VEDIT

CP/M
CUSTOMIZABLE
FULL SCREEN EDITOR

CompuView Products Inc.

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

4.) Appendices

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.

VEDIT also provides a very flexible and powerful command mode for performing search and substitute operations and repetitive editing operations using iteration 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 recovery 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 the user create versions of VEDIT which are best suitable to their hardware, keyboard, needs and desires.

While the typical VEDIT user will spend 99% of their time in the visual mode and only 1% in the commmand mode, this manual deals primarily with the command mode. This is the proper balance for the manual however, because the visual mode is exceptionally easy to learn to use. A little experimentation is the best teacher. The command mode is more difficult to learn and the manual, therefore, describes this mode in detail with many examples. A nice thing about VEDIT is that you can do practically all of your basic editing in the easy to use visual mode and can learn the command mode little by little.

CP/M and MP/M are registered trademark of Digital Research, Inc.

Getting Started

This manual is organized into four main sections. The first section describes the overall operation of VEDIT in both command and visual mode without describing the functions in either mode in detail. The section also describes basic disk file editing concepts and their application to VEDIT. The second section describes the visual mode in detail, while the third 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.

The new user of VEDIT is best off to at least skim the next section and the visual mode description before editing any important files. The anxious new user will probably want to immediately "tinker around" and this is probably the best way to learn the visual mode, as long as no important files are clobbered. Before you can do this however, you will have to go through the customization process described in Appendix A. This delays things a little at the beginning but is well worth the trouble. The customization process leaves a lot of options up to your choice. Since you probably won't know what options to choose the first time, recommendations are made for the first few times you go through the customization process. 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.

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.

Introduction

VEDIT is a full screen, or "visual" editor which currently runs under the CP/M operating system and its derivatives, including MP/M and CDOS. It allows any text file to be created and/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 their 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 command mode allows the execution of normal editor commands, such as for searching, altering and displaying lines. Commands are provided for saving a section of text in a text register and inserting the contents of the text register at any position in the file. Command mode also allows for explicit Read and Write commands to be executed, and new Input or Output files to be opened and closed. The repetitive execution of single commands or sets of commands called Iteration Macros is provided. One command puts the editor into visual mode.

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, and then 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.

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 ...".

The Text Register

The text register is used for saving a temporary copy of text which is independent of the text buffer. Its main purpose is for copying or moving a section or "block" of text from one area of the file to another. The text register is not changed by any disk read/write operations, nor by the "EA" or [RESTART] commands. It can thus also be used to extract a section of text from one file and insert it anywhere in another file. Commands exist for using the text register from both command and visual mode. The text may be saved in one mode and inserted in the other. Inserting the text register does not destroy or change the register. It may therefore be inserted repeatedly at different locations in the file.

In command mode the text save is line oriented, while in visual mode it is character oriented. Visual mode also has an additional text register operation which moves a block of text to the text register and then deletes it from the text buffer. The text register is thus more flexible in visual mode, besides being much easier to use.

Invoking VEDIT

VEDIT is invoked from CP/M by typing a command of one of the following three forms:

```
VEDIT filename.ext

VEDIT

VEDIT infile.ext outfile.ext
```

The first form is the normal form for creating a new file or editing an existing file. The file name may be specified with an optional drive name and file extension in the normal CP/M format. The named file is opened for input if it exists on disk and an auto-read is done on it. If the file does not exist, the message "NEW FILE" is printed. An output file is also opened which will have the specified name when the edit session is over. At that time the input file will have been renamed to 'filename'.BAK. VEDIT will begin in either visual mode or command, depending upon how the "Begin in Visual Mode" switch was set during customization. The normal way to write the output file to disk and exit VEDIT is to enter command mode and give the "EX" command. For example, the procedure to edit a file called MYFILE.TXT would be:

```
VEDIT MYFILE.TXT
```

(The editing would primarily be done in visual mode. Enter command mode when done editing to give the exit command).

```
EX[ESC][ESC]
```

(The file will be written out to disk and you will get the CP/M prompt).

The second form is used when VEDIT is to be loaded into memory and the input and output files are to be specified from command mode. With this form, VEDIT always begins in command mode. The first form is equivalent to the second form followed by the command "EBfilename.ext[ESC][ESC]".

