CompuView VEDIT CP/M Customizable Full Screen Editor

VEDIT

A Visual Editor

User's Manual

Written By

Theodore Green

CompuView Products, Inc.
618 Louise
Ann Arbor, Michigan 48103

DISCLAIMER

CompuView Products, Inc. and the author make no claims or warranties with respect to the contents or accuracy of this publication, or the product it describes, including any warranties of fitness or merchantability for a particular purpose. Any stated or expressed warranties are in lieu of all obligations or liability for any damages, whether special, indirect or consequential, arising out of or in connection with the use of this publication or the product it describes. Furthermore, the right is reserved to make any changes to this publication without obligation to notify any person of such changes.

Table of Contents

Introduction to VEDIT

VEDIT is an editor designed to take full advantage of a CRT display to make editing of a file as fast and easy as possible. The main feature of VEDIT is its visual mode editing which continuously displays a region of the user's file on the screen and allows any changes made to the screen display to become the changes in the file. The screen display is changed by moving the displayed cursor to any place in the file and making necessary changes by typing in new text or hitting a function key. These changes are immediately reflected on the screen and become the changes to the file. The visual mode allows blocks of text to be moved or copied within the file, and can perform automatic indenting for structured programming languages.

The visual mode can also perform the common word processing operations of wrapping words at the end of lines and reformatting paragraphs between right and left margins. It is very easy to send any portion of the text to the line printer. There are also special facilities which simplify program development editing.

VEDIT also provides a very flexible and powerful command mode for performing search and substitute operations and repetitive editing operations using several types of macros. Commands are provided for moving and rearranging blocks of text, and for the extensive file handling, which includes the ability to insert a specified line range of another file at the edit position.

The sophisticated disk buffering in VEDIT is designed to automatically perform the read/write operations necessary for editing files larger than can fit in the main memory at one time. This applies mostly to the visual mode and allows the editing in visual mode to be done with little concern over the size of the file being edited. The user can also recover from common disk write errors, such as running out of disk space, by deleting files or inserting another disk.

Since so many hardware configurations, different keyboards, editing applications and personal preferences exist in the world, VEDIT is supplied with a customization program in order to let users create versions of VEDIT which are best suitable to their hardware, keyboard, applications and preferences.

Getting Started

This manual is organized into five main sections. The first section describes some basic editing concepts and then introduces the main features of VEDIT and the modes of operation. The second section is a tutorial on the use of VEDIT, including how to invoke and exit it, and perform the most common editing operations. It also covers some of the file handling, including splitting and merging files and what to do if you accidentally run out of disk space. The tutorial section is task oriented. Given an editing operation you wish to perform, this section describes which function keys or commands to use to perform the operation. The third section describes the visual mode in detail, while the fourth section is devoted to a detailed description of the command mode. The last section contains appendices of the customization process, a reference guide of the commands and a description of the error messages.

Before you can begin using the editor, you will have to go through the customization process described in Appendix A. While many parameters can be customized, the menu driven operation allows you to limit your attention to a subset of these parameters. In fact, if you are using a CRT terminal and are content to initially use the default keyboard layout (see enclosed sheet), you only need to select your CRT terminal in order to create a ready to use version of VEDIT. Since the customization process does not destroy or alter the "prototype" editor files on disk, but rather creates a new file with your customized editor in it, you may go through the process as often as you like. As you gain experience with VEDIT you will probably perform the customization several times until you get everything just right. You may also create several versions of VEDIT, although that might confuse you more than help.

The new user of VEDIT is best off to at least skim the first section in order to get an overview of the capabilities of VEDIT. Trying out the editor while reading the tutorial section is the best way to gain a working familiarity with most features. The visual (full screen) mode is easy enough to use that it can be learned by experimenting with the various function keys, as long as no important files are accidentally altered.

Once you have had some practice with the visual mode of VEDIT, you will then want to try out the command mode. The command mode is definitely not as easy to use as the visual mode and more references to this manual will be necessary. However, most basic editing can be done entirely in the visual mode, and the command mode can be learned gradually as the need arises. Also, the tutorial introduces the most used commands of this mode.

While you will typically spend 99% of your time in the visual mode and only 1% in the command mode, this manual deals extensively with the command mode. This is appropriate, because the visual mode is exceptionally easy to learn and use. A little experimentation is the best teacher. The command mode, because of its powerful capabilities, is more complex, and more difficult to learn. This manual, therefore, describes this mode in detail with many examples.

The most complex aspect of the command mode are the "macros" which can perform repetitive editing operations, can be saved and loaded from disk and can invoke other macros. One type of useful macro is the auto-startup file which be used to setup programmable function keys on a CRT terminal.

Section I - Overall Description

Introduction

VEDIT is a full screen, or "visual" editor which currently runs under the CP/M, CP/M-86 and MSDOS operating systems and their derivatives, including MP/M, MP/M-86, CDOS and CROMIX. It allows any text file to be created or edited in a visual manner on systems with most types of CRT displays. It has two operating modes: visual mode and command mode. The typical user will spend 99% of the time in the visual mode, the primary editing mode. Here, the screen continuously displays the region of the file being edited, a status line and cursor. Changes are made by first moving the cursor to the text to be changed. You can then overtype, insert any amount of new text and use function keys to perform all changes, which are immediately shown on the screen and become the changes to the file. The word processing functions of word wrap and reformatting of paragraphs are provided. Ten text registers (scratchpad buffers) allow sections of text to be copied and moved for extensive "cut and paste" type operations. Any portion of the text may be sent to the line printer.

The command mode allows the execution of line and character oriented editor commands, such as for searching, altering and displaying lines. It has added capabilities for dealing with the text registers, which may be saved and loaded directly from disk files. Command mode also allows for explicit Disk Read and Write commands, and for new Input or Output files to be opened and closed. A real time and effort saver is the ability to insert a specified line range of another file at any place in the text being edited. The repetitive execution of single commands or sets of commands called Iteration Macros is provided. The text registers can also hold commands which are executed as macros. Three commands are used for changing the various switches, parameters and tab positions which VEDIT uses in both command and visual modes. One command puts the editor into visual mode. Finally one command must be given to exit VEDIT, saving the edited file on disk.

Basic Editing Concepts

The purpose of editing is to create or change a file on disk so that it may be saved for future use and processed by another program, such as a word processing program (text formatter), a compiler, or simply be printed out. When the file is first created, the initial text of the file is entered with the editor, corrections are made, and then saved on disk. When a file is to be changed or "edited", the existing copy of the file is read from the disk into the computer's "main memory", the changes are made by the user with the use of the editor, and an entirely new copy of the file is saved on disk.

Each file on disk has a name, and when a file is created with the editor, the user assigns the file its name. It is helpful to choose names which mean something and are easy to remember. The name LETTER1 is thus better than JV%8-G5F. The CP/M operating system has file names which consist of two parts, the "filename" and the "filetype" or "extension". A "." separates the two parts and the filename may be up to 8 characters long and the extension up to 3 characters long. When a file is to be edited, its name must be specified in order for it to be read from the disk. The new copy of the file may be written to disk with a new name or with the same name as before. The normal way of invoking and exiting VEDIT will cause it to automatically write it with its original name. One question in this case is, "what happens to the original copy of the file?" VEDIT leaves the original copy on disk too, but since you cannot have two files on disk with the same name, the name of the original file is changed to have an extension of ".BAK". This is referred to as the "backup" of the file. Any previous backup of the file on the disk will be deleted by this process.

When a file is read from disk, its contents are stored in the "main memory" of the computer. The portion of the main memory used for saving the file is referred to as the "text buffer". All changes made to the file are made in the main memory or text buffer. When the changes are complete, the file is saved again on disk. This process of reading a file from disk (or creating a new file), making changes to the file and saving it on disk, is referred to as an "edit session". Therefore, two files are being processed while editing. The file being read is called the "input" file and the file being written is called the "output" file. Specifying to the editor which file is to be used for input or output is referred to as "opening" the file. The way VEDIT is normally invoked, i.e. "VEDIT FILE.TXT", the named file is opened for input, and another file is opened for output which will have the same name as the original input file when the edit session is over. At that time the original input file will still exist, but will have been renamed to a backup file, i.e. "FILE.BAK".

In some cases the file to be edited is larger than the maximum size of the text buffer and only a portion of it can be in the text buffer at once and edited. This situation is handled by first reading in the first portion of the file, making the edit changes to it, writing part of the text buffer out to disk to make space in the main memory, reading in more of the file being edited, and so on. (There are a lot more details involved in this process.) In order to edit a portion of the file which has already gone through the text buffer and been written on disk, a new edit session has to be started. VEDIT, especially in visual mode, has the capability to perform this read/write process automatically. When the user reaches the end of the text buffer in visual mode, the beginning of the text buffer is written out to disk (to the output file) and more of the file being edited (the input file) is read or "appended" to the end of the text buffer. This process, when done automatically, is referred to as "auto-buffering". Another automatic process done in both visual and command mode is called "auto-read" which consists of reading the input file until it is all read in, or until the main memory space is almost full.

Visual Mode

In visual mode, the screen continuously displays the current contents of the file in the region you are editing and a cursor. The bottom line of the screen is used for status information and is normally filled with the "-" character. The changes made to the screen display by typing in new text or using control functions become the changes to the file. The characters typed while in visual mode fall into two categories: Displayable characters and Control characters. The displayable characters are displayed on the screen at the cursor position and cause the cursor to move to the right. The user customized keyboard layout determines which control function each control character or escape sequence performs. The control functions fall into two subcategories - cursor movement and visual functions. The cursor movement operations cause no change to the file, but rather move the cursor forward and backward by a character, a word, a line, a paragraph or a screen at a time. Additional cursor movements allow movement to the next tab position and the beginning or end of the text buffer. The cursor can only point to characters in the file, it never points to "space", i.e. a position on a screen line past the end of the text line.

A useful feature is the ability to move or copy a section of text to any other position in the file. This is done by first copying, moving or appending the text to one of the ten text registers (scratch pad buffers), and then inserting the text register at any place or places in the file. (It may also be inserted in another file). Any portion of the text can easily be printed on the line printer and special printer control characters can be imbedded in the text. VEDIT can optionally display the cursor's line number and column position on the status line. Up to ten positions in the text may be marked, so that the cursor can be directly moved to any of these ten positions.

For word processing uses, the visual mode can perform word wrap and reformatting of paragraphs between adjustable right and left margins. When word wrap in on, VEDIT will move an entire word which didn't fit within the right margin to a new line, while you are typing. Reformatting a paragraph also wraps words, but operates on an existing paragraph and the currently set left and right margins. You can therefore decide later to change the margins and re-fit the paragraphs within the new margins.

VEDIT has several unique built in aids for program development. One is automatic indentation for use with structured languages such as Pascal, PL/I and C. When "Indenting" is set, the editor will automatically insert tabs and spaces following each [Carriage Return] to the current indent position. The indent position can be moved right and left by an adjustable indent increment. Many assembly language programmers prefer their program code to be in upper case letters with comments in upper and lower case. VEDIT can accept all lower case keyboard input and automatically convert the labels,

opcodes and operands to upper case while leaving the comments in lower case. It does this by searching on the line being entered or edited for a special character such as ";". To the left of the ";" lower case letters are converted, to the right of the ";" they are not converted. This is referred to as "Conditional Lower to Upper Case Conversion".

The visual mode can handle text lines which are up to 260 characters (256 plus CR LF and two spare) long. Text lines longer than a screen line (usually about 80 columns) are handled by displaying them on multiple screen lines and indicating in the first reserved column those screen lines that are "continuation lines". This indication is usually in the form of the "-" character which can be displayed in reverse video. These continuation lines are created as necessary while you type.

Command Mode

In command mode, the user enters command lines which consist of single commands, strings of commands or iteration macros. Each command line, whether it consists of one command or multiple commands is ended with an <ESC> <ESC>; there is no RETURN.

Each command consists of a single letter or two letters if the first letter is "E" or "R" (Extended and Register commands). Some commands may be preceded by a number to signify that the command is to be repeated, or "iterated". If no number is given, a "1" is used as the default. Wherever a number is allowed, you can also use the "#" character to represent the maximum positive number 32767. Multiple commands may be typed one after another on a command line. They are always executed left to right. Their effect is the same as if each command had been typed on its own command line.

A group of commands, called an iteration macro, may also be executed multiple times as a group by enclosing the group within "[" and "]", and prefixing the "[" with the iteration number for the entire group. (Note: The characters for enclosing iteration macros are printed as "[" and "]" in this manual. Some users may be more familiar with angle brackets and can choose either set during customization.) The effect is to execute the first command of the group through the last command of the group and then start over again with the first command. The group is executed the number of times specified by the iteration macro. The number "#" is useful in iteration macros to signify "forever" or "all". For example, the command "4T" prints out four lines. The command "5[4T]" prints out the same four lines five times for a total of 20 printed lines. The "[" and "]" may also occur within each other ("be nested") for more complicated macro commands. For example, the command "3[5[4T]4L]" would print out the same four lines five times, then move to the next four lines and print them out five times and last, move to the next four lines and print them out five times. The leftmost "3" determines that everything inside the outside "[" and "]" will be executed three times. This may seem a little complicated at first, but it becomes useful with practice.

Many of the commands make a change to the text buffer at the position determined by the "edit pointer". The edit pointer is very much like the cursor in visual mode, it is just not as readily seen. Commands exist to move the edit pointer a character at a time, a line at a time or to the beginning or the end of the text buffer. The number of lines or characters the edit pointer moves is determined by the iteration number for the command. Negative iteration numbers mean backward movement, towards the beginning of the text buffer. One command prints a given number of lines before or after the edit pointer to display the contents of the file and "show" the user where the edit pointer is.

The commands which alter the text all operate from the position of the edit pointer. One deletes characters, one deletes lines, one inserts new text and another searches for a string of characters and changes them to another. Other commands only perform searching without alteration. Two commands are used to manipulate the text register, with one making a copy of the specified lines and the other then inserting this copy at the edit pointer. Another two commands are used to change the switch settings and tab positions. The last two groups of commands deal with the reading and writing of files and with the opening and closing of input and output files.

The commands fall into nine overlapping categories:

Edit pointer movement	-	B, L, C, Z
Display text	-	T
Print text	-	EO
Alter text	-	D, I, K, S, EI
Search	-	F, N, S
Text Register	-	G, M, P, RD, RL, RS, RT, RU
Disk Buffering	-	A, N, W, EA, EX, EQ
File Handling	-	EB, EC, ED, EF, EG, ER, EW
Switch and Tab Set	-	EP, ES, ET

Additionally the "V" command enters the visual mode, and the "U" command prints three memory usage numbers.

The Text Registers

The ten text registers have two primary purposes. One is to hold sections of text which are to be moved or copied to other positions in the file currently being edited. The second is to hold strings of commands which may be executed in command mode as macros. In either case, the registers are holding text; only the manner in which the text is used is different.

Generally, the text registers are all empty when VEDIT is first invoked. The registers are loaded by copying, moving or appending a portion of the main text to the register. This may be done in either command or visual mode, although it is usually easier in visual mode. Alternately, a register may be loaded directly from a disk file. A register may be typed to the console in order to view its contents, and it may be saved on a disk file. The contents of a register may then be inserted at the cursor position in visual mode or at the edit pointer in command mode. The text registers are not changed by any disk read/write operations. They can therefore be used to extract sections of text from one file and insert them anywhere in another file. Inserting a text register does not destroy or change the register. It may therefore be inserted repeatedly at different locations in the file.

When holding regular text, the registers act as scratch pad buffers in that they hold a temporary copy of text which is independent of the main text buffer. This is for the purpose of copying or moving sections or "blocks" of text from one area of the file to another, commonly referred to as "cut and paste" operations. Three operations are possible. One is to simply copy a section of the main text buffer to the register. The second is to move a section of text to the register, in which the section of text is also deleted from the main text buffer. For both the move and copy operations, the section of text can optionally be appended to any text which already exists in the register. Third, the register contents can be inserted anywhere within the main text buffer.

Placing command strings into the registers and executing these commands as macros is a very powerful facility, although it requires some practice to learn. It is a useful manner of saving long command strings which must be executed repeatedly during an edit session. If they are to be reused in the days ahead, they can even be saved on disk. Very sophisticated editing operations are also made possible. For example, say that you have a manuscript on disk as 20 different files and you find that you have consistently mispelled 40 words. This could be a very time consuming editing operation, but it can be greatly simplified with two command macros. One macro will contain the global search and replace for each of the 40 words. The second macro will contain the commands to edit each of the 20 files, and for each file execute the search/replace macro. Once the two macros are created, you execute the second macro and can take a coffee break while the 800 (20 times 40) operations are automatically performed. (ALWAYS make a backup copy of the files before performing complex macros. It is very easy for a small syntax error, or a software or hardware failure to destroy the files being automatically edited).

Auto-Startup

VEDIT will automatically execute a startup file on disk as a command macro. This can be used to setup various VEDIT parameters and to program the function keys on a CRT terminal. When invoked, VEDIT will attempt to read the file "VEDIT.INI" into text register 0, and then execute this register. No error is given if this file is not found. It is strictly optional.

The file VEDIT.INI may contain EP, ES and ET commands to setup the various parameters, switches and tab positions. It may also contain an EB command which allows a particular file to be edited without it being specified when VEDIT is invoked. This may be handy if the same file is edited many times. The startup file may also contain the commands to load other text register with text or commands from other disk files.

Some CRT terminals have programmable function keys which are setup by sending certain (usually obscure) character strings to the terminal. The VEDIT startup file can perform this too. This is best done by loading the character strings into a second text register, typing out the register, and finally emptying the register. The CRT version of VEDIT comes with several example disk files for doing this.

Auto Read / Write and Auto-Buffering

Auto Read/Write refers to any disk file reading or writing which is done by VEDIT without the user having given the "A" or "W" commands in command mode. (See also "Basic Editing Concepts" above). The simplest auto read/write involves reading the input file into the text buffer when the editor is invoked in the normal way, and writing the output file when the editor is exited. More sophisticated auto read/write called "Auto-Buffering" can take place, especially in visual mode. Auto-buffering refers to the read/write operations which VEDIT performs, especially in visual mode, when the user has reached the end of the text buffer and not all of the input file has been read yet. It is only performed in command mode for the "N" command, since it would otherwise interfere with special editing applications. If the text buffer fills up in visual mode while the user is typing in more text, VEDIT will also try to write out 1K byte sections from the beginning of the text buffer to the output file. This is referred to as "Auto-Write". For more details see Appendix A, "Memory Parameters ...".

Disk Write Error Recovery

Since most CP/M systems run with floppy disks which have limited storage capacity, the typical user will occasionally run into a disk write error. This is caused by either running out of disk space, leading to the error message "NO DISK SPACE", or running out of directory space, leading to the error message "NO DIR SPACE". Fortunately, VEDIT allows the user to recover from these errors using one of two recovery procedures. One is to delete files from the disk using the "ED" command until enough space exists to write the rest of file out. The second is to use the "EC" command to allow removing the full disk and inserting another disk on which to complete the write operation. This, however, results in the output file being split into two files on two disks. The two parts may then be merged back into one file with either VEDIT or PIP.

It is best to avoid disk write errors in VEDIT by making sure that enough disk space exists before editing a file. You can use the CP/M STAT command for this purpose.

Which Mode to Use for What

The visual mode is designed to satisfy the majority of all editing needs. The bulk of editing consists of inserting new text, correcting typos, and making revisions, which includes moving blocks of text around. These are all readily handled in visual mode and are best done in that mode. There is probably a three to one time savings in inserting new text and correcting the typos in visual mode over command mode. There is probably a ten to one time savings in making the revisions in visual mode, compared to command mode, even assuming you are very practiced with the commands!

Command mode is most useful in searching for text in the file, performing repetitive edit changes using macros and for extensive file handling. Searching is used to directly access a particular word or string in the file. Command mode is also used to change the various VEDIT switches, parameters and tab positions. Text register operations such as loading and saving them on disk can only be performed in the command mode. The edit pointer in command mode and the cursor in visual mode both serve a similar purpose. When entering visual mode, the cursor takes on the position in the text buffer of the edit pointer in command mode. When exiting visual mode to command mode, the edit pointer takes on the last position of the cursor.

Searching is often used in conjunction with the visual mode command in iteration macros for finding all occurrences of a string in the file and then editing that region of the file in visual mode. The examples in the tutorial section and the command mode section should be followed.

Command mode is also used when the edit session involves more than just making changes to a single file. The file handling commands allow several files to be merged into one file or a file to be split into several smaller ones. Combined with the text register commands in either visual or command mode, portions of one file can be found and copied into the middle of another file. Other possibilities exist and some examples are given in the "Detailed Command Description" of this manual.

Word Processing with VEDIT

VEDIT can be used for two types of word processing. One is stand alone word processing in which the text is composed entirely with VEDIT and then printed out with either VEDIT or a simple print program. In this case the text will have to be formatted with VEDIT exactly the way it is to be printed out. This would include any page headings, page splits, centering of lines, and other details which VEDIT does not perform automatically. VEDIT can, however, format paragraphs between left and right margins. Therefore, if a paragraph is currently ragged, with very different line lengths, VEDIT can format the paragraph between any left and right margins. If after formatting, you decide, for example, that you now want the paragraph indented on both sides, VEDIT can also do this for you automatically.

The second type of word processing uses a "Text Output Processor" which takes care of page headings, centering of lines, justification and many other details. In this case VEDIT is used to create a file which contains short command lines to the text output processor, which does the final printing. VEDIT's facility for word wrap still makes it easier to enter the text, but the formatting of paragraphs becomes more cosmetic since the text output processor generally formats its own paragraphs. Most text output processors take commands which begin with a period "." in the first column of a line. When formatting paragraphs, VEDIT recognizes such commands lines and leaves them alone. All files created by VEDIT are standard text files and are therefore compatible with most text output processors.

Text formatters come with a wide range of capabilities and prices. Many cost under $100, but can still handle almost all word processing functions. Fancier ones cost around $200 and can handle mailing lists, create tables of contents, perform proportional spacing and automatic footnoting. Combining VEDIT with one of the better text output processors will give you more word processing capabilities than found on most of the stand alone word processors. For short documents such as letters, memos and short reports, a stand alone word processor is probably easier to use than the combination of VEDIT and a text output processor. However, as text documents become longer, most stand alone word processors become increasingly inefficient, because they can only edit a very small portion of the text at one time. Some even have an upper limit to the size of text they can handle. With longer documents, such as manuscripts, using VEDIT with a text output processor will get the job done faster and with less effort.

