USER GUIDE

READY TO USE MACROS

VEDIT PLUS includes four ready to use command macros - a print formatting macro, a file comparison macro, a mailing list sort macro and a main-menu macro. A Z80 to 8086 Assembly Language Translator macro and a Mail Merge macro are available as additional cost options.

Print Formatter Command Macro

The supplied command macro PRINT.VDM performs simple print formatting. This macro can be used as an alternative to the "PR" print command which also prints the filename and page number at the top of each page. Much more sophisticated formatters can be written with VEDIT PLUS macros; PRINT.VDM is intended as a macro example which is relatively easy to understand and expand. Possible enhancements include line numbering, header and footer messages and more.

To get ready to use PRINT.VDM enter Command Mode and issue the command:

RLP PRINT.VDM This loads the print formatter.

This loads the macro into text register "P" where it will remain unless you insert other text into the register.

To print your text, issue the command:

MP This prints the entire edit buffer.

The entire file should begin printing. To stop the printing before it is done press **<CTRL-C>**.

If you use PRINT.VDM a lot, you can automatically load it into VEDIT PLUS by placing the command **"RLP PRINT.VDM"** into the VEDIT.INI file, which is executed each time VEDIT PLUS is invoked.

Feel free to modify the PRINT.VDM macro to suit your needs. Modifying this macro is also an informative way to learn more about VEDIT PLUS macros.

File Comparison/Merge Macro

The file comparison macro can compare two text files of arbitrary size. One file, called the "active file" can also be edited while it is being compared to the "template file". The files can be merged by copying blocks of text from the "template" file to the "active" file. You will find many uses for this file comparison. Possibilities include merging the work done by several people or simply determining which is the most up to date version of a file.

The macro uses "windows" to give you a split-screen comparison of the two files. Remember that the macro can only save changes you make to the "active file", not to the "template file". To perform the file comparison issue the commands:

VPLUS	Invokes VEDIT PLUS.
RLZ COMPARE.VDM	Loads the macro into register "Z".
MZ	This starts the macro execution.

Alternatively, you can invoke VEDIT PLUS with "*auto-execution*", which is equivalent to the three commands above:

```
VPLUS -X COMPARE.VDM
```

The macro first prompts for the window configuration you want. You can select a vertical split, a horizontal split or no windows. Next, the macro prompts for the filename of the "active file", which is the file you can also edit. It then prompts for the filename of the "template file".

COMPARE then displays the active file in Visual Mode, with the cursor at the first position of the active file that differs from the template file. Note that COMPARE handles the case where entire lines have been added or deleted. Since you are in Visual Mode, you can edit the active file in any way you wish. If you selected "windows", the template file will also be displayed with its own cursor.

To continue the file comparison, position the cursor where the files are again identical for at least 24 characters and press **[VISUAL EXIT]**. The cursor will then move to the next difference detected. This process is continued until the end of one or both files is reached. You are then prompted with a menu, allowing you to save the active file and/or return to Command Mode.

If you did not select "windows", you can examine the template by pressing [VISUAL ESCAPE]. The following menu will appear:

```
1) Examine template file     3) Resume, no realignment
2) Examine active file       4) Realign template & resume

          5) Stop. Get termination options menu

          Enter Option:
```

This menu gives you the choice of examining the template file, examining (and editing) the active file, or continuing the file comparison. Although you cannot change the template file, you can copy text from the template file to the active file using text registers "0" through "9".

You can also switch to the template file by pressing [WINDOW]-Switch and "P". You can then switch back to the active file by pressing [WINDOW]-Switch and RETURN.

When you press [VISUAL EXIT] while examining the active file, COMPARE takes the 24 characters following the cursor and attempts to match them in the template file; if successful, it resumes the file comparison. This is the same as selecting option 4 from the menu.

On the other hand, when you press [VISUAL EXIT] while examining the template file, COMPARE immediately resumes the file comparison from the current cursor positions. This is the same as selecting option 3 from the menu.

COMPARE attempts to align the template file with the active file by looking within the template file for the 24-character string following the active-file cursor. If it cannot find a match within the twenty lines preceding the template cursor nor within the hundred lines following, it issues the error: "Unable to realign template file".

Realignment failure is most likely due to the active file cursor being positioned where the files are not identical for the next 24 characters. It could also be due to the cursor having been moved too far forward or backward. To continue the comparison, reposition the active file cursor and press [VISUAL EXIT].

Occasionally, COMPARE will be unable to realign the files. To continue the file comparison you must then manually realign the cursor in both files and then select option 3 from the above menu.

Mailing List Sort Macro

The "SORT.VDM" macro can alphabetically sort a simple mailing list consisting of address lines separated by one or more blank lines. The sort is based on the first address line, assumed to be a name. For example, the following list could be sorted:

```
Svinicki, John
236 Bluelake Dr.
Marquette, MI 48123
-(313) 123-4567
-Wrestling Coach

Fortson, Rick
3219 Space Ct.
Albany, NY 14311
-Neuroscientist

Touleyrou, Diane
642 Sunset Blvd.
Miami, FL 32103
-(305) 321-7654
-News Reporter
```

To run the SORT macro first enter VEDIT PLUS:

```
VPLUS

RLZ SORT.VDM        Loads the macro into register "Z".

MZ                  This starts the macro execution.
```

Alternatively, you can invoke VEDIT PLUS with "auto-execution", which is equivalent to the three commands above:

```
VPLUS -X SORT.VDM
```

SORT prompts you for the filename of the file to be sorted and for the name of the file to contain the sorted output. Just press **RETURN** if the latter is the same as the former. SORT will then: sort the list; save the new list on disk; and return to the Command Mode of VEDIT PLUS. You can then use the "**EQ**" command to exit VEDIT PLUS.