The third form is used when one file, "infile.ext" is only to be read in and not altered, and the second file "outfile.ext" is to be created. If the file "outfile.ext" already exists, it will be renamed to "outfile.BAK". This form may also be used when the file to be edited is more than a half diskette in size. In this case "infile.ext" would be the file to be edited, and "outfile.ext" would be the same file name, but with a different drive specification for a drive containing a nearly blank diskette. This third form is equivalent to the second form followed by the command "ERinfile.ext[ESC]EWoutfile.ext[ESC][ESC]".

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 a character at a time, a line at a time 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 in the visual mode is the ability to move or copy a block of text to any other position in the file. This block of text is specified by moving the cursor to the beginning and end of the text block, typing a function key at each end and then moving the cursor to the place in the file it is to be inserted. Typing one more function key inserts a copy of the text at the cursor position.

The visual mode handles text lines which are up to 260 characters (256 plus CR LF and two spare) long. Text lines longer than a screen line are handled by displaying them on multiple screen lines and indicating in the first reserved column those screen lines that are continuations. These continuation lines are created as necessary while you type.

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. This is explained above under "Automatic Read / Write". Its purpose 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. The automatic read/write in visual mode will also begin to write out the text buffer if the memory becomes full and the user continues to type in new text.

Tab characters may be inserted into the text in both command and visual mode. Visual mode can optionally also insert spaces to the next tab position when the Tab key is hit. 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.

As an aid in writing programs in structured languages such as Pascal, PL/I and C, the visual mode also has an Indent and Undent function. 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 further to the right with the [INDENT] key, and moved left with the [UNDENT] key.

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" (Extended command). 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 eight overlapping categories:

Edit pointer movement - B, L, C, Z
Display text - T
Alter text - D, I, K, S, EI
Search - F, N, S
Text Move - G, P
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.

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 "OUT OF 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 iteration macros and for extensive file handling. Searching is used for directly accessing a particular word or string in the file and then entering visual mode. 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. For example, the following command will search for all occurrences of the word "temporary" and let those regions of the file be edited in visual mode.

```
#[Ntemporary$V]$$
```

(The "$" character is used in this manual for the [ESC] control character, since the "$" is echoed any time the [ESC] is typed in command mode.)

Another common operation is to change (substitute) all occurrences of a word to another and check that it was done correctly in visual mode. For example, the following command could be used in a form letter to change the string /name/ to the desired name, check that it was done right in visual mode, and if necessary make the changes in that mode.

```
#[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:

```
#[Sand$or$V]$$
```

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 $$
```

The commands "I" for Insert and "T" for Type are most useful in iteration macros. The "T" can be used to simply 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 "[TAB]" is the tab character and the "[CR]" is the RETURN character which will cause a carriage return and line feed to be inserted and printed.)

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.

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 is set by the user during customization. A "-" is suggested.) The bottom screen line is used for status information consisting of messages. (Some CRT displays allow the messages to appear in reverse video.) 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, but special control characters may be inserted into the text buffer in command mode. The control functions either move the cursor or perform a visual operation.

The visual mode performs auto-buffering when the user reaches the end of the text buffer, and the entire input file has not yet been read. Specifically, if the current screen display reaches the end of the text buffer, the auto-buffering is performed. VEDIT will also perform an auto-write if the text buffer reaches its maximum size 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 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.

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. (They may result from unexacting use of the "D" command 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 on the CRT if necessary. 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] or [CR] and [TAB] are displayable characters, but have special properties. The Carriage Return character [RETURN] 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 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 exist in the text, are displayed in the regular CP/M format by preceding the letter with an "Up Arrow". Although special control characters cannot be entered into the text from visual mode, they can be entered from command mode and will then be displayed correctly in visual mode.

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 visual functions [SET INSERT MODE], [RESET INSERT MODE] and [SWITCH INSERT MODE] are used for switching between NORMAL and INSERT mode. The visual functions for removing text are [DEL] which deletes a character, [EREOL] for deleting (erasing) all remaining characters on the line from 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 visual function [RESTART] starts the edit session over, saving the current file on disk, just as the EA command does. Additionally the visual functions [COPY TO TEXT

REGISTER], [MOVE TO TEXT REGISTER] and [INSERT TEXT REGISTER] are used to move or copy text from one area in the file to another. The text register used is the same as used in command mode, thus the text register may be set in command mode and inserted in visual mode or vice versa.

