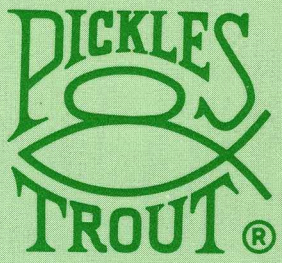
# P&T VEDIT USER'S MANUAL



# P&T VEDIT USER'S MANUAL

**P&T VEDIT USER'S MANUAL**

Copyright © 1980, 1981, 1983 Theodore Green, author

Published by
Pickles & Trout
P.O. Box 1206
Goleta, California, 93116

All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

10 9 8 7 6 5 4

Pickles & Trout is a registered trademark of Pickles & Trout
CP/M is a registered trademark of Digital Research, Inc.
MP/M is a trademark of Digital Research, Inc.
CDOS is a trademark of Cromemco, Inc.
TRS-80 is a trademark of Tandy Corp.

The publisher and author have made a reasonable effort to insure that the computer programs described herein are correct and operate properly and that the information presented in this publication is accurate; however they are sold without warranties either expressed or implied, including, but not limited to, the implied warranties of merchantibility and fitness for a particular purpose. The publisher and author are not liable for consequential damages resulting from the use of this product either individually or in concert with other computer programs. Further, the publisher and author reserve the right to revise this publication and the programs described herein and to make changes from time to time in the contents thereof without obligation of the publisher or author to notify any person or organization of such revision or changes.

# Table of Contents


| Section | | Page |
|---|---|---|
| I.) | Introduction to VEDIT | 3 |
| II.) | Getting Started | 4 |
| | Sample "First" Edit Session | 5 |
| 1.) | Overall Description | 9 |
| | Introduction | 9 |
| | Basic Editing Concepts | 9 |
| | Visual Mode | 11 |
| | Command Mode | 12 |
| | Which Mode to use for What | 14 |
| | Word Processing with VEDIT | 15 |
| | The Text Register | 16 |
| | Automatic Disk Buffering | 16 |
| | Forward Disk Buffering | 17 |
| | Backward Disk Buffering | 18 |
| 2.) | Visual and Command Modes Task Tutorial | 20 |
| | Invoking VEDIT | 20 |
| | Keyboard Characters & Edit Functions | 22 |
| | Cursor Movement | 23 |
| | Page Movement | 24 |
| | Adding New Text | 25 |
| | Visual Functions | 26 |
| | Deleting Text | 26 |
| | Correcting Mistakes Made to a Line | 28 |
| | Repeating Operations | 28 |
| | Indenting Text | 29 |
| | Moving and Copying Blocks of Text | 30 |
| | Emptying a Text Register | 32 |
| | Sending Text to the Printer | 32 |
| | Entering Control Characters in the Text | 33 |
| | Switching Between Visual and Command Mode | 33 |
| | Searching and Substituting | 34 |
| | Saving Already Edited Text | 35 |
| | Begin Editing New File | 35 |
| | Making More Memory Space | 36 |
| | Inserting a Line Range of Another File | 37 |
| | Concatenating Two Files | 38 |
| | Splitting a File into Two or More Files | 39 |
| | Recovery from Full Disk Errors | 40 |
| | Ending the Edit Session | 41 |

3.) Visual Mode . . . . . 42

Entering New Text . . . . . 42
Performing Edit Functions . . . . . 43
The Repeat Function . . . . . 44
The Tab Character . . . . . 44
Displaying Line and Column Numbers . . . . . 45
The Text Register . . . . . 46
Printing Text . . . . . 46
Inserting Control Characters . . . . . 47
Indent and Undent Functions . . . . . 47
Lower to Upper Case Conversion . . . . . 48
Disk Buffering in Visual Mode . . . . . 49
Details of the End of Lines . . . . . 49
Edit Functions (Cursor Movement) . . . . . 50
Edit Functions (Visual Functions) . . . . . 52

4.) Command Mode . . . . . 54

Command Mode Notation . . . . . 55
Editing a Second File . . . . . 55
Search Options and Special Characters . . . . . 55
Iteration Macros . . . . . 57
Text Register . . . . . 60
Printing Text . . . . . 60
Disk Buffering in Command Mode . . . . . 60
Disk Write Error Recovery . . . . . 62
Command Line Editing . . . . . 64
Brief Command Description . . . . . 65
Detailed Command Description . . . . . 69

5.) Appendices

A - Customizing VEDIT . . . . . 96

What is Customization . . . . . 96
When is Customization Necessary . . . . . 96
How to Perform Customization . . . . . 97

Customization Notes . . . . . 107

VEDIT Checksum . . . . . 107
Keyboard Layout . . . . . 107
A Word About The Keyboard . . . . . 109
Memory Size Parameters . . . . . 109

B - Command Reference . . . . . 110
C - Error Messages . . . . . 112
D - VEDIT notes . . . . . 114
E - Preconfigured Keyboard Layout (P&T CP/M 2) . 115

## 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 pressing a function key. These insertions, deletions and corrections are immediately seen on the screen and become the changes to the file.

It is very easy to send any portion of the text to the line printer. One scratchpad buffer may be accessed for "cut and paste" operations. Other features, such as automatic indenting for structured programming languages, simplify and enhance program development editing.

VEDIT also provides a very flexible and powerful command mode for performing search and replace operations, repetitive editing operations using several types of macros and much more. Blocks of text may be copied or moved within the current file and other files in an almost unlimited manner. The extensive file handling allows multiple files to be edited, split and merged and includes the ability to insert a specified line range of another file at the edit position. The powerful command macro capability allows complex editing tasks to be performed automatically. Example of such tasks include numerous search/replace operations on multiple files and source code translations.

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

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

## Getting Started

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

Most users will want to perform the customization after gaining some familiarity with VEDIT. While many parameters can be customized, the menu driven operation allows you to limit your attention to a subset of these parameters. You may customize VEDIT as many times as you wish. 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 with different configurations.

This introductory section includes a "first" sample edit session with VEDIT to familiarize you with the most basic aspects of using the editor. After performing this sample edit session you are best off to at least skim the "Overall Description" section in order to get an overview of the capabilities of VEDIT. Trying out the editor while reading the tutorial section is the best way to gain a working familiarity with most features. The visual (full screen) mode is easy enough to use that it can be learned by experimenting with the various function keys, as long as no important files are accidentally altered.

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

While you will typically spend 99% of your time in the visual mode and only 1% in the command mode, this manual deals extensively with the command mode. This is appropriate, because the visual mode is exceptionally easy to learn and use. The command mode, because of its powerful capabilities, is more complex, and more difficult to learn. The most complex aspect of the command mode are the "macros" which can perform repetitive editing operations..

## Sample "First" Edit Session

This sample edit session assumes that you are using VEDIT as it came with your Pickles & Trout CP/M.

In this edit session you will create a short file with the name of "FIRST.TRY", which you can subsequently type out from the operating system.

To create this sample file invoke VEDIT with the command:

```
A>VEDIT FIRST.TRY <enter>
```

After a short pause in which VEDIT is loaded from disk, it will briefly display the message:

```
NEW FILE
```

This is an indication that a new file is being created and not an existing one being edited. VEDIT will then clear the screen and position the "cursor" in the upper left hand corner of the screen. A row of dashes, called the "Status Line" will appear at the bottom of the screen. On the status line should be displayed the line and column number for the cursor, in this case "LINE: 1 - COL: 1". Having the status line on the bottom screen line is an indication that you are in the "Visual Mode" of VEDIT, in which you can perform full screen editing.

You can now begin to type some text which will appear on the screen and will eventually be made into a file. Type in the following text by typing the <enter> key at the end of each line.

```
Bach simply awed the professional musicians
who met or just observed him. Their descriptions
indicate that Bach, at the head of an orchestra,
was a conductor very much like the great
conductors of today.
```

While typing in the text you can make corrections by using the BACKSPACE key which will move the cursor left and delete the character there. The most important aspect of VEDIT is, of course, that you can easily edit text after it has been typed in. The first step in editing is to position the cursor at the text which needs changing. Although VEDIT has many cursor movement keys, you can get by for now using just the four basic movements UP, DOWN, RIGHT and LEFT (using the arrow keys) As you use these keys you will notice that you can only position the cursor at real characters in the text and at the ends of lines. You cannot position it on the screen where there is no text.

With the cursor positioned correctly, you are ready to make your changes. The three basic types of changes are over-typing existing text, deleting text and inserting text. To over-type any text, simply position the cursor at the first character to be over-typed and type in the new text. This is the easiest way to correct mis-typed letters. "^" indicates the cursor in the following example:

```
Bach simply wade the professional musicians
           ^
```

After typing the word "awed" the new line will appear as:

```
Bach simply awed the professional musicians
           ^
```

Often you need to insert new text into the middle of a line. For this you use the "Insert Mode" of VEDIT. Look over your keyboard layout and find the key which performs the function [SWITCH INSERT MODE] or [ENTER INSERT MODE]. Press the key (or control sequence) and the message "INSERT" should appear on the status line. In Insert Mode, text typed in the middle of a line will not over-type any existing text but will shift the text right in order to make room for the new text. Consider the following line:

```
Bach simply awed the professional musicians
           ^
```

After typing the word " all" in Insert Mode, the line will appear as:

```
Bach simply awed all the professional musicians
               ^
```

Deletions are also an important part of editing. VEDIT's two most used deletion functions are [BACKSPACE] and [DELETE]. (Check how your keyboard layout has assigned these functions.) [Backspace] deletes the character to the left of the cursor, i.e. the character you may have just typed in. [DELETE] deletes the character at the cursor position. The line always closes up after any type of deletion. Consider the following line:

```
Bach simply aweded the professional musicians
           ^
```

After typing the [DELETE] function twice, the line will be:

```
Bach simply awed the professional musicians
           ^
```

Position the cursor in the first line as follows:

```
Bach simply awed the professional musicians
                  ^
```

Type the <enter> key and the line will be split into two lines:

```
Bach simply awed the
professional musicians
```

Now position the cursor at the end of the first line and type the [DELETE] function. This will append the second line to the end of the first, giving you your original line back.

Continue making edit changes until the text is modified to your satisfaction.

At this point you are ready to save your text on disk and return to the operating system. For this you must exit Visual Mode and enter the "Command Mode". This is done by the [VISUAL ESCAPE] function which corresponds to typing the ESC key twice. Try it. The screen should scroll up and the cursor will be on the bottom line following the command mode "*" prompt. There is one very important thing to remember about the command mode:

All commands are ended by typing the ESC key twice!

One common command is to go back into the visual mode. The command is:

V$$ where "$" indicates typing the ESC key.

This will put you back into visual mode with the status line on the bottom line. The cursor will be positioned at the same place in the text, although not necessarily on the screen, as it was when you exited visual mode before. Go back to command mode. If necessary, repeat switching between command and visual mode until you can clearly identify which mode you are in.

If you have a printer connected to your computer you may want to print the text you have just created. This can be done from either the visual mode or the command mode. To print the entire text from command mode give the command:

B#EO$$ where "$" indicates typing the ESC key.

Before giving the command, make sure that your printer is properly connected and "ON LINE".

Finally, while in command mode you can give the command to save your text on disk and leave VEDIT. The command is:

EX$$ where "$" indicates typing the ESC key.

You should now be back in the operating system with its "A>" prompt. You can check that your file is on disk with the directory command

"DIR", and can type it out with the "TYPE" command, i.e.:

```
TYPE FIRST.TRY
```

You should see whatever you last saw on the screen in the visual mode of VEDIT.

This has introduced you to only a small list of VEDIT's capabilities. However you have experimented with the basics which make up 90% of any editing task, and have tried one of the ways of invoking VEDIT and saving the text on disk. The next section gives an overview of more of VEDIT's capabilities, and the following tutorial gives a hands-on introduction to almost all of the visual mode and more of the command mode. Good luck.