It takes a few seconds per address to sort a medium sized file.

Command Mode Menu Macro

The command macro "MENU.VDM" may be used to replace the normal Command Mode prompt "COMMAND:" with a "menu" of the most common Command Mode operations. This macro will be useful to some people and also serves as an example of how to write "menu" macros for VEDIT PLUS.

It is best to use the "*auto-startup*" feature to automatically load the menu macro into VEDIT PLUS each time it is used. This can be done by renaming the supplied file "MENU.INI" to "VEDIT.INI". (VEDIT.INI most likely needs to be in the "\VEDIT" subdirectory.)

Each time you exit Visual Mode, the screen displays a menu. The menu's top line displays which edit buffer is active (see Multiple File Editing) and which file you are editing in it. For instance, the menu might read:

```
                    VEDIT PLUS - MAIN MENU

                       By William J. Vollano Jr.

You Are Editing Buffer "@"    File = "MYFILE.TXT"

    1 = Normal Exit, Saving Your File
    2 = Quit - Any Edit Changes Are NOT Saved
    3 = Edit A New File (Save Existing File)
    4 = Edit Another File (Multiple File Editing)
    5 = Directory
    6 = Erase Files
    7 = Examine / Change Edit Switches
    8 = Examine / Change Edit Parameters
    9 = Exit Menu to Command Mode
    0 = Enter Visual Mode

Enter Your Choice ( 0 - 9 ) :
```

Most choices involve additional prompts and/or confirmation. Choice (3) is used when you want to sequentially edit one file after another. Choice (4) is used when you want to simultaneously edit several files. Choice (5) allows you to display the directory of any drive and MS-DOS subdirectories. Choice (6) allows you to erase any disk file. Choice (7) displays a sub-menu of all the edit switches (ES command) with the currently set switch values. Choice (8) does the same with the edit parameters (EP command). Choice (9) disables the menu and puts you into Command Mode as indicated by the normal "COMMAND:" prompt.

You are encouraged to change this menu macro in any way you wish.

Even if you do not understand its operation, you should be able to change any screen messages by simply editing the file "MENU.VDM".

Technical Note: The menu macro executes in text register "Z"; therefore, register "Z" is unavailable for other uses when this macro is in use.

Z80 to 8086 Assembly Language Translator

(This macro needs to be purchased as an option.)

This macro converts a Z80 or 8080 assembly language file into a ready to assemble 8086 assembly language file. The Z80 file is assumed to use extended Intel mnemonics. Although the entire 8080 instruction set is supported, the Z80 "IX" and "IY" registers, "Bit" instructions and alternate registers are not supported. (The supplied macro ZIL-INT.EXC converts from Zilog to Intel mnemonics.) This description assumes you are familiar with assembly language.

To run the translator give the DOS command:

```
VPLUS -X Z80-8086.VDM
```

Z80-8086 prompts you for the name of the file to be translated. It writes the translated code into a file with the same name, but with an extension of ".A86". The translation takes approximately one minute for each 10K of commented source code.

The resulting code normally requires some assembler directives and a few 8086 instructions to properly set up the 8086 segment registers. The translated code should then be ready to assemble. The following easily corrected assembly errors are possible.

Some Z80 relative jump instructions will be "out of range". Since the 8086 does not have conditional long jump instructions, two 8086 instructions are needed for each conditional long jump. For example, the 8080/Z80 "JZ LABEL" instruction is translated into a "JNZ $+5 ! JMP LABEL" instruction sequence. You will have to hand translate any relative jump instructions which are out of range.

If a Z80 "LDA" or "STA" instruction accesses a two byte storage area, the translated instruction will cause an "Instruction Mismatch" error. This is corrected by preceding the storage reference with "BYTE PTR" in the 8086 code, i.e. "MOV AL, BYTE PTR WORD1". Similarly, byte storage references accessed by 16 bit registers must be preceded with "WORD PTR", i.e. "MOV BX, WORD PTR BYTE2".

Z80 self-modifying code will also lead to 8086 assembly errors, since the 8086 assembler will not allow direct modification of the code segment. However, this can be circumvented. Assume that the Z80 self-modifying code is:

```
MVI   A,0C9H          ;Get machine code for RET instruction

STA   ROUTIN          ;Disable ROUTIN with a RET instruction
```

The translated "MOV ROUTIN,AL" will result in an error because of the attempt to directly change the code segment. The 8086 equivalent which will assemble and work correctly is:

```
MOV   BX,OFFSET ROUTIN

MOV   CS:[BX],0C3H
```

The "CS:" code segment override is very important in case your assembler directives separate the code and data segments. Note that the machine code for an 8086 "RET" instruction is "0C3H" versus "0C9H" for the Z80. Because of the potential for self-modifying code, it is impossible for any translator to guarantee that the translated program will operate correctly.

If you examine the translated 8086 code, you will surely notice that some obvious optimizations are possible. You may also wonder why the seemingly innocuous Z80 "INX H" instruction is translated into the sequence "PUSHF ! INX BX ! POPF". The reason is that the Z80 instruction does not alter any condition flags, while the 8086 instruction does alter the flags. Therefore, the correct translation involves saving the flags.

Since the 8086 does not have direct equivalents of the Z80 "LDAX" and "STAX" instructions, the translator simulates these by copying the "CX" or "DX" register into the "DI" register which can be used for indirect memory references. This is only significant should you later hand optimize any routines and in the process also use the "DI" register.