Indent and Undent Functions

As an aid in 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.

[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. The same rules as for [CURSOR DOWN] apply.
[CURSOR DOWN]	Move the cursor down 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, don't move.
[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.
[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.
[BACK TAB]	This moves the cursor to the first position in the current physical line. If the cursor is already at the first position, the cursor is moved up to the first position of the previous screen line.
[TAB CURSOR]	This moves the cursor to the next tab stop. If there are no more characters on the line, don't move. Note that this only moves the cursor, use the [TAB] key to insert a Tab character.
[ZIP]	This moves the cursor to the end of the text line the cursor is on. If it already is at the end of the line, it moves to the end of the next text line.
[NEXTLINE]	This moves the cursor to the beginning of the next text line.

[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].
[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, to completely delete a line would require the following sequence: [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.
[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 an the message "1 END" is displayed on the status line. When the key is hit while the "1 END" is set, the text block between the first cursor position and the current cursor position is copied to the text 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 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]	A copy of the current text register is inserted at the current cursor position. 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.
[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.

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 keybaord 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 the [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 and also during the execution of the "A", "F", "N", and "T" commands, and in a few other situations.

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

(Please note that 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 ">". The user determines which characters to use during the customization process.)

'n' denotes a positive number. (# represents 32767)
'm' denotes a number which may be negative to denote backwards in the text buffer.

'string', 's1' and 's2' denote strings which may include the [RETURN] key in them. 'string' and 's1' may also include the "wildcard" character "¦", each of which will match any character during the search.

'file' is a disk file name in normal CP/M format with optional disk drive and extension.

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.
G	Insert the contents of the text register 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.
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.
mP	Put 'm' lines of text into the text register. "0P" empties the text register.
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 text register.
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.

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.
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.
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	(5 - 100)
3	Indent Increment	(1 - 20)
4	Lower case convert	(0, 1 or 2)
5	Conditional convert character	(32 - 126)

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)

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.

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 register 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:

nFsl[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 'sl'. The edit pointer will be positioned after the last character of the 'n'th occurrence of 'sl' if it is found. If the 'n'th occurrence of 'sl' is not found, an error will be printed and the edit pointer will be positioned after the last occurrence of 'sl' found, or be left at its original position if no occurrences of 'sl' 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. For the command form "#Fsl[ESC]", an error is only given if no occurrences of 'sl' 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$$`	Find the word "end" if it occurres in the last 100 lines of the text buffer.
`#[Ffix up$V]$$`	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 it" is found and so on.
`F$V$$`	The next occurrence of the previous specified string is found, and visual mode is then entered.

G Get

Example: `G$$`

Description: This command inserts a copy of the text register 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 the text register.

Notes:

See Also: Commands: P
Visual Mode text move.

Examples:

`BG$$`	Inserts the contents of the text register at the very beginning of the text buffer.
`12[G]$$`	Inserts the contents of the text register twelve times at the current edit pointer.

`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.

Itext[ESC] Insert

Example:	Ia word$$ I[RETURN]new line$$
Description:	This command inserts the text 'text' into the text buffer, starting at the current edit pointer. The insertion is complete when the [ESC] character is encountered. The inserted text does not overwrite any existing text. The 'text' may contain the [RETURN] 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:	Some control characters, including the [ESC], can only be inserted with the "EI" command. 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.

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.

nNsl[ESC] Next

Example: `Nbad line$$`   `3Nthird$$`

Description: This command is very similar to the "F" command, except that if the 'n'th occurrence of 'sl' 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 'sl', 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$-4DItype$]$$`	Changes all occurrences of the string "typo" to "type" in the rest of the file.
`Nxcxc$$`	Accesses the last part of the file, assuming the string "xcxc" never occurs in it.

mP Put

Example: 40P$$ -20P$$ 0P$$

Description: This command saves a copy of the specified text lines in the text register. The previous contents of the text register are destroyed. 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 register share the same memory space, saving text in the text register decreases the amount of memory available to the text buffer. Thus the "0P" command should be given when the saved text 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: If the "P" command occurs within an iteration macro, the macro is aborted following the command.