Section 2 - Visual and Command Modes Task Tutorial

This section is a tutorial on the basic editing capabilities of VEDIT. It is task oriented and gives the commands necessary to perform simple editing operations such as inserting text, and more complex tasks such as moving text and concatenating files. As a "Hands-On" tutorial, it is meant to be followed while actually running VEDIT. Later, as a reference, it should serve to explain how to combine commands to perform a desired task.

Not every possibly conceivable text editing situation or sequence of commands is included here. However, we have tried to include a comprehensive list of editing tasks --- some elementary, others with many steps. Tasks are presented so that you should rarely have to look forward in this section to learn something necessary for the completion of the current task. For example, moving the cursor is the first task discussed; it is used in almost every following task.

The labeled boxes in this section represent visual control functions, such as "CURSOR UP" and "INDENT". The actual keys you type to perform the functions are chosen in the VEDIT customization procedure. If you are using one of our example keyboard layouts, refer to the layout sheet. Most layouts use both control characters and escape sequences. Control characters such as <CTRL-Q> are typed by holding down the CONTROL key while typing the "Q". Escape sequences such as ESC-R are typed by first pressing the "ESC" key and then the "R".

The "ESC" key is also used in all command mode commands (two are needed at the end of each command). It is represented in all examples in this manual as an "$", which is also what VEDIT displays on the screen when an ESC is typed.

Invoking VEDIT

To use VEDIT it has to be invoked from CP/M with the proper command. The next page describes all the ways of invoking VEDIT, but the most common is just to type "VEDIT" followed by the name of the file to be edited or created. For example: (The "B>" is the prompt given by CP/M)

```
B>VEDIT LETTER.TXT
```

VEDIT will then read in the file "LETTER.TXT", or if you are creating the file, briefly display the message "NEW FILE". It will then normally go into the "Visual Mode" which displays the beginning of the file on the screen. The bottom line will contain the "Status Line" which consists mostly of dashes "-", and optionally the line and column numbers. Also visible will be the "Cursor" which indicates at what position on the screen you are editing. It will initially be in the upper left hand corner. At this point you are ready to begin editing.

For the purposes of this tutorial it will be best if you begin by editing a file which already exists, instead of creating a new one. If you don't have any such files available, you can copy one of the files with a filename extension of ".DOC" from your VEDIT distribution disk to your work disk. Don't be concerned about making accidental changes to the file, because you can easily quit the edit session in such a way that no files are actually changed. (We assume, of course, that you have made a copy of your distribution disk.)

INVOKING VEDIT

VEDIT FILENAME.EXT	You will land in Visual Mode (status line will appear at the top or bottom of screen) OR Command Mode ("*" prompt), depending on the parameter set by command ES. See "Command Mode Detailed Command Description".
VEDIT	Begin in Command Mode. Choose a file to edit with an EBfilename$$ or perform any other Command Mode command.
VEDIT INFILE.EXT OUTFILE.EXT	"INFILE.EXT" will be read in and not altered, while "OUTFILE.EXT" will be created. If "OUTFILE.EXT" already exists, it will be renamed to "OUTFILE.BAK". This form is equivalent to invoking VEDIT without any filenames (second form) and then issuing the command: ERinfile.ext$EWoutfile.ext$$. Use this form if the edited file is more than half a disk long. In this case, INFILE.EXT is the file to be edited and OUTFILE.EXT is specified to be on another disk drive with a nearly blank disk.

Keyboard Characters

When in the visual mode, you edit the file by typing two basic types of characters - displayable characters and control codes. The displayable characters are all of the letters, numbers and other normal characters on your keyboard. Go ahead and try typing a few words in right now. Notice that as each character is typed, it appears at the cursor position and the cursor then moves to the right. If there already were characters on the line, you have just overwritten them. You will soon see that it is just as easy to insert characters without overwriting. The control codes are used to perform the various editing functions. The keyboard layout that you have customized determines which editing function each control code performs. Control codes can be control characters, such as <CTRL-S>, escape sequences such as ESC-P, or a function key on your keyboard. Function keys generally send a control character or an escape sequence when you type them.

Editing Functions

The editing functions in the visual mode break down into two categories. One type are the "Cursor movement" functions which only move the cursor around on the screen and scroll the screen to display different parts of the file, but do not change the file in any way. Look at the keyboard layout you are using and try typing the control codes for some of the cursor movement functions such as [UP] [DOWN] [RIGHT] and [LEFT]. The following pages describe all of the cursor movements, and you are advised to briefly try them all out. Don't be concerned about remembering them all now. Some are more important than others, and you will get along quite well knowing only [UP], [DOWN], [RIGHT], [LEFT], [ZIP], [PAGE UP] and [PAGE DOWN].

CURSOR MOVEMENT:

Operation	Command Sequence
Move cursor right	CURSOR RIGHT
Move cursor left	CURSOR LEFT
Move cursor up	CURSOR UP
Move cursor down	CURSOR DOWN
Last character of current line	ZIP
Beginning of current paragraph	PREV PARA
Beginning of next paragraph	NEXT PARA
First character of the previous word	PREV WORD
First character of the next word	NEXT WORD

First character of next line	NEXT LINE
First character of current line	BACK TAB

PAGE MOVEMENT:

Purpose: The move to other regions of the file not currently displayed on the screen.

Operation	Command Sequence
Previous Page of text	PAGE UP
Next Page of text	PAGE DOWN
First Page of text (First character)	HOME
Last Page of text (Last character)	ZEND

Adding New Text

The three functions relating to the "Insert Mode" give you two choices for switching between the "Insert" and "Normal" modes. You started in Normal mode, and the displayable characters you typed overwrote any existing characters. When you switch to Insert mode you will see the word "INSERT" on the status line and any character at the cursor position will be squeezed to the right when you type in new characters. Try it to see the difference between the two modes.

You may be wondering about how to insert entire lines into the text. To start a new line you simply type the RETURN key. If the cursor is at the end of a line, this opens up a blank line on the screen on which you can enter text. If you enter a lot of new lines, one after another, the screen will automatically scroll to keep up with you. If the cursor is in the middle of a line when you type RETURN, the line is split into two lines, with the character at the cursor position and all following characters moving to the new line. With the [DELETE] function, explained next, you can also concatenate lines together.

ADDING NEW TEXT:

Operation	Command Sequence
Entering text into the text buffer --- beginning an empty file or continuing at the end of a file.	NONE - Move cursor wherever you like and begin typing. What you see is what you get.
Overtyping (typing over existing text)	1.) Position cursor over first character to be overtyped. 2.) Retype.
Inserting new characters in between existing characters	1.) INSERT — Watch for "Insert" prompt on status line 2.) Type new text 3.) INSERT — "Insert" prompt disappears (or leave INSERT on.)

Visual Functions

The second category of editing functions are called the "Visual" functions which perform such operations as deleting characters or lines, indenting on the left side and moving sections of text to other parts of the file. The following pages describe each of these functions.

Deleting Text

There are functions to delete the previous and the next character and word. Two functions will delete partial or entire lines. These are described on the next page. Go ahead and try out the [DELETE], [BACKSPACE], [EREOL], and [ERLINE] functions. Notice that the [UNDO] function will bring back the original text on the line unless you erased the entire line with the [ERLINE].

To can delete a line with the [ERLINE] function, or if the line is blank, with the [DELETE] function. You can also concatenate two lines by moving the cursor to the end of the first line and typing [DELETE]. Go ahead and try all of this, especially splitting lines with a RETURN and concatenating lines with a [DELETE].

DELETING TEXT:

Operation	Command Sequence
Deleting characters backwards	BACK SPACE
Deleting characters forwards	DELETE
Erase from cursor to end of line	EREOL
Erase entire line cursor is on	ERLINE
Delete word to left of cursor	DEL PREVIOUS WORD
Delete word to right of cursor	DEL NEXT WORD
Delete paragraphs and blocks of text	1.) Position cursor over first character in the paragraph to be deleted. 2.) MOVE TO TEXT REGISTER 3.) Position cursor past last character in paragraph to be deleted.

3.) Position cursor past last character in paragraph to be deleted.

4.) [MOVE TO TEXT REGISTER]

5.) Type digit "0 - 9" to specify which text register to use.

6.) [MOVE TO TEXT REGISTER]

7.) [MOVE TO TEXT REGISTER]

8.) Type same digit "0 - 9" to empty the text register.

CORRECTING MISTAKES MADE TO A LINE:

[UNDO]

This command returns the line the cursor is on to its appearance before the cursor was most recently put on that line.

This does not mean that, by putting the cursor on a previous line you changed, undo will give you the original line.

Indenting Text

If you don't want your text to begin in the first column, you can let VEDIT automatically indent your text with the [INDENT] and [UNDENT] functions. The section "Visual Mode - Indent and Undent" explains these functions, but it is easier to understand them through experimentation. Type a RETURN to start a new blank line, then type the control code for [INDENT]. Notice that the cursor has moved right by 4 spaces to column 5 (unless you have changed this parameter). Type and few words and another RETURN. This time the cursor will begin immediately in column 5. You have set the "Indentation Position" to column 5, and it will stay there until you increase it with another [INDENT] or move it back with [UNDENT]. To achieve the indentation, VEDIT inserts the most Tabs and fewest spaces to the indent position. You can confirm this by moving the cursor over these leading Tabs and spaces, and if you like, you can also delete them or insert characters before this "Left margin". VEDIT only creates this indentation when you type RETURN, when "Word Wrap" is being used and when paragraphs are formatted.

INDENTING TEXT BLOCKS:

Operation	Command Sequence
Increase the amount of Indentation. (Move left margin to the right)	INDENT
Decrease the amount of Indentation. (Move left margin to the left)	UNDENT
To change Indent/Undent increment:	
1.) Enter command mode	VISUAL ESCAPE
2.) Issue command, where n= # of columns indented each time. EX: EP 3 4$$ will indent to 5th column, 9th column, etc.	EP 3 n$$
3.) Enter visual mode again	V$$

WORD WRAP AND FORMATTING PARAGRAPHS

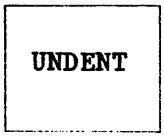
Note: Before the word wrap or the format paragraph function will work, the right margin must be set in command mode (unless it was set during customization). The left margin is set with the INDENT and UNDENT functions.

1.) Set right margin and invoke word wrap at column n. This allows n columns to be used. Word wrap begins at column n + 1.

EP 7 n$$

2.) Move left margin to the right.

3.) Move left margin to the left.



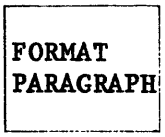
4.) Set addition indent for the begin of a paragraph with spaces.

-- OR --

Use a tab. See ET command to change tab positions.

5.) Type in the paragraph. Words will be wrapped as needed.

6.) Reformat a paragraph to current left and right margins.



Moving and Copying Blocks of Text

A useful facility in VEDIT is the ability to move blocks of text to other regions in the file, to duplicate blocks of text and to delete blocks of text. These are done through the use of the "Text Registers" and the functions [COPY TO TEXT REGISTER], [MOVE TO TEXT REGISTER] and [INSERT TEXT REGISTER]. The text registers are simply regions in memory in which VEDIT can store text which is independent of the text you are editing. A block of text is any amount of text from one character to an entire file. You can copy a block of your text to a text register, in which case your text is unaltered, or you can move a block of your text to a text register, in which case it is deleted from your text. Alternately, these copy and move operations can append the block of text to any existing text in the register. At any time you can insert a text register into your main text, which does not alter the text register.

The following page describes the steps to copy a block of text from one area of the file to another. Note that at step 3.), the cursor must be positioned just AFTER the block of text. If you wish to include the end of a line, this would be the first column of the following line. You could of course position the cursor at the end of the line, but in this case the carriage return which ends the line would not be included in the text move.

If you wanted to move the paragraph, you would use the same procedure, except use the [MOVE TO TEXT REGISTER] function instead of [COPY TO TEXT REGISTER]. In this case the text will also be deleted from your main text and from the screen.

If you type [COPY TO TEXT REGISTER] or [MOVE TO TEXT REGISTER] twice at the same location, the text register will be emptied. If all registers are empty, the "TEXT" message will disappear from the status line. You can therefore delete a block of text by moving it to a text register and then emptying the text register.

MOVING TEXT WITHIN THE FILE:

1.) Position cursor over first character in block to be moved.

2.) | MOVE TO TEXT REGISTER | — Message "1 END" on status line

3.) Position cursor past last character in block to be moved.

4.) | MOVE TO TEXT REGISTER | — Message "REGISTER [+] 0-9" on status line

5.) Type a digit "0 - 9" to specify which register to put text into. — Type optional "+" before digit if new text is to be appended to any existing text in register, instead of overwriting it.

6.) Position cursor at position to insert the text.

7.) | INSERT TEXT REGISTER | — Message "REGISTER [+] 0-9" appears

8.) Type same digit as above (or the digit of another register).

NOTES:

1.) If you get a "FULL" message at step 4, there is insufficient memory for the Text Register to contain the entire text block. Nothing was inserted into the Text Register.

2.) Following the text insert in step 6, the cursor is positioned at either the beginning or end of the inserted test depending upon ES command's switch 4.

3.) In step 3, in order to include the CR-LF of the line, position the cursor at the beginning of the next line.

4.) Alternately you may reverse steps 1) and 3), i.e. either end of the block may be set first.

EMPTYING A TEXT REGISTER

Purpose: It is best to empty a text register when its contents are no longer needed. This frees up more memory space too.

1.) | MOVE TO TEXT REGISTER |

2.) | MOVE TO TEXT REGISTER |

Type command key twice with cursor at same position to empty the register.

3.) Type a digit "0 - 9" to specify which register to empty out.

SENDING TEXT TO THE PRINTER

1.) Be certain you printer is on, and the "on line" or "select" function on the printer is enabled. (See your printer manual).

2.) Position cursor at beginning of text block.

3.) | PRINT TEXT | Will get "1 End" message on status line.

4) Position cursor at end of text block.

5.) | PRINT TEXT | Printer should start now.

6.) <CTRL-C> If you wish to stop the printing.

ENTERING COMMAND MODE

Besides the "Visual Mode" in which all editing is done on the screen at the cursor position, VEDIT has a command mode, where all editing is done by typing in commands which are always ended by typing the <ESC> key twice. The command mode is definitely not as easy to use as the visual mode, but fortunately you don't need to know very much of it in order to use VEDIT very successfully. One thing you do have to know is how to enter command mode in order to end the edit session. This is done by typing the control code for [VISUAL ESCAPE]. Go ahead and try it. The screen will scroll up one line and the command mode prompt "*" will appear below the status line. The command to enter visual mode is "V" and since all commands end in <ESC><ESC>, you should type the "V" and the <ESC> or Escape key twice to get back into visual mode. Notice that the cursor is at the same position in the text (but not necessarily on the screen) as it was when you exited visual mode.

SWITCHING FROM VISUAL MODE TO COMMAND MODE

Function to exit visual mode into command mode. "Edit Pointer" takes on last position of cursor.	VISUAL ESCAPE
Same as above, but does not abort any command, such as a global search.	VISUAL EXIT

SWITCHING FROM COMMAND MODE TO VISUAL MODE

Command to enter visual mode. Text registers are preserved. Cursor takes on last position of command mode "Edit pointer".	V$$

SEARCHING AND SUBSTITUTING

Enter the following commands while in Command Mode. Since <ESC> is echoed with a "$", type <ESC> in the command sequence wherever "$" appears.

Take cursor to beginning of Text Buffer (not necessarily the beginning of the file) and find 1st occurrence of "word".	BFword$$
Do same as BFword$$ except enter Visual Mode after word is found.	BFword$V$$
Find next occurance of "word" and enter Visual Mode. Make changes in Visual Mode and return to Command Mode using [VISUAL ESCAPE]. The command will repeat until the end of text buffer.	#[Fword$V]$$
Search through the entire text for 1st occurance of "word" and substitute "newword".	Sword$newword$$
Replace "word" with "newword" throughout file.	#Sword$newword$$

SAVE ALREADY EDITED TEXT AND CONTINUE

Purpose: You should make it a habit to regularly save your text on disk during a long edit session. This way you will lose less work in case of a power, hardware or software failure, or if someone accidentally turns off the computer. Saving the text every hour and whenever you leave the computer is suggested.

1.) | VISUAL
ESCAPE | — If in visual mode, enter command mode.

2.) EA$$ — Write file to disk; same file will be used to continue edit session.

MAKING MORE MEMORY SPACE

Purpose: When using the text registers extensively, you may run out of memory space for performing the desired operations. This is usually indicated by a *BREAK* in command mode, or a "FULL" in visual mode. You should first try and empty out any text registers which are no longer needed. If this does not give you enough room, you can write out the first part of the text if it is already edited.

1.) Position cursor past end of text which does not need changing (it's been corrected already).

2). | VISUAL
ESCAPE | — Enter command mode.

3). 0W$$ — Write this text out to disk. More room is now available.

INSERT A LINE RANGE OF FILE 1 INTO FILE 2

Note: Both files must be on same disk.

1.) VEDIT file1	Line number appears on status line; note upper and lower range of lines you want copied by positioning cursor on those lines. See ES command if line number does not appear on status line.
2.) EQ$$	Exit VEDIT without writing the file to disk.
3.) VEDIT file2	Edit the file in which you want the lines inserted.
4.) Position cursor where the lines should be inserted.	
5.) VISUAL ESCAPE	Enter Command Mode.
6.) EGfile1[a,n]	Lines a -> n will be copied from file1 to file2 beginning at edit pointer (cursor) position. To copy an entire file, leave off the "[a,n]".

If you get a *BREAK* message there was insufficient memory to insert the entire text, as much as possible was inserted. To make more space for other files, text, etc., try emptying some of the text registers or writing the first part of the text out to disk, as described earlier.

CONCATENATING TWO FILES

Purpose: It is sometimes desirable to append one file to the end of another. This is readily done with VEDIT. In this example the text in file "file2" is appended to the end of the text in "filel" and the combined text is written to the file "file3". The three files can be on different disks.

Note: This assumes that the entire file 'filel' fits into memory.

1.) VEDIT	Invoke VEDIT without a filename. VEDIT will come up in command mode.
2.) ERfilel$0A$$	Setup the first input file for reading, and read it in. This assumes that the entire 'filel' fits into memory.
3.) (optional) EC$$	Only needs to be done if the disk with 'file2' is not in one of the drives. After the EC$$, make sure that the disk with 'file2' and the disk to hold 'file3' are in the drives.
4.) EWfile3$$	Setup the output file which will hold the combined text.
5.) ERfile2$0a$$	Read the second input file. All of it does not need to fit into memory.
6.) EX$$	This writes out the complete file 'file3' and exits VEDIT.

SPLITTING A FILE INTO TWO OR MORE FILES

Purpose: It is sometimes desirable to split a large file into several smaller ones. This example assumes that the splits are simple — the front, middle and end sections of a large file are copied to their own files. More complex splitting can be done with the text registers. In this example 'file1' is split into 'file2', 'file3' and file4'.

1.) VEDIT	Invoke VEDIT without a filename. VEDIT will come up in command mode.
2.) ERfile1$0A$$	Setup the large input file for reading, and read it in. The entire 'file1' need not fit into memory.
3.) EWfile2$$	Setup first output file.
4.) V$$	In visual mode position the cursor at the first character of the second part of the large file. Return to command mode.
5.) 0WEF$$ 0A$$ EWfile3$$	Write the first part of the large file to 'file2' and close it. 0A will read in more of 'file if necessary. Setup the second output file.
6.) V$$	See step 4. Not needed if only splitting into two parts.
7.) (optional) 0WEF$$ 0A$$ EWfile3$$	Not needed if splitting into two parts. Write the 2nd part of large file to 'file3' and close it. 0A will read in more of 'file1' if necessary. Setup the third output file.
8.) EX$$	Write the rest of the large file to the last output file and exit VEDIT.

RECOVERY FROM FULL DISK ERRORS

Purpose: If you attempt to write more text to disk than the disk can hold, you will get a "NO DISK SPACE" error and a return to command mode. Occasionally you may get a "NO DIR SPACE" error which means the disk has insufficient directory space to hold the rest of the file. You can recover from both of these errors by deleting old files on the disk, or by writing the rest of the file to another disk.

1.) EX$$ NO DISK SPACE *BREAK*	You attempt to finish the edit session, but you get the disk full error.
2.) EDoldfile$$	Delete one or more old files on the disk.
3.) EX$$	Continue finishing the edit session.

-- OR --

2.) EF$$	Save whatever fit onto disk. We will call this Part 1.
3.) EC$$	Command to insert another disk (with empty space) into one of the drives. Type RETURN after changing disks.
4.) EWpart2$$	Setup to write the rest of the text to the file 'part2' on the new disk.
5.) EX$$	Write the rest of the text out to 'part2', and exit VEDIT.

You will now have to concatenate the two parts on the two disks back into one file. See "Concatenating Two Files".

Ending the Edit Session

To end the edit session and exit VEDIT, you must be in the command mode and issue one of the commands "EX" or "EQ". There is an world of difference between these two commands. "EX" is the normal command to end an edit session, and the text you were editing will be written out to disk. The "EQ" command on the other hand aborts the edit session and DOES NOT write the text out to disk. You won't use "EQ" very often. Since this is just a practice session with VEDIT, the text you are currently editing is probably all butchered up and you don't want it written out to disk. Therefore the "EQ" command is the appropriate way to exit VEDIT now. Of course, if you would like to preserve your current text, you should exit with the normal "EX" command. If you give the "EQ" command, VEDIT will ask for verification before it actually aborts the edit session.

EXIT VEDIT TO CP/M:

1.) Exit Visual mode to Command mode.

VISUAL
ESCAPE

2.) Exit Command mode to CP/M (MSDOS).

EX$$	File closed and written out to disk.
-- OR --	
EQ$$	Abort - This does not write out file to disk.

Section 3 - Visual Mode

Properties

In visual mode the screen continuously displays the region of the file being edited and a cursor. The left most column does not contain text, but rather is reserved for the line continuation indicator. (The character used for the line continuation indicator can be set by the user during customization. A "-" is the default.) The bottom screen line is used for status information consisting of messages. (Some CRT displays allow the messages to appear in reverse video.) This status line can also optionally indicate what line number in the file and what column the cursor is on. Characters typed while in visual mode take effect immediately when typed. There are two basic kinds of keyboard characters - Displayable characters and Control characters. Displayable characters simply appear on the screen and are either inserted or overtype the existing text. Control characters consist of either ASCII control characters, characters with the high order bit (Bit 8) set, or escape sequences. The customization process determines which control function the control characters perform. Unused control characters are ignored in visual mode. It is possible to also insert any control character into the text. The control functions either move the cursor or perform a visual operation.