COMMAND MODE

Command Mode Notation

**$** is a shorthand for "<ESC>". Wherever "$" appears in a Command Mode example, press the <ESC> key.

<ESC> is the special control character <ESC> generated by pressing the <ESC> key. It is entered into the *command line* to end "*search strings*", "*text strings*" and filenames. This manual often uses the shorthand of "$" for <ESC> because "$" is what VEDIT PLUS displays in Command Mode for the <ESC> character. (In rare cases where you have no <ESC> key, you can customize any other control character to act as the <ESC> key - see Installation Task 8.5.)

<TAB> represents the tab character - press the <TAB> key.

RETURN indicates pressing the **RETURN** key (labelled "<--'" on the IBM PC and "Enter" or "CR" on most other keyboards). Pressing **RETURN** ends the *command line* and starts a new screen line. It also enters the <CR> <LF> pair into the command line.

<CR> represents "carriage return", which is generated by pressing the **RETURN** key. Note that pressing **RETURN** usually generates two control characters - "carriage return" and "line feed", represented as "<CR><LF>".

<CTRL-*x*> Control characters such as "<CTRL-U>" are typed by holding down the **CTRL** key like a SHIFT key and typing the indicated letter, in this case "U".

[ ] The bracket characters are used for iteration loops.

Command Lines

In Command Mode you are prompted for "*command lines*" by the "COMMAND:" prompt. The first few command prompts are preceded by a help message to remind you of the most used commands. Each command line you enter consists of a single command, multiple commands or a special sequence of commands called an "iteration loop". Each command line is ended by pressing **RETURN** or the <ESC> key twice, at which time the command line is executed.

Since no commands are executed until you press **RETURN** (or <ESC> twice), the line may be edited with most common line editing

characters. They are described in detail below. Once execution begins, it may often be aborted by pressing **<CTRL-C>**. This results in the "*BREAK*" message and a new command prompt. VEDIT PLUS checks for the **<CTRL-C>** before any new command is executed, when anything is displayed on the screen and during commands which repeatedly access the disk.

Commands such as "I", "F" and "S", which take "*text string*" arguments must end in the "*text delimiter*", typically **<ESC>**. If you press **RETURN** before the final delimiter, the **<CR><LF>** pair becomes part of the string and the command waits for the rest of its string. This is indicated by the command prompt changing to "-" as a reminder.

Prompt: "-" This means VEDIT PLUS is waiting for the text delimiter.

If you have made a mistake, receive the "-" prompt and do not know what the delimiter is, press **<CTRL-C>** to abort the command.

Commands such as "**ED**" are followed by a "*filename*". The filename may be followed by a **RETURN**, which also ends the command line. However, if the filename is to be followed by other commands, you must end the filename with a **<SPACE>**, a **<TAB>** or an **<ESC>**.

In the very rare case that the command line should exhaust the amount of memory space available to it, VEDIT PLUS will beep and not accept any more characters. You will have to edit the current command line in order to end it and should then rectify the full memory situation.

Occasionally you may see a "<<" displayed at the end of a line preceding the command prompt. This only indicates that the previous line did not end in a carriage return-line feed. It typically occurs with the "-T" command.

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 back the cursor up.
<CTRL-U>	Delete the entire command line, display a "#" and start new line.
<CTRL-X>	Identical to **<CTRL-U>**.

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 **RETURN** or any other special character.

Command Syntax

Each command consists of a one or two letter mnemonic. Many commands are preceded by a number which is a numeric argument to the command. The meaning of this number, which can also be negative, depends upon the particular command. 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 number 65535. Some commands are followed by additional arguments such as text strings, filenames or text register 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. For clarity's sake, you can leave a space between the commands. For example, the three command lines, each with a single command:

```
B       <RETURN>
#PR     <RETURN>
V       <RETURN>
```

are equivalent in operation to the command line with three commands:

```
B #PR V  <RETURN>          (or you can leave the spaces out)
```

A group of commands may be repeatedly executed by enclosing the desired group of commands within brackets "[" and "]". Such a group of commands is called an "*iteration loop*". The initial "[" is preceded by a number, called the "*iteration count*", which specifies how often the group of commands will be repeated.

You will often want to use a sequence of commands, particularly iteration loops, over and over again. You can do this by storing the sequence of commands in one of the 36 text registers and then executing the commands in the text register. Any sequence of commands executed in a text register is called a "*command macro*". Command macros may be stored on disk and loaded back into a text register for later re-use.

Command Operation

Many of the commands operate on the text 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 edit buffer. The number of lines or characters the edit pointer moves is determined by the numeric argument for the command. Negative numbers mean backward movement, towards the beginning of the edit buffer. The "T" command types out a given number of lines before or after the edit pointer to display the contents of the file and "show" where the edit pointer is.

The commands which alter the text all operate from the position of the edit pointer. The search and replace commands all start their search at the edit pointer position. The "V" command puts the editor into Visual Mode and initially sets the cursor at the edit pointer position. When returning to Command Mode, the edit pointer is set from the cursor position.

Controlling Screen Display

Any screen output in Command Mode from commands such as "T", "RT", "EL" and "ED" can be temporarily stopped for easier reading by pressing <CTRL-S>. Pressing any other key, but typically another <CTRL-S>, will then resume the screen output. You can also abort the command by pressing <CTRL-C>.

The "RT" (type text register) command also responds to <CTRL-S> characters that are in the text, automatically stopping the screen display when the <CTRL-S> "stop character" is encountered. The display resumes with the next key typed. This is useful with macros which display several screens of text, such as a long menu. However, be careful when printing files which contain <CTRL-S>, since it may have an unexpected effect on the printer.