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

Examples:

120P120K$$	The text lines are saved in the text register and then deleted from the text buffer.
-23T$$ -23P$$	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 commmand 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$$`	Changes all occurrences of "typo" to "type".
`#[Stypo$type$0LT]`	Changes all occurrences of "typo" to "type" and prints out every line that was changed.
`#[Sname$smith$V]$$`	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 number of memory bytes used by the text register (length of the text register).

Notes: These three numbers will not always add up to the same total, since several other 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 register is 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: 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]$$
Description:	This command will insert a specified line number range of the file "file" into the text buffer at the edit pointer. If insufficient memory exists to insert the entire file segment, as much as possible will be inserted and a *BREAK* message will be printed. Use a line range of [1,10000] to insert an entire file.
Notes:	The line numbers of a file can be printed by PIP using the [N] option.
See Also:	Commands: A, ER
Example:	EGlibrary.asm[34,65]$$ Lines 34 through 65 of the file "LIBRARY.ASM" are inserted into the text buffer at the edit pointer.

nEI Edit Insert

Example:	12EI$$
Description:	This command will insert the character whose decimal value is "n" into the text buffer at the edit pointer. This is useful for entering special control characters into the text buffer. One application would be printer formatting characters. While many control characters can be entered using the "I" command, some can only be entered with the EI command. Characters with a decimal value between 128 and 255 can also be entered with the EI command. Only the "End of File" marker with a value of 26 cannot be entered. Control characters are displayed in both command and visual mode by preceding the letter with an "Up Arrow".
Notes:	
See Also:	Commands: I
Example:	8EI$$ A backspace character is inserted into the text buffer at the edit pointer.

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 5 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)

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

Notes: While the parameter values were specified in hexadecimal during customization, they must be specified in decimal in command mode.

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

Example: EP 1 6$$ This sets the "Indent Increment" to six.