In visual mode, the disk buffering can perform automatic Read and Write to handle files which are larger than the size of available main memory. Specifically, if the current screen display reaches the end of the text buffer, and the entire input file has not yet been read, the auto-buffering is performed. VEDIT will also begin to write out the text buffer (auto-write) if the memory becomes full while the user is typing in more text. At this point the first 1K text bytes will attempt to be written to the output file. If no output file is open, or if the cursor is within the first 1K of the text buffer, no writing occurs and the "FULL" message appears instead on the status line. Both the auto-buffering and the auto-write may be disabled by the "Auto Buffering in Visual Mode" switch.

The purpose of auto-buffering is to make the size of the file as invisible to the user as possible. It is not always completely invisible, however, since editing the portion of the file which has already passed through the text buffer requires starting a new edit session.

Each text line is assumed to end in a <CR> <LF> pair as is required for other CP/M programs, and the <LF> is the true delimiter of the text lines. Typing the RETURN (or <CR>) key inserts a <CR> <LF> pair at the cursor position. Deleting the end of a line, will delete both the <CR> and the <LF>. While VEDIT, in visual mode, will never create a line ending in just a <CR> or <LF>, such lines are handled in visual mode, although displayed differently. (Such lines can be

created in command mode). If a line ends in only a <LF>, the next line will be displayed with a starting position directly below the end of the previous line. If a line contains a <CR> not followed by a <LF>, the character following the <CR> will be displayed in the reserved column of the same screen line and the rest of the characters will overwrite previous characters. (This is not very eloquent, but is just what most terminals would do). Such lines may be fixed by deleting the offending lone <CR> or <LF> with the [DEL] key and then inserting the <CR> <LF> pair with the RETURN key.

Displayable Characters

When a displayable character is typed, it appears on the screen at the current cursor position and the cursor then moves to its right. VEDIT has two modes for inserting new characters, NORMAL and INSERT mode. When a displayable character is typed in NORMAL mode it appears at the cursor position and any character which was there is simply overwritten. The only exception to this is the <CR> <LF> pair, which is not overwritten, but is squeezed to the right. Also, typing the RETURN does not overwrite any character, but rather moves any character at the cursor position to the next line. In INSERT mode, no character is ever overwritten, but rather is squeezed to the right when a new character is typed at its position. In either mode, a new screen line, called a continuation line, is begun if the the text line becomes longer than the screen line. Visual functions exist to enter Insert Mode, revert to Normal mode, or to switch between the modes. The editor always starts in Normal mode.

The keyboard characters RETURN and TAB are displayable characters, but have special properties. The RETURN (or <CR>) key causes a <CR> and line feed <LF> pair to be inserted into the text and a new line to be begun on the screen. If it is typed while the cursor is pointing within a text line, that line is effectively split into two lines.

The [TAB CHARACTER] key causes insertion of a tab character, or optionally, spaces to the next tab position. The tab character itself is displayed with spaces on the screen to the next tab position, even though the spaces do not exist in the text buffer.

Any control characters, other than <CR>, <LF> and <TAB> which are in the text, are displayed in the common CP/M format by preceding the letter with an "Up Arrow". The control function "[NEXT CHAR LITERAL]" allows any control character except <CTRL-Z> (which is not allowed by CP/M) to be inserted into the text. Alternately, the command mode "EI" command can be used to insert control and special characters which cannot be produced by the keyboard.

Control Functions

The control functions fall into two categories: Cursor Movement and Visual Function. The cursor movement keys only move the cursor to some other position in the text and do not actually change the text. The cursor may be moved forward and backward by a character, a word, a line, a paragraph or a screen at a time. Up to ten positions in the text may be "remembered" with invisible markers which allow the cursor to be directly moved to these positions. These and other movements are individually described later in this section.

Some of the visual functions perform editing functions such as deleting, while others change the Insert mode, change the indentation, manipulate the text registers,and print text. The visual functions for deleting text are [DEL] which deletes a character, [EREOL] for deleting (erasing) all remaining characters on the line beginning at the cursor position, [ERLINE] for deleting the entire text line, and [BACKSPACE] which moves the cursor to the left and deletes the character there. The previous or next words in the text can also be deleted with the [DEL PREVIOUS WORD] and [DEL NEXT WORD] functions.

The visual functions [SET INSERT MODE], [RESET INSERT MODE] and [SWITCH INSERT MODE] are used for switching between NORMAL and INSERT mode. The visual function [RESTART] starts the edit session over, saving the current file on disk, just as the EA command does. These and other visual functions are fully described in this section.

The Tab Character

One displayable character which acts a little different is the [TAB CHARACTER], which is normally assigned to the Tab Key or <CTRL-I>. When the Tab key is typed, it inserts the tab character into the text and this is displayed with spaces to the next tab position. The tab positions are variable, but are normally set to every 8 positions. You can tell the difference between the tab character and spaces by the way the cursor moves over them. The cursor moves over each space individually, but moves over the Tab as a unit, i.e. a single [CURSOR RIGHT] might move you from column 1 to column 9. This reflects the fact that the Tab is a single character and should be treated as such. When the cursor is at the Tab character, it is displayed at the left side of the displayed spaces. If you wish to insert other characters before the Tab and leave the Tab in the file, you must be in the Insert mode. Otherwise the first character you type will overwrite the Tab. The Tab character is commonly used when writing programs and aligning tabular data. Text paragraphs are normally indented using just spaces.

The [TAB CHARACTER] and the [TAB CURSOR] function must not be confused. The latter is strictly a cursor movement function and has nothing to do with Tab characters. It only moves the cursor right to the character at the next tab position. It is very similar to typing [CURSOR RIGHT] repeatedly, except that it does not move the cursor past the end of the line. If you find that you have customized the Tab key to be [TAB CURSOR] you are advised to change the Tab key to be [TAB CHARACTER] as it should be.

Optionally, the [TAB CHARACTER] function can insert spaces to the next tab position. This would be equivalent to you typing in the spaces. While this uses up more disk space and is not recommended for normal applications, it is useful for applications which require an exact layout which is not compatible with the tab positions of other programs. This option may be changed with the "ES" command.

Displaying Line and Column Numbers

If desired, VEDIT can display the line number in the file that the cursor is on and/or the cursor's column position on the status line at the bottom of the screen. The display of these two numbers is controlled by a parameter which is initially set during the customization, but may be changed from command mode with the "EP" command. (The example customization sets the parameter to display both numbers). The cursor's column position is simply the horizontal position on the current line. The line number in the file is a count of the current number of preceding lines in the file, including any which have already been written out to disk. The line number is the same as would be printed by the CP/M PIP program with the "N" switch. The line number for a particular line will therefore decrease if some of the preceding lines are deleted, and will increase if some lines are inserted into the preceding text. These numbers are not updated immediately following every cursor movement, but only after the user pauses typing for about 1/2 of a second.

Setting and Jumping to Text Markers

Up to ten positions within the text can be invisible marked, allowing the cursor to be directly moved to these positions. The positions are marked by typing the [SET TEXT MARKER] key. The status line will then prompt for a digit "0 - 9". Type a digit. To move the cursor to a marked position type the [GOTO TEXT MARKER] key and the appropriate digit following the status line prompt. The cursor will then move to the beginning of the line with the marked position.

The marked positions are relative to the text and not absolute positions. This means that the markers will adjust themselves as text is inserted and deleted. All markers are initially set to the Home position. If text containing a marker is written to disk, that marker will be reset to the Home condition. Unfortunately, performing a RESTART function or an "EA" command will therefore reset all markers. Some forms of the "S" command (search and replace) will allow the markers to drift by a few positions. This is usually not noticable because accessing a marker moves the cursor to the beginning of the line containing the marker.

The Text Registers

The visual functions [COPY TO TEXT REGISTER] and [MOVE TO TEXT REGISTER] are used to copy or move text from the main text buffer to one of the text registers. The function [INSERT TEXT REGISTER] is then used to insert the contents of a text register at the cursor position. These functions are usually used to move or copy text from one area in the file to another. A section of text can also be moved

to a text register, whose contents are then written in command mode to a disk file. Command macros are also created and edited in visual mode. When complete they are moved to a register which is then executed in command mode. The text registers used are the same as used in command mode, thus the text registers may be set in command mode and inserted in visual mode or vice versa.

The text to be copied or moved is specified by first moving the cursor to the beginning of the text and marking it by typing the appropriate function key. The message "1 END" will appear on the status line. The cursor is then moved just past the last character of the text and the function key typed again. The status line will then prompt for a digit "0 - 9" to specify which register. The digit may optionally be preceded by the a "+" to indicate that the text is to be appended to any text which may already be in the register. After typing the digit, the status line message will change to "TEXT". In case of [COPY TO TEXT REGISTER], the main text buffer will be unchanged. However, in case of [MOVE TO TEXT REGISTER] the text will be deleted from the main text buffer. Typing [INSERT TEXT REGISTER] and a digit "0 - 9" will insert the specified register at the cursor position. Depending upon the "Point past register insert" switch (see ES command), the cursor will be positioned either at the beginning or the end of the inserted text.

Whether the beginning or the end of the text is first marked is actually unimportant. It is also immaterial whether you type [COPY TO TEXT REGISTER], [MOVE TO TEXT REGISTER] or even [PRINT TEXT] when you mark the first end of the text. Only when you mark the second end must the correct key be typed. If there is unsufficient memory space for the text register, or to insert the register, the message "FULL" will appear on the status line and the operation will be aborted.

Printing Text

VEDIT can print out any portion of the text which is currently in the main text buffer. This can be done from both the visual and the command modes of the editor. It is easiest to do in visual mode and is similar to the method of moving text to the text register. First the cursor is positioned at one end of the text to be printed and the [PRINT TEXT] function key pressed. (This is ESC - P in the example keyboard layout). Then the cursor is positioned at the other end of the text to be printed and the [PRINT TEXT] pressed again, which causes the text to be printed. To print the entire text move the cursor to the beginning and end of the text with the [HOME] and [ZEND] functions, and type [PRINT TEXT] at each end. The printing can be aborted by typing a CTRL-C. Many printers use control characters or escape sequence to control such things as character size, font style and overstrike. These special control sequences can be imbedded directly into the text.

Inserting Control Characters

VEDIT can insert control characters into the text. These may be special printer control characters, the [ESC] character, or control characters for other purposes. Only CTRL-Z cannot be inserted because it is defined by CP/M to signify the end of the file. The [NEXT CHAR LITERAL] function places the next character typed on the keyboard into the text. In this manner any control character which can be generated from the keyboard can be placed into the text. In case a character must be inserted which cannot be generated from the keyboard, the command mode "EI" command can be used. This command can insert any character with a decimal value between 00 and 255 (except CTRL-Z) into the text.

Indent and Undent Functions

As an aid in word processing and writing programs in structured languages such as Pascal, PL/I and C, the visual mode has the [INDENT] and [UNDENT] functions. These functions allow the editor to automatically pad up to the "Indent position" with tabs and spaces, when a new line is created with the RETURN key. The [INDENT] key moves the Indent position to the right by the "Indent increment", and the [UNDENT] key moves the Indent position back to the left. If the cursor is on a new line, or before any text on the line, when the [INDENT] or [UNDENT] is pressed, the cursor and any following text will also move to the new Indent position.

Normally the "Indent position" is zero and when a RETURN is typed, a <CR> <LF> pair is inserted into the text, and the cursor moves to column 1 of the next line. After the [INDENT] key is pressed once and a RETURN typed, the cursor will be positioned not in column 1, but rather at the first indent position, i.e., column 5 if the "Indent increment" is set to four. Pressing the [INDENT] key again will position the cursor still farther to the right after each RETURN, i.e., to column 9. Each time the the [UNDENT] key is pressed, the indent position moves back toward the left until it is back at zero.

The exact number of tabs and spaces inserted into the text buffer, to pad up to the "Indent position", is related to the currently set tab positions and the "Indent Increment". The padding will consist of the most tabs and fewest spaces in order to save memory and disk space. For example, assume that the "Indent increment" is set to the common value of four (4) and the tab positions at every eight (8). When the "Indent position" is eight, the padding will consist of one tab; when the "Indent position" is twenty, the padding will consist of two tabs and four spaces. On the other hand, if the tab positions were set to every four, only tabs would be used in the padding. Note that if the "Expand Tab with spaces" switch is set, only spaces will be used for padding. This would use up lots memory and disk space.

disk space.

Lower to Upper Case Conversion

Several modes are available for converting between lower and upper case letters as they are being typed on the keyboard. There are three options for converting from lower to upper case:

1.) No conversion is made.
2.) All lower case letters are converted to upper case.
3.) Conditional conversion of lower case to upper case for assembly language programming and other special applications.

The second option is similar to the "Caps Lock" on a keyboard, the 26 lower case letters are converted to upper case. The third option is specifically designed for assembly language programming. In this mode lower case letters are converted to upper case if they occur to the left of a special character, typically ";". To the right of the ";" they are not converted. In this manner an assembly language program can be edited in all lower case letters and VEDIT will automatically convert the labels, opcodes and operands to upper case, while leaving the comment fields alone. This can also be used for Fortran programs and other special applications. These options and the special character are set with the "EP" command.

Upper and lower case letters can also be reversed, i.e., lower case converted to upper case and upper case converted to lower case. This is specifically designed for the Radio Shack TRS-80 Model I, whose keyboard normally produces upper case letters and lower case with the Shift key. This reversal is done immediately when a keyboard character is received and before any resulting lower case letter is converted to upper case as described above. The letters are also reversed for the command mode. This mode may also be handy in the case where most text is to be entered in upper case, but where an occasional lower case character is also needed. This mode is selected with the "ES" command.

Word Processing Oriented Functions

VEDIT has functions for moving the cursor to the beginning of the next word or the preceding word, and functions to delete the next or the previous word. The [NEXT WORD] function moves the cursor to the first letter of the next word. The [PREVIOUS WORD] function moves the cursor to the first letter of the current word, or if already there, to the beginning of the previous word. The [DEL NEXT WORD] function deletes the entire word and any following spaces if the cursor is at the beginning of the word. If the cursor is in the middle of a word, it deletes only that portion of the word at and to the right of the cursor. [DEL PREVIOUS WORD] deletes the previous word and any following spaces if the cursor is at the beginning of a word. If the cursor is in the middle of a word, it deletes only that portion of the word to the left of the cursor. The delete-word functions never delete carriage returns, but rather just moves over them when they are encountered.

Words are allowed to have imbedded periods in them, such as in "i.e.". A comma "," always ends a word, even if the comma is not followed by a space. The special characters ")", "]" and "}" also separate words from each other, as do spaces, tabs and carriage returns. All other characters are allowed in words.

The cursor can also be moved to the beginning of the previous or the next paragraph with the [PREVIOUS PARAGRAPH] and [NEXT PARAGRAPH] functions. VEDIT considers a paragraph to end when an empty line, a blank line, a text output processor command line, or a line which is indented by at least two spaces or a tab is encountered. Text output processor command lines are considered to be any line which begins with a ".", "!" or a "@" in the first column.

Word Wrap and Indentation

To simplify the entering of word processing type text, the WORD WRAP facility may be used. This facility allows the user to specify a right margin beyond which no text should appear. If you attempt to enter a new line beyond this margin, VEDIT will move the word which didn't fit within the margin to the next line, leave the cursor in the same position in the word, and add a carriage return to end the previous line. Word wrap will only occur when the cursor is at the end of a line being entered, or when in INSERT mode. If you do not wish the text to begin in the left most column, you may set a left margin with the [INDENT] and [UNDENT] functions. A left hand margin may be set independent of whether word wrap is enabled. The right hand margin can be greater than the screen line length, in which case VEDIT will display a continuation line before the word wrap takes place. The word wrap facility is enabled by setting the right margin parameter. A value of zero turns word wrap off. This parameter is initially set during the customization and can be changed with the

"EP" command. For example, to set the right margin at column 70, the following command is given in command mode:

EP 7 70[ESC][ESC]

Formatting Paragraphs

In word processing it is frequently desirable to format a paragraph so that all of the text appears between certain left and right margins. The [FORMAT PARAGRAPH] function performs this. The left margin is set by the [INDENT] and [UNDENT] functions, while the right margin is the same as used for word wrap. When a paragraph is formatted or reformatted, any spaces and tabs which make up an indentation on the left side are ignored and effectively deleted before any new indentation is created, but with one exception. The exception is the first line of a paragraph. Any additional indentation that the first line has over any second line in the paragraph is preserved.

To format a paragraph, the cursor may be placed anywhere in the paragraph or in the text output processor command lines which precede the paragraph. The text output processor command lines will not be formatted or changed in any way. After formatting, the cursor will be positioned at the beginning of the next paragraph. A series of paragraphs may therefore be formatted by just repeatedly typing the [FORMAT PARAGRAPH] function. A paragraph will only be reformatted if the right margin is greater than the left indent margin. Setting word wrap off, therefore also disables the formatting function.

[HOME] Move the cursor to the very first character in the text buffer.

[ZEND] Move the cursor to the very last character in the text buffer.

[CURSOR UP] Move the cursor up one line, to the same horizontal position if possible. If the position is beyond the end of the line, move to the end of the line, if the position is in the middle of a tab, move to the end of the tab. If there is no line, it won't move.

[CURSOR DOWN] Move the cursor down one line, to the same horizontal position if possible. The same rules as for [CURSOR UP] apply.

[CURSOR RIGHT] Move the cursor to the next character in the text. If currently at end of line, move to beginning of next line. If there is no line, don't move.

[CURSOR LEFT] Move the cursor to the previous character in the text. If currently at beginning of line, move to end of previous line. If there is no line, don't move.

[BACK TAB] Move the cursor to the first position in the current screen line. If cursor is already at the first position, move to beginning of previous screen line.

[TAB CURSOR] Move the cursor to the character at the next tab position. If cursor is at the end of a line, don't move. Note that this only moves the cursor, use the [TAB] key to insert a Tab character.

[ZIP] Move the cursor to the end of the text line the cursor is on. If it already is at the end of a line, it moves to the end of the next text line.

[NEXTLINE] Move the cursor to the beginning of next text line.

[PREVIOUS WORD] Move the cursor to the first character of the current word, or if already there, to the beginning of the previous word.

[NEXT WORD] Move the cursor to the first character of next word.

[PREVIOUS PARA] Move the cursor to be beginning of the current pragraph, or if already there, to the beginning of the previous paragraph.

[NEXT PARA] Move the cursor to the beginning of next paragraph.

[PAGE UP] This scrolls the screen to give a similar effect to typing [CURSOR UP] for 3/4 screen lines.

[PAGE DOWN] This scrolls the screen to give a similar effect to typing [CURSOR DOWN] for 3/4 screen lines.

[SET TEXT MARKER] Followed by a digit "0 - 9". Sets an invisible text marker which will automatically adjust as text is inserted and deleted.

[GOTO TEXT MARKER] Followed by a digit "0 - 9". Moves the cursor to the beginning of the line containing the specified text marker. If the marker has not been set or has been reset, moves the cursor home.

[SET INSERT MODE] Change the mode to INSERT if not already there.

[RESET INS MODE] Change the mode to NORMAL if not already there.

[SWITCH INS MODE] Switch the mode to the opposite. Note that normally either [SET INS MODE] and [RESET INS MODE] or [SWITCH INS MODE] would be implemented during the VEDIT Customization process.

[DELETE] Delete the character at the cursor position. The cursor doesn't move. A lone <CR> or <LF> will also be deleted, but a <CR> <LF> pair will both be deleted as one.

[BACKSPACE] Move the cursor left and delete the character at that position. Does not delete a <CR> <LF>.

[DEL PREVIOUS WORD] Delete the previous word and any following spaces if the cursor is at the beginning of a word. Otherwise delete only that portion to the left of the cursor.

[DEL NEXT WORD] Delete the entire word and any following spaces if the cursor is at the beginning of a word. Otherwise delete that portion of the word at and to the right of the cursor.

[EREOL] This deletes all characters from the cursor position to the end of the text line but not the final <CR><LF> pair unless the text line only consists of the <CR><LF>, in which case the <CR><LF> is deleted. For example, the following sequence will delete an entire line:

[BACK TAB] [EREOL] [EREOL].

[ERLINE] This deletes the entire text line. Use of [BACK TAB] [EREOL] is actually preferable, since the latter does not close up the screen line and frequently allows the [UNDO] to restore the original line.

[UNDO] This rewrites the screen and ignores the changes made to the text line the cursor is on.

[NEXT CHAR LITERAL] The next character, whether a displayable character, a control character, or a character with its high order bit set, will be placed into the text buffer.

[INDENT] This increases the "Indent Position" by the amount of the "Indent Increment". The editor will then automatically pad with tabs and spaces to the Indent position following each RETURN. The padding will also take place on the current line if the cursor is before any text on the line.

[UNDENT] This decreases the "Indent Position" by the amount of the "Indent Increment", until it is zero. One [UNDENT] therefore effectively cancels one [INDENT].

[COPY TO TEXT REG] The first time this key is hit, the position of the cursor is remembered, and the message "1 END" is displayed on the status line. When the key is hit while the "1 END" is set, the status line prompts for a digit "0 - 9" indicating the text register to be used. The text block between the first cursor position and the current cursor position is then copied to the text register. Optionally the digit may be prefixed with a "+" to indicate that the text is to be appended to any text already in the register. Assuming there is enough memory space for this "copy", the message "TEXT" is then displayed on the status line in place of the "1 END". If insufficient memory space exists, no copy is made, the "1 END" is erased and the "FULL" message appears on the status line. Hitting this key twice at the same cursor position will empty the specified text register. Note that either the beginning or the end of the text block may be set first.

[MOVE TO TEXT REG] This is similar to [COPY TO TEXT REG], except that the text block is deleted from the text buffer after it is moved to the text register.

[INSERT TEXT REG] Followed by a digit "0 - 9" indicating which text register's contents are to be inserted at the current cursor position. The register itself is not changed. If there is insufficient memory space for the entire "copy", nothing is inserted and the "FULL" message will appear on the status line. Moving the cursor to another line will clear the "FULL" message.

[PRINT TEXT] This is activated similar to the [COPY TO TEXT REG], only no digit needs to be typed. The block of text is then printed on the CP/M listing device. A CTRL-C will abort the print out.

[FORMAT PARAGRAPH] This will format the paragraph that the cursor is in so that all of the text appears between a left and right margin. The left margin is the current Indent Position, and the right margin is the current Word Wrap column. Following the format, the cursor will be positioned at the beginning of the next paragraph. Text output processors commands will not be formatted.