Notes: If after invoking VEDIT you end up in the command mode (a "*" prompt and no status line), this indicates that you either did not specify a filename after "A>VEDIT", or customized VEDIT to begin in command mode (See customization Task 4.3). All of our ready to run versions will start in visual mode when a filename is specified.

# Section I - Overall Description

## Introduction

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

The main purpose of the command mode is for performing repetitive editing tasks, explicit file handling and accessing the additional text register operations. The command mode allows the execution of common line and character oriented editing commands, including searching, altering, inserting and much more. Single commands and groups of commands may be repeated any desired number of times. A powerful aspect of the command mode are macros, which allow groups of commands to be executed repetitively. The file handling commands allow explicit disk read/write operations, and files to be opened and closed. A specified line range of another file can also be inserted at any place in the text being edited. Finally, the "EX" command is given to exit VEDIT, saving the edited file on disk.

## Basic Editing Concepts

The purpose of editing is to create or modify a file on disk so that it may be saved for future use and processed by another program, such as a word processor (text formatter), a compiler, or simply be printed out. When a file is first created, the initial text of the file is entered with the editor, corrections are made, and the text is then saved on disk. When a file edited, the existing copy of the file is read from the disk into the computer's "main memory", the changes are made with the use of the editor, and the modified text is then saved as a new file 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 are meaningful and easy to remember. The name LETTER1 is

thus better than JV%8-G5F. The CP/M operating systems 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 disk. The modified file may be written to disk with a new file name or with the original name. The normal way of invoking and exiting VEDIT will cause it to write it with its original name. One question in this case is: "What happens to the original text file?" VEDIT leaves the original file 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 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 main memory used for saving the file is referred to as the "text buffer". All changes made to the file are made in the text buffer. When the editing is 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 specified 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 too large for all of it to be stored in the text buffer at one time. VEDIT handles such a file by reading the first part of the file into the text buffer, in which you can make any desired changes. After this first part is edited, VEDIT will write the early lines of the text buffer to the output file and read in more unedited lines from the input file. This is repeated until the entire file is edited. If desired, VEDIT can also read edited text back from the output file for further editing. VEDIT can perform this read/ write process automatically and almost invisibly to the user. In particular, when the user reaches the end of the text buffer in visual mode, the beginning of the text buffer is written out to disk (to the output file) and more of the file being edited (the input file) is read or "appended" to the end of the text buffer. This process, when done automatically, is referred to as "auto-buffering". Another automatic process done in both visual and command mode is called "auto-read" which consists of reading the input file until it is all read in, or until the main memory space is almost full.

## Visual Mode

In visual mode, the screen continuously displays the current contents of the file in the region you are editing and a cursor. The bottom line of the screen is used for status information and is normally filled with the "-" character. The changes made to the screen display by typing in new text or using control functions become the changes to the file. The characters typed while in visual mode fall into two categories: Displayable characters and Control characters. The displayable characters are displayed on the screen at the cursor position and cause the cursor to move to the right. The user customized keyboard layout determines which edit function each control character or escape sequence performs.

The edit functions fall into two subcategories - cursor movement and visual functions. The cursor movement operations cause no change to the file, but rather move the cursor forward and backward by a character, a line, 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 be positioned at real characters in the text and at the end of lines. It never points to "space", i.e. a screen position where there is no text. VEDIT can optionally display the cursor's line number and column position on the status line.

VEDIT has two modes for inserting new text: "Insert" mode and "Normal" mode. In normal mode, the new text will overwrite the existing text. In insert mode, the existing text is not overwritten, but rather is squeezed to the right as the new text is typed. New lines are started by simply typing the <enter> key. Typing the <enter> key in the middle of a line splits it into two lines.

Text can be deleted on a character, line or block basis. One can delete the character to the left or at the cursor position, an entire line, or only the portion to the right of the cursor.

A useful feature is the ability to move or copy a section of text to any other position in the file. ("Copy" implies that the original text is not deleted, while "move" implies that the original text is deleted.) This is done by first copying, moving or appending the text to the text register (scratch pad buffer), and then inserting the text register at any place or places in the file. (It may also be inserted in another file). Blocks of text are deleted by moving the block to the text register and emptying the text register, or just "forgetting" about it. Any portion of the text can easily be printed on the line printer and special printer control characters can be imbedded in the text.

Several options are available for dealing with the Tab character. Normally when the "Tab" key on the keyboard is typed, a tab character is placed into the text. A tab character is displayed as spaces to the next tab position. The tab positions may be set as desired; they

are normally at every 8th position. Optionally, typing the "Tab" key can have VEDIT insert spaces to the next tab position into the text.

VEDIT has several unique built in aids for program development. One is automatic indentation for use with structured languages such as Pascal, PL/I and C. When "Indenting" is set, the editor will automatically insert tabs and spaces following each [Carriage Return] to the current indent position. The indent position can be moved right and left by an adjustable indent increment. Many assembly language programmers prefer their program code to be in upper case letters with comments in upper and lower case. VEDIT can accept all text in lower case and automatically convert the labels, opcodes and operands to upper case while leaving the comments in lower case. It does this by searching on the line being entered or edited for a special character such as ";". To the left of the ";" lower case letters are converted, to the right of the ";" they are not converted. This is referred to as "Conditional Lower to Upper Case Conversion".

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

Note: If you are just beginning to use VEDIT, you may wish to skip to the "Visual Mode Task Tutorial" at this point.

## Command Mode

In command mode, the user enters command lines which consist of single commands, concatenated 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 <enter>.

Each command consists of a single letter or two letters if the first letter is "E" (Extended commands). Some commands may be preceded by a number to signify that the command is to be repeated, or "iterated". If no number is given, a "1" is used as the default. Wherever a number is allowed, you can also use the "#" character to represent the maximum positive number 32767. Several commands are followed by additional arguments such as text strings or file names. 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 preceding 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 ">" 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. If no explicit number is specified, it defaults to "#" which signifies "forever" or "all". For example, the command "4T" types out four lines. The command "5[4T]" displays the same four lines five times for a total of 20 displayed lines. The "[" and "]" may also occur within each other ("be nested") for more complicated commands. For example, the command "3[5[4T]4L]" will display the same four lines five times, then move to the next four lines and display them five times and last, move to the next four lines and display them 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 for performing repetitive editing operations.

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 types out 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 available for dealing with the text register. Three commands are used to change the various switches, parameters and tab positions which VEDIT uses in both command and visual modes. One command puts the editor into visual mode. The last two groups of commands deal with the reading and writing of files and with the opening and closing of input and output files.

The commands fall into nine overlapping categories:

| | | |
|---|---|---|
| Edit pointer movement | - | B, L, C, Z |
| Display text | - | T |
| Print text | - | EO |
| Alter text | - | D, I, K, S, EI |
| Search | - | F, N, S |
| Text Register | - | G, P |
| Disk Buffering | - | A, N, W, EA, EQ, EX, EY |
| File Handling | - | EB, EC, ED, EF, EG, EK, 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.

## 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!

Any edit operation which can be performed in visual mode can also be performed in command mode. However, straight forward modifications, insertions and deletions are much easier done in the visual mode. Unless they are part of iteration macros, the equivalent of the "L", "C", "T", "D", and "I" commands are best done in visual mode.

Command mode is most useful in searching for text in the file, performing repetitive edit changes using macros and for extensive file handling. Searching is used to directly access a particular word or string in the file. Most macros center around searching for a string and then performing some edit changes in that region of the file. Command mode is also used to change the various VEDIT switches, parameters and tab positions. The edit pointer in command mode and the cursor in visual mode both serve a similar purpose. When entering visual mode, the cursor takes on the position in the text buffer of the edit pointer in command mode. When exiting visual mode to command mode, the edit pointer takes on the last position of the cursor.

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

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

## Word Processing with VEDIT

VEDIT can be used for two types of word processing. One is stand alone word processing in which the text is composed entirely with VEDIT and then printed out with either VEDIT or a simple print program. In this case the text will have to be formatted with VEDIT exactly the way it is to be printed out. This would include any page headings, page splits, centering of lines, and other details which VEDIT does not perform automatically.

The second type of word processing uses a "Text Output Processor" which takes care of page headings, centering of lines, justification and many other details. In this case VEDIT is used to create a file which contains the text and short command lines to the text output processor, which does the final printing. Most text output processors take commands which begin with a period "." in the first column of a line. All files created by VEDIT are standard text files and are therefore compatible with most text output processors.

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

## The Text Register

The purpose of the text register is to hold sections of text which are to be moved or copied to other positions in the file currently being edited.

The text register is empty when VEDIT is first invoked. The register is loaded by copying, moving or appending a portion of the main text to the register. This may be done in either command or visual mode, although it is usually easier in visual mode. The contents of the register may then be inserted at the cursor position in visual mode or at the edit pointer in command mode. The text register is not changed by any disk read/write operations. It can therefore be used to extract a section of text from one file and insert it anywhere in another file. Inserting a text register does not destroy or change the register. It may therefore be inserted repeatedly at different locations in the file.

The register acts as a scratch pad buffer in that it holds a temporary copy of text which is independent of the main text buffer. This is for the purpose of copying or moving sections or "blocks" of text from one area of the file to another, commonly referred to as "cut and paste" operations. Three operations are possible. One is to simply copy a section of the main text buffer to the register. The second is to move a section of text to the register, in which the section of text is also deleted from the main text buffer. Third, the register contents can be inserted anywhere within the main text buffer.

## Automatic Disk Buffering

Auto Disk Buffering refers to any disk file reading or writing which VEDIT performs automatically, without the user having given explicit read or write commands. (See also "Basic Editing Concepts" above). The simplest auto disk buffering (called "Auto-Read") 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. VEDIT can also perform more sophisticated disk buffering when editing very large files. This can be done in either the forward direction, "Forward Disk Buffering", or in the backward direction, "Backward Disk Buffering". The following headings describe these two types of auto-buffering.

If the text buffer fills up in visual mode while the user is typing in more text, VEDIT will attempt to write out 1K byte sections from the beginning of the text buffer to the output file. This is referred to as "Auto-Write". If the 1K section of text cannot be written out, either because auto buffering is disabled, or because the cursor is positioned within it, VEDIT will display the message "FULL"

on the status line. No more text can then be inserted, until the user explicitly writes some text to disk or allows VEDIT to automatically write it out.

While it is most convenient to normally have auto-buffering enabled, there are times when an experienced user will want to disable it. This can be done from command mode with the "ES" command. Generally, when explicit file reading and writing commands are being given, it is best to disable auto-buffering, which might read or write a file unexpectantly.

## Forward Disk Buffering

When VEDIT edits a file it reads text from the Input file into the Text Buffer, where it is edited, and writes the edited text to the Output file. For a small text file, the operation is quite simple. The entire Input file is initially read into the text buffer for editing. When editing is complete, the text buffer is written to the Output file. In order to edit files which are too large to fit into memory all at one time (i.e. files which are larger than 42 Kbytes in a 64K system), the procedure becomes more complicated. Only a portion of the Input file is initially read into the text buffer for editing. In order to edit the rest of the file, some of the text buffer must be written to the Output file, and then more of the Input file read in for editing. This must be repeated until the entire file has been edited.

Conceptually, it might help to consider the displayed screen a "window" into the text buffer. This "window" may be readily moved anywhere within the text buffer with the [PAGE UP], [PAGE DOWN] and other cursor movement functions. Furthermore, the text buffer may be considered a "window" into the file. Moving this text buffer window toward the end of the file is referred to as "forward disk buffering", and moving it back toward the beginning of the file as "backward disk buffering".