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 CP/M and start
                    over.
```

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)

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.

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. An error is printed if no output file is open, and no other action is taken.
Notes:	
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.

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].

Customizing VEDIT

Introduction

VEDIT has to be customized before the first time it is used, and can then be customized again, when the user has a new keyboard, a new CRT terminal, just wishes to change some default parameters or try a new keyboard layout. Please note that it does not have to be 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.

VEDIT is supplied as a disk file with an extension of ".SET", i.e. VEDITZM.SET, VEDITZC.SET, which contains 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" which is the executable version of the customized editor.

The customization is done with the supplied programs VEDSET.COM for the memory mapped versions, and VDSETCRT.COM for the CRT terminal versions. Running VEDSET (or VDSETCRT) simply involves typing a control key or a number in response to the questions it asks. 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 versions of VEDIT, each for a special application.

Getting started

The following paragraphs describe the various aspects of the customization in some detail. You do not need to fully understand these details in order to get VEDIT up and running, since the enclosed "Example Keyboard Layout" and the next section "Running VEDSET ..." give recommended values for every question. 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.

Determining the desired keyboard layout for the cursor movement and function keys is the first step of the customization. Since it could be a difficult step, 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 logical 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 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 lables 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.)

The second decision during customization is to determine the desired Tab positions and whether tabs should be expanded with spaces. Unless you have some special application, don't expand tabs with spaces, it will use up lots of disk space. Where you set the tab positions will only be applicable to VEDIT since most CP/M utility programs set the tab positions at every 8th position. This is thus the best choice for VEDIT, too. One exception would be where you do a lot of word processing with the same tab positions. Another exception would be if you are using a structured language compiler which perhaps set tabs at every 4th position for easier indenting. The values you enter for the tab positions, the switches and the parameters below, are the defaults, they may be changed while running VEDIT. Assuming you want the tabs at every 8th position, the tab positions would be:

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

These are also the default positions.

Five special characters can also be customized. The first is the line continuation indicator used in visual mode in reserved column 0. We suggest a "-" or reverse video "-", codes "2D" or "AD" hex. The second is the command mode "Escape character". This will normally be the "ESCAPE" or "ESC" key with a hex code of "1B". If your keyboard does not have an "ESC" key, you will need to choose some other control character, perhaps a "CTRL-Z" with a hex code of "1A". The next two characters are the enclosing brackets for iteration macros. They are printed as "[" and "]" in this manual (codes 5B and 5D hex). Some users may be more familiar with the "<" and ">" angle brackets (codes 3C and 3E hex). Use either set, but it may help if your keyboard produces one set without needing the Shift key. The fifth character only applies to memory mapped versions, and is the character used for the blinking "underline" cursor. While this would normally be the underline character, (code 5F hex), users with displays which do not produce reverse video, such as the Sorcerer, may wish to try a hex code of "7F" which is commonly a solid block.

You even have the choice of whether the messages on the status line appear in reverse video on CRTs which support reverse video. Some displays, such as on the Sorcerer and TRS-80 Model I, do not produce reverse video.

Next you have to decide on the default settings of several more switches and parameters (switches (2) - (7) and parameters (1) - (5)). Remember that the switch and parameter settings can be changed while running VEDIT. Switch (2) determines whether auto-buffering is enabled during visual mode. The first time around, we suggest you enable auto-buffering. After some practice and reading the section on Auto Read/Write, you may decide otherwise. Switch (3) determines whether VEDIT starts in Visual mode or command mode. The first time around, we suggest you set this switch on. Switch (4) determines whether the edit pointer's position (or cursor's in Visual mode) is moved just past the newly inserted text, when the text register is inserted. Again, for the first time around, we suggest that you set this switch on. After some practice with the text register, you will know which way you prefer to have this switch. Switch (5) determines whether the difference between upper and lower case letters is ignored in searches. We suggest you set this switch on. Switch (6) determines whether the screen is cleared when visual mode is exited to command mode. We suggest you set this switch off. Switch (7) 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 vise 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.

Parameter (1) is only applicable to memory mapped versions, and determines the cursor type. 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. Parameter (2) determines the memory mapped cursor's blink rate. Start with the value suggested by the VEDSET prompt. Parameter (3) determines the "Indent Increment". A value of 4 is common when structured programming languages are being used. Parameter (4) controls lower to upper case conversion. This is described under the "ES" command. Start with a value of "0" for no conversion. Parameter (5) is related to parameter (4) and again described under the "ES" command. Supplying a value of "3B" hex, makes the ";" the special conditional character.

Two more parameters that can be customized are dependent upon the memory size of CP/M you are running. For details on these two parameters please refer to the section below. While these two parameters can be specified for many special applications, it is best to follow the table below the first few times, until you have a good 'feel' of the operational characteristics of VEDIT. The first value must be specified in bytes between 1024 and 32768, and the second value must be specified in K bytes between 1 and 32. (A "K byte" is a unit of 1024 bytes. 1024 = 2 ** 10.)

CP/M size	Value for Spare	Value for Transfer
16K	1526	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

(In particular, do not make the "Spare Memory for Auto-Read" 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 41K. 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 last information needed for customization pertains to your display board or CRT terminal. First, you need to know the number of lines and the number of characters per line that it 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 Procesor Technology VDM board, which should have a 00 output to Port C8 hex. (The SOL-20 requires a 00 output to Port FE hex).

Running VEDSET or VDSETCRT

The customization is done with the supplied programs VEDSET.COM for the memory mapped versions, and VDSETCRT.COM for the CRT terminal versions. Depending upon which version you have, your diskette may contain several ".SET" files, which will be described on a separate sheet entitled "Description of Files".

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
```

VEDSET (VDSETCRT) will now prompt with questions which are answered by typing the control keys to setup the keyboard, or with a number, or 'Y' for Yes or 'N' for No. The questions with a numeric answer also require a RETURN at the end of the line. Unless otherwise specified, typing a RUBOUT or CTRL-U will ignore the input for that question and repeat the last question. The following steps describe the answer to each question.

1.) (CRT Terminal Only)

The CRT customization program begins by displaying a menu of CRT terminals which are directly supported. Type any key after the first part of the menu, since the menu is two screens long. When prompted, enter the number of the CRT terminal which you wish to use VEDIT on. You may of course perform the customization several times for different CRT terminals. Reference the disk file README.CRT in case your CRT terminal is not listed in the menu.

2.) 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.