If the screen output in Command Mode is too fast, you can slow it down with the "EP 6" parameter. For example, the following command adds 50 seconds delay to every displayed line:

```
EP 6 50          Add 50 milliseconds delay per line.
```

Displaying Current Settings

Several commands change various settings used by VEDIT PLUS. These settings remain in effect until you explicitly change them again. You can display the current settings by typing the corresponding command followed by an immediate **RETURN**. The commands to display these settings are:

EP <RETURN>	Display the "edit parameter" values
ES <RETURN>	Display the "edit switch" values
ET <RETURN>	Display the tab position values
PP <RETURN>	Display the "print parameter" values
ER <RETURN>	Display the input (read) filename
EW <RETURN>	Display the output (write) filename
EU <RETURN>	Display current drive and subdirectory

These commands are fully described later, but it is always safe to enter them with an immediate **RETURN** to see the current values. Note that several of the Visual Mode menu-functions also change the settings.

Help Command

Separate interactive on-line help is available in the Command and Visual Modes. There are two types of help available in Command Mode via the "**H**" and "**EH**" commands. "**H**" gives help with any particular command, such as a list of all the edit switches or parameters. "**EH**" is more oriented toward help with common editing tasks.

Both help commands are interactive - they first display several screens of a help menu from which you pick the desired help topic. Alternatively, you can skip the menu by directly following the "**H**" or "**EH**" with the topic name. For example, for a list of all edit switches give the command:

H ES	List edit switches, skipping help menu

Technically, the "**H**" and "**EH**" commands and even the Visual Mode **[HELP]** function operate identically, only the particular help files used on disk are different. The "**H**" command uses the file "VPHELP.HLP", "**EH**" uses "VPEHELP.HLP" and **[HELP]** uses "VVHELP.HLP".

The three help files are standard text files which may be edited

with VEDIT PLUS. If you desire, you can change and expand the on-line help to suit your purposes. You can also very easily create on-line help for a print formatter, such as V-PRINT, or for a compiler you are using. This is described below.

All three help files are designed for screens 24 lines deep by 80 characters wide. If your screen is smaller, you will have to edit the help files. This is described below.

Modifying On-Line Help Files (Technical)

If you are a new VEDIT PLUS user, you should skip this technical topic until later.

The three on-line help files used by VEDIT PLUS are standard text files and are easily changed and/or expanded. Once you are comfortable with VEDIT PLUS you may want to experiment with the help files and alter them to better serve your particular needs. Since two help commands are available in Command Mode, you may even want to let one command give you on-line help for some other program, such as the V-PRINT print formatter or the programming compiler you are using. It is very easy to even create a custom help file from scratch.

The simplest type of modification is changing the keyboard layout displayed by the **[HELP]** function. You will need to do this if you are not using the preconfigured keyboard layout for the IBM PC version, or the "Default Keyboard Layout" for all other versions. To change the displayed keyboard layout, edit the file "VVHELP.HLP". The keyboard layout appears at the very beginning of the file, exactly as it appears on the screen. Simply edit the control/function key names shown after each edit function until they correspond with your customized keyboard layout.

ng the file
you will not
e following
d if you are
he operation

For this discussion we will assume that you are examini
"VPHELP.HLP" in Visual Mode. (If you print the file,
be able to see the control characters in it.) Th
complex sounding discussion is much easier to understan
actually viewing the help files and are familiar with t
of the on-line help.

ucture, just
one screen
a prompt to
isplayed as
. First, it
ows the user
onse is just
ext **<CTRL-S>**
VEDIT PLUS

All three supplied help files have an identical str
their content is different. Each help file begins with
full of straight text. The last line on the screen is
the user. Notice that it is followed by a **<CTRL-S>** (d
"^S" in Visual Mode). The **<CTRL-S>** serves two purposes
stops the screen display at this point. Second, it all
to enter a response on the keyboard. If the resp
RETURN, the following text in the help file up to the n
is displayed. If the user enters a text "string",

searches the remaining help file for that string enclosed by backslashes "\". Once found, the text following the "\string\" is displayed. The text is displayed until either another <CTRL-S> is reached or a backslash in column one is reached. The latter also ends the help command.

Notice how the menu screens in VPHELP.HLP are simply separated by a <CTRL-S>. Following the text for the menu screens is the text for each topic which can be selected from the menu. The first topic is the "A" command. Notice that a "\A\" appears just before the "A" command description. This is followed by a "\B\" and a description of the "B" command. This is repeated for each help topic.

The text for a help topic usually ends with the backslash "\" starting the next topic. However, the topic could display a sub-menu ending with a <CTRL-S>. Any user response would then be used as a search string to a sub-topic. In this way the on-line help can be hierarchical, with any desired number of sub-menus.

The [HELP] function and associated VVHELP.HLP file use one additional feature. Following the keyboard layout menu, the user is prompted to press the function/control key corresponding to the desired visual function. In the VVHELP.HLP file this prompt is followed by a <CTRL-V>. The <CTRL-V> is similar to <CTRL-S> in that it stops the display and waits for a user response. With <CTRL-V>, however, the response is expected to be a control/ function key. This key is converted into a two letter mnemonic code corresponding to the edit function it performs. This two letter code is then used as a search string in the customary way.

For example, the function/control key corresponding to [CURSOR UP] is converted into the code "CU". A "\CU\", therefore, appears at the beginning of the help text for [CURSOR UP]. A response of just RETURN is handled a little differently. RETURN is also an edit function with a code of "RT". The second screen of the keyboard layout, therefore, must begin with a "\RT\". In effect, the second and third menu screens are sub-menus.