VEDIT's automatic forward disk buffering greatly simplifies the editing of these larger files. Forward disk buffering is usually performed automatically in visual mode and is almost invisible to the user (except for disk access time). The forward disk buffering takes place in visual mode when the user moves the cursor to the last page of the text buffer (by [PAGE DOWN], [ZEND], etc.), and all of the Input file has not been read. VEDIT will then attempt to read in more of the Input file and if necessary write text to the Output file. The minimum amount to be read from the Input file is determined by the user in Task 6.2 of the customization. This is referred to as "Minimum Transfer Kbytes". If this much free memory is available, the Input file is read until the memory is "nearly" full. "Nearly" is defined as leaving the number of bytes free that you specified in Task 6.1 of the customization. If this much free memory is not available,

"Minimum Transfer Kbytes" will be written from the beginning of the text buffer to the Output file, before more of the Input file is read. See also Appendix A, "Memory Size Dependent Parameters".

Forward disk buffering is only done automatically in visual mode if it was enabled during customization or with the "ES" command (the command is "ES 2 1$$", where "$" is the [ESC] key). It should normally be enabled. The disk buffering may also be controlled manually in the command mode with the "A" and "W" commands. Knowledge of these commands is not necessary for most applications, since the automatic disk buffering accommodates most needs.

Auto-buffering is only performed in command mode for the "N" command, since it might otherwise interfere with special editing applications. The auto-buffering allows the "N" command to search for a string throughout the entire file.

### Backward Disk Buffering

VEDIT's backward disk buffering augments the forward disk buffering to further simplify the editing of large files. It can also be performed automatically in visual mode in such a way as to be almost invisible to the user (except for disk access time). However, for best results, it must be used with some care, because you are more likely to run out of disk space. Although VEDIT always lets you recover from running out of disk space, it is more complicated if you are using backward disk buffering.

Occasionally, you may want to edit some text which has already passed through the text buffer and has been written to the Output file. Without backward disk buffering, you would have to restart the edit session from the beginning (with the "EA" command). The backward disk buffering, however, lets VEDIT read text from the Output file back into the beginning of the text buffer for further editing. However, before reading text back from the Output file, VEDIT needs to make space free in the text buffer. VEDIT does this by writing text from the end of the text buffer out to a temporary disk file. (The file name is VEDIT.REV. The text within the file is stored in reverse order, since the file is written from the end of the text buffer forward.)

Backward disk buffering uses additional disk space to hold the temporary VEDIT.REV file. As the Output file is written, disk space is also used up. Reading from the Input file does not free up any disk space, nor does reading back from the Output file. Without backward disk buffering, the maximum file size which may be edited is therefore 1/2 a disk, unless the Input and Output files are on different drives, in which case the maximum file size is a full disk. With backward disk buffering, the maximum file size is reduced to 1/3 a disk if everything is on the same drive, or else 1/2 a disk if the

Output file is on another drive. The VEDIT.REV file is always on the default drive. (With a 3 drive system you could safely edit a file one disk in length, by making the default, the Input and Output drives all different.) These file size limitations arise because in the worst case, VEDIT will need to create a VEDIT.REV file which is nearly as large as the Output file, which is generally as large as the Input file.

If you do use backward disk buffering and run out of disk space, you can still recover without loosing any edited text. The procedure is described below. To be on the safe side, we recommend that you customize VEDIT with backward disk buffering turned OFF and forward disk buffering turned ON. If while editing you decide you would like backward disk buffering, and are confident you have the disk space, you can turn it on the command mode command:

ES 2 2$$ where "$" is the [ESC] key

To calculate if you have enough disk space for backward disk buffering, perform a "STAT *.*" on the disk. If the amount of free space is twice the size of the file you wish to edit, you are usually safe (unless the Output file will be significantly larger than the Input file). You can include any ".BAK" version of the file to be edited in the amount of free space available. If the amount of free space is not at least equal to the size of the file being edited, you will encounter a disk full error even without backward disk buffering. It is always best be to sure that there is enough free disk space before editing a file.

If you are at least two-thirds through a large file and wish to begin editing from the beginning again, it will generally be faster to restart the edit session (with the "EA" command), rather than using backward disk buffering.

Before you decide that backward disk buffering is never worthwhile, let us say that it is very useful with large capacity disk systems such as 8" double density or Hard disks where there is usually plenty of free disk space.

## Section 2 - Visual and Command Modes Task Tutorial

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

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

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

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

### Invoking VEDIT

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

```
A>VEDIT LETTER.TXT<enter>
```

VEDIT will then read in the file "LETTER.TXT", or if you are creating the file, briefly display the message "NEW FILE". It will then normally go into the "Visual Mode" which displays the beginning of the file on the screen. The bottom line will contain the "Status Line" which consists mostly of dashes "-", and optionally the line and column numbers. Also visible will be the "Cursor" which indicates at

what position on the screen you are editing. It will initially be in the upper left hand corner. At this point you are ready to begin editing.

For the purposes of this tutorial it will be best if you begin by editing a file which already exists, instead of creating a new one. If you don't have any such files available, use a copy of the SAMPLE.TXT file on your master diskette.

## INVOKING VEDIT

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

## Keyboard Characters

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

## Editing Functions

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

## CURSOR MOVEMENT

| Operation | Command Sequence |
|---|---|
| Move cursor right | CURSOR RIGHT |
| Move cursor left | CURSOR LEFT |
| Move cursor up | CURSOR UP |
| Move cursor down | CURSOR DOWN |
| First character of current line | BACK TAB |
| Move cursor to next tab position | TAB CURSOR |
| Last character of current line | ZIP |
| First and last character of current line | LINE TOGGLE |
| First character of next line | NEXT LINE |

| Operation | Command Sequence |
|---|---|
| Top and bottom screen lines | SCREEN TOGGLE |
| Move cursor up by scrolling | SCROLL UP |
| Move cursor down by scrolling | SCROLL DOWN |

## PAGE MOVEMENT

Purpose: To rapidly access other regions of the file not currently displayed on the screen.

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

## Adding New Text

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

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

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

## Visual Functions

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

## Deleting Text

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

You can delete an entire line with the [ERLINE] function. You can also concatenate two lines by moving the cursor to the end of the first line and typing [DELETE]. Go ahead and try all of this, especially splitting lines with a <enter> and concatenating lines with a [DELETE].

Paragraphs or blocks of text are deleting by moving them to the text register and then emptying the text register.

## DELETING TEXT

| Operation | Command Sequence |
|---|---|
| Deleting characters backwards | BACK SPACE |
| Deleting characters forwards | DELETE |
| Erase from cursor to end of line | EREOL |
| Erase entire line cursor is on | ERLINE |
| Delete paragraphs and blocks of text | 1.) Position cursor over first character in the paragraph to be deleted. |
| | 2.) MOVE TO TEXT REGISTER |
| | 3.) Position cursor past last character in paragraph to be deleted. |
| | 4.) MOVE TO TEXT REGISTER |
| | 5.) MOVE TO TEXT REGISTER |
| | 7.) MOVE TO TEXT REGISTER |

## CORRECTING MISTAKES MADE TO A LINE

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

| UNDO |
|---|

This does not mean that, by putting the cursor on a previous line you changed, UNDO will give you the original line. It only affects the line that is currently being edited.

## REPEATING OPERATIONS

Purpose: It is often necessary to repeat an edit operation such as inserting the same character, deleting many lines, or moving the cursor many "pages" forwards or backwards. By using the [REPEAT] function key, it is possible to perform these repeated operations without having to type the same key over and over again. Pressing the [REPEAT] key once gives a repeat value of "4" (see status line). Pressing it again multiplies the value to "16", and finally "64". Any other value may be selected by typing in a number between 00 and 255. Once the repeat value is correct, simply type the desired key or control sequence which is to be repeated.

| Operation | Command Sequence |
|---|---|
| Delete four lines of text | REPEAT, ERASE LINE |
| Page forward by 16 pages | REPEAT, REPEAT, PAGE DOWN |
| Delete 30 characters | REPEAT, 3, 0, DELETE |

| | | | | |
|---|---|---|---|---|
| Insert 20 blank lines | REPEAT | 2 | 0 | <enter> |
| Insert 40 "*" characters | REPEAT | 4 | 0 | * |

Note: The number displayed on the status line will be blank for a repeat value of 00. Type in 000, and any character, if you decide that you don't want to repeat any operation after all.

## Indenting Text

If you don't want your text to begin in the first column, you can let VEDIT automatically indent your text with the [INDENT] and [UNDENT] functions. The section "Visual Mode - Indent and Undent" explains these functions, but it is easier to understand them through experimentation. Type a <enter> to start a new blank line, then press [INDENT]. Notice that the cursor has moved right by 4 spaces to column 5 (unless you have changed this parameter). Type a few words and another <enter>. This time the cursor will begin immediately in column 5. You have set the "Indentation Position" to column 5, and it will stay there until you increase it with another [INDENT] or move it back with [UNDENT]. VEDIT inserts the most Tabs and fewest spaces needed for the indent position. You can confirm this by moving the cursor over these leading Tabs and spaces. VEDIT only creates this indentation when you type <enter>.

### INDENTING TEXT BLOCKS

| Operation | Command Sequence |
|---|---|
| Increase the amount of Indentation. (Move left margin to the right) | INDENT |
| Decrease the amount of Indentation. (Move left margin to the left) | UNDENT |

To change Indent/Undent increment:

| | | |
|---|---|---|
| 1.) | Enter command mode | VISUAL<br>ESCAPE |
| 2.) | Issue command, where n= # of columns indented each time. EX: EP 3 4$$ will indent to 5th column, 9th column, etc. | EP 3 n$$ |
| 3.) | Enter visual mode again | V$$ |

## Moving and Copying Blocks of Text

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

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

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

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

## MOVING TEXT WITHIN THE FILE

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

2.) | MOVE TO |
| TEXT |
| REGISTER |

Message "1 END" on status line

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

4.) | MOVE TO |
| TEXT |
| REGISTER |

Message "TEXT" on status line

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

6.) | INSERT |
| TEXT |
| REGISTER |

NOTES:

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

2.) Following the text insert in step 5, the cursor is positioned at either the beginning or end of the inserted test depending upon ES command's switch 4.

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

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

## EMPTYING A TEXT REGISTER

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

1.) | MOVE TO TEXT REGISTER |

2.) | MOVE TO TEXT REGISTER | Type command key twice with cursor at same position to empty the register.

## SENDING TEXT TO THE PRINTER

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

2.) Position cursor at beginning of text block.

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

4.) Position cursor at end of text block.

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

6.) <CTRL-C> To stop the printing.

## ENTERING CONTROL CHARACTERS INTO THE TEXT

1.) | NEXT CHAR LITERAL |

This will enter the next character into the text, even a control character. You probably want to be in INSERT mode before this command.

2.) Type a control character which will be entered into the text at the cursor position.

## ENTERING COMMAND MODE

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

## SWITCHING FROM VISUAL MODE TO COMMAND MODE

Function to exit visual mode into command mode. "Edit Pointer" takes on last position of cursor.

| VISUAL ESCAPE |

Same as above, but does not abort any command, such as a global search.

| VISUAL EXIT |

## SWITCHING FROM COMMAND MODE TO VISUAL MODE

| | |
|---|---|
| Command to enter visual mode. The text register is preserved. Cursor takes on last position of command mode "Edit pointer". | V$$ "$" denotes the [ESC] key |

## SEARCHING AND SUBSTITUTING

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

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

## SAVE ALREADY EDITED TEXT AND CONTINUE

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

| | | |
|---|---|---|
| 1.) | VISUAL<br>ESCAPE | If in visual mode,<br>enter command mode. |
| 2.) | EA$$ | Write file to disk; same file will be used to continue edit session. |

## BEGIN EDITING NEW FILE

Purpose: It is not necessary to exit VEDIT and invoke VEDIT again from the operating system in order to edit another file. Since the contents of the text register is not lost when you begin editing another file from within VEDIT, it is very easy to copy or move a portion of one file to another.

| | | |
|---|---|---|
| 1.) | VISUAL<br>ESCAPE | If in visual mode,<br>enter command mode. |
| 2.) | EY$$ | Save current file on disk and empty the text buffer. |
| 3.) | EBnewfile.ext$$ | Begin editing the file "newfile.ext", which may be an existing file, or a file to be created. |
| 4.) | V$$ | Enter visual mode for full screen editing of the file. |

## MAKING MORE MEMORY SPACE

Purpose: When using the text register extensively, you may run out of memory space for performing the desired operations. This is usually indicated by a *BREAK* in command mode, or a "FULL" in visual mode. You should first try and empty out the text register. If this does not give you enough room, you can write out the first part of the text if it is already edited.