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 IN ESCAPE SEQUENCE
Simply answer with a [RETURN] if you are not using escape

sequences or are typing them in by hand. 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)

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, since the following question will let you start over again.

WAS KEYBOARD LAYOUT CUSTOMIZED CORRECTLY? (Y OR N)
Enter 'Y' if the keyboard layout was customized correctly. Else enter 'N' to repeat this step.

3.) DO YOU WISH TO USE THE DEFAULT TAB POSITIONS? (Y OR N)
Enter "Y" if you want the tabs at every 8th position, which is the normal for CP/M. Otherwise, enter "N" and the following message appears:

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 CTRL-U or RUBOUT to start the question over.

4.)

	Suggest
HEX CODE FOR SCREEN CONTINUATION CHARACTER	2D
HEX CODE FOR COMMAND MODE ESCAPE CHARACTER	1B
HEX CODE FOR COMMAND ITERATION LEFT BRACKET	5B
HEX CODE FOR COMMAND ITERATION RIGHT BRACKET	5D
HEX CODE FOR CURSOR CHARACTER	5F

Enter the number in hexadecimal and a RETURN following each question. Always enter a value, there are NO DEFAULTS. Typing a CTRL-U or RUBOUT will start over with the first character. Note that the last prompt relating to the cursor only appears with the memory mapped customization.

5.) (CRT Terminal only)

ENTER DECIMAL VALUE (4MHZ = 76, 2MHZ = 38)
Enter a value of "38" if you running a 2 Mhz processor or "76" if you are running a 4 Mhz processor. Interpolate for other processor speeds. This value is only used for CRTs which require time delays for some functions. The maximum value is 255.

6.) REVERSE VIDEO ON STATUS LINE (0=NO, 1=YES)
If your CRT or video display board produces reverse video, answer with a "1" for Yes. If you have a Sorcerer, TRS-80 Model I, or some other display which does not produce reverse video, answer with a "0" for No.

7.)

		Suggest
(1) EXPAND TAB WITH SPACES	(0=NO 1=YES)	0
(2) AUTO BUFFERING IN VISUAL MODE	(0=NO 1=YES)	1
(3) BEGIN IN VISUAL MODE	(0=NO 1=YES)	1
(4) POINT PAST TEXT REG. INSERT	(0=NO 1=YES)	1
(5) IGNORE UC/LC DISTINCTION ...	(0=NO 1=YES)	1
(6) CLEAR SCREEN ON VISUAL EXIT	(0=NO 1=YES)	0
(7) REVERSE UPPER AND LOWER CASE	(0=NO 1=YES)	0

(1) CURSOR TYPE	(0, 1 or 2)	1
(2) CURSOR BLINK RATE (10 is fastest)	(10 - 100)	See Prompt
(3) INDENT INCREMENT	(1 - 20)	4
(4) LOWER CASE CONVERT	(0, 1 or 2)	0
(5) CONDITIONAL CONVERT CHARACTER	(20 - 7E)	3B

Enter the number in hexadecimal and a RETURN following each question. Type a CTRL-U or RUBOUT to start over with the first switch or parameter. CRT version users may answer parameters (2) and (3) with an arbitrary value. Note that the prompts for parameters (1) and (2) relating to the cursor only appear with Memory Mapped versions of VEDIT.

8.) SIZE IN DECIMAL OF SPARE MEMORY FOR AUTO READ
See the table on page 57 for a recommended value depending upon your memory size. Please read the note below the table too. Enter the decimal number followed by a RETURN. The number must be in the range 1024 - 32768 or an error and a reprompt of the question will be given. Type a CTRL-U or RUBOUT to restart the question.

9.) SIZE IN DECIMAL OF FILE MOVE TRANSFERS IN K BYTES
See the table on page 57 again for a recommended value. Enter the decimal number signifying the multiple of 1K (1024) bytes desired, followed by a RETURN. The number entered must be in the range 1 - 32.

10.) ENTER NUMBER OF SCREEN LINES IN DECIMAL
Enter the number of lines on your CRT display and a RETURN. 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.

11.) ENTER LINE MOVEMENT FOR PAGING IN DECIMAL
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.

12.) ENTER TOP LINE FOR CURSOR IN DECIMAL
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. "3" is a good starting point.