Any text file, such as a command reference for another program, can be converted into a help file accessible from within VEDIT PLUS. The first step is to simply place <CTRL-S> "*stop characters*" within the file after each screen full of text. If you then rename the file to be "VPEHELP.HLP" the text will be viewable, one screen at a time, via the "EH" command. You do not even have to get fancy and use menus with search strings.

There is no need for a custom help file to be editing oriented. By using the menu and sub-menu features, you could create a help system for virtually any topic, even non-computer topics.

FILE EDITING COMMANDS

NOTE:

For simplicity, the following discussion assumes you are only editing one file at a time. The following heading "Multiple File Editing" explains the differences when you are simultaneously editing two or more files.

Exiting with Saving

You can perform the equivalent of the Visual Mode **[FILE]-Exit** function with the "**EX**" command. It will save the file being edited to disk and exit VEDIT PLUS.

EX	Save file on disk and exit VEDIT PLUS.

Alternatively, you can save the file on disk and remain in VEDIT PLUS with the "**EY**" command. "**EY**" is useful when you are finished editing one file and want to edit a new file.

EY	Save file on disk and remain in VEDIT PLUS.

Quitting without Saving (Abandoning)

You can perform the equivalent of the Visual Mode **[FILE]-Quit** function with the "**EQ**" and "**EZ**" commands. These commands let you quit editing without saving the edit changes. "**EQ**" abandons the file and returns you to the operating system, while "**EZ**" abandons the file and remains in VEDIT PLUS with an empty edit buffer.

EQ	Abandon file and quit VEDIT PLUS.
EZ	Abandon file and remain in VEDIT PLUS.

Both commands require confirmation of the decision to abandon the file. You can skip the confirmation prompt by including a "**Y**" in the quit command: "**EQY**" or "**EZY**".

Note: The "**EZ**" command does not affect the contents of the text registers. When simultaneously editing several files, "**EQ**" and "**EZ**" only affect the current file - "**EQ**" only quits the current edit buffer and remains in VEDIT PLUS.

Save File and Continue Editing

You can perform the equivalent of the Visual Mode **[FILE]-Save** function with the "**EA**" command. If you are spending a lot of time editing a file, it is a good habit to routinely save the file on disk and then continue editing it. Otherwise, all of your edit changes could be lost should a power or hardware failure occur. This also protects you from your own mistakes. The command to save a file on disk and then continue editing it is:

EA Save file on disk and continue editing it.

The "**EA**" command does not affect your current editing position, the text markers or the text registers.

NOTE: The "**EA**" command starts a new edit session. Therefore, if an "**EA**" is later followed by an "**EQ**", you will only ignore those changes made after the "**EA**" command. Those changes made before the "**EA**" command will have already been saved on disk.

Editing a New File

You can perform the equivalent of the Visual Mode **[FILE]-New** function with a combination of the "**EY**" and "**EB**" commands. This allows you to finish editing one file and begin editing another file. (Although VEDIT PLUS allows you to edit several files simultaneously, it is a little easier to edit one file after another.)

The "**EY**" command saves the current file being edited on disk, in preparation for editing another file. The "**EB**" command is followed by the name of the new file to be edited. The commands to finish editing one file and begin editing a new file (NEWFILE.TXT) are:

EY EB *newfile.txt*

Notice that spaces may be added between commands and in front of a filename to improve readability. The filename must be terminated with a **RETURN** or an **<ESC>**.

When VEDIT PLUS is invoked you can specify a single file to be edited, or a separate input file and output file. Similarly, the "**EB**" command allows you to also specify a separate input file and output file.

For example, if you want to edit the text in the file "INFILE.TXT" and in the process create a new file with the name "OUTFILE.TXT", invoke VEDIT PLUS with the command:

```
VPLUS INFILE.TXT OUTFILE.TXT
```

You can achieve the equivalent result from within VEDIT PLUS (typically following "**EY**" to edit a new file or following an "**EE**" command to simultaneously edit another file) with the command:

```
EB INFILE.TXT OUTFILE.TXT
```

The two filenames following the "**EB**" command must be separated by one or more spaces. The last filename must be terminated by **<ESC>** or a **RETURN**.

Directory Display

You can perform the equivalent of the Visual Mode **[FILE]**-Directory function with the "**ED**" command. "**ED**" lists the directory of the disk. Drive specifiers and the "*wildcard characters*" "**?**" and "*****" can also be used. Some examples are:

ED	Lists the directory of the current drive.
ED A:	Lists the directory of drive "A".
ED *.ASM	Lists the directory of all ".ASM" files.

Deleting (Erasing) Files

Files can be deleted with the "**EK**" command. In case the disk becomes full and VEDIT PLUS gives you a "NO DISK SPACE" or "NO DIR SPACE" error, you can first use the "**ED**" command to determine what files can be deleted. Then use "**EK**" followed by the name of the file to be deleted. Some examples are:

EK OLDFILE.TXT	Deletes one file.
EK *.BAK	Deletes all ".BAK" files.

Before deleting the files, "**EK**" displays a list of the files to be deleted and requests confirmation. This is a safeguard, especially when using wildcard characters.

NOTE: WHEN DELETING FILES, DO NOT DELETE ANY ".$$$" or ".R" FILES FROM WITHIN VEDIT PLUS! THESE ARE THE TEMPORARY EDIT FILES VEDIT PLUS IS USING. DELETING THESE FILES WILL RESULT IN LOST TEXT. THE EDITOR ITSELF - VPLUS.COM - CAN BE DELETED IF NECESSARY.