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

2.) | VISUAL<br>ESCAPE | Enter command mode.

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

4.) V$$ — Enter visual mode for full screen editing of the file.

## INSERT A LINE RANGE OF FILE 1 INTO FILE 2

Note: Both files must be on same disk.

1.) VEDIT file1 — Line number appears on status line; note upper and lower range of lines you want copied by positioning cursor on those lines. See ES command if line number does not appear on status line.

2.) EQ$$ — Exit VEDIT without writing the file to disk.

3.) VEDIT file2 — Edit the file in which you want the lines inserted.

4.) Position cursor where the lines should be inserted.

5.) | VISUAL ESCAPE | — Enter Command Mode.

6.) EGfile1[a,n] — Lines a -> n will be copied from file1 to file2 beginning at edit pointer (cursor) position. To copy an entire file, leave off the "[a,n]".

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

## CONCATENATING TWO FILES

Purpose: It is sometimes desirable to append one file to the end of another. This is readily done with VEDIT. In this example the text in file "file2" is appended to the end of the text in "file1" and the combined text is written to the file "file3". The three files can be on different disks.

Note: This assumes that the entire file 'file1' fits into memory.

| | |
|---|---|
| 1.) VEDIT | Invoke VEDIT without a filename. VEDIT will come up in command mode. |
| 2.) ERfile1$0A$$ | Setup the first input file for reading, and read it in. This assumes that the entire 'file1' fits into memory. |
| 3.) (optional)<br>EC$$ | Only needs to be done if the disk with 'file2' is not in one of the drives. After the EC$$, make sure that the disk with 'file2' and the disk to hold 'file3' are in the drives. |
| 4.) EWfile3$$ | Setup the output file which will hold the combined text. |
| 5.) ERfile2$0a$$ | Read the second input file. All of it does not need to fit into memory. |
| 6.) EX$$ | This writes out the complete file 'file3' and exits VEDIT. |

## SPLITTING A FILE INTO TWO OR MORE FILES

Purpose: VEDIT allows you to split a large file into several smaller ones. This example assumes that the splits are simple -- the front, middle and end sections of a large file are copied to their own files. In this example 'file1' is split into 'file2', 'file3' and file4'.

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

## RECOVERY FROM FULL DISK ERRORS

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

| | |
|---|---|
| 1.) EX$$<br><br>NO DISK SPACE<br>*BREAK* | You attempt to finish the edit session, but you get the disk full error. |
| 2.) EKoldfile$$ | Delete one or more old files on the disk. Don't delete any ".$$$" files or the "VEDIT.REV" file. |
| 3.) EX$$ | Continue finishing the edit session. |

-- OR --

| | |
|---|---|
| 2.) EF$$ | Save whatever was already written to disk. We will call this Part 1. |
| 3.) EWa:part2$$ | Setup to write the rest of the text to the file 'part2' on another disk drive, in this case drive "A:". |
| 4.) EX$$ | Write the rest of the text out to 'a:part2', and exit VEDIT. |

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

Note: See the section "Disk Write Error Recovery" if you do not have enough space on any drive to save all of the file. The procedure becomes more complicated, but you can still save all of your changes and additions to the file.

## Ending the Edit Session

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

### EXIT VEDIT TO CP/M:

1.) Exit Visual mode to Command mode.

```
VISUAL
ESCAPE
```

2.) Exit Command mode to CP/M .

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

## Section 3 - Visual Mode

### Properties

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

VEDIT's interruptable screen updating allows the screen to be updated in the fastest way possible when you are performing rapid screen changes. You do not have to wait for the screen to finish updating before you continue editing. Operations such as [PAGE DOWN] require the entire screen to be updated. If you type another [PAGE DOWN] while the screen is being updated, VEDIT will interrupt the unwanted update and restart to display the most current screen. VEDIT, therefore, does not necessarily update the screen in the order in which you perform edit changes. It will skip the intermediate screen displays and go directly the current screen display.

### Entering New Text

When a normal text 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 text, NORMAL and INSERT mode. When a text character is typed in NORMAL mode it appears at the cursor position and any character which was there is simply overwritten. In INSERT mode, no character is ever overwritten, but rather is squeezed to the right when a new character is typed at its position. In either mode, a new screen line, called a continuation line, is begun if the the text line becomes longer than the screen line. Visual functions exist to enter Insert Mode, revert to Normal mode, or to switch

between the modes. The mode which the editor starts in is set during customization and is a matter of personal preference.

Two convenient exceptions to the operation of Normal and Insert mode pertain to the ends of lines and the <enter> key. Text typed at the end of a line is always inserted before the (invisible) <CR> <LF> pair, which ends each text line. Also, typing the <enter> key does not overwrite any character, but rather moves the rest of the line beginning with the character at the cursor position to a new text line.

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

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

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

## Performing Edit Functions

The edit functions fall into two categories: Cursor Movements and Visual Functions. The cursor movement keys only move the cursor to some other position in the text and do not actually change the text. The cursor may be moved forward and backward by a character, a line, or a screen at a time. One position in the text may be "remembered" with an invisible marker which allows the cursor to be directly moved back to this position. These and other movements are individually described later in this section.

Some of the visual functions perform delete operations, while others change the Insert mode, change the indentation, manipulate the text register, and print text. Provided that the cursor has not been moved from the current line, the line can usually be restored to its original form (before any deletions or insertions were made) by using the [UNDO] function.

## The Repeat Function

Often it is desirable to repeat a typed character such as "*" or "-" when preparing tables, or to repeat an edit function such as [PAGE DOWN] in order to move quickly through the file. These can be performed by the the [REPEAT] function.

When the [REPEAT] key is pressed, a "4" will appear in the left side of the status line. This is the repeat value. Pressing [REPEAT] again will increase the value to 16, and pressing it again a value of 64. If you want any other repeat value, you can simply type it in, i.e. "70". Allowable values are between 00 and 255. (The number will be blank for a value of 00). Once the repeat value is correct, simply type the displayable character or the edit function which is to be repeated. For example, to create the top of a box consisting of 50 "*", type [REPEAT], "50" and "*". Or to delete 16 lines type [REPEAT], [REPEAT], [ERASE LINE].

With VEDIT's interruptable screen updating, only the final screen will be shown when using the [REPEAT] key. Since some operations such as deleting a line may take a second to perform on a very large file, you may notice some delay when using the "Repeat" function. (If you are deleting more than 20 lines, it may be quicker to use the "K" command in command mode.) While the repeated function is being executed, the cursor will remain on the status line. If for some reason you want to abort the repeated function, you can do so by typing [ESC] and the "Space" key. This is not customizable.

## The Tab Character

One displayable character which acts a little different is the [TAB CHARACTER], which is normally assigned to the Tab Key or <CTRL-I>. When the Tab key is typed, it inserts the tab character into the text, which is displayed with spaces to the next tab position. The tab positions are variable, but are normally set to every 8 positions. You can tell the difference between the tab character and spaces by the way the cursor moves over them. The cursor moves over each space individually, but moves over the Tab as a unit, i.e. a single [CURSOR RIGHT] might move you from column 1 to column 9. This reflects the fact that the Tab is a single character and should be treated as such. When the cursor is at the Tab character, it is displayed at the left side of the displayed spaces. If you wish to insert other characters before the Tab and leave the Tab in the file, you must be in the Insert mode. Otherwise the first character you type will overwrite the Tab and shift the rest of line left. The Tab character is commonly used when writing programs and aligning tabular data. Text paragraphs are best not indented by using a Tab, but rather typing four or five spaces.

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

Optionally, the [TAB CHARACTER] function can insert spaces to the next tab position. This is equivalent to you typing in the spaces. While this uses up more disk space and is not normally recommended, it is useful in some applications. This option may be changed with the "ES" command (the command is "ES 1 1$$"). Although the screen display for these two options is identical, they are actually very different, especially to programs other than VEDIT.

If you set the tab positions in VEDIT to anything other than the default, you may find that other programs will display your text unlike what you wanted. This is due to your VEDIT tab positions being incompatible with the tab positions of the other programs, which usually have fixed tabs at every 8 positions. If you send text files with tab characters to a large mainframe computer, you may find that the tabs are lost in the transfer. (Many mainframes do not have tab characters internally.) These two cases are good candidates for allowing the Tab key to insert spaces to the next tab position.

## Displaying Line and Column Numbers

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

## The Text Register

The most straight forward use of the text register is for cut and paste type operations. It can hold a section, which has been "cut" and needs to be "pasted" elsewhere.

The visual functions [COPY TO TEXT REGISTER] and [MOVE TO TEXT REGISTER] are used to copy or move text from the main text buffer to the text register. The function [INSERT TEXT REGISTER] is then used to insert the contents of a text register at the cursor position. These functions are usually used to move or copy text from one area in the file to another. The text registers used are the same as used in command mode, thus the text registers may be set in command mode and inserted in visual mode or vice versa.

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

Whether the beginning or the end of the text is first marked is actually unimportant. It is also immaterial whether you type [COPY TO TEXT REGISTER], [MOVE TO TEXT REGISTER] or even [PRINT TEXT] when you mark the first end of the text. Only when you mark the second end must the correct key be typed. You can abort the operation by typing the <ctl-minus> key in response to the status line prompt for a digit. If there is insufficient memory space for the text register, or to insert the register, the message "FULL" will appear on the status line and no text will have been moved or copied.

## Printing Text

VEDIT can print out any portion of the text which is currently in the main text buffer. This can be done from both the visual and the command modes of the editor. It is easiest to do in visual mode and is similar to the method of moving text to the text register. First the cursor is positioned at one end of the text to be printed and the [PRINT TEXT] function key pressed. (This is ESC - P in the example keyboard layout). Next, the cursor is positioned at the other end of the text to be printed and the [PRINT TEXT] pressed again, which causes the text to be printed. To print the entire text move

the cursor to the beginning and end of the text with the [HOME] and [ZEND] functions, and type [PRINT TEXT] at each end. The printing can be aborted by typing a CTRL-C). Many printers use control characters or escape sequences to control such things as character size, font style and overstrike. These special control sequences can be imbedded directly into the text using the [NEXT CHAR LITERAL] function or the "EI" command.

## Inserting Control Characters

VEDIT can insert control characters into the text. These may be special printer control characters, the [ESC] character, or control characters for other purposes. Only CTRL-Z cannot be inserted because it is used by CP/M to mark the end of a file. The [NEXT CHAR LITERAL] function places the next character typed on the keyboard into the text. Any control character which can be generated from the keyboard can be placed into the text. In case a character must be inserted which cannot be generated from the keyboard, the command mode "EI" command can be used. This command can insert any character with a decimal value between 00 and 255 (except CTRL-Z, decimal 26) into the text.

## Indent and Undent Functions

As an aid in word processing and writing programs in structured languages such as Pascal, PL/I and C, the visual mode has the [INDENT] and [UNDENT] functions. These functions allow the editor to automatically pad up to the "Indent position" with tabs and spaces, when a new line is started with the <enter> 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 <enter> 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 <enter> 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 <enter>, 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 will use up lots of memory and disk space.

## Lower to Upper Case Conversion

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

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

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

Upper and lower case letters can also be reversed, i.e., lower case converted to upper case and upper case converted to lower case. This reversal is done immediately when a keyboard character is received and before any resulting lower case letter is converted to upper case as described above. The letters are also reversed for the command mode. This feature may be handy when most text is to be entered in upper case, but where an occasional lower case character is also needed. This mode is selected with the "ES" command.