13.) ENTER BOTTOM LINE FOR CURSOR IN DECIMAL
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" 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.

14.) ENTER SCREEN LINE LENGTH IN DECIMAL
Enter the number of characters per line your CRT display has and a RETURN. This number must be in the range 20 - 255. This value will be 80 for most CRT terminals and either 64 or 80 for most Memory Mapped displays. The value for the MATROX video display board is 128.

15.) ENTER LENGTH OF DISPLAYED LINE IN DECIMAL
Enter the number of characters per line you want VEDIT to actually display and a RETURN. This value will normally be 80 or 64 since it usually is equal to the screen line length, unless for some reason you don't wish to use the full line length. This number must be less than or equal to the above length of a screen line. The value for the MATROX video display board is 80.

16.) (Memory Mapped Only)

ENTER ADDRESS OF SCREEN IN HEXADECIMAL
Enter the memory address of the beginning of the video board in hexadecimal and a RETURN.

17.) (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 number" pairs needed for initialization.

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

The number of 'data byte'/ 'port number' pairs specified must be entered in hexadecimal with each number followed by RETURN. Typing a CTRL-U or RUBOUT will ignore any values for that pair and reprompt with the "ENTER DATA BYTE" question for that pair. (The Processor Technology VDM board requires one pair, a "00" sent to port "C8" hex, and the SOL-20 a "00" sent to port "FE" hex.)

18.) WAS THE CUSTOMIZATION DONE CORRECTLY (Y OR N)
Enter 'Y' if it appears that the customization was performed correctly, and your customized version of VEDIT will be created on disk. Otherwise, enter 'N' if you want to start over again at step 2. At step 2 you can skip customizing the keyboard layout again.

More on the Memory Parameters for Customization.

The first parameter determines how many bytes of memory are free after VEDIT does an auto-read (such as following an EB command) on files too large to fit in memory all at the same time. 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. The CP/M operating system (BDOS and BIOS) takes up about 4K of memory and VEDIT up to 11K. The rest of the memory space is for the text buffer and text register. Thus a 24K CP/M system would have a 9K buffer, and a 48K system a 33K buffer. Choosing a 1K (1024 byte) multiple makes the disk read/write work a little bit faster. The second parameter specifies the size of file transfers during auto-buffering and for the 'N' command. See the section on auto-buffering for details. 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 read the number of K bytes specified during customization under "Size of File Transfers". 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 during customization under "Spare Memory for READ".

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.

This page is intentionally blank.

'n' denotes a positive number. 'm' denotes a number which may be negative to denote backwards in the file.

'string','s1' and 's2' are strings of characters which may include anything except an [ESC]. The special character "¦" will also match any character during the search.

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'.
G	Insert the contents of the text register.
Itext[ESC]	Insert the 'text' into the text buffer.
mK	Kill 'm' lines.
mL	Move the edit pointer by 'm' lines.
nNstring[ESC]	Search for 'n'th occurrence of 'string' in file.
mP	Put 'm' lines of text into the text register.
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.

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".	
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)
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)
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.	

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

OUT OF 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.

CompuView Products Inc.

618 Louise, Ann Arbor, Michigan 48103
Telephone (313) 996-1299

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. The ".DOC" files contain the manual. These ".DOC" files are only supplied with 8" disks.

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.
VDOC1.DOC	Contains the Overall Description of VEDIT.
VDOC2.DOC	Contains the Visual Mode Description of VEDIT.
VDOC3.DOC	Contains the Command Mode Description of VEDIT.
VDOC4.DOC	Contains the appendices for the description of VEDIT, including directions for the customization process.

VEDIT CP/M
Visual Editor

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. It's general purpose nature will handle all standard text files and is suitable for Word Processing, Fortran, Assembler, Basic, C-Basic and more.

Visual Mode

The main feature of VEDIT is it's 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. The typed text will appear on the screen at the cursor position and will either overwrite the existing text or be inserted without overwriting. The bottom screen line is reserved for status information, such as whether Insert mode is on, the status of the text register and error messages.