MS-DOS / PCDOS Pathnames

With the MS-DOS (and PCDOS) versions of VEDIT PLUS any filename may optionally include a standard "*pathname*" to any subdirectory. For example, the command to edit the file "LETTER.TXT" in the subdirectory (relative to the Root) "BUSINESS" would be:

```
EB \BUSINESS\LETTER.TXT
```

The command to list all files in the same subdirectory would be:

```
ED \BUSINESS\
```

Note above that you must include the second "\". The "ED" command lists both files and subdirectories - subdirectories are indicated with a "*" instead of the normal "." at the end of the filename.

Pathnames can also be specified when VEDIT PLUS is invoked. For example you could edit the file "LETTER.TXT" above with the invocation:

```
VPLUS \BUSINESS\LETTER.TXT
```

CP/M User Numbers

The CP/M (and CP/M-86) versions of VEDIT PLUS support filenames with an optional user number. The user number is specified by following the filename with a "=" and the number. Some examples are:

`ED =5`	Display directory of user "5".
`ED B:=4`	Display directory of drive "B", user number "4".
`EB LETTER.TXT=10`	Edit file LETTER.TXT in user 10.
`VPLUS LETTER.TXT=10`	Invoke VEDIT PLUS

Changing Current (Logged-In) Drive / Directory

If you are constantly accessing files in another MS-DOS subdirectory (or CP/M user number) it is easier to change to that subdirectory with the "EU" command. That way you won't have to specify the pathname each time a file is referenced. For example, to access files in the subdirectory "BUSINESS" give the command:

```
EU \BUSINESS
```

The "**EU**" command can also be used to change to another drive. For example:

EU C:	Change to drive C:.
EU C:\BUSINESS	Change to drive C: and subdirectory "BUSINESS".

You can verify which drive and subdirectory are current with the command:

EU	Display the current drive and subdirectory.

With CP/M the "**EU**" command may be used to change to (log into) a different user number. For example:

EU 4	Change to user number 4.
EU B4:	Change to drive B: and user number 4.
EU	Display the current drive and user number.

AUTO-STARTUP & AUTO-EXECUTION

Automatic Startup

VEDIT PLUS can automatically execute a startup file on disk as a command macro. This can be used to set up various editing parameters or to program function keys on a CRT terminal. When invoked, VEDIT PLUS attempts to read the file "VEDIT.INI" into text register "Z", and then execute this register as a command macro.

The file VEDIT.INI may contain "**EP**", "**ES**" and "**ET**" commands to set up the various parameters, switches and tab positions. The startup file may also contain commands to load other text registers with text or commonly used command macros. For example, the VEDIT.INI file could include the command "**RLP PRINT.VDM**" to automatically load the Print Formatter macro into register "P" each time VEDIT PLUS is invoked.

Some CRT terminals have programmable function keys which are initialized by sending (usually obscure) character strings to the terminal. The VEDIT PLUS startup file can perform this automatically. It is best done by loading the character strings into a second text register, typing out the register, and finally emptying the register. The CRT versions of VEDIT PLUS come with several example files to program a CRT's function keys.

To accommodate personal preferences and hardware configurations, it is possible to select on which drives VEDIT PLUS searches for the VEDIT.INI and the three help files - VPHELP.HLP, VPEHELP.HLP and VVHELP.HLP. This is described under the following topic "Automatic File Searching".

The entire auto-startup feature can also be turned off during installation. (See Task 9.1.) This is recommended when you don't have a VEDIT.INI file, since it will save a few seconds each time VEDIT PLUS is invoked.

Auto-Execution

VEDIT PLUS also has an "*auto-execution*" feature which allows you to specify a different command file to execute in place of, or in addition to, the VEDIT.INI file. The auto-execution of a command file is controlled with two invocation switches:

-X Execute the following file in addition to VEDIT.INI.

-I Execute the following file in place of VEDIT.INI.

For example, the command to print the file DATAFILE.DAT using the supplied Print Formatter macro is:

```
VPLUS -I PRINT.VDM DATAFILE.DAT
```

In this case the VEDIT.INI file is not executed; PRINT.VDM is executed in its place. If the commands in VEDIT.INI must also be executed, perhaps to properly initialize your hardware, use the "**-X**" switch instead. With the "**-X**" switch, the commands in VEDIT.INI are executed first, followed by the commands in PRINT.VDM. In either case, the auto-execution file is loaded into text register "Z" and executed from there.

The auto-execution feature makes it much easier for a user without any knowledge of VEDIT PLUS to run an application macro such as the supplied File Comparison, Mailing List Sort and Print Formatting macros or one which you have written. It also saves a few keystrokes.

As another example, instead of executing the main menu macro MENU.VDM from the VEDIT.INI file, you can use it when desired with the command:

```
VPLUS -X MENU.VDM
```

When an auto-execution file is specified, VEDIT PLUS will search for this file in the same way that it searches for a VEDIT.INI file. Auto-execution can be used even if "*auto-startup*" has been disabled - in this case "**-I**" and "**-X**" are equivalent since the VEDIT.INI will not be executed.

HINTS:

1. You can disable the VEDIT.INI auto-startup feature on an occasional basis by using "**-I**" followed by a non-existent file.

2. You can execute the VEDIT.INI file when auto-startup is disabled by using "**-I VEDIT.INI**".

Automatic File Searching

VEDIT PLUS can automatically search another drive and/or subdirectory for the three help files, the VEDIT.INI file, any auto-execution file, and the file specified for the "**RX**" and "**RL**" commands. The file searching is controlled by two installation parameters (see Tasks 9.2 and 9.3).

Installation Task 9.2 determines whether VEDIT PLUS searches for files on the current (logged-in) drive. When enabled with a value

of "1", VEDIT PLUS first searches for the files on the current drive and in the current MS-DOS subdirectory (or CP/M user number). If Task 9.2 is set to a value of "2", VEDIT PLUS additionally searches on the current drive and in the root directory (or CP/M user number 0).