[VISUAL EXIT] Visual Mode is exited to Command Mode. The current cursor position in the text buffer will become the command mode edit pointer position. Any text register is preserved. Depending upon the value of the "Clear screen on visual exit" switch, the command prompt will appear either on a clear screen or just below the status line.

[VISUAL ESCAPE] This is identical to the [VISUAL EXIT], except that any current iteration macro is aborted.

[RESTART] The text buffer and any unappended portion of the input file is written to the output file. The output file is closed and then reopened as the Input and Output file. The file is then read into the text buffer again.

Section 4 - Command Mode

Properties

In command mode all character output goes to the current CP/M console output device. The user enters command lines, which consist of single commands, strings of commands or iteration macros. Each command line is ended with an <ESC> <ESC>, at which point the command line is executed. The <ESC> is also used to delimit search strings. In the event that your keyboard does not have an <ESC> key, you may customize the command mode escape character to be any other control character.

Each character typed is echoed by VEDIT and none are processed by CP/M. Thus a <CTRL-C> has a different meaning in VEDIT and does not cause a return to CP/M. The <ESC> is echoed with a "$", which is also used in the examples in this manual to signify the <ESC> key. The <RETURN> or <CR> key is echoed with a <CR> <LF> pair, and the pair is also entered into the command line. Although this causes a new line to be printed, it is still part of the command line and DOES NOT end the command line.

The user is prompted for a new command line by the "*" character. If, while typing, the command line should exhaust the amount of memory space available to it, (the text buffer, text register and command line all share the same memory space) VEDIT will send the "Bell" character to the console and neither accept nor echo any more characters. The user will then have to edit the current command line in order to end it and then rectify the full memory situation. Even when the memory is full, (see "U" command) up to ten characters may be typed on the command line.

Before the command line is ended and begins executing, the line may be edited with most common line editing characters. They are described in detail below under "Line Editing". Once execution begins, it may often be aborted by typing the <CTRL-C> character. This causes a *BREAK* and a new command mode prefix to be printed. VEDIT checks for the [CTRL-C] before any new command is executed, during the execution of the "A", "F", "N", and "T" commands, and in a few other situations.

Command Mode Notation

$ denotes the <ESC> control character. Wherever "$" appears in a command mode example, type the <ESC> key.

<TAB> represents the TAB character while "<CR>" represents RETURN.
<CR> Type the respective keys, not the literal representation.

<ESC> represents the ESC key or alternate command mode escape
<CTRL- > character selected during customization. Other control characters produced by holding the CTRL key and typing a letter are represented by "<CTRL-letter>".

[The bracket characters used for iteration macros are printed
] as "[" and "]" in this manual. Some users may be more familiar with the angle brackets "<" and ">". You can choose which characters to use during the customization process.

Search Options and Special Characters

There are two search options which are useful for some applications, particularly when using text register macros. One allows strings to be delimited without using the <ESC> character. The second allows search error messages to be suppressed. Two special characters have significance in strings being searched. The first is the wildcard character "|", which will match any character in the text being searched. The second is [CTRL-Q], which allows the following control character to appear literally in the string.

The commands "F", "N", "S" and "I" are followed by a text string, which is normally delimited with an <ESC>. An option allows an explicit delimiting character to begin and end the text string. With this option, the character immediately following the "F", "N", "S" or "I" command is the delimiter. Any character can be the delimiter, but "/" is a common choice. Note that the text string itself cannot contain the delimiting character. This option can be invoked by preceding the command with a "@". For examples, the commands on the left side are equivalent to those on the right.

Fspeled$V$	@F/speled/V$$
Sspeled$Spelled$v$$	@S/speled/spelled/V$$
4Fpoint$V$$	4@F:point:$$
Ia new line$$	@I/a new line/$$

This explicit delimiter option can also be made the default by setting it with the "ES" command, or during customization. With the option ON, the "@" character is no longer needed. Although using this option requires more characters to be typed, many users find that it makes the commands more understandable. It also allows the <ESC> character to be searched, which is useful when editing macros. For example, the following command searches for the string "h<ESC> <ESC>":

```
@F/h<ESC><ESC>/V$$
```

Note that the <ESC> <ESC> therefore does not end a command if it appears between explicit delimiters. Since it is easy to forget the second delimiter and type <ESC> <ESC>, the command mode prompt changes from its normal "*" to "-" indicating that the command has not yet ended.

The command "F$$" will always search for the last used string, even if the explicit delimiter was used for the original string or is currently in effect.

```
F$$         Search for last used string.
```

Search error messages can be suppressed by preceding the "F", "N" or "S" command with a ":". Alternately the suppression may be turned ON with the "ES" command or during customization. This is primarily useful with text register macros which contain many "S" commands, and where the macro should not terminate if some of the strings are not found.

A useful feature for some search operations is the special "|" character. Each "|" in the string being searched will match any character in the text. The search string "C|N" will match "CAN", "C1N", "C N" and others. Similarly, "C||E" will match "CONE", "C NE" and others.

The literal character <CTRL-Q> operates similar to the [NEXT CHAR LITERAL] in visual mode, in that the next character is treated literally and not interpreted. This is the only way to search for characters such as <CTRL-R>, <CTRL-U> and <CTRL-H> which are also used for line editing. It is also an alternate way to search for the <ESC> character. For example, the following examples insert text containing a <CTRL-H> and search for the same text:

```
Iword<CTRL-Q><CTRL-H>$$

Fword<CTRL-Q><CTRL-H>$$
```

These two commands both search for the string "h<ESC>":

```
@F/h<ESC>/$$                Fh<CTRL-Q><ESC>$$
```

CP/M and VEDIT both require that lines end in a <CR> <LF> pair. However, when files are transferred from mainframe computers, the lines often end in a <CR> without the <LF>. These lone <CR> must be changed to <CR> <LF> pairs. One cannot simply search for a <CR> by typing the RETURN key because it is expanded into <CR> <LF>, unless the RETURN is preceded with a "<CTRL-Q>. Therefore, the command to change all lone <CR> to <CR> <LF> pairs is:

```
b#S<CTRL-Q><CR>$<CR>$$
```

Iteration Macros

An iteration macro is a group of Command Mode commands which repeats with or without user intervention as many times as desired. They are most useful in searching and substitution tasks (changing all instances of a misspelled word, for example).

An iteration macro's general construction is: a string of commands enclosed with brackets "[" and "]", prefixed by a number which tells VEDIT how many times to iterate, and ended with <ESC> <ESC>. The following example changes the first three occurrences (if found) of " teith" to " teeth".

Example: `3[S teith$ teeth$]$$`

It is very important to observe the placement of any necessary <ESC> to terminate strings and filenames when using iteration macros. For example, "BF word$$" needs no "$" between the "B" command and the "F" command, but in "S name$ smith$V$$", the "$"s are necessary after " name" and after " smith". The following example changes the first occurrence of " teith" to " teeth]", which is not the intention.

Wrong: `3[S teith$ teeth]$$`

If desired, each command may be ended with one <ESC>, in which case you won't have to remember whether the command must be ended in an <ESC> or not.

Iteration Prefixes:

Besides any integer, the iteration macro may be prefixed with a "#". This is used when the iteration is to continue as long as possible. "#" represents the maximum positive number 32767. If no prefix is given, "1" is assumed. The following example changes all occurrences of " teith" to " teeth".

Example: #[S teith$ teeth$]$$

It is normal to get the error message "CANNOT FIND ..." when performing a search or substitute command for all occurrences of a string, because the command is literally searching for 32767 occurrences. However, the error will not occur for the "#S" command.

Using Visual Mode in Iteration Macros:

Search and substitute operations are often used in conjunction with the visual mode in order to edit the region, or to confirm that the substitute was done correctly. For example, the following command will search for all occurrences of the word "temporary" and let those regions of the text be edited in visual mode.

#[Ftemporary$V]$$

The following command could be used in a form letter to change the string "-name-" to the desired name, check that it was done correctly in visual mode, and if necessary make any edit changes.

#[S-name-$Mr. Jones$V]$$

The Visual Mode has two ways of exiting back to Command Mode in order to help in using iteration macros. The [VISUAL EXIT] simply exits and lets any command iteration continue. The second, [VISUAL ESCAPE] exits to Command Mode, but also aborts any iteration macro. The latter is used when the user realizes that the iteration macro is not doing what was intended and does not want the macro to further foul things up. For example, in order to change all occurrences of the word "and" to "or", the following command may have been given:

#[Sandorv]$$

The user might then see in Visual Mode that the word "sand" was changed to "sor", which was not the intention. The [VISUAL ESCAPE] would stop the command and the following correct command could then be given:

```
#[S and$ or$v]$$
```

If it is unnecessary or undesirable to view each substitution in Visual Mode, the previous substitute operation could take the simpler form:

```
#S and$ or$$
```

Note that this is not an iteration macro, but rather just a form of the "S" command. Because it executes much quicker, it is preferable to the equivalent command:

```
Slow:          #[S and$ or$]$$
```

The commands "I" for Insert and "T" for Type are useful in iteration macros. The "T" can be used to type out the lines that are changed in an iteration macro without going into Visual Mode. The "I" command is useful when the same text is to be inserted into the text buffer many times. For example, to begin creating a table of 60 lines, where each line begins with a <TAB> and ".....", the following command could be used before the rest of the table was filled in Visual Mode:

```
60[I<TAB>.....<CR>$]$$
```

The <CR> will be expanded into a <CR> <LF> pair.

Iteration macros only work from the edit pointer position forward, unless a particular command has a negative prefix. Therefore be sure to place the edit pointer at the beginning of the text buffer, file, or other area you're working in so that all occurrences are found.

An iteration will continue until its iteration count is exhausted or until an error occurs. Common errors are unsuccessful search operations. A special situation is encountered when using search commands ("F" and "S") in iteration macros with search error suppression enabled. When a search is unsuccessful, no error is given, but the iteration is stopped, and execution continues with the command following the iteration. This may be an outer level iteration. Recall that the commands "#S" and "#F" are only unsuccessful if no occurrences are found.

Text Registers

Eight commands are available for using the ten text registers in command mode. Lines of text may be copied to a register with the "P" command:

35P5$$	Copy the next 35 lines to register 5.
-6P+4$$	Append previous 6 lines to register 4.
0P2$$	Empty out register 2.

The "G" command inserts the contents of the specified register at the edit pointer:

G2$$	Insert register 2 at edit pointer.

The "RS" command will save the contents of the specified register in a disk file. Various portions of a file or files may therefore be appended to a text register, which is then saved as a new disk file.

RS4b:filesave.reg$$	Save contents of register 4 in "filesave.reg" on drive "B".

Similarly the "RL" command will load a register from a disk file. This can be useful for merging several files together in complex ways.

RL4b:filesave.reg$$	Load register 4 from "filesave.reg" on drive "B".

The contents of a text register can be display on the console with the "RT" command:

RT9$$	Type out contents of register 9.

The "RT" command expands control characters and displays <ESC> with a "$". Since this is not suitable for initializing a terminal (programmable function keys, etc.), the "RD" command is provided, which does not expand control characters:

RD9$$	Dump out contents of register 9.

The "RU" command displays the number of characters contained in each of the text registers. The sum of these ten values is the last number displayed by the "U" command. If this sum is not zero, the status line message "TEXT" appears in visual mode.

The "M" command executes the contents of the specified register as a command macro, as described in the following section.

Text Register Macros

The text registers may hold commands which can be executed just as if they had been typed in by hand. Frequently used commands, particularly long iteration macros, can be saved in the text registers. These commands are referred to as "command macros" or just "macros" for short. The macros are usually created and edited in visual mode and are then moved to the appropriate register. The macros can also be saved on disk and be retrieved from disk (see RL and RS commands). Macros offer so many capabilities that it is impossible to cover all of the possibilities.

A macro is invoked with the "M" command:

```
M6$$      Executes macro in register 6.
```

A macro may contain an "M" command to invoke the macro in another register. This can be done up to a level of 5.

A common use of macros is for large search and replace operations. Take the example of a long manuscript split into 20 files in which 40 words were consistently mispelled. The task of correcting the words in all 20 files can be done with 2 macros. One will contain the search and replace for the 40 words. The second will edit each file, and for each file execute the search/replace macro. The first macro would appear as:

```
ES 8 1
ES 9 1
b#s/word1/fix1/
b#s/word2/fix2/
b#s/word3/fix3/
 .......
b#s/word40/fix40/
```

The first two commands specify that explicit terminators are to be used and that search errors are to be suppressed. Since explicit terminators are used, the <ESC> character is not needed anywhere. Macros do not need to end in <ESC> <ESC>. Search errors must be suppressed, because otherwise, if any word is not found the entire macro will abort.

The second macro reads in each of the 20 files, executes the first macro, writes the file back to disk, and continues with the next file. It is assumed that the first macro is in register 1.

```
EBfile1.txt<ESC>
M1
#WEF
EBfile2.txt<ESC>
M1
#WEF
 ........
EBfile20.txt<ESC>
M1
#WEF
```

Note that each filename must be ended with an <ESC>. Assuming that this macro is in register 0, the following command would invoke the macro to perform the search and replace on all files:

```
M0$$
```

It is often desirable to save such complex macros on disk for future use. The commands to save these two macros might be:

```
RS1macro1.exc$$
RS0macro2.exc$$
```

Similarly the commands to retrieve them from disk are:

```
RL1macro1.exc$$
RL0macro2.exc$$
```

The commands to display them on the console are:

```
RT1macro1.exc$$
RT0macro2.exc$$
```

Macros are most easily created in visual mode and then moved to the appropriate text register. They can be edited by appending the text register to the end of the current text buffer in visual mode, making the changes and moving them back to the register.

Printing Text

Text can be printed from command mode with the "EO" command. This command takes a numeric argument similar to the "T" command to specify how many lines before or after the edit pointer are to be printed. For example, "40EO" will print the following 40 lines, while "-5EO" will print the preceding 5 lines. Additionally, the command "0EO" will print all lines from the beginning of the text buffer to the current edit pointer. (The edit pointer is the same as the cursor position when you change from visual to command mode). Therefore, the command to print the entire text is:

```
Z0EO        Print entire text on line printer.
```

Command Line Editing

Several common control characters are recognized in command mode as line editing characters. They are:

<CTRL-H> or <BACKSPACE>	Delete the last character typed and echo a <CTRL-H> to the console.
<RUBOUT> or <DELETE>	Delete the last character typed and echo the deleted character to the console.
<CTRL-R>	Doesn't change the command line, but echoes the entire command line back to the console.
<CTRL-U>	Delete the entire command line and send a "#" to the console.
<CTRL-X>	Identical to <CTRL-U>.

'n' denotes a positive number. (# represents 32767)
'm' denotes a number which may be negative to denote backwards in the text buffer.
'r' denotes a digit "0 - 9" specifying a text register.

'string', 's1', 's2' and 'text' denote strings which may include the RETURN key in them. May use explicit terminators, or else must end in <ESC>.

'file' is a disk file name in normal CP/M (MSDOS) format with optional disk drive and extension. Any leading spaces are ignored. Must be ended with an <ESC>.

nA	Append 'n' lines from the input file to the end of the text buffer. "0A" performs an auto-read.
B	Move the edit pointer to the beginning of the text buffer.
mC	Move the edit pointer by 'm' positions.
mD	Delete 'm' characters from the text.
E	First letter of extended two letter commands.
nFstring<ESC>	Search for the 'n'th occurrence of 'string' in the current text buffer and position the edit pointer after it. Only first 32 characters of 'string' are searched.
Gr	Insert the contents of text register 'r' at the edit pointer.
Itext<ESC>	Insert the 'text' into the text buffer at the edit pointer.
mK	Kill 'm' lines.
mL	Move the edit pointer by 'm' lines and leave at the beginning of that line.
Mr	Execute text register 'r' as a command macro.
nNstring<ESC>	Search for the 'n'th occurrence of 'string' and read more of the file from disk if necessary. The edit pointer is positioned after last 'string' if found, else not moved or left at the beginning of the text buffer.

mPr	Put 'm' lines of text into text register 'r'. 'r' may be preceded by "+" to append to the register. "0Pr" empties text register 'r'.
Ss1<ESC>s2<ESC>	Search for the next occurrence of 's1' within the current text buffer, and if found, change to 's2'.
mT	Print (type) 'm' lines.
U	Print # of free bytes remaining / # bytes in text buffer/ # bytes in combined text registers.
V	Go into visual mode. Set cursor position from current edit pointer.
nW	Write 'n' lines to the disk from the beginning of the text buffer and delete from the text buffer. 0W writes out the text buffer up to the current line.
Z	Move the edit pointer to the last character in the text buffer.

SPECIAL CHARACTERS

|	The search wildcard character. Each "|" will match any character in the text being searched. For "F", "N" and "S" commands.
<CTRL-Q>	Literal character. Next character, usually a control character, is taken literally and not interpreted. Allows searching and inserting of control characters including <ESC>.
@	Immediately precedes "F", "I", "N" or "S" to indicate that explicit terminating characters are being used.
:	Immediately precedes "F", N" or "S" to indicate that search error messages are to be suppressed.

EXTENDED COMMANDS

EA	Restart the editor by completely writing the output file, closing it, and then opening the output file again with an EB. The text register is not disturbed.
EBfile	Open the file "file" for both Read and Write and then perform an auto-read if the input file exists. If the file does not exist, "NEW FILE" is printed. Gives error if an output file is still open.
EC	Allow user to change disks, primarily for write error recovery.
EDfile	Delete (erase) the file "file" from the disk. This is primarily intended for write error recovery.
EF	Close the current output file.
EGfile[line range]	Insert the specified line number range of the file "file" into the text buffer at the edit pointer. If no line range is specified, the entire file is inserted.
nEI	Insert the character whose decimal value is "n" into the text buffer at the edit pointer. Only the value "26" is not allowed since this is the CP/M "End of File" marker. Values of 128 to 254 are allowed.
mEO	Send 'm' lines to the line printer. "0EO" prints from the beginning of the text buffer to the current line.
EP n k	Change the value of parameter "n" to "k". Currently there are the following parameters:

1	Cursor type (Mem Mapped Only)	(0, 1 or 2)
2	Cursor blink rate (Mem Mapped Only)	(5 - 100)
3	Indent Increment	(1 - 20)
4	Lower case convert	(0, 1 or 2)
5	Conditional convert character	(32 - 126)
6	Display line and column position (0 = none, 1 = line, 2 = column, 3 = both)	(0 - 3)
7	Word Wrap column (0 = Off)	(0 - 255)

EQ	Quit the edit session and leave disk files exactly as before the session started.
ERfile	Open the file "file" for input. Gives error if file does not exist.
ES n k	Change the value of switch "n" to "k". Currently there are the following switches:

1	Expand Tab with spaces	(0=NO 1=YES)
2	Auto buffering in visual mode	(0=NO 1=YES)
3	Start in visual mode	(0=NO 1=YES)
4	Point past text reg. insert	(0=NO 1=YES)
5	Ignore UC/LC distinction in search	(0=NO 1=YES)
6	Clear screen on Visual Exit	(0=NO 1=YES)
7	Reverse Upper and Lower case	(0=NO 1=YES)
8	Suppress search errors	(0=NO 1=YES)
9	Explicit string terminators	(0=NO 1=YES)

ET	Set new tab positions. The ET is followed by up to 30 decimal numbers specifying the tab positions. Since the positions start at 1, the normal positions would be: 9 17 25 33 etc.
EV	Print the VEDIT version number.
EWfile	Open the file "file" for output. Any existing file by that name will be renamed to "file.BAK" following an EF or EX. Gives error if an output file is already open.
EX	Exit back to CP/M after writing the text and any unappended part of the input file to the output file. Gives error if no output file is open.

TEXT REGISTER COMMANDS

RDr	Dump contents of register 'r' on console. Control characters are not expanded.
RLrfile	Load register 'r' from file 'file'.
RSrfile	Save contents of register 'r' in file 'file'.
RTr	Type contents of register 'r' on console. Control characters are expanded, <ESC> is represented as "$".
RU	Display number of characters (size) in each text register.

nA Append

Example: 100A$$ 0A$$

Description: This command will append 'n' lines from the input file to the end of the text buffer. Fewer lines will be appended if there is insufficient memory space for 'n' lines, or there are not 'n' lines remaining in the input file. If 'n' is 0, an auto-read is performed, which reads all of the input file or until the main memory is almost full. The command can be issued (with 'n' not zero) after an auto-read to read in more of the file. An error is given if there is no input file open when this command is issued. The input file can be opened with the EB and ER commands, or when VEDIT is invoked from CP/M.

Notes: No indication is given if fewer than 'n' lines were appended. Use the "U" command to see if anything was appended. If the text buffer is completely full, the text registers cannot be used and visual mode will not work well.

See Also: Commands: U, W, EB, EG, ER
Auto-Read

Examples:

```
ERTEXT.DOC$$
0A$$
```

The file 'TEXT.DOC' is opened and all of the file is read in, or until the memory is almost full.

B Beginning

Example: B$$

Description: This command moves the edit pointer to the beginning of the text buffer. The beginning of the text buffer will not be the beginning of the text file if a "W" command or an auto-write was done. In this case, use the "EA" command to move back to the beginning of the text file.

Notes:

See Also: Commands: EA, Z

Examples: B12T$$ Moves the edit pointer to the beginning of the text buffer and prints the first 12 lines.

mC Change

Example: 12C$$ -4C$$

Description: This command moves the edit pointer by 'm' character positions, forwards if 'm' is positive and backwards if 'm' is negative. The edit pointer cannot be moved beyond the beginning or the end of the text buffer, and an attempt to do so will leave the edit pointer at the beginning or the end respectively. Remember that every line normally ends in a <CR> <LF> (carriage return, line feed), which represents two character positions.

Notes:

See Also: Commands: D, L

Examples: Fhello$-5C$$ Searches for the word "hello", and if it is found, positions the edit pointer at the beginning of the word.

mD Delete

Example: 12D$$ -4D$$

Description: This command deletes 'm' characters from the text buffer, starting at the current edit pointer. If 'm' is positive, the 'm' characters immediately at and following the edit pointer are deleted. If 'm' is negative, the 'm' characters preceding the edit pointer are deleted. Fewer than 'm' characters will be deleted if the ends of the text buffer are reached.

Notes:

See Also: Commands: C, K

Examples: 100<FBIKES$-D$>$$ The 'S' will be deleted from up to 100 occurrences of the word 'BIKES'.