Easy to use and full cursor control

VEDIT provides a full array of easy to use cursor movements, including cursor Up, Down, Right, Left, Back Tab, Next Tab, Next Line, Page Up, Page Down, Home and Zip to the end of lines. The operation of the cursor movements is designed to allow the user to perform common operations with the typing of as few keys as possible. For example, the Zip key moves the cursor to the end of the text line, or if the cursor is already at the end, to the end of the next text line. The cursor always points to true text characters and never to nonexistent spaces past the end of a text line.

Function keys allow for character deletion, line deletion, and for lines to be concatenated or split. One visual function even 'undoes' the changes just made to the text line in case the user made a mistake, i.e. erased the line by accident.

Block Move Too

Blocks of text may also be moved in visual mode simply by moving the cursor to the beginning and end of the text block and hitting a function key at each end to save a copy of the text block in the text register. The cursor may then be moved to any place in the file, or even to another file; hitting one more function key inserts a copy of the text register at the cursor position.

Flexible Command Mode and Extensive File Handling

For full flexibility, a superset of the ED commands is included in command mode, allowing search and substitute operations, repetitive editing operations, text move and extensive file handling (ER,EW,EB,EF,EX,EA,...). By use of the text register and the file handling commands, a block of text may readily be copied from one file to another. The file handling commands also allow for a file to be split into smaller files, for several files to be merged together, and much more. A backup of the original files is always preserved. Nested iteration macros allow for sophisticated, repetitive editing operations. The visual mode may be specified as a command within iteration macros, allowing for example, all regions of a file containing a specified string to be edited in visual mode.

Special Features

- Included is a setup program which allows you to customize VEDIT to your screen size (up to 70 lines and 200 columns), screen address and keyboard layout. You decide which key or control code to use for each cursor movement or visual function. Even keyboards producing the full 256 codes are supported. The setup program also allows the user to decide the default values for the tab positions, various switch settings and several parameters.

- The Tab key allows insertion of a tab character or spaces to settable tab positions. The settable tab positions allow, for example, a word processing user to set the same positions in VEDIT as are set for a text formatting program, in order to see what the final product will look like. The tab key may be expanded with spaces, for users with special text layout requirements.
- The visual mode handles text lines longer than the screen by writing them on multiple screen lines and indicating in the first reserved column those that are continuation lines. Continuation lines are automatically created as necessary while you type.
- The sophisticated disk buffering 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 to be done with little concern over the size of the file. This Auto-Read/Write may also be disabled.
- Unlike several other screen oriented editors and word processing packages on the market, VEDIT never has trouble keeping up with the fastest typists.

Applications: VEDIT is ideally suited for work in Basic, C-Basic, Fortran and Assembler, because of its ability to handle long lines and very large files, and its capabilities for copying portions of source code from one file to another. Even Basic users will find that it is much easier to enter a new program, or make extensive changes, with VEDIT rather than the Basic editor. VEDIT is also well suited to Word Processing uses, readily allowing word searches, and having a very easy to use block move in visual mode. The customization program allows full use of all the special keys on word processing keyboards, a feature rarely found elsewhere. Many users will find that VEDIT and a good Text Output Formatter give them more capabilities than any single word processing package. With all of these capabilities, VEDIT will still operate in even the smallest CP/M system, and allows a 29 Kbyte text buffer in a 40K system.

Availability: VEDIT is currently available on 8" disk for CP/M systems with most memory mapped displays, including the VDM, SSM, VIO, Matrox, and the Piiceon video board. It will shortly be available for the Sorcerer in at least some disk formats. It will also be available for several smart CRT terminals in June, including the Heath H19, DEC VT100 and Hazeltine 1500. Please check with us on the availability for the TRS-80 Model II, other CRT terminals, and for the disk format you need.

Ordering: Specify your video board or CRT terminal type, the 8080/Z80 or Z80 code version, and the disk format desired. All packages include VEDIT, the customization program, and an extensive 56 page manual. VEDITS (all features of VEDIT except command mode) and the entire manual text on disk are available on request. A disk with both the 8080/Z80 and Z80 code versions may be ordered for $30 extra.

Standard Package: For CRTs, Sorcerer, Piiceon	**$110**
Memory Mapped Package: For memory mapped video boards	**$100**
Manual: Price refunded with software purchase	**$15**

CompuView Products Inc.

1531 JONES DRIVE ANN ARBOR, MICHIGAN 48105
CALL ANYTIME: (313) 996-1299