Installation Task 9.3 determines whether VEDIT PLUS searches for the files on a specified drive. The search is first on the specified drive and the special subdirectory "\VEDIT". If not found, it will last search on the specified drive and the root directory. (Under CP/M this search will only be on user number 0.) Setting up the "\VEDIT" subdirectory is described in the Introduction under "Organizing Your Files".

For convenience in accessing macro files from within VEDIT PLUS, the "**RX**" and, optionally, the "**RL**" commands search for their specified files using this same extended search. The command form "**+RL** *filename*" selects the extended search option. For example, the command to load the file comparison macro from the typical "\VEDIT" subdirectory is:

```
+RLZ COMPARE.VDM          Search for COMPARE.VDM file.
```

NOTE: For normal operation, at least one of the two installation parameters pertaining to file searching must be enabled. Otherwise the auto-startup, auto-execution and on-line help features will not be available. Likewise, extended file searching for the "**RX**" and "**+RL**" commands will be disabled.

MULTIPLE FILE EDITING

VEDIT PLUS has 36 text registers, named "0" through "9" and "A" through "Z". Besides being temporary storage areas for holding blocks of text, the contents of the text registers can be directly edited (in Visual or Command Mode).

The command to edit a text register is "**EEr**", where '**r**' is the register name. The "**EE**" command is equivalent to the Visual Mode **[WINDOW]-S**witch function. Note that **[WINDOW]-S**witch can be used even if you haven't split you screen into "windows". For example:

EE3	Command to edit text register 3.
[WINDOW]-S-3	Equivalent command in Visual Mode.

The first time you edit a text register, the text register is converted into an "*edit buffer*". Edit buffers are a little different from text registers - the main difference being that edit buffers can be used to edit a separate file.

VEDIT PLUS makes no distinction between edit buffers which are text registers and the main edit buffer. The main edit buffer has the name of "@". If you prefer, think of VEDIT PLUS as having 37 text registers - when it is invoked, you are first editing register "@".

At any time only one edit buffer is the "*current*" or "*active*" one being edited. Except for the main edit buffer, the name of the current edit buffer is always displayed on the status line, i.e. "E0" - "E9" or "EA" - "EZ". Once a text register is made into an edit buffer it remains an edit buffer. Use the "**EE**" command (or **[WINDOW]-S**witch) to switch from one edit buffer to another.

The primary purpose for additional edit buffers is to simultaneously edit several files - one is edited in the main edit buffer and the others are edited in selected text registers. For example, assume that you want to simultaneously edit the three files "REPORT.TXT", "TABLE.TXT" and "INDEX.TXT". Invoke VEDIT PLUS with the first file and use two text registers to edit the other files - picking mnemonic names:

VPLUS REPORT.TXT	Invoke VEDIT PLUS with the main file.
[VISUAL EXIT]	Go into Command Mode.
EET	Begin editing in register "T".
EB TABLE.TXT	Edit the file TABLE.TXT.
EEI	Begin editing in register "I".
EB INDEX.TXT	Edit the file INDEX.TXT.

At this point you are editing the file "INDEX.TXT". To switch to editing the file REPORT.TXT in the main edit buffer give the command:

EE@ Switch to editing the main edit buffer.

As a typing convenience, any "EE" command not followed by a valid text register name switches back to the main edit buffer. Therefore, the easier to type "EE." or even "EE<RETURN>" switches back to the main edit buffer.

Edit Buffer Details

All text registers converted into edit buffers and the main edit buffer share the same properties. The primary property of edit buffers is that they can edit separate files - each of unlimited size and with automatic disk buffering.

The "RU" (register usage) command displays the number of bytes stored in each text register/edit buffer. It precedes the number with a "*" to indicate which text registers are also edit buffers. Since edit buffers differ in important ways from text registers, it is important to make a clear distinction between them.

Many text register commands can also be performed on edit buffers - you can insert the contents of another edit buffer into the current edit buffer, save the edit buffer on disk or execute it as a command macro. The main difference is that you cannot change the contents of an edit buffer except when it is the "*active*" edit buffer. Therefore, you can perform the [BLOCK]-Insert function with edit buffers, but cannot perform the [BLOCK]-Copy or [BLOCK]-Move functions. Also, you cannot use the "RC", "RQ", "RI", or "RL" commands on an edit buffer. Attempting to do so results in the error "INVALID EDIT BUFFER OPERATION". Since the primary purpose for edit buffers is to edit additional disk files, this limitation on changing an edit buffer helps prevent you from inadvertently altering a disk file.

It is possible to change an edit buffer back into a text register. When you issue an "EX" or "EQ" command (saving or not saving any file in it), VEDIT PLUS changes the current edit buffer into an empty text register and switches you to another edit buffer. The "EX" and "EQ" commands are more fully described below. Only the main edit buffer always remains an edit buffer, even if it has been emptied.

In practice there is a limit on how many files can be edited at one time because of memory constraints. This is fully described later under "Memory Management". On a typical 256K machine, four or five large files is probably the practical limit, but 20 or more small

files could be edited at any one time.

Sometimes VEDIT PLUS needs to perform disk buffering when switching from one edit buffer to another - part of the edit buffer you are leaving will be written to disk in order to make more memory space free. This is fairly rare on a machine with more than 196K of memory, but is common with CP/M versions. This auto-buffering on the "EE" command can be disabled with the command form "-EEr". (See "Memory Management" for more details.)

Moving Text Between Edit Buffers

You can copy or move text from one edit buffer to another by using an intermediate text register.