## Disk Buffering in Visual Mode

In visual mode, the disk buffering can perform automatic Read and Write to handle files which are larger than the size of available main memory. Specifically, if the current screen display reaches the end of the text buffer, and the entire input file has not yet been read, Forward Disk Buffering is performed.

Auto-buffering in the backward direction is performed when the cursor is at the beginning of the text and you type [HOME]. Therefore, typing [HOME] [HOME] from anywhere in the text will perform backward disk buffering, reading back text which has already been written to the Output file. Nothing will happen if there is no text to read back in, or if backward disk buffering was not enabled during customization or with the "ES" command. The amount that VEDIT will buffer backwards is set by Task 6.2 during the customization. This is generally between 4 and 14 Kbytes. To move back further in the file, just repeat the [HOME] function.

VEDIT will also begin to write out the text buffer (auto-write) if the memory becomes full while the user is typing in more text. At this point VEDIT will attempt to write the first 1K text bytes 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 on the status line. Both the auto-buffering and the auto-write may be disabled by the "Auto Buffering in Visual Mode" switch. The command "ES 2 0" will turn off all auto buffering. The command "ES 2 1" will enable forward auto buffering. The command "ES 2 2" will enable both forward and reverse auto buffering.

## Details of the End of Lines

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 text lines. Typing the <enter> (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>. Although VEDIT, in visual mode, will never create a line ending in just a <CR> or <LF>, such lines are handled in visual mode, although displayed differently. (Such lines can be created in command mode). If a line ends in only a <LF>, the next line will be displayed with a starting position directly below the end of the previous line. If a line contains a <CR> not followed by a <LF>, the character following the <CR> will be displayed in the reserved column of the same screen line and the rest of the characters will overwrite previous characters. (This is not very eloquent, but is also what most terminals will do.) Such lines may be corrected by deleting the offending lone <CR> or <LF> with the [DEL] key and then inserting the <CR> <LF> pair with the <enter> key.

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

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

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

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

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

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

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

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

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

[LINE TOGGLE] Is a combination of [ZIP] and [BACK TAB]. First moves the cursor to the end of the text line. If it already is at the end of a line, it moves to the beginning of the screen line.

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

[SCROLL UP] Similar to [CURSOR UP], except that the cursor remains on the same screen line and the screen moves down instead.

[SCROLL DOWN] Similar to [CURSOR DOWN], except that the cursor remains on the same screen line and the screen moves up instead.

| | |
|---|---|
| [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. |
| [SCREEN TOGGLE] | Move the cursor first to the last allowed screen line, or if already there, to the first allowed screen 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, the following sequence will delete an entire line:

[BACK TAB] [EREOL] [EREOL].

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

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

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

[REPEAT] The next text character or edit function is repeated. This is either a multiple of 4 or a number typed in. Type "000" and any character to abort.

[INDENT] This increases the "Indent Position" by the "Indent Increment". The editor will then automatically pad with tabs and spaces to the Indent position following each <enter>. 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 "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 saved, and 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. If insufficient memory space exists no copy is made 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] The text register's contents are inserted at the current cursor position. The register itself is not changed. If there is insufficient memory space for the entire "copy", nothing is inserted and the "FULL" message will appear on the status line. Moving the cursor to another line will clear the "FULL" message.

[PRINT TEXT] This is activated similar to the [COPY TO TEXT REG]. The block of text is printed on the CP/M listing device. A CTRL-C will abort the printing.

[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. The text register is preserved. Depending upon the value of the "Clear screen on visual exit" switch, the command prompt will appear either on a clear screen or just below the status line.

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

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

# Section 4 - Command Mode

## Properties

In command mode the user enters command lines, which consist of single commands, concatenated commands or iteration macros. Each command line is ended by typing the ESC key twice, at which point the command line is executed. The ESC is also used to delimit search strings and file names. In the event that your keyboard does not have an <ESC> key, you may customize the command mode escape character to be any other control character.

Each character typed is echoed by VEDIT and none are processed by CP/M. Thus a <CTRL-C> has a different meaning in VEDIT and does not cause a return to CP/M. The ESC key 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 <CTRL-C>. This causes a *BREAK* and a new command prompt "*" to be displayed. VEDIT checks for the <CTRL-C> before any new command is executed, during the execution of the "A", "F", "N", and "T" commands, and in a few other situations.

See the section "Getting Started" for the procedure to switch between visual and command modes.

## Command Mode Notation

| | |
|---|---|
| $ | denotes the <ESC> control character. Wherever "$" appears in a command mode example, type the <ESC> key. |
| <TAB> | represents the TAB character. |
| <CR> | represents the carriage return charater (<enter>). |
| <ESC><br><CTRL- > | represents the ESC key or alternate command mode escape character selected during customization. Other control characters produced by holding the CTRL key and typing a letter are represented by "<CTRL-letter>". |
| [<br>] | The bracket characters used for iteration macros are printed as "[" and "]" in this manual. Some users may be more familiar with the angle brackets "<" and ">". You can choose which characters to use during the customization process. |

## Editing a Second File

When you are done editing one file and need to edit another file, it is not necessary to exit VEDIT and then re-invoke VEDIT for the second file. The "EY" command makes it easy to write out the file being edited and close it, in preparation for editing another file. It performs the equivalent of the "EX" command without leaving VEDIT. The command to finish editing a file and begin editing another file (NEWFILE.TXT) is:

```
EY$EBnewfile.txt$$
```

## Search Options and Special Characters

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

The commands "F", "N", "S" and "I" are followed by a text string, which is normally delimited with an <ESC>. An option allows an explicit delimiting character to begin and end the text string. With

this option, the character immediately following the "F", "N", "S" or "I" command is the delimiter. Any character can be the delimiter, but "/" is a good choice. Note that the text string itself cannot contain the delimiting character. This option can be invoked by preceding the command with a "@". For examples, the commands on the left side are equivalent to those on the right.

```
Fspeled$V$              @F/speled/V$$

Sspeled$spelled$v$$     @S/speled/spelled/V$$

4Fpoint$V$$             4@F:point:$$

Ia new line$$           @I/a new line/$$
```

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

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

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

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

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

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

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

The literal character <CTRL-Q> operates similar to the [NEXT CHAR LITERAL] in visual mode, in that the next character is treated

literally and not interpreted. This is the only way to search for characters such as <CTRL-R>, <CTRL-U> and <CTRL-H> which are also used for line editing. It is also an alternate way to search for the <ESC> character. For example, the following examples insert text containing a <CTRL-H> and search for the same text:

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

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

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

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

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

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

## Iteration Macros

An iteration macro allows a group of commands to be repeated with or without user intervention as many times as desired. They are most useful in searching and replace tasks (changing all instances of a misspelled word, for example).

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

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

The iteration macro operates by executing the first command of the group through the last command, and then starting over again with the first command. The entire group will be executed the number of times specified by the iteration count. If no explicit iteration count is given, it defaults to "#" (32767) which signifies "forever" or "all".

It is very important to observe the placement of any necessary <ESC> to terminate strings and filenames when using iteration macros. For example, "BF word$$" needs no "$" between the "B" command and the

"F" command, but in "S name$ smith$V$$", the "$"s are necessary after " name" and after " smith". The following example changes the first occurrence of " teith" to " teeth]", which is not the intention.

Wrong: 3[S teith$ teeth]$$

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

Iteration Counts:

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

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

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

Using Visual Mode in Iteration Macros:

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

[Ftemporary$V]$$

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

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

The Visual Mode has two ways of exiting back to Command Mode in order to help in using iteration macros. The [VISUAL EXIT] simply exits and lets any iteration macro 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:

```
Wrong:      [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:

```
Right:      [S and$ or$v]$$
```

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

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

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

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

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

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

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

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

An iteration will continue until its iteration count is exhausted or until an error occurs. A common error is an unsuccessful search operations. In fact, many iterations will normally stop with an unsuccessful search error message. A special situation concerns using search commands ("F" and "S") in iteration macros where search error suppression is enabled. In this case, when a search is unsuccessful,

no error is given, but the iteration is stopped, and execution continues with the command following the iteration. This may be an outer level iteration. Recall that the commands "#S" and "#F" are only unsuccessful if no occurrences are found.

## Text Register

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

| | |
|---|---|
| 35P$$ | Copy the next 35 lines to the register. |
| -6P$$ | Copy the previous 6 lines to the register. |
| 0P$$ | Empty out the register. |

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

| | |
|---|---|
| G$$ | Insert the register at edit pointer. |

## Printing Text

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

| | |
|---|---|
| Z0EO | Print entire text on line printer. |

## Disk Buffering in Command Mode

While the disk buffering can be fully automatic in visual mode, it is not done automatically in command mode because it would almost surely interfere with the explicit file handling often done in command mode. VEDIT has a full set of commands for reading and writing files. Commands must be issued in order to read a file, write a file and perform forward disk buffering. In some cases it will be easier to switch into visual mode and allow it to perform the disk buffering automatically.

The "ER" command opens a file for reading, but does not actually read anything in. The file can be read with the "nA" command. Similarly, the "EW" command opens a file for writing, but does not write anything out. Text can be written out with the "nW" command. Forward disk buffering in command mode, therefore, requires successive "W" and "A" commands.

Some commands perform automatic reading/writing when invoked. The "EB" command performs an auto-read which reads in the entire file from disk or until the text buffer is nearly full. The "EY" command performs all the reading and writing to finish editing and saving a file without leaving the editor. The "N" command can perform forward disk buffering to find occurrences of a string anywhere in the file. These commands operate regardless of the setting of the "Auto Buffering in Visual Mode" switch.

As described earlier, backward disk buffering is accomplished by writing text from the end of the text buffer to the temporary "VEDIT.REV" file, and reading back text already written to the Output file. VEDIT can perform this disk buffering automatically in visual mode. Commands are also provided for you to do this manually in command mode. Because of the complexity of these commands, we suggest you not use them until you are thoroughly familiar with all other aspects of VEDIT's file handling.

The "-nA" and "-0A" commands allow text which has already been written to the Output file to be read back into the text buffer. "-nA" will read "n" lines back from the output file, or until the text buffer is full, or the output file is empty. "-0A" will read lines back until the text buffer is "nearly" full or the output file is empty.

The "-0W" command will write text from the current edit pointer (cursor position) to the end of the text buffer out to the temporary VEDIT.REV file. The "-0W" command will also open the VEDIT.REV file if necessary. Its only purpose is to make more memory space available for performing the "-nA" command, or any other time you need more memory space. Note that there is no "-nW" command.

Whenever an "A" command is issued or VEDIT performs auto disk buffering, VEDIT will always read the contents of the VEDIT.REV file back into the text buffer, before reading any more from the Input file. You, therefore, do not need to explicitly remember whether or not there is any text in the VEDIT.REV file.

## Disk Write Error Recovery

Since most CP/M systems run with floppy disks which have limited storage capacity, the typical user will occasionally encounter a "Full Disk" error condition. This is caused by either running out of disk space, leading to the error message "NO DISK SPACE", or running out of directory space, leading to the error message "NO DIR SPACE". Fortunately, VEDIT allows you to recover from these errors using one of two recovery procedures. One is to delete files from the disk using the "EK" 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 operation. The following paragraphs describe these procedures in some detail, and an example is given in the Tutorial.

The best policy is to avoid "Full Disk" errors by making sure that there is enough space before your begin editing. If you are editing files more than 1/3 disk in length, it is best to read the Input file from one drive and write the Output file on another drive. For example, if the Input file and VEDIT are on drive A and the disk in drive B is blank, give the command:

