASCII collating sequence in which all

lower case letters are 20-hex (or 32 decimal) higher than the corresponding upper case letters.

```
@!/LOOP/ @!/declare a label/
Z-."E@B/OUT/"
         @!/we're done when we're out of chars/
:EC4     @!/get ASCII of next char and delete it/
V4"U     @!/if it's upper case then .../
V4+32W4' @!/we add 32 to make it lower/
V4EI     @!/and insert it, whether changed or not/
         @!/just do one line for the example/
         @!/that is, stop on a CR (=13 decimal)/
V4-13"N@B/LOOP/"
@!/OUT/
```

Many enhancements could be made to this basic command string to make it "smarter" in its translation. One might be the following.

The next command string also converts upper case letters to lower case letters, but it will leave the first upper case character after a period as an upper case character instead of translating everything. We do this as follows. When a period comes along (ASCII representation is 46 decimal), we record this fact by setting value register 0 to 1. And just before translating a character from upper to lower case, we check this value register, and we bypass the translation if the value is 1.

```
1W0      @!/don't convert first char/
@!/LOOP/
         @!/test for being done/
Z-."E@B/OUT/"
:EC4     @!/get a char (and delete it)/
         @!/test for "."/
V4-46"E1W0'
         @!/now, is it upper case?/
V4"U     @!/do this if so/
         @!/first testing the bypass flag in 0/
V0"EV4+32W4"
         @!/don't translate if bypass flag is on/
0W0'     @!/reset it in any case/
V4EI     @!/insert the char, changed or not/
V4-13"N@B/LOOP/"
         @!/and stop on a carriage return/
@!/OUT/
```

The next example demonstrates the interactive features of Z-TEL. The command string accepts a string of decimal digits and outputs the number in its hexadecimal representation.

Assuming text register 2 has the following command string (which will be invoked as a macro below):

```
V3-9"G   @!/non-decimal ?/
V3+55ET  @!/display corresponding hex/
'@B/OUT/"
```

```
V3+48ET  @!/otherwise just print the number/
@!/OUT/
```

then the following command string will do this conversion.

```
@!/START/
0W1       @!/register 1 accumulates number in/
@X/
-> /      @!/display a prompt/
@!/LOOP/
EK4       @!/get a character from console/
          @!/rubout gives another chance/
V4-127"E@X/<RUBOUT>/@B/START/'
          @!/evaluate on a CR/
V4-13"E@B/EVAL/'
          @!/if a digit, add it in/
V4"D10*V1W1V4-48+V1W1@B/LOOP/'
          @!/no good otherwise/
@X/ ??/@B/START/
@!/EVAL/ @!/start of hex display routine/
@X/
/
V1/4096W3
          @!/print first hex digit if non-zero/
V3"NM2'   @!/textreg 2 does printing/
          @!/now shrink V1/
V1-((V1/4096)*4096)W1
V3W2      @!/save as zero-suppression flag/
          @!/and print second hex digit/
V1/256W3
V2+V3"NM2'
V3+V2W2
          @!/shrink it again/
V1-((V1/256)*256)W1
V1/16W3
          @!/and print third hex digit/
V2+V3"NM2V3+V2W2'
          @!/and once more .../
V1-((V1/16)*16)W3
M2        @!/print the last one/
@X/h
/
```

Appendix C

"Quick-Glance" Reference Guide

A. Basic Commands

A	"Append lines"	[<n>] A
B	"Branch"	B <string>
C	"move pointer by Character"	[<n>] C
D	"Delete character"	[<n>] D
E	(not a command by itself; see section B)	
F	(not a command by itself)	
G	"Get text from register"	[:] G <text-register>
H	(not a command; "H" is equivalent to "0,Z")	
I	"Insert string"	[@] I<string>
J	"Jump"	[<n>] J
K	"Kill lines or text"	[<n> [,<m>]] K
L	"move pointer by Lines"	[<n>] L
M	"invoke Macro"	M <tr>
N	"Non-stop search"	[<n>][@] [f] N<string> [;]
O	"Output lines"	[<n>] O
P	"Put text into register"	[<n> [,<m>]] [:] P <tr>
Q	(not defined)	
R	(not implemented)	
S	"Search"	[<n>][@][f]S<string>[;]
T	"Type text"	[<n> [,<m>]] T
U	(not defined)	
V	"Value of"	V <value-reg>
W	"set value"	<n> W <value-reg>
X	"display string"	[@] X <string>
Y	"display context"	<n> Y
Z	(not a command)	