E Extended Commands

Example: EX$$ EV$$

Description: This is not a command by itself but just the first letter of all the extended commands.

Notes: No error is given if just E$$ is given.

See Also: Extended commands.

Examples:

nFs1<ESC> Find

Example: Fmispell$$ 10Fwords$$ F$$

Description: This command searches the text buffer, beginning from the current edit pointer, for the 'n'th occurrence of the string 's1'. The edit pointer will be positioned after the last character of the 'n'th occurrence of 's1' if it is found. If the 'n'th occurrence of 's1' is not found, an error will be given (unless suppressed) and the edit pointer will be positioned after the last occurrence of 's1' found, or be left at its original position if no occurrences of 's1' were found. If no string is specified, the search will reuse the previously specified string. The switch "Ignore Upper/Lower case distinction" will determine if the search will ignore the distinction between upper and lower case letters. If the search is to include parts of the file not yet in the text buffer, use the "N" command.

Notes: The search is always forward, never backwards. While ignoring the upper/lower case distinction is usually more convenient, the search will take longer. Remember that the "wild card" character can be used. The "@" character allows an explicit delimiting character. For the command form "#Fs1<ESC>", an error is only given if no occurrences of 's1' are found.

See Also: Command: N

Examples:

BFhello$$ Searches for the word "hello" from the beginning of the text buffer.

#[3Ffirst$-5DIthird$]$$ Changes every third occurrence of the word "first" to "third".

Z-100LFend$$ Find the word "end" if it occurs in the last 100 lines of the text buffer.

#[@F/fix up/V]$$ Finds the next occurrence of the string "fix up" and enters Visual mode. Any changes can be made in Visual mode. When Visual mode is exited, the next occurrence of "fix up" is found and so on.

F$V$$ The next occurrence of the previous specified string is found, and visual mode is then entered.

Gr Get

Example: G4$$

Description: This command inserts a copy of text register 'r' at the current edit pointer. If there is insufficient memory space for the entire copy, nothing is inserted and an error message is given. If the text register is empty, nothing is inserted. The contents of the text register are not affected by this command. The "P" command or visual mode is used to place text in a text register.

Notes:

See Also: Commands: P
Visual Mode Text Registers

Examples:

BG9$$	Inserts the contents of text register 9 at the very beginning of the text buffer.
12[G2]$$	Inserts the contents of text register 2 twelve times at the current edit pointer.
132P3$132K$$ EA$$ 10LG$$	Moves 132 lines of text, by saving it in text register 3, killing the original lines and inserting the text after the tenth line of the file, in the situation where the beginning of the file is no longer in the text buffer.

Itext<ESC> Insert

Example: Ia word$$ I<CR>new line$$

Description: This command inserts the text 'text' into the text buffer, starting at the current edit pointer. The insertion is complete when the <ESC> (or explicit delimiter) character is encountered. The inserted text does not overwrite any existing text. The 'text' may contain the <CR> key, which is expanded to carriage return - line feed. If insufficient memory space exits for the 'text', an error will be printed and only part of the 'text' will have been inserted. The edit pointer is moved just past the inserted text. This command is probably best used in iteration macros, since normal text insertion is much easier to do in visual mode.

Notes: Control characters including <ESC> can be inserted by preceding them with the literal character <CTRL-Q>. The "@" charactera allows an explicit delimiting character to be used. The tab character is not expanded with spaces as is optional in visual mode.

See Also: Commands: EI

Examples:

200[I<CR><TAB>$]$$ Inserts 200 new lines, each beginning with a tab character.

Iunder<CTRL-Q><CTRL-H> $$ Inserts the text "under", a BACKSPACE and the underline character. This will underline the "r" on some printers.

@I/a word/$$ Inserts the text "a word" into the text buffer.

@I/EP 7 70<ESC><CR>/$$ Inserts the command line "EP 7 70 <ESC>" into the text, including a RETURN.

mK Kill

Example: 4K$$ -3K$$ 0K$$

Description: This command performs a line oriented deletion (or killing) of text. If 'm' is positive, all characters from the current edit pointer and up to and including the 'm'th [LF] are deleted from the text buffer. If 'm' is negative, all characters preceding the edit pointer on the current line and the 'm' preceding lines are deleted. If 'm' is 0, all characters preceding the edit pointer on the current line are deleted. Fewer than 'm' lines will be killed if either end of the text buffer is reached.

Notes:

See Also: Command: D, T

Examples:

#[Ftemp line$0LK]$$	Kills all lines which contain the string "temp line".
-10000K$$	Kills all text before the edit pointer.
#P#K$$	Saves the rest of the text from the edit pointer in the text register and then deletes it from the text buffer.

mL	Lines

Example: 120L$$ -14L$$ 0L$$

Description: This command performs a line oriented movement of the edit pointer, and the edit pointer is always left at the beginning of a line. If 'm' is positive, the edit pointer is left following the 'm'th <LF>. If 'm' is negative, the edit pointer is left at the beginning of the 'm'th preceding line. If 'm' is 0, the edit pointer it moved to the beginning of the current line. Attempting to move past the ends of the text buffer will leave the edit pointer at the respective end. This command makes no changes to the text buffer.

Notes:

See Also: Commands: C, T

Examples: #[Stypo$type$0LT]$$ Changes all occurrences of "typo" to "type" and prints out every line that was changed.

Mr Macro

Example: MO$$

Description: This command executes the contents of register 'r' as a command macro. Any legitimate command or sequence of commands may be executed as a macro. Macro are most easily created and edited in visual mode. They may also be saved and loaded from disk. A macro may invoke another text register, which in turn may invoke another, up to a nesting depth of 5. Macros are most convenient for holding long command sequences which are repeatedly used, saving the effort of retyping them each time.

Notes: Macros do not need to end in an <ESC> <ESC>. RETURNs may be used to separate commands in order to improve readability.

See Also: Commands: G, P, RL, RS
Visual Mode Text Registers, Text Register Macros.

Examples: See section "Text Register Macros" for an example.

nNsl<ESC>	Next

Example: Nbad line$$ 3@N/third/$$ N$$

Description: This command is very similar to the "F" command, except that if the 'n'th occurrence of 's1' is not found in the text buffer, auto-read/writes are performed to read in more of the input file until the 'n'th occurrence is found or the end of the input file is reached. If the 'n'th occurrence still is not found, an error is printed. The edit pointer is also positioned very similar to the "F" command, except in the event the 'n'th occurrence is not found and neither the 'n-1'th occurrence nor the original edit pointer position is any longer in the text buffer. In this case the edit pointer is positioned at the beginning of the text buffer. Using this command with an 's1', which you know does not exist, can be used to access the last part of a large file.

Notes: All Notes for the "F" command also apply.

See Also: Command: F
Auto Buffering

Examples: #[Ntypo$-4DItype$]$$ Changes all occurrences of the string "typo" to "type" in the rest of the file.

#[@N/typo/-4D@I/type/]$$ Alternate form of the same command using explicit delimiters.

Nxcxc$$ Accesses the last part of the file, assuming the string "xcxc" never occurs in it.

	mPr	Put
Example:	40P1$$	-20P+2$$ 0P3$$

Description: This command saves a copy of the specified text lines in text register 'r'. The previous contents of the text register are destroyed, unless the 'r' is preceded with a "+" indicating that the text is to be appended. The range of lines saved is the same as for the "K" or "T" commands. If 'm' is zero, the text register is simply emptied, and nothing is saved in it. Since the text buffer and the text registers share the same memory space, saving text in the text registers decreases the amount of memory available to the text buffer. Thus the "0Pr" command should be given when the text in a register is no longer needed. This command does not change the text buffer. If there is insufficient memory space for the text copy, the text register is only emptied, nothing is saved in it and an error is printed. The saved text is inserted in the text buffer with the "G" command or in Visual mode.

Notes: Using the "P" command to change the contents of a register which is currently executing as a macro is not recommended.

See Also: Commands: G, K, T
Visual Mode text move

Examples:

120P1$120K$$	The text lines are saved in text register 1 and are then deleted from the text buffer.
-23T$$ -23P6$$	The text lines are printed for verification before they are saved in the text register.

nSs1<ESC>s2<ESC> Substitute

Example: Stypo$type$$ #Sname$Mr. Smith$$

Description: This command performs 'n' search and substitute operations. Each operation consists of searching for the next occurrence of 's1' in the text buffer and changing it to 's2'. An error is printed if 's1' is not found. If there is insufficient memory space for inserting 's2', 's1' will have been changed to as much of 's2' as possible and an error is printed. The edit pointer is positioned after 's2', if 's1' is found, or else is left at its original position if 's1' is not found. For the command form "#Ss1<ESC>s2<ESC>, an error is only given if no occurrences of 's1' are found. See the "N" command example on how to perform a "substitute" if all of the file is not in the text buffer.

Notes: All Notes for the "F" command apply here too. A command like #Sfishes$fish$$ will execute much faster than the equivalent command #[Sfishes$fish$]$$.

See Also: Commands: F, N, I

Examples:

#Stypo$type$$ Changes all occurrences of "typo" to "type".

#[Stypo$type$0LT]$$ Changes all occurrences of "typo" to "type" and prints out every line that was changed.

ES 9 1$$
#[S/typo/type/0LT]$$ Alternate form of above command. Explicit terminators can now be used without "@" prefix.

#[Sname$smith$V]$$ Change the next occurrence of "name" to "smith" and enter into Visual mode. Any changes can be made in Visual mode and when Visual mode is exited, the next occurrence of "name" will be searched and so on.

#Sgarbage$$ Deletes all occurrences of the string "garbage" from the rest of the text buffer.

mT Type

Example: 14T$$ -6T$$ 0T$$

Description: This command prints (types) the specified lines. If 'm' is positive, all characters from the edit pointer up to and including the 'm'th <LF> are typed. If 'm' is negative, the previous 'm' lines and all characters up to just preceding the edit pointer are printed. If 'm' is 0, only the characters on the present line preceding the edit pointer are printed. Fewer than 'm' lines will be printed if either end of the text buffer is reached. Note that "0TT" will print the current line regardless of the position of the edit pointer on it. This command does not move the edit pointer. This command is most useful in iteration macros for printing selected lines. Visual mode should be used for looking at sections of a file.

Notes:

See Also:

Examples: #[Fmoney$0TT]$$ Prints out every line in the text buffer with the string "money" in it.

U Unused (Free Memory)

Example: U$$

Description: This command prints the number of memory bytes free for use by the text buffer or text register, followed by the number of memory bytes used by the text buffer (length of the text buffer), followed by the combined number of memory bytes used by the text registers (length of the text registers).

Notes: These three numbers will not always add up to the same total, since several other small buffers all use the same memory space. If the number of free bytes goes below 260, the "FULL" flag will be set when in visual mode.

See Also:

Examples:

V Visual

Example: V$$

Description: This command enters Visual Mode. The visual cursor position will be set from the current edit pointer position. Visual mode is exited with either the "Visual Exit" or the "Visual Escape" character. At that time the edit pointer will be set from the cursor position.

Notes: The text registers are preserved.

See Also: Visual Mode

Examples: Fhere$V$$ Find the word "here" and enter visual mode.

nW Write

Example: 20W$$ #W$$ 0W$$

Description: This command writes 'n' lines from the beginning of the text buffer to the output file and then deletes these lines from the text buffer. If there are less than 'n' lines in the text buffer, the entire text buffer is written out and deleted. If 'n' is zero, the entire text buffer up to the line the edit pointer is currently on, is written out. The edit pointer is moved to the new beginning of the text buffer. If no output file is open, an error is printed and no text is output nor deleted. The output file can be opened with an "EW" or "EB" command or when VEDIT is invoked.

Notes: No indication is given if less than 'n' lines were written.

See Also: Commands: A, EB, EW, EX

Examples:

```
EWpart1.txt$$
24W$$
EF$$
EWpart2.txt$$
EX$$
```

The first 24 lines of the text buffer are written out to file "PART1.TXT" and the rest of the text buffer is written out to file "PART".TXT" and the edit session is completed.

Z Zip

Example: Z$$

Description: This command moves the edit pointer to the last character in the text buffer.

Notes: This command does not move the edit pointer to the last character in the file if the last part of the file is not yet in the text buffer. See the "N" command on how to bring the last part of the file into the text buffer.

See Also: Commands: B, N

Example:

Z-100L$$	Positions the edit pointer to the 100th line before the end of the text buffer.
Z-12T$$	Prints the last twelve lines in the text buffer.
Nxcxc$Z-12T$$	Prints the last twelve lines in the file, assuming the string "xcxc" never occurs in it.

EA Edit Again

Example: EA$$

Description: This command writes the entire text buffer out to the output file, followed by the remainder of the input file if any and closes the output file. All file backup and renaming is performed as with the "EF" or "EX" command. The output file is then reopened as both the input and output file and an auto-read on the input file is performed. This command thus starts a new edit session and is functionally similar to an "EX" command followed by invoking VEDIT again with the name of the current output file. This command has two main purposes. First, it acts a method of saving the currently edited file on disk as a safeguard against losing the file due to a user error, or hardware, software or power failure. Second, it acts as a method of accessing the beginning of a large file after it has been written out to disk. This is especially true in the case a block of text is to be moved from the rear of a large file to the front, since the contents of the text register are not affected by the "EA" command. If the "Start in Visual Mode" switch is set, the editor will go into visual mode following the "EA" command.

Notes: Any commands following the "EA" on the command line will be ignored, since the command line is cleared.

See Also: Commands: B, G, EX
Visual Restart

Example:

```
132P132K$$
EA$$
10LG$$
```

Moves 132 lines of text, by saving it in the text register, killing the original lines and inserting the text after the tenth line of the file, in the situation where the beginning of the file is no longer in the text buffer.

EBfile<ESC> Edit Backup

Example: EBfile.txt$$

Description: This command opens the file 'file' for both input and output and then performs an auto-read on the file. It is similar to the sequence of commands:

ERfile<ESC>EWfile<ESC>0A$$

except that if the file does not yet exist on disk, the message "NEW FILE" is printed. If an output file is still open, an error is printed and the command has no other effect.

Notes: The term "backup" is used here to describe this command since the term is used by some other editors to perform a similar operation. Remember that VEDIT always creates a "backup" of a file on disk, if its name is used as the name of the output file.

See Also: Commands: W, ER, EW

Example:

```
#W$EF$$
EBnewfile.txt$$
```

The entire text buffer is written out to the current output file, that file is closed, and the file "NEWFILE.TXT" is opened for input and output and read in.

```
ERpart1.txt$0A$$
EBpart2.txt$$
```

The file "PART1.TXT" is read into the text buffer, the file "PART2.TXT" is then made the current input and output file and is appended to the end of the previous file "PART1.TXT".

EC Edit Change (Disk)

Example: EC$$

Description: This command must be given before the user attempts to change any logged-in disks in order to recovery from a disk write error, or to read files from another disk. An error is printed if the current disk has an output file which has not been closed. In this case it should be closed with the "EF" command. This command is used in the event of a disk write error where the user does not wish to delete any files with the "ED" command. In this case the "EF" command should be given to close that part of the output file which has been written to the original disk. Then issue the "EC" command. It will prompt with a message when the original disk can be removed and a new disk inserted. Type a [RETURN] after the new disk is inserted and then issue an "EW" command to open a file for output. The user can then issue any "W" command or the "EX" command. When the edit session is over the output file is in two parts on two disks. They can easily be merged with a PIP command or with VEDIT. See the "ER" command for this. This command can also be used to switch to another disk before an "ER" or "EG" command.

Notes: Be sure that the entire input file has been read into memory before issuing the "EC" command.

See Also: Commands: ED, EF
Disk Write Error Recovery.

Example: EC$$ Will give prompt: INSERT NEW DISK AND TYPE RETURN when the user should remove the old disk and insert a new disk.

EDfile<ESC> Edit Delete

Example: EDfile.txt$$

Description: This command will erase the file 'file' from the disk. This is the easiest method of recovering from a disk write error in order to make more disk space or a free entry in the directory. The "EC" command can also be used for disk write error recovery.

Notes: Be sure that you do not delete the file which is currently open for output. Don't delete the input file until all of it has been read into memory.

See Also: Commands: EC
Disk Write Error Recovery

Example: EDoldfile.txt$$ The file "OLDFILE.TXT" is deleted from the disk making more disk space and a free directory entry.

EF Edit Finish (Close)

Example: EF$$

Description: This command closes the output file and the file is saved on disk. No file is saved on disk before either this command or an "EX" command is executed. A backup of any existing file on disk with the same name as the output file is created by renaming it with a file extension of ".BAK".

Notes: Since the output file is actually opened with the CP/M file extension ".$$$", the .$$$ file is first closed, then any existing file on disk with the same name as the output file is renamed to .BAK, and last, the .$$$ file is renamed to the true output file name.

See Also: Commands: EW, EX

Example: EWsave.txt$$
#W$EF$$ The contents of the text buffer is written out as the file "SAVE.TXT" and that file is then closed.

EGfile[line range] Edit Get (File)

Example: EGfile.txt[1,100]$$ EGfile.txt$$

Description: This command will insert a specified line number range of the file "file" into the text buffer at the edit pointer. If insufficient memory exists to insert the entire file segment, as much as possible will be inserted and a *BREAK* message will be printed. If no line range is specified, the entire file is inserted.

Notes: The line numbers of a file can be printed by PIP using the [N] option.

See Also: Commands: A, ER

Example: EGlibrary.asm[34,65]$$ Lines 34 through 65 of the file "LIBRARY.ASM" are inserted into the text buffer at the edit pointer.

nEI Edit Insert

Example: 12EI$$

Description: This command will insert the character whose decimal value is "n" into the text buffer at the edit pointer. This is useful for entering special control characters into the text buffer, especially characters which cannot be generated from the keyboard. Characters with a decimal value between 128 and 255 can also be entered with the EI command. Only the "End of File" marker with a value of 26 cannot be entered. Control characters are displayed in both command and visual mode by preceding the letter with an "Up Arrow".

Notes:

See Also: Commands: I

Example: 8EI$$ A backspace character is inserted into the text buffer at the edit pointer.

92EI$$ A "\" is inserted into the text with the EI command, because it cannot be generated from the keyboard.

mEO — Output to Printer

Example: 40EO$$ -20EO$$ 0EO$$

Description: This command sends the specified lines to the line printer device. If 'm' is positive, all characters from the edit pointer up to and including the 'm'th <LF> are printed. If 'm' is negative, the previous 'm' lines and all characters up to just preceding the edit pointer are printed. If 'm' is 0, the entire text from the beginning of the text buffer up to the current edit pointer are printed. Fewer than 'm' lines will be printed if either end of the text buffer is reached. This command does not move the edit pointer.

Notes: The print out can be stopped by typing CTRL-C.

See Also: Commands: T
Printing Text from visual mode

Example: ZOEO$$ Prints out the entire text buffer and positions the edit pointer at the end of the text.

EP n k<ESC> — Edit Parameters

Example: EP 1 4$$ EP 3 30$$

Description: This command changes the value of parameter 'n' to 'k'. Currently there are 7 parameters. The numbers are specified in decimal and separated by spaces or commas. The default values of these parameters are determined during the customization process. An error is given if 'n' is specified out of range. The parameters are:

1	Cursor type	(0, 1 or 2)
2	Cursor blink rate	(5 - 100)
3	Indent Increment	(1 - 20)
4	Lower case convert	(0, 1 or 2)
5	Conditional convert character	(32 - 126)
6	Display line and column number	(0, 1, 2 or 3)
7	Word Wrap column	(0 - 255)

Parameter (1) determines the type of cursor displayed in visual mode for memory mapped versions The CRT terminal versions use the terminal's cursor instead. The cursor types are: 0=Underline, 1=Blinking Reverse Video Block, 2=Solid Reverse Video Block.

Parameter (2) determines the cursor's blink rate for cursor types 0 and 1 above.

Parameter (3) determines how much further the editor will indent each time the [INDENT] key is typed. The indent position after typing the [INDENT] key four times is therefore the "Indent Increment" multiplied by four.

Parameter (4) determines whether lower case characters are converted to upper case. For value (0) no conversion takes place, for (1) all lower case are converted to upper case, and for (2) lower case are converted to upper case, unless the cursor is past a "special" character on the text line. This "special" character is set by parameter (5). All of this is primarily applicable to assembly language programming, where it is desirable to have the Label, Opcode and Operand in upper case and the comment in upper and lower case.

Parameter (5) sets the conditional upper/lower case convert character used for parameter (4) above.

Parameter (6) determines if the the line number in the file that the cursor is on, and / or the cursor's horizontal position are displayed on the status line. The values are: 0 = Both off, 1 = Line number displayed, 2 = column displayed and 3 = both displayed.

Parameter (7) is the Word Wrap column. It is also the right margin used when formatting paragraphs. A value of 0 disables both Word Wrap and formatting. It should be turned off when editing programs!

Notes: The parameter values are specified in decimal.

See Also: Commands: ES
Customization, Visual Mode, Indent and Undent Functions

Examples:

EP 1 6$$	This sets the "Indent Increment" to six.
EP 7 70$$	This sets the Word Wrap column to 70.

EQ Edit Quit

Example: EQ$$

Description: This command quits the edit session without writing out the text buffer or closing any output file. Its main purpose to "quit" after one has made a mistake editing and it seems best to leave everything on disk just the way it was before this edit session began. DO NOT confuse this command with the "EA" command; their results are quite opposite. Remember that the "EA" command starts a new edit session.

Notes: Any output file with the file extension ".$$$" will also be deleted. Any original file on disk with the same name as the output file, but with an extension of ".BAK" will have been deleted if more than 128 characters were written to the (now deleted) output file. With the exception of this possible backup file, all other files will exist on disk just as they did before the aborted edit session.

See Also: Commands: EA

Example:

#K$$	Shoot!! Meant -#K$$
EQ$$	Since a bad mistake was made in the above command, it is best to abort this edit session, go back to the operating system and start over. All edit changes are lost.

ERfile<ESC> Edit Read

Example: ERnewfile.txt$$

Description: This command opens the file 'file' for input (reading). Nothing is read into the text buffer with this command. The "A" command or an auto-read is used to actually read the input file. If the same file was already open for input, the file is "rewound", so that the file can again be read from the beginning. An error is printed if the file 'file' does not exist. Files can also be read from disks which are not currently running by using the "EC" command. Issue the "EC" command, insert the new disk into a drive which is not being used for any output file and open a file for reading with the "ER" command. This may be necessary in case a file has been split into two parts during a disk write error recovery.