```
VEDIT infile.ext b:outfile.ext
```

The simplest and most common recovery is to delete files from the disk which is full. If you keep your disks fairly full, you might find it useful to keep a recent DIRectory or STAT listing handy to check for files that may be deleted. If you find files which you can delete, you are all set. You can then re-issue the command which led to the full disk error. Any ".BAK" files can usually be deleted. You can also consider deleting any files which you know are backed-up on other disks. Never delete the ".$$$" and "VEDIT.REV" files from within VEDIT. (You can delete them from the operating system, in the unlikely event they appear on the directory there.)

If you are still reading this in order to learn more about VEDIT, STOP. You are very unlikely to ever require the following procedures. They are described here for completeness only.

There may be times when you cannot delete enough files to finish the edit session. You then have several alternatives. One is to close the current output file (with the "EF" command) and create a second output file on another drive. An example is:

```
EF$$              Close the current output file.
EWA:PART2$$       Create an output file on another drive.
EX$$              Exit the edit session.
```

You can then use VEDIT or PIP to merge the two partial output files back into one file. (See the Tutorial for merging files.)

If all the disks in the drives are full, you will have to either change disks using the "EC" command or delete the Input file. In either case you want to read as much of the Input file and hope that there is enough room to read all of it. Begin by issuing the command:

```
#A$$
```

Then look at the end of the text buffer to see if all of the file was read in. If not, the recovery will be more complicated. If all of the Input file has been read, it is often simplest to delete this Input file from disk with the "EK" command. This will make enough space available for the rest of the Output file. If you delete the Input file, there will be no ".BAK" backup file when you exit VEDIT. For example, if the file you are editing is "LETTER.TXT", you could give the following commands:

```
EKletter.txt$$     Delete the Input file.
EX$$               End the edit session.
```

Alternately, if you need to keep the original Input file, you can use the "EC" command to change disks and write the second part of the Output file to an empty disk. First issue the "EF" command to close the current Output file. The "EC" command will allow you to insert another (empty) disk into any drive. Example commands are:

```
EF$$               Close the current output file.
EC$$    <enter>
EWPART2$$          Create a second output file.
EX$$               End the edit session.
```

You will then have to use VEDIT or PIP to merge your two Output files back into one file. This procedure has several potential shortcomings. If you were using backward disk buffering, you may get the error message "REV FILE OPEN", in which case you cannot change any disks. You will then have to make more space on the existing disks by deleting files, possibly the Input file.

If you were unable to read the entire Input file into the text buffer, the procedure becomes still more complicated. (Try again to make more space free on the existing disks!) If you have a copy of your Input file on a backup disk, delete the Input file, which should free enough disk space to end the edit session. All text which you just edited or entered will be in the Output file, but the Output file will be missing the last portion of the Input file which was never read in. You must examine the Output file to see how much is missing. Then copy your backup of the original Input file to a blank disk. Edit this file by deleting the entire front portion up to the text which is missing from the partial Output file. Exit VEDIT. Then use VEDIT or PIP to merge the Output file and the unread portion of the original Input file back together. This is a complicated procedure, but at least none of your edited text is lost.

If in the previous paragraph you know that you don't have a backup copy of the Input file, you will have to use the "EC" command procedure to write a second Output file to a blank diskette. However, by using the "EC" command, VEDIT will not be able to continue reading any unread portion of the Input file. You will therefore have to merge the Output file from the original disk with the second Output file, with the unread portion of the Input file.

If you cannot change disks because you were using backward disk buffering, you will have to make more space free on the existing disks by deleting files. If you want to avoid the complexities of deleting the Input file, you can delete any ".COM" or ".CMD" files, including VEDIT, which you can probably restore from a backup disk.

## Command Line Editing

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

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

If you wish to search for one of these characters in the text, or use one within any other string, you must precede it with a <CTRL-Q>. <CTRL-Q> causes the following character to be taken literally, and not be interpreted as a line editing character, a <CR> or any other special character.

'n' denotes a positive number. (# represents 32767)
'm' denotes a number which may be negative to denote backwards in the text buffer.
'string', 's1', 's2' and 'text' denote strings which may include the <enter> key in them. May use explicit terminators, or else must end in <ESC>.
'file' is a disk file name in normal CP/M (MSDOS) format with optional disk drive and extension. Any leading spaces are ignored. Must be ended with an <ESC>.

---

| | |
|---|---|
| nA | Append 'n' lines from the input file to the end of the text buffer. "0A" performs an auto-read. |
| -nA | Read back 'n' lines from the Output file. "-0A" reads back until the text buffer is nearly full. |
| B | Move the edit pointer to the beginning of the text buffer. |
| mC | Move the edit pointer by 'm' positions. |
| mD | Delete 'm' characters from the text. |
| E | First letter of extended two letter commands. |
| nFstring<ESC> | Search for the 'n'th occurrence of 'string' in the current text buffer and position the edit pointer after it. Only the first 32 characters of 'string' are searched. |
| 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. The edit pointer is moved past 'text'. |
| mK | Kill (delete) '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 perform auto-disk buffering to 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 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. |
| -0W | Write all lines from the edit pointer to end of text buffer to the "VEDIT.REV" file. This makes more memory space free. |
| Z | Move the edit pointer past the last character in the text buffer. |

SPECIAL CHARACTERS
---

| | |
|---|---|
| \| | The search wildcard character. Each "\|" will match any character in the text being searched. For "F", "N" and "S" commands. |
| <CTRL-Q> | Literal character. Next character, usually a control character, is taken literally and not interpreted. Allows searching and inserting of control characters including line editing characters, <CR> and <ESC>. |
| @ | Immediately precedes "F", "I", "N" or "S" to indicate that explicit terminating characters are being used. |
| : | Immediately precedes "F", N" or "S" to indicate that search error messages are to be suppressed. |
| # | Represents the maximum positive number 32767. It is used to signify "forever" or "all occurrences of". |

## EXTENDED COMMANDS

**EA** — Saves the file being edited on disk and then edits the file again from its beginning. Similar to "EY" followed by "EB" command.

**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. Used for write error recovery, or just to edit files on other disks.

**EF** — Close the current output file.

**EGfile[line range]** — Insert the line range of the file "file" into the text buffer at the edit pointer. If no line range is specified, the entire file is inserted.

**nEI** — Insert the character whose decimal value is "n" into the text buffer at the edit pointer. The value "26" is not allowed since this is the "End of File" marker. Values of 128 to 254 are allowed.

**EKfile** — Erase (kill) the file "file" from the disk. This is intended for making more space free on the disk.

**mEO** — Send 'm' lines to the line printer. "0EO" prints from the beginning of the text buffer to the current line.

**EP n k** — Change the value of parameter "n" to "k". Currently there are the following parameters:

| | | |
|---|---|---|
| 1 | Cursor type (Mem Mapped Only) | (0, 1 or 2) |
| 2 | Cursor blink rate (Mem Mapped Only) | (5 - 100) |
| 3 | Indent Increment | (1 - 20) |
| 4 | Lower case convert | (0, 1 or 2) |
| 5 | Conditional convert character | (32 - 126) |
| 6 | Display line and column position | (0 - 3) |

(0 = none, 1 = line, 2 = column, 3 = both)

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

ET — Set new tab positions. The ET is followed by up to 30 decimal numbers specifying the tab positions. Since the positions start at 1, the normal positions would be: 9 17 25 33 etc.

EV — Print the VEDIT version number.

EWfile — Open the file "file" for output. Any existing file by that name will be renamed to "file.BAK" following an EF or EX. Gives error if an output file is already open.

EX — Exit back to CP/M after writing the text and any unappended part of the input file to the output file. Gives error if no output file is open.

EY — Finishes editing a file by writing the entire text buffer and any remaining portion of the Input file to the Output file and closing it. Usually followed by an "EB" command.

nA Append
-- ------

Example: 100A$$ 0A$$ -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.

The special forms "-nA" and "-0A" will read back 'n' lines from the Output file into the beginning of the text buffer. "-0A" reads all of the Output file back or until the text buffer is almost full. Nothing is read back if there is no Output file or it is empty.

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
Automatic Disk Buffering

Examples: ERTEXT.DOC$$

| | |
|---|---|
| 0A$$ | The file 'TEXT.DOC' is opened and all of the file is read in, or until the memory is almost full. |
| -0A$$ | Read as much of the Output file as will fit back into the beginning of the text buffer. |

## 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 or backward disk buffering to move back to the beginning of the text file.

Notes:

See Also: Commands: EA, Z
Backward Disk Buffering

Examples: B12T$$ Moves the edit pointer to the beginning of the text buffer and types 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. The extended commands are described later in this section.

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

See Also: Extended commands.

Examples:

| nFs1<ESC> | Find |
|---|---|

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

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

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

See Also: Command: N

Examples:

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

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

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

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

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

## 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 Register

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. |
| 132P$132K$$<br>EA$$<br>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$$ I<CR>new line$$

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

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

See Also: Commands: EI

Examples:

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

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

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

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

mK Kill

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

Description: This command performs a line oriented deletion (or killing) of text. A positive 'm' deletes all characters from the current edit pointer and up to and including the 'm'th [LF]. Hence the command "#K$$" deletes all text from the cursor to the end of the text buffer. A negative 'm' deletes all characters preceding the edit pointer on the current line and the 'm' preceding lines. 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 all text following the edit pointer in the text register and then deletes it from the buffer.

mL Lines

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

Description: This command performs a line oriented movement of the edit pointer, always leaving it 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 is 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 type out every line that was changed.

| nNsl<ESC> | Next |
|---|---|

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

Description: This command is very similar to the "F" command, except that if the 'n'th occurrence of 'sl' is not found in the text buffer, forward disk buffering is 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 given. The edit pointer is positioned very similarly to the "F" command. However, with the "N" command it is possible that the 'n'th occurrence is not found, and that the previous occurrence is no longer in the text buffer due to auto-buffering. In this case the edit pointer is positioned at the beginning of the text buffer. Using this command with a search string, 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. The error "NO OUTPUT FILE" occurs if no output file is open for performing forward disk buffering.

See Also: Command: F
Auto Buffering

Examples:

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

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

Nxxxx$$ Accesses the last part of the file, assuming the string "xxxx" 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 registers decreases the amount of memory available to the text buffer. Thus the "0P" command should be given when the text in the register is no longer needed. This command does not change the text buffer. If there is insufficient memory space for the text copy, the text register is only emptied, nothing is saved in it and an error is given. The saved text is inserted in the text buffer with the "G" command or in Visual mode.

Notes:

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

Examples:

| | |
|---|---|
| 120P$120K$$ | The text lines are saved in the text register and are then deleted from the text buffer. |
| -23T$$<br>-23P$$ | The text lines are typed 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 given 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 given. The edit pointer is positioned after 's2', if 's1' is found, or else is left at its original position if 's1' is not found. For the command form "#Ss1<ESC>s2<ESC>, an error is only given if no occurrences of 's1' are found. See the "N" command example on how to perform a "substitute" if all of the file is not in the text buffer.

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

See Also: Commands: F, N, I

Examples:

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

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

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

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

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

mT Type
-- ----

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

Description: This command types out (displays) 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 typed out. If 'm' is 0, only the characters on the present line preceding the edit pointer are typed out. Fewer than 'm' lines will be typed out if either end of the text buffer is reached. Note that "0TT" will display 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 displaying selected lines. Visual mode should be used for looking at sections of a file.

Notes:

See Also:

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

U Unused (Free Memory)
- ----------

Example: U$$

Description: This command displays 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 small buffers all use the same memory space. If the number of free bytes goes below 260, the "FULL" flag will be set when in visual mode.

See Also:

Examples:

## V Visual

Example: V$$

Description: This command enters Visual Mode. The visual cursor position will be set to the edit pointer position. Upon return to the command mode (by [VISUAL EXIT] or [VISUAL ESCAPE]) the edit pointer will be set to the cursor position.

Notes: The text register is preserved.

See Also: Visual Mode

Examples: Fhere$V$$ Finds "here" and enters visual mode.

## nW Write

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

Description: This command writes 'n' lines from the beginning of the text buffer to the output file and deletes them from the buffer. If there are less than 'n' lines in the buffer, the entire buffer is written out and deleted. If 'n' is zero, the entire buffer up to the line the edit pointer is on, is written out. The edit pointer is moved to the new beginning of the buffer. If no output file is open, an error is given and nothing is done. The output file can be opened with an "EW" or "EB" command or when VEDIT is invoked.

The form "-0W" writes the text buffer, from the line the cursor is on to the end of the text, to the temporary VEDIT.REV file, creating the file if necessary. This is primarily used to free up more memory space before backward disk buffering using the "-0A" command.

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

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

Examples:

```
EWpart1.txt$$   The first 24 lines of the text buffer
24W$$           are written out to file "PART1.TXT" and
EF$$            the rest of the text buffer is written
EWpart2.txt$$   out to file "PART2.TXT" and the edit
EX$$            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$$ | Positions the edit pointer to the 100th line before the end of the text buffer. |
| Z-12T$$ | Types out the last twelve lines in the text buffer. |
| Nxcxc$Z-12T$$ | Types out 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. (Backward disk buffering may be more convenient in some cases.) This is especially true when a block of text is to be moved from the rear of a large file to the front. The contents of the text register is not affected by 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, Backward Disk Buffering

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 displayed. If an output file is still open, an error is given and the command has no other effect.