B. Extended Commands

EA	"Edit Again"	EA
EC	"value of Character"	[:] EC <value-reg>
EF	"End output File"	EF
EI	"Extended Insert"	<n> EI
EK	"get from Keyboard"	[:] EK <value-reg>
EL	"print to list device"	[<n>[,<m>]] EL
EN	"Extended Non-stop search"	[<n>] [@] [f] EN <string>
ES	"Extended Search"	[<n>] ES <text register>
ET	"Extended Type-out"	<n> ET
EV	"Extended Value of"	EV <text register>
EX	"Exit to Monitor"	EX
EY	"Yank"	EY

C. Special-character Commands

```
=     "display value"                 <n>[,<m>][:]=
\     "insert value"                  <n>[,<m>][:]\
%     "increment register"            [<n>]%<value-reg>
?     set trace                       ?
#     reset trace                     #
.     "dot"                           .
<     iteration                       [<n>]< .... >
;     exit iteration on failed search   ;
*     "save previous command"         *<text-register>
,     argument separator              ,
<linefeed> "LT" if first character
<backspace> "-LT" if first character
!     "define label"                  [@]!<string>
"     conditional branch              <n> "<branch cond>
      E - branch if <n>  = 0
      N - branch if <n>  # 0
      L - branch if <n>  < 0
      G - branch if <n>  >  0
      D - branch if <n>  is (in ASCII) a digit
      A - branch if <n>  is (in ASCII) alphabetic
      V - branch if <n>  is (in ASCII) lower case alphabetic
      W - branch if <n>  is (in ASCII) upper case alphabetic
'     end of conditional branch     '
<control-e> erase entire input; break execution
<control-f> retype full input string
<control-o> inhibit output
<control-r> take next character as is
<control-t> retype current input line
<control-u> erase current input line
<control-x> trap to ZAPPLE
<rubout> erase previous character
:     (not a command by itself)
@     (not a command by itself)
```

D. Value Registers with Predefined Meanings

A	conversion inhibit flag
C	carriage-return enable flag
D	duplex flag
E	end-of-file flag
F	free space
I	iteration depth
L	line length
N	nulls required after a carriage return
O	old "dot" after a successful search
R	lines read from the reader device
S	string start position after a successful search
T	teletype/crt flag
U	upper case/upper-lower case flag
W	lines written to the punch device
X	extent of matched string after a successful search
Y	marker character for "Y" command
Z	total size of workspace

Appendix D

Additions for a CP/M Version of Z-TEL

To take advantage of the features offered by the CP/M operating system, several changes were made to Z-TEL. These changes include some new commands, some new error messages, and some minor changes to the existing command set to improve its capabilities in a disk environment. This appendix documents all these.

Under CP/M, Z-TEL is started by typing either "Z-TEL" or "Z-TEL <filename>". The second form will cause an automatic "EB<filename>" followed by a "0A", before the initial prompt is given.

A. New or modified commands

EB EB <disk string>

The "edit with backup" command opens the specified file for editing and opens a temporary output file. On closing (e.g. "EA" or "EX"), the input file is renamed to ".BAK" and the temporary output file is renamed to the input file name. An "ER" or "EW" command after an "EB" is permissible, but will cause the name changing here to be bypassed during closing.

Note: <disk string> ::=

[<dev1>:] <file name> [.<file ext>] [<dev2>:]

where <dev1> is the input device (if absent, the logged in device is assumed); where <dev2> is the output device (if absent, the input device name is assumed). <Dev2> is used only for the "EB" command; <dev1> is used for the output device in the "EW" command.