Notes:

See Also: Commands: A, EC, EB, EW

Example:

```
ERparts.inv$$
20A$$
```

The file "PARTS.INV" is opened for input and twenty lines from it are appended to the end of the text buffer.

ES n k<ESC> Edit Set

Example: ES 1 0$$ ES 3 1$$

Description: This command changes the value of switch 'n' to 'k'. Currently there are 7 switches. The numbers are specified in decimal and separated by spaces or commas. The default values of these switches are determined during the customization process. An error is given if 'n' is specified out of range. The switches are:

1	Expand Tab with spaces	(0=NO 1=YES)
2	Auto buffering in visual mode	(0=NO 1=YES)
3	Start in visual mode	(0=NO 1=YES)
4	Point past text reg. insert	(0=NO 1=YES)
5	Ignore UC/LC distinction in search	(0=NO 1=YES)
6	Clear screen on visual exit	(0=NO 1=YES)
7	Reverse Upper and Lower case	(0=NO 1=YES)
8	Suppress search errors	(0=NO 1=YES)
9	Use explicit string terminators	(0=NO 1=YES)

Switch (1) determines whether or not the tab key in visual mode is expanded with spaces to the next tab position. If not, a tab character is inserted into the text buffer. Except for special applications, the tab key would not normally be expanded with spaces.

Switch (2) determines whether or not auto-buffering is enabled in visual mode. The editing of a large file is usually simpler with this switch on, since the user does not need to give explicit Read/Write commands. If some more complicated file handling, with explicit Read/Write commands (ER, EW, A, W) is being done, the switch should then temporarily be set off.

Switch (3) determines whether or not the edit session will begin in visual mode. Changing this switch while running VEDIT will only apply to the "EA" command.

Switch (4) determines the edit pointer's position (or cursor's in visual mode) following insertion of the text register. If the switch is off, the edit pointer is not moved, and is thus left at the beginning of the newly inserted text. If the switch is on, the edit pointer is moved just past the newly inserted text.

Switch (5) determines whether VEDIT will make a distinction between upper and lower case letters in searches and substitutes using the "F", "N" and "S" commands. Most users will probably wish to ignore the distinction, so that the string "why" will match "Why", "WHY" and "why". Setting the switch to "1" will make VEDIT ignore the distinction between upper and lower case characters during searches.

Switch (6) determines whether the screen will be cleared when visual mode is exited and command mode entered. If the screen is not cleared, the command mode prompt "*" will appear below the status line. Setting the switch to "1" will clear the screen when visual mode is exited.

Switch (7) determines whether all letters typed on the keyboard will be reversed with respect to upper and lower case. It should normally be OFF, but does allow a user with an upper case only keyboard to enter lower case letters. Setting the switch to "1" will make VEDIT reverse all keyboard letters in both commmand and visual mode.

Switch (8) determines whether search errors will be suppressed. If not suppressed, not finding a string will cause an error message and the command to be aborted. Search errors are usually only suppressed for command macros.

Switch (9) determines whether explicit string terminators can be used without having to specify the "@" command modifier. This is a matter of personal preference, but is useful with command macros.

Notes:

See Also: Customization, Visual Mode

Example: ES 1 1$$ This enables tabs typed in visual mode to be expanded with spaces.

ET Edit Tab

Example: ET 20 40 60 80 100 120$$

Description: This command changes the tab table used by VEDIT for displaying tab characters, and in Visual mode, when the "Expand Tab" switch is set, for expanding tab characters. Up to 30 tab positions are allowed and they must be in the range 1 - 254. The default positions are set during customization. For word processing the tabs can be set to the same positions as are specified for the text formatting program in order to see how they will look in the final product. An error is printed if a bad position is given. No tab is needed at position 1, and counting starts at 1 (not at zero). Thus the normal tab positions would be:

9 17 25 33 41 49 57 65 73 81 89 97
105 113 121 129

Notes: For use in Visual mode, there must be at least one tab position per screen line, i.e. at least one tab every 64 or 80 positions.

See Also: Customization, Visual Mode, Indent and Undent Functions

Example:

EV Edit Version

Example: EV$$

Description: This command prints the VEDIT version number. This number should be used in any correspondence you have with us concerning the operation of VEDIT. This command can also be used inside iteration macros to give some indication of the progress being made in long executing macros.

Notes:

See Also:

Example:

EWfile<ESC> Edit Write

Example: EWnewdat.inv$$

Description: This command opens the file 'file' for output and subsequent writing. No text is actually written by this command. Some file must be opened for output in order to save any text on disk. A file can also be opened by the "EB", "EA" commands and when VEDIT is invoked from CP/M. If a file is already open for output, an error is printed and no other action takes place.

Notes: The file opened is actually a temporary file with the same name, but with an extension of ".$$$". The file is not made permanent and given its true name until it is closed with the "EF", "EA", or "EX" commands. At that time, any existing file on disk with the same name as the output file is backed up by renaming it with an extension of ".BAK". Any existing file on disk with that name and the .BAK extension will be deleted when more than 128 bytes (the first sector) are written to the output file.

See Also: Commands: W, EA, EF, EX

Example:

```
EWpart1.txt$$
24W$$
EF$$
EWpart2.txt$$
EX$$
```

The first 24 lines of the text buffer are written out to file "PART1.TXT" and the rest of the text buffer is written out to file "PART".TXT" and edit session is completed.

```
ERa:bigfile.asm$$
EWb:bigfile.asm$$
0A$v$$
```

Typical procedure for editing a file which is too big for both it and its Backup to fit on the same disk. In this case, it is read from disk A: and written to disk B:. Just be sure that disk B: is nearly empty.

EX Edit Exit

Example: EX$$

Description: This is the normal exit from VEDIT when the file currently being edited is to written out to disk. This command writes the entire text buffer out to the output file, followed by the remainder of the input file if any, closes the output file and exits back to CP/M. All file backup and renaming is done as with the "EF" command. The error "NO OUTPUT FILE" is printed if no output file is open, and no other action is taken. The error "NO DISK SPACE" results if there is insufficient disk space to save the entire file.

Notes: In case of a "NO DISK SPACE" error, the rest of the file can be saved by either deleting some files on disk (ED command), or writing the rest out to another disk (EC command).

See Also: Commands: EB, EF, EW, EA, EQ

Example:

```
VEDIT FILE.TXT
V$$
EX$$
```

The editor is invoked in the normal way to edit a file. The file is edited in visual mode, and when done, the normal exit back to CP/M is made.

```
EX$$
NO DISK SPACE
EDoldfile$$
EX$$
```

Disk full error.

Because the disk is full, an old file is deleted and the "EX" command given again to finish saving the file on disk.

```
EX$$
NO DISK SPACE
EF$$
EC$$
EWpart2$$
EX$$
```

Disk full error.
Close the partial file.

Because of disk is full, we save what will fit as part 1 of the file, and save the rest in the file "PART2" on another disk. We can subsequently use the "ER" command to merge the two parts back into one file.

RDr Register Dump

Example: RD3$$

Description: This command types (dumps) out the contents of text register 'r' on the console. Control and Tab characters are not expanded or converted. The command is most useful for sending initialization sequences to a CRT terminal, such as sequences to setup programmable function keys. <CTRL-C> can be used to stop the command. The "RT" command should be used to view the registers, since control characters are then expanded.

Notes:

See Also: Commands: RT
Auto-Startup

Example: RD5$$ The contents of text register 5 are dumped (sent) to the console.

RLrfile<ESC> Register Load

Example: RL4macro1.exc$$

Description: This command loads text register 'r' from the file 'file'. The entire file is loaded and the file itself is not affected. If there is insufficient memory space to load the entire file, as much as possible will be loaded and a *BREAK* message will be printed. Used to load text files which are then inserted in the text buffer, and to load command macros.

Notes: The text register number should always be specified, but if left out, register 0 will be used.

See Also: Commands: RS, EG
Text registers in visual and command modes.

Example: RL4macro.exc$$ Loads the file "MACRO.EXC" into text register 4.

RSrfile<ESC> Register Save

Example: RS4macro1.exc$$

Description: This command save the contents of text register 'r' in the created file 'file'. The register contents are not affected. If there is insufficient disk space for the entire file, as much as possible is saved and the error "NO DISK SPACE" is given. The error "NO DIR SPACE" results if there is insufficient directory space on the disk. Commonly used to save a section of text in its own disk file, or to save a command macro for later use.

Notes: See note for "RL" command. Any existing file on the disk with the name 'file' will be deleted. If there is insufficient disk space to save the register, try deleting some files or insert another disk and give the command again after using the EC command.

See Also: Commands: RL

Example: RS4macro.exc$$ Saves the contents of text register 4 in the file "MACRO.EXC".

RTr Register Type

Example: RT3$$

Description: This command types out the contents of text register 'r' on the console. This is commonly used to remind oneself what is in a particular register. <CTRL-C> can be used to stop the command. Control characters are expanded and <ESC> is represented as a "$". The "RD" command will dump out a text register without expanding control characters.

Notes: It is frequently easier to insert the text register at the end of the text buffer and view it in visual mode.

See Also: Commands: RD

Example: RT5$$ The contents of text register 5 are displayed on the console.

RU Register Used

Example: RU$$ 50P4$RU$$

Description: This command displays the number of character (size) of each text register. Commonly used to see which register hold any text and how much they hold.

Notes: The sum of the displayed values is the third number displayed by the "U" command. If any of the registers hold text, the status line message "TEXT" is displayed in visual mode.

See Also: Commands: U

Example: RU$$ Display the sizes of the text registers.

40P3$RU$$ Save 40 lines of text in register 3 and then check how many bytes are now in the text registers.

CUSTOMIZING VEDIT

WHAT IS CUSTOMIZATION?

Customization is the process of installing VEDIT on your computer in order to adapt it to your particular CRT terminal or video board and your preference in keyboard layout. It also allows you to set various VEDIT parameters according to your applications. The customization is menu driven. You can select to perform some aspects of the customization and leave all other aspects at their previously set or default values. You therefore don't need to understand the entire customization process in order to install VEDIT.

Setting up a new keyboard layout is one aspect of the customization. It allows almost any control character, escape sequence or special function key to be used for the visual mode cursor movements and editing functions. The changeable parameters include the Tab positions, the right margin at which word wrap takes place, and many others. Another aspect is related to your screen size, including the number of lines and columns.

The first part of this appendix gives the step by step instructions for the customization. The later part "Customization Notes" covers some of the customization issues in greater depth.

WHEN IS CUSTOMIZATION NECESSARY?

VEDIT has to be customized before the first time it is used, and can then be customized again, when you have a a new CRT terminal, wishes to change some default parameters or just wish to try a new keyboard layout. It is not customized every time you use it. The greatest benefit you receive from the customization process is probably the ability to determine your own keyboard layout, which can utilize any special function keys and accommodate personal preferences.

For some computers, such as the IBM Personal Computer and the IBM Displaywriter, we supply an "up and running" version. The keyboard layout for these preconfigured versions will be listed among the sheets separate from the manual. Even with a preconfigured version, you will probably want to customize your own later. If you don't have a preconfigured version of VEDIT, you can create one easily. See "Quick Customization" described under the section "How to Perform Customization". With a CRT version of VEDIT, you generally only need to select the CRT terminal from the menu in order to complete the customization. The keyboard layout will be the "Example Keyboard

Layout" which does not assume any function keys. If you have a terminal such as a Televideo 920C or a Zenith Z19, you will probably want to reconfigure the keyboard layout according to the example layouts sheets we supply for these terminals.

VEDIT is supplied as a disk file with an extension of ".SET", i.e., VEDITZM.SET and VEDITZC.SET, which contain the "prototype" editor to be customized. The customization process does not alter the .SET file, but rather creates a new file with the file extension of ".COM" (or ".CMD" for CP/M-86), which is the executable version VEDIT. Depending on your version, you may have several .SET files. See the sheet "Description of Files".

The customization is done with the supplied programs VEDSET.COM for the memory mapped versions, and VDSETCRT.COM for the CRT terminal versions. Since the customization program is fairly easy to run, you will probably run it several times in the first week until you have everything "just right". You can of course also create several configurations of VEDIT, each for a special application. To help remind you of which configuration you are using, you can create a custom signon message for each, which will be displayed when VEDIT is invoked.

TWO TYPES OF VERSIONS --- CRT AND MEMORY MAPPED

There are two primary versions of VEDIT - CRT versions and Memory Mapped versions. The CRT version supports practically every terminal on the market. The particular terminal VEDIT is to work with is selected from a large menu of terminals which appears when VEDIT is customized. The IBM Displaywriter is supported by the CRT version.

The memory mapped version supports most S-100 and Multibus display boards. Special memory mapped versions are also available for the TRS-80 Models I, II and 16. The memory mapped customization is a little more complex and requires the screen size in addition to the screen's address. The general purpose memory mapped version contains a file on disk which describes the patches necessary to implement bank select.

In addition to the primary versions, there are several specialized versions. One is for the IBM Personal Computer. This version is memory mapped, but skips some parameters, such as screen address which are fixed by the hardware. It also asks additional questions concerning specific features of the IBM PC, such as display attributes. Separate sheets describe this in more detail. There is also a version for the PIICEON V-100 and the TDL VDB, both of which are I/O mapped display boards.

HOW TO PERFORM CUSTOMIZATION

STEP 1 - ENTER COMMAND SEQUENCE FROM OPERATING SYSTEM

In order to customize the CRT version of VEDIT you will need the files "VDSETCRT.COM" and "CRT.TBL" on your work disk in addition to the appropriate ".SET" file. For the memory mapped version, you will need the file "VEDSET.COM" and the appropriate ".SET" file. The enclosed sheet "Description of Files" gives a further description of the files on the distribution disk.

To begin customization, first specify the VEDSET or VDSETCRT file (.COM or .CMD) then the .SET file, then the name of a file you wish the editor to be called. (VEDIT is a good choice.) If your disk has more than one ".SET" file, select the appropriate one from the separate sheet "Description of Files". There is no need to give the file extensions to these names. Remember to end the command line with a RETURN.

Assuming you wish to customize the Z80 CRT version, with a file name of VEDITZC.SET, the customized editor is to be called VEDIT and the files VEDITZC.SET and VDSETCRT.COM are on the currently logged in disk, the command to run VDSETCRT would be:

```
VDSETCRT VEDITZC VEDIT
```

A similar command for the 8080 Memory Mapped version would be:

```
VEDSET VEDIT8M VEDIT
```

The command for the TRS-80 Model II Pickles & Trout version is:

```
VEDSET VEDIT2P VEDIT
```

A running VEDIT (a VEDIT.COM file) may be customized as well. This allows some aspects of the customization to be changed without having to repeat the entire process. A typical command to do this would look as follows:

```
VEDSET OLDVEDIT.COM NEWVEDIT.COM
```

where "VEDSET" may also be VDSETCRT, "OLDVEDIT" is the VEDIT you want to change, and "NEWVEDIT" is the name of the new VEDIT.

If you receive a "Checksum Error", please see the second part of this section for an explanation.

STEP 1.5 - (CRT ONLY). CHOOSE YOUR CRT TERMINAL

VDSETCRT will display a menu of terminals from which you select the number corresponding to your terminal. The list is two screens long; type any key after looking over the first screen. Following the prompt enter the number corresponding to the terminal you are using.

In the rare case that your terminal does not appear on the menu, you have two choices. If you are technically inclined you can change the file "CRT.ASM" which contains the tables corresponding to each of the terminals. Two entries "Customer 1" and "Customer 2" are intended to be changed by the user. Alternately, contact us for support.

Technically inclined users may wish to read the file "VEDITCRT.DOC" for related information. Hazeltine and Intertube users should also read this file.

STEP 2 - LOOK OVER MAIN MENU TASKS

TASKS:

1). Perform all new keyboard layout
2). Add alternate keys to existing layout
3). Set special characters
4). Set ES and EP parameters
5). Set screen parameters
6). Set other parameters
7). Set signon message
8). Customization complete; return to operating system

Tasks (1) and (2) are used to determine the keyboard layout, task (7) sets the signon message and (8) writes the customized VEDIT out to disk. The remaining tasks change the various parameters. The prompts for many of these are followed by a number in parentheses, which is a suggested value. If you use this value you must type it in, there in NO DEFAULT value. Questions with a numeric answer also require a RETURN after the answer. To ignore input for a particular question, type either the RUBOUT (DELETE) key or a CTRL-U. After each task is performed, the program returns to the main menu. At this point another part of the customization can be performed or a previous step repeated if a mistake was made.

QUICK CUSTOMIZATION:

If you have the CRT version of VEDIT, the normal screen size of 24 by 80 and wish to bring VEDIT up quickly with the "Example Keyboard Layout", you need only to select your terminal from the CRT terminal menu and then immediately select task (8) in the menu to complete the initial customization. (The default memory size parameters will work well in any size system.) If you like, you can go ahead and do that now, and read the rest of this section later.

If you have a memory mapped system or a CRT terminal with a size other than 24 by 80, you will need to perform task (5) in the customization in order to bring VEDIT up. When you are ready to try out something other than the "Example Keyboard Layout" you can perform task (1).

If task (1) is not performed, the resulting editor will respond to the control codes in the "Example Keyboard Layout". Similarly, if tasks (3) through (6) are not performed, the editor will be setup with the parameters in the "Example Customization".

Note: If you have a Televideo, a Z19 or other terminal for which we supply a special keyboard layout sheet, you must perform task (1) to give you the keyboard layout. Just selecting the terminal will NOT give you the special layout.

STEP 3 - TASK 1: PERFORM ALL NEW KEYBOARD LAYOUT

ENTER ESCAPE MODE CHARACTER #1

If you choose to use escape sequences, or your keyboard produces escape sequences with special function keys, type the escape character, or the function key lead-in character, most commonly ESC. Else type RETURN, which will then also slip the remaining questions about escape characters.

ENTER ESCAPE MODE CHARACTER #2

A second escape mode character may also be specified, typically for other function keys. If not needed, type RETURN. (A "CTRL-A" for the Televideo 920C).

ENTER COMMON 2ND CHARACTER #1 IN ESCAPE SEQUENCE

Simply answer with a RETURN if you are not using escape sequences or are typing them in by hand. (A RETURN will skip the next

question.) However, some terminal's special function keys send 3 character escape sequences where the second character is always the same and should be ignored. In this case type in the second character. (A "?" for the Heath H19) (A capital letter "O" for the DEC VT-100)

ENTER COMMON 2ND CHARACTER #2 IN ESCAPE SEQUENCE

Some terminals, particularly ANSI standard ones, have a second character which should be ignored in the second character position. Type the character, or if there is no such second character, type RETURN. (A "[" for the DEC VT-100) (RETURN for H19, Televideo, etc)

UPPER/LOWER CASE ESCAPE SEQUENCES EQUIVALENT (Y/N) ?

If you answer NO, the editor will make a distinction between, for example, ESC H and ESC h. This is annoying if you hand type escape sequences and you should answer with a "Y". However, the function keys on terminals such as the Televideos send escape sequences which distinguish between upper and lower case. Here you would have to answer "N".

TYPE CONTROL CHARACTERS FOR ....

When prompted for each visual operation, you may press a special function key, a control character or enter an escape sequence. Disallowed characters are the normal displayable characters. Typing one of these will give an error and a reprompt. If you inadvertently attempt to use the same key code for a second operation, an error and a reprompt for the operation will be given. If you do not want to use a particular function, just type RETURN to ignore the function. Specifically, you will probably want to use either [SET INSERT MODE] and [RESET INSERT MODE] or [SWITCH INSERT MODE], but not all three functions. You probably won't use [RESTART], since the function is also available in command mode. Otherwise choose something for [RESTART] which you are very unlikely to hit by mistake. Don't confuse [TAB CURSOR] with the tab character, since it is a cursor movement operation. If you make a mistake, just type RETURN for the rest of the functions and perform this task again.

STEP 4 - TASK 2: ADD ALTERNATE KEYS TO EXISTING LAYOUT

Task (2) allows you to use alternate control codes for any of the editing functions. For example, your keyboard may have cursor keys which you have customized as the four basic cursor movements in VEDIT. However, out of habit you are still using

CTRL-S, CTRL-F, CTRL-E and CTRL-C to move the cursor. You can select task two to enter any such alternate control codes to use for any editing function. You must type the RETURN key for those functions you don't wish to invoke by an alternate control key.

Task (2) can also be used to specify the initial control code to use for an editing function if none was specified in task (1), i.e., you ignored the function by typing a RETURN for it. The functional difference between tasks (1) and (2), is that task (1) first clears out any existing keyboard layout, while task (2) builds on the existing layout.

STEPS 5 -> 10 - SET NON-KEYBOARD PARAMETERS:

Answer questions in decimal or hexadecimal as prompted, then hit RETURN. There are no default settings, so always enter a value. Type a CTRL-U or hit the DELETE (RUBOUT) key to repeat the question.

STEP 5 - TASK 3: SET SPECIAL CHARACTERS

3.1) HEX CODE FOR SCREEN CONTINUATION CHARACTER (2D)

This is the line continuation indicator used in Visual Mode in reserved column 0. Most common is a hyphen (Hex 2D) or reverse video hyphen (Hex AD).

3.2) HEX CODE FOR COMMAND ESCAPE CHARACTER (1B)

This is the command mode Escape character which should be the "ESC" or "ESCAPE" key, hex code of "1B", if your keyboard has it. If your keyboard doesn't have an ESC key, choose another control character, perhaps CTRL-Z, Hex code of 1A.

3.3) HEX CODE FOR COMMAND ITERATION LEFT BRACKET (5B)
3.4) HEX CODE FOR COMMAND ITERATION RIGHT BRACKET (5D)

The Command Iteration Brackets are those which delimit iteration macros --- groups of Command Mode commands. This manual represents these as "[" and "]", hex codes of 5B and 5D. If you may prefer to use "<" and ">", hex codes of 3C and 3E. Use either set, but it may help if your keyboard

produces one set without needing the SHIFT key.

3.5) (Memory Mapped Only)
HEX CODE FOR CURSOR CHARACTER (5F)

This is the character used as the blinking "underline" cursor. While normally the underline character (code 5F hex), some users, particularly those with a Sorcerer, may wish to try a hex code of "7F" which is commonly a solid block.

3.6) (Memory Mapped Only)
HEX CODE FOR SCREEN CLEAR CHARACTER (20)
HEX CODE FOR STATUS LINE CHARACTER (2D)
HEX CODE FOR TAB EXPAND CHARACTER (20)

VEDIT normally clears the screen with spaces (code 20 hex), uses a '-' (code 2D hex) on the status line and displays tab characters with spaces. These may be changed for special applications, or if your display requires other characters. For example, the Polytechnic VTI requires that Bit 7 be set for normal characters. Therefore, the character codes would be "A0", "AD" and "A0" respectively.