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

See Also: Commands: W, ER, EW

Example:

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

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

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

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

## EC Edit Change (Disk)

Example: EC$$

Description: This command must be given before the user attempts to change any logged-in disks in order to recovery from a disk write error, or to read files from another disk. An error is given 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 "EK" 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 an [<enter>] 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: EK, EF
Disk Write Error Recovery.

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

## 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, EY

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

## EGfile[line range] Edit Get (File)

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

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

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

See Also: Commands: A, ER

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

## nEI Edit Insert

Example: 12EI$$

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

Notes:

See Also: Commands: I

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

92EI$$ A "\" is inserted into the text with the EI command.

## EKfile<ESC> Edit Kill

Example: EKfile.txt$$ EK????????.bak$$

Description: This command will erase (kill) 'file' from the disk. This is the easiest method of freeing disk or directory space to recover from a disk write error.

Notes: Never erase any ".$$$" files or the "VEDIT.REV" file from within VEDIT! These are the temporary files VEDIT is using. Don't delete the input file until all of it has been read into memory.

See Also: Commands: EC
Disk Write Error Recovery

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

EK????????.bak$$ Deletes all files with an extension of ".BAK" from the default drive.

mEO Output to Printer

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

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

Notes: The print out can be stopped by typing CTRL-C in CP/M.

See Also: Commands: T, RP
Printing Text from visual mode

Example: Z0EO$$ Prints the entire text buffer and places the edit pointer at the end of the text.

EP n k<ESC> Edit Parameters

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

Description: This command changes the value of parameter 'n' to 'k'. Currently there are 6 parameters. The numbers are specified in decimal and separated by spaces or commas. The default values of these parameters are determined during the customization process. An error is given if 'n' is specified out of range. The parameters are:

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

Parameter (1) determines the type of cursor displayed in visual mode. The cursor types are: 0=Underline, 1=Blinking Reverse Video Block, 2=Solid Reverse Video Block.

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

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

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

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

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

Notes: The parameter values are specified in decimal.

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

Examples:

| | |
|---|---|
| EP 3 6$$ | This sets "Indent Increment" to six. |
| EP 1 1$$ | Sets visual cursor to blinking reverse block. |

## EQ Edit Quit

Example: EQ$$

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

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

See Also: Commands: EA

Example:

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

| ERfile<ESC> | Edit Read |
|---|---|

Example: ERnewfile.txt$$

Description: This command opens the file 'file' for input (reading). Nothing is read into the text buffer with this command. The "A" command or an auto-read is used to actually read the input file. If the same file was already open for input, the file is "rewound", so that the file can again be read from the beginning. An error is given 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 9 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 2=BACK) |
| 3 | Start in visual mode | (0=NO 1=YES) |
| 4 | Point past text reg. insert | (0=NO 1=YES) |
| 5 | Ignore UC/LC distinction in search | (0=NO 1=YES) |
| 6 | Clear screen on visual exit | (0=NO 1=YES) |
| 7 | Reverse Upper and Lower case | (0=NO 1=YES) |
| 8 | Suppress search errors | (0=NO 1=YES) |
| 9 | Use explicit string terminators | (0=NO 1=YES) |

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

Switch (2) determines whether auto-buffering is enabled in visual mode. "0" disables auto-buffering, "1" enables only forward disk buffering, and "2" enables both forward and backward disk buffering. We recommend a default value of "1". Before using "2", make sure you have sufficient free disk space. Use "0" when you are giving explicit Read/Write commands. This will prevent unexpected disk read and write from occurring while you are editing in visual mode.

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 command and visual mode.

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

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

Notes:

See Also: Customization, Visual Mode

Example: ES 1 1$$ This causes 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 positions 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 displayed 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 are:

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 displays 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 given 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 way to save the file being edited on disk and exit VEDIT. It writes the entire text buffer to the output file, followed by any unread portion of the input file, closes the output file and exits VEDIT. All file backup and renaming is done as with the "EF" command. The error "NO OUTPUT FILE" is given if no output file is open. The error "NO DISK SPACE" results if there is insufficient disk space to save the entire file.

Notes: In case of a "NO DISK SPACE" or "NO DIR SPACE" error, see the heading "Disk Write Error Recovery" for the procedure to save your file.

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

Example:

```
VEDIT FILE.TXT  The editor is invoked in the normal way
V$$             to edit a file in visual mode. The new
EX$$            file is then saved on disk.
```

## EY Finish Edit Session

Example: EY$$

Description: This command writes out the file being edited to disk and closes it, in preparation for editing another file. First the text buffer is written to the Output file. Any unread portions of the input file at then transferred to the Output file and the Output file closed. All file backup and renaming of files is done as with the "EF" command. This command is equivalent to "EX", but without leaving the editor. The error "NO OUTPUT FILE" is displayed if no output file is open.

Notes: See notes for EF and EX commands.

See Also: Commands: EX, EF

Example:

```
EF$$
EBnewfile.txt   The current file is saved on disk, and
                editing on a new file "NEWFILE.TXT"
                initiated.
```

# CUSTOMIZING VEDIT

## WHAT IS CUSTOMIZATION?

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

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

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

## WHEN IS CUSTOMIZATION NECESSARY?

VEDIT is supplied already customized and ready to run for P&T CP/M 2. You may begin to use it at once. The keyboard layout for this ready to use version is given in Appendix E. After using VEDIT for a while you may wish to change some default parameters or try a new keyboard layout. The greatest benefit you receive from the customization process is probably the ability to determine your own keyboard layout so you can accommodate your personal preferences.

You will find four disk files on your master diskette which relate to VEDIT. VEDIT.COM is a preconfigured version that is ready to run. VEDIT.SET is an unconfigured version that you may configure if you want a different configuration. VEDSET.COM is the customization program that allows you to configure VEDIT. SAMPLE.TXT is a sample text file that you may practice on when learning how to use VEDIT.

You may customize a copy of VEDIT as many times as you want. If you are changing VEDIT's configuration, you will probably customize it several times until you have everything "just right". You can of

course also create several configurations of VEDIT, each for a special application. To help remind you of which configuration you are using, you can create a custom signon message for each, which will be displayed when VEDIT is invoked.

## HOW TO PERFORM CUSTOMIZATION

### STEP 1 - ENTER COMMAND SEQUENCE FROM OPERATING SYSTEM

In order to customize VEDIT you will need the files VEDSET.COM and VEDIT.SET. To begin customization, type a command line like the following:

```
VEDSET VEDIT Newfile
```

where "Newfile" is the name you want the configured version of VEDIT to have. Typical choices would be "VEDIT" or "EDIT". You need not type the ".SET" and ".COM" extensions on the file names.

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

```
VEDSET OLDVEDIT.COM NEWVEDIT
```

where "OLDVEDIT" is the VEDIT you want to change (you need to type the ".COM" here), and "NEWVEDIT" is the name of the new VEDIT.

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

### STEP 2 - LOOK OVER MAIN MENU TASKS

TASKS:

1). Perform all new keyboard layout
2). Add alternate keys to existing layout
3). Set special characters
4). Set ES and EP parameters
5). Set screen parameters

6). Set other parameters
7). Set signon message
8). Display or print keyboard layout
9). Customization complete; return to operating system

Tasks (1) and (2) are used to determine the keyboard layout, task (8) can print the current keyboard layout, task (7) sets the signon message and (9) writes the customized VEDIT out to disk. The remaining tasks change the various parameters. The prompts for many of these are followed by a number in parentheses, which is a suggested value. To use the suggested value you must type it in, there is NO DEFAULT value. Questions with a numeric answer also require a <enter> after the answer. To ignore input for a particular question, type either the RUBOUT (DELETE) key or a CTRL-U. After each task is performed, the program returns to the main menu. At this point another part of the customization can be performed or a previous step repeated if a mistake was made. Typing a CTRL-C from the main menu aborts the customization.

STEP 3 - TASK 1: PERFORM ALL NEW KEYBOARD LAYOUT

ENTER ESCAPE MODE CHARACTER #1

If you choose to use escape sequences, or your keyboard produces escape sequences with special function keys, type the escape character, or the function key lead-in character, most commonly ESC. Else type <enter>, which will then also skip the remaining questions about escape characters. For P&T CP/M 2 users, ESC is highly recommended.

ENTER ESCAPE MODE CHARACTER #2

A second escape mode character may also be specified, typically for other function keys. If not needed, type <enter>. This is typically not needed for P&T CP/M 2 users.

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

If you answer NO, the editor will make a distinction between, for example, ESC H and ESC h. This is annoying if you hand type escape sequences and you should answer with a "Y" unless you have a very good reason not to.

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. The control codes or escape sequences are displayed as you type them in. Use Task (8) to print out the final keyboard layout for your reference. 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 <enter> 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 <enter> for the rest of the functions and perform this task again.

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

Task (2) allows you to use alternate control codes for any of the editing functions. For example, your keyboard may have cursor keys which you have customized as the four basic cursor movements in VEDIT. However, out of habit you are still using CTRL-S, CTRL-F, CTRL-E and CTRL-C to move the cursor. You can select task (2) to enter any such alternate control codes to use for any editing function. Type the <enter> key for those functions you don't wish to invoke by an alternate control sequence.

When running task (2) you should answer the escape character questions the same way as you did for task (1).

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

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

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

STEP 5 - TASK 3: SET SPECIAL CHARACTERS

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

This is the line continuation indicator used in Visual Mode in reserved column 0. Most common is a hyphen (Hex 2D) or reverse video hyphen (Hex AD). P&T CP/M 2 users may wish to use Hex 1F which displays as an up arrow.

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

The Command Iteration Brackets are those which delimit iteration macros --- groups of Command Mode commands. This manual represents these as "[" and "]" with hex codes of 5B and 5D. You may prefer to use "<" and ">" with hex codes of 3C and 3E. Braces (Hex code 7B and 7D) are easy to use on TRS-80 microcomputers.

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

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

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

Instead of inserting the tab character into the file, spaces to the next tab position are inserted when the [TAB CHARACTER] function is typed. This is useful if another program interacting with your file doesn't interpret tab characters at the same tab positions. Since many spaces use up extra disk space, don't turn this switch on unless you need to.

4.2) AUTO-BUFFERING IN VISUAL MODE
(0=NO, 1=YES, 2=AND BACKWARD) (1)

Auto-buffering is described in section 1. of this manual. You may select no auto-buffering "0", auto-buffering only in the forward direction "1", or auto-buffering in both the forward and backward direction "2". Consider the advantages and disadvantages of backward file buffering before selecting option "2". We recommend option "1" during customization.

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

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

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

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

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

This determines whether the difference between upper and lower case letters is ignored. We suggest you set this to "Yes". A search for "the" will then also find "The", "THE", etc.

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

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

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

This determines whether the case of all letters typed on the keyboard will be reversed, ie., upper case is converted to lower case and vice versa. Only in very unusual situations would you want to do this so respond with 0.