ED ED <disk string>

The "file delete" command deletes the specified file. This is useful when the disk is full and a file must be removed for editing to continue. (Alternatively, one can end the current file with the "EF" command and establish a new file on a different disk with the "EW" command.) Caution is urged with the use of this command. (Note: the syntax and description of <disk string> can be found under the "EB" command.)

EF EF

The "EF" command is changed slightly for the CP/M version of Z-TEL. The file is closed and a <control-z> is written out, as in the standard version. Then, before returning to the editor, any name changing necessary is done: the temporary file is changed from ".$$$" to the requested name after the previous file (if any) is renamed. See the description under "EX" below.

EO EO

The "edit over" command does everything the "EA" command does, with the addition of doing an "EB" on the new file -- if an "EB" was used before. If the "EB" command wasn't used, "EO" is the same as "EA". (Note: "EO" is currently implemented to be the "EA" command.)

EQ EQ

The "EQ" command hasn't changed for the CP/M version. It should be noted, however, that a temporary file (with the extension ".$$$") is left on disk after the command is executed.

ER ER <disk string>

The "edit read" command opens the specified file for input. (See note under the "EB" command for the syntax and description of <disk string>.) The "ER" command does not cause any data to be transferred into the text buffer; the "A" (append) command must be used. The "ER" command also resets value register "R" to zero.

EW EW <disk string>

The "edit write" command opens the specified file for output. (See note under the "EB" command for details of <disk string>). The first device specified is used, not the second device specifier. The "EW" command resets value register "W" to zero.

EX EX

The "exit" command is the normal method used to end an editing session. If "EB" (as opposed to "ER" or "EW") was used to open the disk file, the previous backup file (if any) is deleted, the input file is renamed to ".BAK", and the temporary output file is renamed to the name of the original input file. A situation can arise where a file is found on disk with the same name as the one about to be used for the new output file. This

situation arises when the output disk is different from the input disk for the "EB" command or when the string passed to the "EW" command represents a file already on disk. (Note that the "EW" command will not do anything with backup files.) The old file on the output disk is renamed to ".PRV" rather than ".BAK" since it hasn't been "updated" and isn't a backup copy of the file just edited. When all the name changing (if any) is done, control is returned to the monitor.

B. New value register descriptions

Value register "R" contains the number of lines read from the reader device. This meaning isn't changed by CP/M, but the register is reset to zero whenever an "ER" or "EB" command is executed. This is still a read-only register.

Value register "W" contains the number of lines written to the punch device from either the last "EB" or "EW" command. The "W" value register is a read only register.

C. New error messages

161 No file for input. The "A" ("append") routine was called but no file had been opened. (Files are opened using the "ER" and "EB" commands.)

162 No file for output . An "O" command ("output lines") or an "EX" command was issued (among a few others) without an output file having been opened first. Output files are opened with the "EW" and "EB" commands.

163 File not found. The file requested (by an "ER" command) was not on the specified disk.

"DIRECTORY FULL"

The specified disk has no room for the directory entry for this file.

"FILE EXTENSION ERROR"

The file name extension was unacceptable to CP/M.

"DISK FULL"

The output disk has no more room for the file. When this message appears during an editing session, the file should be closed (using the "EF" command) and a new output file should be opened on a different drive (using "EW") or a file on the output disk should be deleted using the "ED" command.

"FILE NOT FOUND"

The requested file is not on the specified disk.

"CLOSE ERROR"

This error shouldn't occur. If it does, either the editor made a mistake somewhere or the disk or drive was not write-enabled.

"[NEW FILE]"

This message means that the requested file was not found and a new file has been opened for output. This message will appear when the "EB" command is used to open a new file (including the implicit "EB" when starting up the editor from CP/M by typing "Z-TEL <filename>".)

"FILE NAME ERROR"

The file name specified was unacceptable to CP/M.

"REMEMBER: YOU HAVE ANOTHER 'XXX' "

If an "EW" command is used with a name for which a file already exists, this message will appear. On closing, the old file will be renamed to "<file>.PRV" to reflect its status as a previous copy. Since it's not a backup copy, it's not renamed to "<file>.BAK". The new file will then be renamed to the specified name.