STEP 6 - TASK 4: SET ES SWITCHES AND EP PARAMETERS

This task selects the default values for these parameters. They can be changed while running VEDIT by using the ES and ET commands. All numeric values are in decimal.

4.1) EXPAND TAB WITH SPACES (0 = NO, 1= YES) (0)

Instead of inserting the tab character into the file, spaces to the next tab position are inserted when the [TAB CHARACTER] function is typed. This is useful if another program interacting with your file doesn't interpret tab characters; this function also takes extra disk space. Do not turn this switch on unless you need to.

4.2) AUTO-BUFFERING IN VISUAL MODE (0=NO, 1=YES) (1)

We suggest you enable auto-buffering, which allows VEDIT, in Visual Mode, to automatically read and write a file as needed.

4.3) BEGIN IN VISUAL MODE (0=NO, 1=YES) (1)

This determines whether VEDIT starts in Visual or Command Mode. We suggest you set this switch to "Yes".

4.4) POINT PAST TEXT REGISTER INSERT (0=NO, 1=YES) (1)

This determines whether the cursor (or Edit Pointer in Command Mode) will be positioned at the beginning or the end of text inserted from a text register. We suggest that you initially set this switch to "Yes". After some practice with the text registers you will know which way you prefer it.

4.5) IGNORE UPPER/LOWER CASE DISTINCTION IN SEARCH (0=NO, 1=YES) (1)

This determines whether the difference between upper and lower case letters is ignored. We suggest you set this to "Yes".

4.6) CLEAR SCREEN ON VISUAL EXIT (0=NO, 1=YES) (0)

This determines whether the screen is cleared when Visual Mode is exited to Command Mode. For most applications you will want to answer "No".

4.7) REVERSE UPPER & LOWER CASE (0=NO, 1=YES) (0)

This determines whether all letters typed on the keyboard will be reversed with regard to upper and lower case, i.e., upper case letters are converted to lower case and vice versa. Only in very unusual situations would you want to set this switch on, so set it off. For the TRS-80 Model I, you should set this switch on, since the keyboard reverses upper and lower case.

4.8) IGNORE SEARCH ERRORS (0=NO, 1= YES) (0)

This switch should normally be off. Otherwise there will be no message if a Find or Substitute is unsuccessful. This switch can be set with the ES command prior to executing some types of command macros.

4.9) USE EXPLICIT TEXT DELIMITERS (0=NO, 1=YES) (0)

This switch, if set ON, allows you to delimit each string in commands such as Substitute or Find with any character. The most commonly used ones are "/", ";", or ";", but any character may be used without specifying it beforehand.

We suggest turning this parameter off in the beginning because almost none of our examples use this feature. It may be set with the ES command before you begin issuing other commands.

4.10) (Memory Mapped Only)
CURSOR TYPE (0, 1, 2) (1)

This parameter determines the cursor type in memory mapped versions. The cursor types are 0=Blinking Underline, 1=Blinking Reverse Video Block, 2=Solid Reverse Video Block. Most users seem to prefer type "1", but you must use "0" if your display does not produce reverse video.

4.11) (Memory Mapped Only)
CURSOR BLINK RATE (10 - 100) (See Prompt)

This determines the memory mapped cursor's blink rate. Start with the value suggested by the VEDSET prompt. A smaller number causes the cursor to blink faster.

4.12) INDENT INCREMENT (1 --20, SUGGEST 4)

This determines the "Indent Increment". A value of 4 is common when structured programming languages are being used.

4.13) LOWER CASE CONVERT (0=NO, 1=YES, 2=CONDITIONAL) (0)

This parameter is useful for assembly language programs. If you choose "0", no conversion will occur. If you choose "1", all lower case keyboard character will be converted to upper case. If "2" is chosen, the answer to the next question will determine before which character lower to upper case conversion will occur. For example, Z80 assembler uses ";" as a comment delimiter. To the left of the ";" lower case letters are converted to upper case. To the right of the "; " in the comment field, no conversion is done.

4.14) DECIMAL CODE FOR CONDITIONAL CONVERSION CHARACTER (59)

This is the "Conditional Conversion" character used when the previous parameter is set to "2". A value of "59" decimal, makes the ";" the special conditional character.

4.15) LINE AND COLUMN DISPLAY (0=NONE, 1=LINE, 2=COLUMN, 3=BOTH) (3)

This determines whether the Visual Mode status line will display the line number and column position the cursor is on. It is usually useful to know both.

4.16) RIGHT MARGIN FOR WORD WRAP IN DECIMAL (0=OFF)

The Word Wrap column defines the right margin column. A value of 00 turns Word Wrap off. Words typed beyond the right margin will be wrapped to the next line. The right margin is also used for the [FORMAT PARAGRAPH] function. The right margin can be greater that the screen line length.

STEP 7 - TASK 5: SET SCREEN PARAMETERS

5.1) ENTER NUMBER OF SCREEN LINES IN DECIMAL (24 or 25)

Enter the number of lines on your CRT display. While most terminals have 24 lines, some have a 25th "Status Line". On some of these, it is possible for VEDIT to place its status line on the 25th line. These terminals are marked with a "*" following the terminal's name in the menu. To use the 25th line, answer this question with a "25". Note that the Intertec Intertube II must be specified as having 25 lines. The IBM Personal Computer also has 25 lines.

5.2) ENTER LINE MOVEMENT FOR PAGING IN DECIMAL (20)

Enter the number of screen lines you wish [PAGE UP] and [PAGE DOWN] to move through the text by. About 4/5 of the total number of screen lines is suggested, i.e., "12" for a 16 line display and "20" for a 24 line display.

5.3) ENTER TOP LINE FOR CURSOR IN DECIMAL (3)

This sets the top screen line the cursor can normally be on, before the screen will begin to scroll down. This, therefore, is the minimum number of lines you will always

see before the line you are editing.

5.4) ENTER BOTTOM LINE FOR CURSOR IN DECIMAL (20)

This is similar to the previous step, except that it sets the bottom line range for the cursor. This number must be greater than or equal to the "Top Line for Cursor" setting, and at most be one less than the "Number of Screen Lines", since the very bottom line is only used for status. "4" less than the number of screen lines is a good starting point.

5.5) (Memory Mapped Only)
ENTER SCREEN LINE LENGTH IN DECIMAL (80 or 64)

Enter the number of bytes per screen line your display has. This number must be in the range 20 -255. It will be 64 or 80 for most Memory Mapped Displays and therefore the same as the next question. However it will be greater for displays which have invisible attribute bytes at the end of each line. The MATROX display board is like this and requires a value of 128.

5.6) ENTER LENGTH OF DISPLAYED LINE IN DECIMAL (80 or 64)

This is the number of characters per line VEDIT will display. Normally you would want this value equal to the screen line length, usually 80 or 64. The TRS-80 Model II and 16, the IBM Personal Computer and the MATROX display board require a value of 80.

5.7) (Memory Mapped Only)
ENTER ADDRESS OF SCREEN IN HEXADECIMAL

Enter the memory address of the beginning of the video in hexadecimal and a RETURN. Many 16 x 64 boards have an address of CC00 hex. The TRS-80 Model II has an address of F800 hex.

5.8) (Memory Mapped Only)
ENTER NUMBER OF VIDEO BOARD INITIALIZATION BYTES

Enter "0" if your board requires no initialization. Otherwise enter a number between "1" and "5" for the number of "data byte"-"port address" pairs needed for initialization. Most memory mapped system need no initialization (including TRS-80 Models I, II and 16). (One exception is the Processor Technology VDM, which requires a

"00" output to port "C8" hex, and the SOL-20 a "00" output to port "FE" hex).

```
ENTER [RUBOUT] OR [CTRL-U] TO START PAIR OVER
ENTER DATA BYTE
ENTER PORT ADDRESS
```

The specified number of "data byte"-"port address" pairs is entered in hexadecimal with each number followed by RETURN. Typing CTRL-U or RUBOUT will reprompt with the "ENTER DATA BYTE" question for that pair.

STEP 8 - TASK 6: SET OTHER PARAMETERS

6.1) SIZE IN DECIMAL OF SPARE MEMORY FOR AUTO-READ IN BYTES

See the table below for a recommended value depending upon your memory size. The number must be in the range 1024 - 32768. Use RUBOUT or <CTRL-U> if you mistype the number.

MEMORY SIZE	SPARE MEMORY FOR (For 6.1)	VALUE FOR TRANSFER (For 6.2)
20K	2304	3
24K	3072	4
28K	4096	5
32K	4096	6
36K	5120	7
40K	6144	8
44K	6144	9
48K	7168	10
52K	7168	11
56K	8192	12
60K	8192	13
64K	8192	14

--- Minimum system size = 20K.
--- 1 K byte is a unit of 1024 bytes (1024 = 2 ** 10).
--- For CP/M systems, the memory size is the CP/M size, which should be on your CP/M disk label or displayed when you first boot.
--- Do not make the Spare Memory for Auto Read more than two times larger than the value given in the table or it may produce a non-operational editor. This value represents the number of bytes free in the text buffer AFTER a file

larger than available memory space is read. For example, in a 56K system the available memory is about 41K. If the table value of 8192 was chosen and a very large file edited, VEDIT would initially read in the first 33K of the file, leaving 8192 bytes free. This extra space is necessary for insertion of new material. Use the "U" command to verify actual free space.
See "Customization Notes" for more details.

6.2) SIZE IN DECIMAL OF FILE MOVE TRANSFERS IN K BYTES

Choose the value from column 3 of the above table which corresponds to your memory size. This parameter specifies the amount of the file read into the text buffer when auto-buffering is done. The number entered must be in the range 1 - 32.

6.3) DO YOU WISH TO USE DEFAULT TAB POSITIONS? (Y/N)

The default tab positions are set at every 8th position, for example, 9 17 25 41 49 57 65 73 81 89 etc. This is the most common tab setting; if you change the tabs, the change will apply to VEDIT only. Tab positions may be reset inside VEDIT using the ET command.

If you enter "N", this prompt is given:

ENTER UP TO 30 TAB POSITIONS IN DECIMAL

Enter the desired tab positions, separating the numbers with spaces or commas and following the last number with a RETURN. Don't be concerned if your input line goes off the right side of your terminal or screen. Note that you need no tab at position 1 and that the positions are counted starting from 1, not 0. You must also specify at least one tab position per screen line and the highest allowed position is 254. Entering a number outside of the range 1 - 254 will give an error and a reprompt of the question. If you make a mistake, type RUBOUT or <CTRL-U> to start the question over.

6.4) ENTER DECIMAL VALUE (4mhz = 76, 2mhz = 38)

Enter "76" if your processor speed is 4mhz, "38" if the speed is 2mhz. Interpolate for other processor speeds. This value is only used for CRT's which require time delays for some functions and is not critical. The maximum value is 255.

6.5) REVERSE VIDEO ON STATUS LINE (0 = NO, 1 = YES) (1)

If your CRT or video display board produces reverse video, answer "1". If you have a Sorcerer, TRS-80 Model I, or a CRT terminal which does not produce reverse video, answer "0".

STEP 9 - TASK 7: SET SIGNON MESSAGE

This message will appear briefly whenever you invoke VEDIT. It can be used to help you identify how the particular VEDIT was customized. The message may be up to 64 characters long. An example message might be:

Bob's Televideo 920C, Word Wrap = 70.

STEP 10 - TASK 8: CUSTOMIZATION COMPLETE; RETURN TO OPERATING SYSTEM

This writes the customized VEDIT out to disk.

Customization Notes

This section describes some aspects of the customization in more detail. You do not need read this section in order to get VEDIT up and running. However, once you are more familiar with VEDIT, you will probably want to gain a better understanding of the customization in order to create a more "personalized" version of VEDIT.

VEDIT Checksum

To help insure that your distribution diskette is intact, the customization performs a checksum on the VEDIT file being customized. If there is a fault, a warning error message is given. If you encounter this error make sure that you have copied the files from the distribution diskette properly. If all else fails, try running the customization from the distribution diskette. If this still results in the error, please contact us for an exchange diskette. If you patch the VEDIT file, this error will result. In this case it can be ignored, and the new VEDIT file will contain a new checksum so that the error will not occur again unless the file becomes modified again.

Keyboard Layout

Determining the desired keyboard layout for the cursor movement and function keys is the first task of the customization. Since it could be a difficult task, several example keyboard layouts are enclosed to help out the new user. The best layout will depend to some extent upon your keyboard, especially if you have one with extra keys which produce control codes. If extra keys are available, you may want to allocate them to the most used visual operations such as the cursor movements. The more extra keys you have, the easier it becomes to remember the layout.

If and when you decide to try out your own layout, you will want to avoid placing the keys you least want to hit by accident, such as [Erase End Of Line] or [Home], right next to the cursor movement keys. In the event that you have no or few special keys, most visual operations will involve holding the CONTROL key while you type a letter, or using escape sequences. In this case, the layout may be tight and difficult to organize. One strategy is to use mnemonic letters, such as CTRL-D for [DELETE] and CTRL-U for [CURSOR UP], etc. Another is to arrange the keys in some geometric manner, such as the cursor movement keys on one side of the keyboard and the visual function keys on the other side. You can also simplify the layout by using at least a few escape sequences, especially for functions you do not use often, or don't want to hit by accident. Trying out some combinations on paper is probably the easiest way to accomplish the

layout task.

Besides responding to the customary control characters, VEDIT also handles multi character escape sequences. These may be user typed, or may result from pressing a special function key. For example, instead of typing the single character CONTROL-Q, the user may type two characters, i.e. ESC and Q, to perform a visual operation. All escape sequences begin with one of two user defined escape characters (sometimes called Lead-in characters). While the ESC is a common key to use as an escape character, any other ASCII character may be used as the escape character, even displayable ones like "@". The special function keys on some keyboards, like the Heath H19, Televideo 920C and IBM 3101 also send multi character escape sequences. Some terminals, like the IBM 3101, also send a Carriage Return at the end of escape sequences. The keyboard customization detects this automatically and the user need not be concerned with it.

When performing the keyboard customization, it asks the question: "Ignore upper/lower case difference in escape sequences?" If you answer NO to this question, the editor will made a distinction between, for example, "ESC-H" and "ESC-h". Therefore, if you entered the escape sequence with a lower case "h" during customization, the editor would not respond to the escape sequence with an upper case "H". This is annoying if you hand type most of the escape sequences, since at times you may have the SHIFT or a CAPS-LOCK depressed. You would therefore want to answer the question with a YES. However, the function keys on some terminals, such as the Televideos, send escape sequences which distinguish between upper and lower case letters. In this case you will want to answer the question with NO. If you find that you have made a mistake with this question, you can skip performing the entire keyboard customization again, by performing task (2) in the customization, answering this and the other three questions pertaining to escape sequences correctly and simply typing a RETURN for all of the function prompts.

When laying out the keyboard, you may therefore use any combination of control characters, special function keys and escape sequences for the visual operations. Some users will prefer to use function keys and control characters for the most used visual operations, and escape sequences for the less used operations. If escape sequences are used, a key like ESC or FORM FEED is suggested for the escape mode character. Any other character may then follow, including numbers, control characters or even another escape character. Many keyboards have a numeric pad and these numbers can be used in escape sequences. For example, use ESC - 8 for [CURSOR UP], ESC - 2 for [CURSOR DOWN], ESC - 4 for [CURSOR LEFT] and so on. In this case you may wish to attach descriptive labels on top of the numeric keys. An Escape and Control character combination would be a good choice for operations you don't want to hit by mistake, like [HOME], [ZEND] or [RESTART EDITOR]. You may use an escape sequence consisting of two escape characters in a row. In fact, if ESC is the escape character, then "ESC - ESC" is the suggested sequence for the

function [VISUAL ESCAPE]. In the unusual case that a displayable character like "@" is used as the escape character, a "@ - @" cannot be used for a visual operation, since in this case, "@ - @" will be treated by VEDIT as the normal "@" character.

While all of this is complicated enough already, there are a few pitfalls to avoid too. (You are well advised to use one of the example keyboard layouts at first.) The only key which is predefined is the RETURN or CR key which is also CTRL-M and cannot be used for any visual operation. The special function keys on some keyboards send a code identical to a control character. You may therefore unintentionally attempt to use the same control code for two visual operations. In this case, VEDSET or VDSETCRT will give an error message and request a new key for that function. Some keyboards have special function keys which send a character with data bit 7 set (sometimes called the parity bit). These work properly since the VEDIT programs decode all 8 bits. (Technical note: An escape sequence treats the second character as having Bit 7 set. The escape mode characters themselves must not have Bit 7 set.)

A Word About Keyboards

With the simplest keyboards, each visual operation will have to be activated by holding the CONTROL key and typing some letter or using an escape sequence. Moving up, keyboards will have keys for Backspace, Tab and Line Feed, which can be used to perform the described function. Some keyboards with a numeric pad can send control codes by holding the SHIFT or CONTROL key and typing one of the pad keys. Numeric pad keys can always be used as part of escape sequences. The pad can then be used for most of the visual operations. In some cases, the keyboard will have many special keys, which send a control code just by typing one of them. In the ideal case, these control codes will be sent with the highest data bit set. (This is Bit 8 and is often called the parity bit. The ASCII standard code does not use Bit 8 and even a "Full ASCII" keyboard will send nothing on Bit 8 or else parity information). Some very special keyboards, usually ones with 70-100 keys on them, use Bit 8 to decode all those keys. Since VEDIT and VEDSET decode all 8 data lines from the keyboard, these fancy keyboards can be used to their full advantage.

Screen Size Parameters

VEDIT can be customized for any screen size up to 70 lines by 200 columns. To set these parameters you need to know the number of lines and the number of characters per line that your CRT terminal or video display board produces. 16 x 64 and 24 x 80 are the most common values. You also have the choice of how many columns on a line are

actually used. You want to use all of them, unless you have a special application or unusual hardware.

For the memory mapped versions, you also need to know the beginning address of the display board in memory in hexadecimal and whether it requires any data bytes output to a port to initialize it. For example, many 16 x 64 boards have an address of CC00 hex. Most of these 16 x 64 boards do not need any initialization, one exception being the Processor Technology VDM board, which should have a 00 output to Port C8 hex. (The SOL-20 requires a 00 output to Port FE hex).

Memory Size Dependent Parameters

The first parameter "Spare Memory for Auto-Read" determines how many bytes of memory are free after VEDIT does an auto-read (such as following an EB command). This size must be specified between 1024 and 32768. A reasonable size is about 1/4 of the size of the text buffer for small systems and a little less for large systems. Choosing a 1K (1024 byte) multiple makes the disk read/write work a little bit faster.

In particular, do not make this value more than 2 times larger than the value in the table, or you may produce a non-operational editor. This value is NOT the amount of memory VEDIT will use for the text buffers, since VEDIT always sizes memory and uses all that is available. Rather, this value is the number of bytes that is free in the text buffer after a file is read which is larger than the available memory space. For example, in a 56K system the available memory is about 39K. If the table value of "8192" was used, and a very large file edited, VEDIT would initially read in only the first 33K of the file, leaving "8192" bytes free. This can be verified with the "U" command.

The second parameter "Size of File Transfers" specifies the size of file transfers during auto-buffering and for the 'N' command. For normal use, a value about 1/3 the size of the text buffer is good. (Specifying a value larger than one half the maximum text buffer size may create a non-working version of VEDIT.) When auto-buffering is initiated, an attempt is made to append this number of K bytes to the end of the text. If there is insufficient memory space for appending this many bytes, this many bytes are written from the beginning of the text buffer to the output file. An auto-read is then performed which reads in the rest of the input file, or until the memory is filled to within the number of spare bytes specified by "Spare Memory for Auto-Read".

'n' denotes a positive number. (# represents 32767)
'm' denotes a number which may be negative to denote backwards in the file.
'r' denotes a digit "0 - 9" specifying a text register.

'string','s1' and 's2' denote text strings.

'file' is a file name in the normal CP/M (MSDOS) format with optional drive and extension specified.

nA	Append 'n' lines from the input file. (0A)
B	Move the edit pointer to text beginning.
mC	Move the edit pointer by 'm' positions.
mD	Delete 'm' characters from the text.
E	First letter of extended two letter commands.
nFstring<ESC>	Search for 'n'th occurrence of 'string'.
Gr	Insert the contents of text register 'r'.
Itext<ESC>	Insert the 'text' into the text buffer.
mK	Kill 'm' lines.
mL	Move the edit pointer by 'm' lines.
Mr	Execute text register 'r' as a command macro.
nNstring<ESC>	Search for 'n'th occurrence of 'string' in file.
mPr	Put 'm' lines of text into text register 'r'.
Ss1<ESC>s2<ESC>	Search for and change 's1' to 's2'.
mT	Type 'm' lines.
U	Print # of unused, used and text register bytes.
V	Go into visual mode.
nW	Write 'n' lines to the output file. (0W)
Z	Move edit pointer to end of text.

SPECIAL CHARACTERS

|	Search wildcard character. Each "|" will match any character in the text being searched.
<CTRL-Q>	Literal Character. Next char. is taken literally.
@	Precedes F, I, N, S command to indicate explicit terminator.
:	Precedes F, N, S command to suppress search error message.

VEDIT prints a message (on the CP/M console device) when the user should be notified of an unusual or special condition. All messages are descriptive, and the user should not normally have to refer to this appendix in order to understand the message or error. The messages fall into three categories: fatal errors, non-fatal errors and other messages. Fatal errors result in an abort of the disk operation being performed and a return to command mode if possible, else a return to CP/M. These are caused by certain disk errors described below. The non-fatal errors usually just signify that a typo was made or that some small detail was overlooked. These only result in a message and the user can try again.

FATAL ERRORS

NO DISK SPACE — The disk became full before the entire output file was written. As much of the output file as possible was written. Refer to the section on disk write error recovery.

CLOSE ERROR — The output file could not be closed. This is a very unusual condition, but may occur if the disk becomes write protected.

READ ERROR — An error occurred reading a file. This error should never occur, since CP/M itself normally gives an error if there was a problem reading the disk.

NO DIR SPACE — There was no directory space left for the output file. Refer to the section on disk write error recovery.

NON-FATAL ERRORS

INVALID COMMAND — The specified letter is not a command.

CANNOT FIND... — The specified string could not be found. This is the normal return for iteration macros which search for all occurrences of a string.