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

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

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

This switch, if set ON, allows you to delimit each string in commands such as Substitute or Find with any character. The most commonly used ones are "/", ";", or ";", but any character may be used.

We suggest turning this switch OFF initially because almost none of our examples use this feature. It may be set with the ES command before you begin issuing other commands.

4.10) CURSOR TYPE (0, 1, 2) (1)

This parameter determines the cursor type as follows: 0=Blinking Underline, 1=Blinking Reverse Video Block, 2=Solid Reverse Video Block. You may wish to try the different types using the "EP 1" command before configuring VEDIT.

4.11) CURSOR BLINK RATE (10 - 100) (See Prompt)

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

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

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

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

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

the ";" lower case letters are converted to upper case. To the right of the "; " in the comment field, no conversion is done.

4.14) DECIMAL CODE FOR CONDITIONAL CONVERSION CHARACTER (59)

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

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

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

STEP 7 - TASK 5: SET SCREEN PARAMETERS

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

Enter the number of screen lines you wish [PAGE UP] and [PAGE DOWN] to move through the text by. 20 is suggested as a starting value.

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

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

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

This is similar to the previous step, except that it sets the bottom line range for the cursor. This number must be greater than or equal to the "Top Line for Cursor" setting, and at most 23, since the very bottom line is only used for status. 20 is a good starting point.

STEP 8 - TASK 6: SET OTHER PARAMETERS

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

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

| MEMORY SIZE | SPARE MEMORY FOR (For 6.1) | VALUE FOR TRANSFER (For 6.2) |
|---|---|---|
| 20K | 2048 | 3 |
| 24K | 2048 | 4 |
| 28K | 3072 | 5 |
| 32K | 4096 | 6 |
| 36K | 5120 | 7 |
| 40K | 5120 | 8 |
| 44K | 6144 | 9 |
| 48K | 6144 | 10 |
| 52K | 7168 | 11 |
| 56K | 7168 | 12 |
| 60K | 8192 | 13 |
| 64K | 8192 | 14 |

--- Minimum system size = 20K.
--- 1 K byte is a unit of 1024 bytes (1024 = 2 ** 10).
--- For CP/M systems, the memory size is the CP/M size, which should be on your CP/M disk label or displayed when you first boot.
--- Do not make the Spare Memory for Auto Read more than two times larger than the value given in the table or it may produce a non-operational editor. This value represents the number of bytes free in the text buffer AFTER a file larger than available memory space is read. For example, in a 64K system the available memory is about 46K. If the table value of 8192 was chosen and a very large file edited, VEDIT would initially read in the first 38K of the file, leaving 8192 bytes free. This extra space is available for insertion of new material. Use the "U" command to verify actual free space. See "Customization Notes" for more details.

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

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

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

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

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

ENTER UP TO 30 TAB POSITIONS IN DECIMAL

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

6.4) BEGIN IN INSERT MODE (0=NO 1=YES) (0)

During full screen editing, you are either in "Normal" or "Insert" mode. This question lets you select which mode the editor begins in. Answering this question is a matter of personal preference. The status line always indicates which mode you are in.

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

It is suggested that P&T CP/M 2 users specify 1.

STEP 9 - TASK 8: SET SIGNON MESSAGE

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

Configured for PASCAL programming.

STEP 10 - TASK 8:DISPLAY OR PRINT KEYBOARD LAYOUT

Selecting this task results in the following question:

DISPLAY ON PRINTER (0) OR CONSOLE (1) ?

Type a "0" (and <enter>) if you wish to have the keyboard layout printed, or "1" to see it displayed on the console screen. The display will be similar to our example keyboard layout sheets, and will also show any alternate keyboard sequences that may be used for each function. This is a handy way to make a record of the keyboard layout. If you forget the keyboard layout, you can run the customization program on the runable VEDIT file and select Task 8 to print out the keyboard layout. For example, if your VEDIT is in the file VEDIT.COM and your customization program is VDSETCRT.COM you can give the command:

```
VDSETCRT VEDIT.COM JUNK
```

Select Task 8 to print the keyboard layout, and then type CTRL-C to abort the customization process.

STEP 11 - TASK 9: CUSTOMIZATION COMPLETE; RETURN TO OPERATING SYSTEM

This writes the customized VEDIT out to disk.

# Customization Notes

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

## VEDIT Checksum

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

## Keyboard Layout

Determining the desired keyboard layout for the cursor movement and function keys is the first task of the customization. Refer to the keyboard layout (Appendix E) for the preconfigured VEDIT supplied with P&T CP/M 2. The best layout will depend on your personal preferences.

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 Line] or [Home], right next to the cursor movement 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 [UNDO], etc. Another is to arrange the keys in some geometric manner, such as having 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 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. 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 "@".

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

When laying out the keyboard, you may therefore use any combination of control characters, special function keys and escape sequences for the visual operations. Some users will prefer to use function keys and control characters for the most used visual operations, and escape sequences for the less used operations. If escape sequences are used, the ESC key is suggested for the escape mode character. Any other character may then follow, including numbers, control characters or even another escape character. An Escape and Control character combination is 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 the preconfigured keyboard layout at first.) The only key which is predefined is the <enter> 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 will give an error message and request a new key for that function.

## A Word About The Keyboard

Since the TRS-80 keyboards have a limited number of function keys, you will need to use control keys and escape sequences for many of the visual editing functions. You may use the named keys (ie. <tab>, <back space>, <hold>, etc.) for any function you wish but it is usually best to assign functions to them that match the name on the key reasonably well. Note that the function keys on the TRS-80 keyboards generate standard control codes, hence if you assign them to visual mode functions, the control codes that they generate may not be used for other functions.

## Memory Size Dependent Parameters

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

In particular, do not make this value more than 2 times larger than the value in the table, or you may produce a non-operational editor. This value is NOT the amount of memory VEDIT will use for the text buffers, since VEDIT always sizes memory and uses all that is available. Rather, this value is the number of bytes that is free in the text buffer after a file is read which is larger than the available memory space. For example, in a 64K system the available memory is about 46K. If the table value of "8192" was used, and a very large file edited, VEDIT would initially read in only the first 38K of the file, leaving "8192" bytes free. This can be verified with the "U" command.

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

'n' denotes a positive number. (# represents 32767)

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

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

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

| | |
|---|---|
| nA | Append 'n' lines from the input file. (0A) |
| -nA | Read 'n' lines back from output 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 r | 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) |
| -0W | Write lines from edit pointer to VEDIT.REV file. |
| Z | Move edit pointer to end of text. |

## SPECIAL CHARACTERS

| | |
|---|---|
| \| | Search wildcard character. Each "\|" will match any character in the text being searched. |
| <CTRL-Q> | Literal Character. Next char. is taken literally. |
| @ | Precedes F, I, N, S command to indicate explicit terminator. |
| : | Precedes F, N, S command to suppress search error message. |
| # | Represents maximum positive number 32767. Signifies "forever" or "all occurrences of". |

## EXTENDED COMMANDS

| | | |
|---|---|---|
| EA | | Restart the editor. (EX and EB). |
| EBfile | | Open "file" for Read & Write, perform an auto-read. |
| EC | | Change disks for reading or write error recovery. |
| 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". |
| EKfile | | Erase (kill) the file "file" from the disk. |
| mEO | | Send 'm' lines to the line printer. (0EO) |
| EP n m | | Change the value of parameter "n" to "m". |
| | 1 | Cursor type (0, 1 or 2) |
| | 2 | Cursor blink rate (10 - 100) |
| | 3 | Indent Increment (1 - 20) |
| | 4 | Lower case convert (0, 1 or 2) |
| | 5 | Conditional convert character (32 - 126) |
| | 6 | Display line and column number (0, 1, 2 or 3) |
| 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 2=BACK) |
| | 3 | Start in visual mode (0=NO 1=YES) |
| | 4 | Point past text reg. insert (0=NO 1=YES) |
| | 5 | Ignore UC/LC search distinction (0=NO 1=YES) |
| | 6 | Clear screen on visual exit (0=NO 1=YES) |
| | 7 | Reverse Upper and Lower case (0=NO 1=YES) |
| | 8 | Suppress search errors (0=NO 1=YES) |
| | 9 | Explicit string terminators (0=NO 1=YES) |
| ET | | Set new tab positions. |
| EV | | Print the VEDIT version number. |
| EWfile | | Open the file "file" for output. Create Backup. |
| EX | | Normal exit back to CP/M after writing output file. |
| EY | | Finish writing and close 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
---

| | |
|---|---|
| NO DISK SPACE | The disk became full before the entire output file was written. As much of the output file as possible was written. Refer to the section on disk write error recovery. |
| CLOSE ERROR | The output file could not be closed. This is a very unusual condition, but may occur if the disk becomes write protected. |
| READ ERROR | An error occurred reading a file. This error should never occur, since CP/M itself normally gives an error if there was a problem reading the disk. |
| NO DIR SPACE | There was no directory space left for the output file. Refer to the section on disk write error recovery. |
| REV FILE OPEN | You cannot change disks because the VEDIT.REV file is open while performing backward disk buffering. |

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 conveniently handled in visual mode. When the cursor is on such a line only the first 258 characters will be displayed. The line may be broken into smaller lines by deleting two characters with the [Back Space], typing [RETURN] to split the line in two and typing in the two deleted characters again. Alternately, enter command mode and give the command "I<CR>$$".

2.) The text being edited can contain characters which have their high order bit (Bit 7) set. These are displayed unmodified meaning that they will show up in reverse video. The character with a value of FF hex (255 decimal) should not be used, because VEDIT will compress it out when performing the "S" command.

The preconfigured version of VEDIT that comes with P&T CP/M 2 uses the following keys and escape sequences for the visual mode edit functions. You may change them by following the configuration procedure in Appendix A.

| | | |
|---|---|---|
| Escape mode character | <esc> | 1st char of escape seq. |
| [home] | <ctl-Q> | beginning of text buffer |
| [zend] | <ctl-Z> | end of text buffer |
| [cursor up] | <up arrow> | |
| [cursor down] | <down arrow> | |
| [cursor right] | <right arrow> | |
| [cursor left] | <left arrow> | |
| [back tab] | <ctl-F> | beginning of line |
| [tab cursor] | <break> | move to next tab stop |
| [zip] | <ctl-X> | end of line |
| [next line] | <hold> | beginning of next line |
| [line toggle] | <ctl-O> | alternate ends of line |
| [scroll up] | <esc> <f1> | move text up 1 line |
| [scroll down] | <esc> <f2> | move text down 1 line |
| [page up] | <f1> | move up about 1 screen |
| [page down] | <f2> | move down about 1 screen |
| [screen toggle] | <ctl-P> | alternate ends of screen |
| [backspace] | <backspace> | delete char before cursor |
| [delete] | <ctl-K> | delete char under cursor |
| [erase to end of line] | <ctl-J> | |
| [erase line] | <esc> J | delete line (can't undo) |
| [undo] | <ctl-U> | restore line |
| [tab character] | <tab> | put tab in text |
| [next character literal] | <esc> L | put next key in text |
| [set insert mode] | <esc> V | enter insert mode |
| [reset insert mode] | <esc> B | cancel insert mode |
| [switch insert mode] | <ctl-V> | toggle insert mode on/off |
| [repeat] | <ctl-N> | repeat next key |
| [indent] | <ctl-E> | increase indent |
| [undent] | <ctl-D> | decrease indent |
| [copy to text register] | <ctl-R> | |
| [move to text register] | <ctl-T> | |
| [insert text register] | <ctl-Y> | |
| [print text block] | <esc> P | |
| [visual escape] | <esc> <esc> | command mode, stop macro |
| [visual exit] | <ctl-W> | command mode, cont. macro |
| [restart editor] | not assigned | |

**PICKLES & TROUT®**

P.O. BOX 1206, GOLETA, CA 93116