NESTING ERROR — You cannot nest macros deeper than 8 levels.

BAD PARAMETER — Something was specified wrong with your "EI", "EP", "ES" or "ET" command.

NO INPUT FILE — There is no input file open for doing a read or append.

NO OUTPUT FILE — There is no output file open for doing a write, a close or an exit with the "EX" command. If you have already written out the text buffer and closed the output file, exit with the "EQ" command.

CANNOT OPEN TWO — You cannot have two output files open and there is already one open. Also given if an output file is open at the time of an "EC" command. Perhaps you want to close it with the "EF" command.

BAD FILE NAME — The file name you gave does not follow the CP/M conventions.

FILE NOT FOUND — The file you wanted to open for input does not exist. Maybe you specified the wrong drive.

OTHER MESSAGES

NEW FILE — The file specified with the EB command or with the invocation of VEDIT did not exist on disk and a new file has been created. If you typed the wrong file name, you may want to start over by issuing the "EQ" command.

BREAK — The command execution was stopped because insufficient memory space remained to complete the command (I, S, G, P and EG). For the "I", "S" and "EG" commands, as much text as possible was inserted. For the "G" and "P" commands, no text at all was copied or inserted. The message is also printed when command execution is stopped because you typed [CTRL-C] on the keyboard in command mode.

QUIT (Y/N)? — This is the normal prompt following the "EQ" command. Type "Y" or "y" if you really want to quit and exit to CP/M, otherwise type anything else.

INSERT NEW DISK AND TYPE [RETURN]
This is the normal prompt for inserting a new disk with the "EC" command.

We are interested in hearing from users about any changes or additions they would like to see in VEDIT, or even just information about their application. We are also interested in suggestions regarding this manual. Each suggestion will receive personal attention and helpful suggestions have a good chance of being incorporated in future releases, since we are continuously expanding the features of VEDIT.

Currently we know of the following limitations to VEDIT.

1.) Lines longer than 258 characters, not including the CR,LF are not handled well in visual mode. When the cursor is on such a line only the first 258 characters will be displayed. The line may be broken into smaller lines by deleting two characters with the [Back Space], typing [RETURN] to split the line in two and typing in the two deleted characters again. Alternately, enter command mode and give the command "I<CR>$$".

2.) The text being edited can contain characters which have their high order bit (Bit 7) set. These are displayed unmodified in the memory mapped version, which is often suitable for foreign language and other special characters. The CRT versions will, however, attempt to display them in reverse video, which may or may not produce the desired result. The character with a value of FF hex (255 decimal) should not be used, because VEDIT will compress it out when performing the "S" command.

EXTENDED COMMANDS

EA		Restart the editor. (EX and EB).	
EBfile		Open "file" for Read & Write, perform an auto-read.	
EC		Change disks for reading or write error recovery.	
EDfile<ESC>		Delete (erase) the file "file" from the disk.	
EF		Close the current output file.	
EGfile[line range]		Insert the specified line number range of the file "file" into the text buffer at the edit position.	
nEI		Insert the character whose decimal value is "n".	
mEO		Send 'm' lines to the line printer. (0EO)	
EP n m		Change the value of parameter "n" to "m".	
	1	Cursor type	(0, 1 or 2)
	2	Cursor blink rate	(10 - 100)
	3	Indent Increment	(1 - 20)
	4	Lower case convert	(0, 1 or 2)
	5	Conditional convert character	(32 - 126)
	6	Display line and column number	(0, 1, 2 or 3)
	7	Word Wrap column (0 = Off)	(0 - 255)
EQ		Quit the current edit session.	
ERfile		Open the file "file" for input.	
ES n m		Change the value of switch "n" to "m".	
	1	Expand Tab with spaces	(0=NO 1=YES)
	2	Auto buffering in visual mode	(0=NO 1=YES)
	3	Start in visual mode	(0=NO 1=YES)
	4	Point past text reg. insert	(0=NO 1=YES)
	5	Ignore UC/LC distinction in search	(0=NO 1=YES)
	6	Clear screen on visual exit	(0=NO 1=YES)
	7	Reverse Upper and Lower case	(0=NO 1=YES)
	8	Suppress search errors	(0=NO 1=YES)
	9	Explicit string terminators	(0=NO 1=YES)
ET		Set new tab positions.	
EV		Print the VEDIT version number.	
EWfile		Open the file "file" for output. Create Backup.	
EX		Normal exit back to CP/M after writing output file.	

TEXT REGISTER COMMANDS

RDr	Dump contents of register 'r' on console.
RLrfile	Load register 'r' from file 'file'.
RSrfile	Save contents of register 'r' in file 'file'.
RTr	Type contents of register 'r' on console.
RU	Display size of each text register. br

CompuView Products Inc.

VEDIT

DESCRIPTION OF FILES ON DISK

The following is a brief description of the files currently supplied on diskette. The files actually supplied on your diskette depend upon which version and package you purchased. You will have to perform the customization process, described in the manual, to produce a runnable version of VEDIT.

VDSETCRT.COM	The program used to perform the customization for the CRT versions. The manual describes the use of this program and the "VEDITZC.SET" or "VEDIT8C.SET" files below.
VEDSET.COM	The program used to perform the customization for the memory mapped versions. Use with the "VEDITZC.SET" or "VEDIT8C.SET" files below.
VEDITZC.SET	File for producing the Z80 CRT version.
VEDIT8C.SET	File for producing the 8080 CRT version.
VEDITZM.SET	File for producing the Z80 Memory mapped version.
VEDIT8M.SET	File for producing the 8080 Memory mapped version.
Note:	The ".SET" files with a "L" as the last character of the file name allow up to 70 screen lines, instead of 33 lines for the normal versions.

1955 Pauline Blvd., Suite 200 Ann Arbor, Michigan 48103 Telephone (313) 996-1299

EXAMPLE KEYBOARD LAYOUT FOR THE TELEVIDEO 920C

This is an example keyboard layout which uses the special keys on the Televideo 920C terminal.

"ESCAPE MODE CHARACTER #1"	[ESC]	
"ESCAPE MODE CHARACTER #2"	CTRL-A B	Used by special function keys.
"COMMON 2ND CHARACTER #1..."	NOT USED	Type [RETURN]
"UPPER/LOWER CASE ESCAPE..."	0	(1 = YES, 0 = NO)

[HOME]	ESC - H	
[ZEND]	ESC - Z	
[CURSOR UP]	[Up Arrow]	
[CURSOR DOWN]	[Down Arrow]	
[CURSOR RIGHT]	[Right Arrow]	
[CURSOR LEFT]	[Left Arrow]	
[BACK TAB]	[F5] [CTRL T]	
[TAB CURSOR]	[F6] F6=F35	Useful for fast cursor movement.
[ZIP]	[F7] F5 = F35	
[NEXT LINE]	[F4] HOME	
[PREVIOUS WORD]	[CTRL-B] P	
[NEXT WORD]	[CTRL-F] N	
[PREVIOUS PARAGRAPH]	ESC - A B	
[NEXT PARAGRAPH]	ESC - V A	
[PAGE UP]	[CTRL-R]	
[PAGE DOWN]	[CTRL-V] F3	
[BACKSPACE]	[F8] BS	
[DELETE]	[DEL]	
[ERASE TO END OF LINE]	[F10] F4=F15	Also called [EREOL] in manual.
[ERASE LINE]	[CTRL-X]	
[DEL PREVIOUS WORD]	[CTRL-S]	
[DEL NEXT WORD]	[CTRL-G]	
[UNDO]	[F11] F2	
[TAB CHARACTER]	TAB	
[NEXT CHAR LITERAL]	ESC - Q L	
[SET INSERT MODE]	NOT USED	Type [RETURN]
[RESET INSERT MODE]	NOT USED	Type [RETURN]
[SWITCH INSERT MODE]	[F3] F1	
[INDENT]	[F2] CTRL D	
[UNDENT]	[F1] CTRL O	
[COPY TO TEXT REGISTER]	ESC - X C	
[MOVE TO TEXT REGISTER]	ESC - X M	
[INSERT TEXT REGISTER]	ESC - X I	
[PRINT TEXT]	ESC - P	
[SET TEXT MARKER]	ESC - X S	
[GOTO TEXT MARKER]	ESC - X G	
[FORMAT PARAGRAPH]	ESC - X F	
[VISUAL ESCAPE]	ESC - ESC	
[VISUAL EXIT]	[CTRL-E]	Used to exit to command mode.
[RESTART EDITOR]	NOT USED	Use "EA" Command.

Notes:

1.) The [RETURN] key is not changeable, and since it is the same as [CTRL-M], the [CTRL-M] cannot be used for any other operation.

2.) The cursor keys produce the codes CTRL-H, CTRL-J, CTRL-K and CTRL-L, these codes may therefore not be used for a second function.

EXAMPLE KEYBOARD LAYOUT

This is an example keyboard layout which assumes that there are no special keys available. Refer to Appendix A of the manual for a method of using any numeric keypad.

"ESCAPE MODE CHARACTER #1"	[ESC]	
"ESCAPE MODE CHARACTER #2"	NOT USED	Type [RETURN]
"COMMON 2ND CHARACTER #1..."	NOT USED	Type [RETURN]
"UPPER/LOWER CASE ESCAPE..."	1	(1 = YES, 0 = NO)

[HOME]	ESC - H	
[ZEND]	ESC - Z	
[CURSOR UP]	[CTRL-E]	
[CURSOR DOWN]	[CTRL-C]	
[CURSOR RIGHT]	[CTRL-F]	
[CURSOR LEFT]	[CTRL-S]	
[BACK TAB]	[CTRL-A]	
[TAB CURSOR]	[CTRL-R]	Useful for fast cursor movement.
[ZIP]	[CTRL-G]	
[NEXT LINE]	[CTRL-Z]	
[PREVIOUS WORD]	[CTRL-K]	
[NEXT WORD]	[CTRL-L]	
[PREVIOUS PARAGRAPH]	ESC - W	
[NEXT PARAGRAPH]	ESC - X	
[PAGE UP]	[CTRL-W]	
[PAGE DOWN]	[CTRL-X]	
[BACKSPACE]	[CTRL-H]	Or use BACK SPACE key.
[DELETE]	[CTRL-B]	Or use DEL or RUBOUT.
[ERASE TO END OF LINE]	[CTRL-N]	Also called [EREOL] in manual.
[ERASE LINE]	ESC - N	
[DEL PREVIOUS WORD]	ESC - K	
[DEL NEXT WORD]	ESC - L	
[UNDO]	[CTRL-J]	
[TAB CHARACTER]	[CTRL-I]	Or use TAB key.
[NEXT CHAR LITERAL]	ESC - Q	
[SET INSERT MODE]	NOT USED	Type [RETURN]
[RESET INSERT MODE]	NOT USED	Type [RETURN]
[SWITCH INSERT MODE]	[CTRL-V]	
[INDENT]	[CTRL-P]	
[UNDENT]	[CTRL-Q]	
[COPY TO TEXT REGISTER]	ESC - C	
[MOVE TO TEXT REGISTER]	ESC - M	
[INSERT TEXT REGISTER]	ESC - I	
[PRINT TEXT]	ESC - P	
[SET TEXT MARKER]	ESC - S	
[GOTO TEXT MARKER]	ESC - G	
[FORMAT PARAGRAPH]	ESC - F	
[VISUAL ESCAPE]	ESC - ESC	
[VISUAL EXIT]	ESC - E	Used to exit to command mode.
[RESTART EDITOR]	NOT USED	Use "EA" Command.

Note:

1.) The [RETURN] key is not changeable, and since it is the same as [CTRL-M], the [CTRL-M] cannot be used for any other function.

EXAMPLE CRT CUSTOMIZATION

A typical CRT customization session is listed below. Since the enclosed keyboard layouts give examples for Task 1, this session performs only Tasks 3 through 6. The two values depending upon memory size are based on a 40K system, however, they may be used for larger size systems too. For clarity sake, each reply below is preceded by "--", which does not appear is the actual customization. Each numerical reply must be followed by the [RETURN] key.

```
3.)  HEX CODE FOR SCREEN CONTINUATION CHARACTER   (2D) -- 2D
     HEX CODE FOR COMMAND ESCAPE CHARACTER        (1B) -- 1B
     HEX CODE FOR COMMAND ITERATION LEFT BRACKET  (5B) -- 5B
     HEX CODE FOR COMMAND ITERATION RIGHT BRACKET (5D) -- 5D

4.)  EXPAND TAB WITH SPACES          (0=NO, 1=YES) -- 0
     AUTO-BUFFERING IN VISUAL MODE   (0=NO, 1=YES) -- 1
     BEGIN IN VISUAL MODE            (0=NO, 1=YES) -- 1
     POINT PAST TEXT REG. INSERT     (0=NO, 1=YES) -- 1
     IGNORE UPPER/LOWER CASE DISTINCTION IN SEARCH (0=NO, 1=YES) -- 1
     CLEAR SCREEN ON VISUAL EXIT     (0=NO, 1=YES) -- 0
     REVERSE UPPER AND LOWER CASE    (0=NO, 1=YES) -- 0
     IGNORE SEARCH ERRORS            (0=NO, 1=YES) -- 0
     EXPLICIT STRING TERMINATORS     (0=NO, 1=YES) -- 0

     INDENT INCREMENT  (1 - 20, SUGGEST 4) -- 4
     LOWER CASE CONVERT (0=NO, 1=YES, 2=CONDITIONAL) -- 0
     DECIMAL CODE FOR CONDITIONAL CONVERSION CHAR. (59) -- 59
     LINE AND COLUMN DISPLAY (0=NONE, 1=LINE, 2=COLUMN, 3=BOTH) -- 3
     RIGHT MARGIN FOR WORD WRAP IN DECIMAL (0=OFF) -- 0

5.)  ENTER NUMBER OF SCREEN LINES IN DECIMAL -- 24
     ENTER LINE MOVEMENT FOR PAGING IN DECIMAL -- 20
     ENTER TOP LINE FOR CURSOR IN DECIMAL -- 3
     ENTER BOTTOM LINE FOR CURSOR IN DECIMAL -- 20
     ENTER LENGTH OF DISPLAYED LINE IN DECIMAL -- 80

6.)  SIZE IN DECIMAL OF SPARE MEMORY FOR AUTO READ -- 6144
     SIZE IN DECIMAL OF FILE MOVE TRANSFERS IN K BYTES -- 8

     DO YOU WISH TO USE THE DEFAULT TAB POSITIONS? (Y OR N) -- Y

     ENTER DECIMAL VALUE  (4MHZ = 76, 2MHZ = 38) -- 76

     REVERSE VIDEO ON STATUS LINE  (0=NO, 1=YES) -- 1
```

EXAMPLE KEYBOARD LAYOUT FOR THE H19

While VEDIT automatically puts the keypad into Shifted Mode, VDSETCRT does not, and for the customization you should enter the Shifted Mode by going into Local Mode and typing ESC and 't'.

"ESCAPE MODE CHARACTER #1"	[ESC]	
"ESCAPE MODE CHARACTER #2"	NOT USED	Type [RETURN]
"COMMON 2ND CHARACTER #1..."	?	
"COMMON 2ND CHARACTER #2..."	NOT USED	Type [RETURN]
"UPPER/LOWER CASE ESCAPE..."	0	(1 = YES, 0 = NO)

[HOME]	[CTRL-Q]	
[ZEND]	[CTRL-Z]	
[CURSOR UP]	[Up Arrow]	
[CURSOR DOWN]	[Down Arrow]	
[CURSOR RIGHT]	[Right Arrow]	
[CURSOR LEFT]	[Left Arrow]	
[BACK TAB]	[BLUE]	
[TAB CURSOR]	[RED]	Useful for fast cursor movement.
[ZIP]	[WHITE]	
[NEXT LINE]	[F5]	
[PREVIOUS WORD]	[CTRL-D]	
[NEXT WORD]	[CTRL-F]	
[PREVIOUS PARAGRAPH]	[CTRL-T]	
[NEXT PARAGRAPH]	[CTRL-B]	
[PAGE UP]	[IC]	
[PAGE DOWN]	[IL]	
[BACKSPACE]	[BACK SPACE]	
[DELETE]	[DC]	
[ERASE TO END OF LINE]	[DL]	Also called [EREOL] in manual.
[ERASE LINE]	[CTRL-X]	
[DEL PREVIOUS WORD]	[CTRL-S]	
[DEL NEXT WORD]	[CTRL-G]	
[UNDO]	[F4]	
[TAB CHARACTER]	[TAB]	
[NEXT CHAR LITERAL]	ESC - Q	
[SET INSERT MODE]	NOT USED	Type [RETURN]
[RESET INSERT MODE]	NOT USED	Type [RETURN]
[SWITCH INSERT MODE]	[F3]	
[INDENT]	[F2]	
[UNDENT]	[F1]	
[COPY TO TEXT REGISTER]	[CTRL-J]	
[MOVE TO TEXT REGISTER]	[CTRL-K]	
[INSERT TEXT REGISTER]	[CTRL-L]	
[PRINT TEXT]	[CTRL-P]	
[SET TEXT MARKER]	ESC - S	
[GOTO TEXT MARKER]	ESC - G	
[FORMAT PARAGRAPH]	ESC - F	
[VISUAL ESCAPE]	ESC - ESC	
[VISUAL EXIT]	[CTRL-E]	Used to exit to command mode.
[RESTART EDITOR]	NOT USED	Use "EA" Command.

Note: You may experience extra characters appearing when using the cursor UP and DOWN keys, especially in conjunction with the REPEAT key. This is caused by the function keys sending their multi character codes at 9600 Baud, which is too fast for any non-interrupt driven software. This is best solved by making your system interrupt driven. If this is not possible, implement the cursor movements with control characters and use the keypad for other functions.

CompuView Products Inc.

SOFTWARE UPDATE OPTION

FOR VEDIT VERSIONS 1.14, 1.36, 1.66

CompuView Products, Inc. offers you a software update option for the VEDIT software package you have just purchased. We recognize that not many companies in the micro computer field offer a software update service, but we ask that you consider ours. While this update service is not free, it will give most users benefits which will more than offset its cost. Please note that you need not purchase this option to receive support to correct any problems in our software, in the form of technical assistance and any necessary "patches". All licensed users receive such support.

Why should you purchase the software support?

The software support option keeps you up to date with the new releases of VEDIT, and also gives you a significant discount on one of our other products. Specifically, the software update service gives you the following benefits:

1.) You will receive two new releases of VEDIT (for the same configurations as you purchased) on disk. The new releases will include new features, enhancements and support for more hardware configurations. The new releases will also include update pages to the manual (often completely new manuals). Over the past 2 1/2 years we have had a major new release about every six months and you should expect to receive the releases at this interval. Therefore, the first new release will be sent to you between 2 and 6 months and the second release between 6 and 10 months following your original purchase of VEDIT. (Please see the restrictions below.)

2.) You will receive a $25 discount on one other product (list priced $75 or more) you buy from us within 2 years of the original purchase.

Restrictions:

A) You must be a licensed user of VEDIT.

B) You should purchase the software update option within 60 days of the original purchase of VEDIT. After 60 days you can still purchase the software update option, but will not be entitled to the $25 discount on one other product. After 90 days, there may be additional charges.

1955 Pauline Blvd., Suite 200 Ann Arbor, Michigan 48103 Telephone (313) 996-1299

C) The new releases only apply to VEDIT - Full Screen Editor, and not to any other software products which we may market under the name VEDIT. The new releases will be for the same configurations as you originally purchased, i.e. Z80 Memory Mapped version, or 8080 CRT version, etc.

Note: All licensed users may purchase additional versions of VEDIT for different hardware configurations at a price of generally between $20 and $50 plus media. (8080 and Z80 versions for $30 extra, Memory Mapped and CRT versions for $30 extra, 8086 for $100 extra.)

We encourage all purchasers of this update option to send us a list of enhancements they would like to see incorporated. Here are some of the new features we are currently planning on adding to VEDIT (Note, we cannot guarantee when these features will be available) :

1.) A split screen display with a visual display of the file in the upper part, and allowing you to enter command mode commands in the lower part.

2.) Easy to use visual mode search and replace functions. This would include conditional replace. You would decide from visual mode whether or not the replace should take place.

3.) More extensive TECO like pattern matching capabilities, conditional branching, expression evaluation, etc.

4.) Repeat key in visual mode, allowing visual functions or displayable characters to be repeated.

5.) Reverse disk I/O to automatically move backwards in files too large to fit in memory.

6.) Interruptable screen updating. This will allow functions which rewrite a significant part to the screen to interrupt, saving screen writing time.

7.) 8088 / 8086 versions - allow up to 1 MByte of memory to be used for the text, text registers, etc.

Ordering the Software Update Option: The cost of this option is $45 plus a shipping charge of $5 in the US, $8 in Canada and $12 everywhere else. Please send your remittance (check, money order or charge card number and expiration date) within 60 days of your purchase of VEDIT. We will immediately send you a receipt acknowledging your purchase of the option, and then the new releases according to the schedule described above.

CompuView Products, Inc.
Suite 200
1955 Pauline Blvd.
Ann Arbor, MI 48103

CompuView Products Inc.

PLEASE READ THIS FIRST

Before you get started, we ask that you read these notes.

1.) Please read the enclosed Software License Agreement before you break the seal on the diskette.

2.) Please be sure that this package contains the manual and separate sheets entitled "Notes on the Software License Agreement", "Software License Agreement", "Example Keyboard Layout", "Example Customization" and "Software Update Option". The CRT version should also have example keyboard layouts for the Televideo 920C and the Heath H19. The TRS-80 Model I and II versions should each contain three additional sheets, and the SuperBrain and the Apple versions one additional sheet.

3.) Please read the enclosed Software Support Option to see if this is of interest to you. All licensed users receive assistance and support should they have any problems with our software. Additionally, users purchasing the Software Support Option receive enhanced versions of the software for a period of a year.

4.) Please make a copy of the supplied diskette before running the programs on it. You can use your "PIP" program for this purpose and copy each file to another diskette.

5.) Before you can begin using VEDIT, you will have to first perform the customization in order to configure the editor to your system. To do this, you should refer to the example Keyboard Layouts, the "Example Customization", and the step by step instructions in Appendix A of the manual. The IBM Personal Computer and IBM Displaywriter versions come with a "ready to run" VEDIT, for which the initial customization is not necessary.

6.) Should you have any problems installing or using the software, please contact us. You will help both yourself and other users by informing us of even small difficulties you may have with the software or documentation.

1955 Pauline Blvd., Suite 200 Ann Arbor, Michigan 48103 Telephone (313) 996-1299