1. Make the edit buffer containing the text active with the "EE" command if necessary.

2. Copy or move the text from the edit buffer to an available text register with the [BLOCK]-Copy or [BLOCK]-Move functions (or the "RC" command).

3. Give the "EE" command for the edit buffer which is to receive the text.

4. Position the cursor at the place the text is to be inserted.

5. Insert the text with the [BLOCK]-Insert function (or the "RG" command).

If the text being copied is already in a disk file, you may want to use the "EL" command to locate it and then use the "EG" command to copy it directly from the disk file.

Exiting and Quitting During Multiple File Editing

The "EX" and "EQ" commands operate a little differently when editing multiple files than when editing only one file. "EX" saves the file and exits back to the operating system, while "EQ" abandons the file and exits back to the operation system.

When there is at least one other edit buffer, "EX" and "EQ" do not exit VEDIT PLUS, but rather only exit the current edit buffer - in effect performing an automatic "EE" command. In the process of exiting the edit buffer, "EX" and "EQ" also convert the edit buffer back into an empty text register (except for the main edit buffer).

If an edit buffer has no output file associated with it, the "EX"

command gives the error "NO OUTPUT FILE". This is a reminder that any contents in the buffer cannot yet be saved on disk. Use the "**EW**" command to open an output file and then save the contents with the "**EX**" command.

Generally it is very easy to exit and save all files being edited with successive "**EX**" commands. If you are confident that all edit buffers which are to be saved have an output file open, you can use the special "**EXA**" (**Exit All**) command to exit all buffers and exit VEDIT PLUS. This command should be USED WITH CARE, because any text without an open output file will be lost. We recommend successive "**EX**" commands until you are familiar with VEDIT PLUS. When developing complex command macros, you may find yourself simultaneously editing small macros in 10 or 20 edit buffers. The "**EXA**" command is convenient here, since it allows you to exit VEDIT PLUS without having to exit each of the many edit buffers.

Similarly, the "**EQA**" (Quit All) command quits all edit buffers without saving any changes and exits VEDIT PLUS.

WINDOW COMMANDS

The **[WINDOW]** function allows windows to be managed from Visual Mode. Command Mode allows windows to be managed with additional flexibility. Window commands can be used individually or can be used in command macros for "pop-up windows" and much more. Macros can perform smart "CRT emulation" within each window with cursor positioning, line and screen erase and control over reverse video and/or color.

Although windows are typically used to simultaneously display multiple edit buffers, windows are completely independent of edit buffers. Windows may have any single character name - even names which are not valid text register names.

Windows are created by spitting an existing window into two windows, either vertically or horizontally. Any new or resulting window may then be split further. A window may be as small as one line and/or 15 columns (not including the border drawn around the window). Overlapping windows (ala Macintosh) are not supported. There are four commands for creating windows, depending upon how the existing window is to be split:

YWBw n	Create window 'w' of 'n' lines at bottom.
YWLw n	Create window 'w' of 'n' columns at left.
YWRw n	Create window 'w' of 'n' columns at right.
YWTw n	Create window 'w' of 'n' lines at top.

Once a window is created, you can switch to it with the command:

YWSw	Switch to window 'w'.
YWS	Switch to the default window "@".

Note that this command is very different from the **[WINDOW]**-Switch function - "**YWS**" only switches to a different window, while **[WINDOW]**-Switch switches to a different edit buffer! In effect, **[WINDOW]**-Switch performs an "**EE**" and a "**YWS**" command.

Three more window commands are:

YWDw	Delete window 'w'.
YWD	Delete the current window.
YWI	Initialize - delete all windows and initialize to the default "@" window for the entire screen. Reset to the installed attributes.
YWZ	Zoom the current window to full screen.

Changing Screen/Window Color

The "**YEA**" command allows the foreground and background color of a window to be changed (IBM PC CGA and EGA only) and/or a window to be displayed in reverse video (all machines). The default colors are set during installation. "**YEA**" changes the color for the current window and all windows created thereafter; different windows can be displayed in different colors.

For non-IBM PC versions, "**YEA**" sets reverse video and "**0YEA**"" sets normal video.

For monochrome IBM PC, "**112YEA**" sets reverse video and "**7YEA**" (or just "**YEA**") sets normal video. For color IBM PC, the command form is "**nYEA**" where 'n' is taken from the table below:

n	Character	Background	n	Character	Background
16	Black	on Blue	1	Blue	on Black
32		Green	33		Green
48		Cyan	49		Cyan
64		Red	65		Red
80		Magenta	81		Magenta
96		Brown	97		Brown
112		White	113		White
2	Green	on Black	3	Cyan	on Black
18		Blue	19		Blue
50		Cyan	35		Green
66		Red	67		Red
82		Magenta	83		Magenta
98		Brown	99		Brown
114		White	115		White
4	Red	on Black	5	Magenta	on Black
20		Blue	21		Blue
36		Green	37		Green
52		Cyan	53		Cyan
84		Magenta	69		Red
100		Brown	101		Brown
116		White	117		White
6	Brown	on Black	7	White	on Black
22		Blue	23		Blue
38		Green	39		Green
54		Cyan	55		Cyan
70		Red	71		Red
86		Magenta	87		Magenta
118		White	103		Brown

SEARCHING

Most searching and replacing of text is done from Visual Mode with the **[FIND]** and **[REPLACE]** functions. The following describes the additional flexibility available with the corresponding Command Mode commands "**F**" and "**S**". It also describes "*pattern matching*" in detail. Pattern matching is a very powerful and useful search feature which can also be used with **[FIND]** and **[REPLACE]**.

Forward and Backward Searching

The text to search for is specified with a "*search string*". Usually the search string consists of the exact characters you want to search for. For example, the command to search for the next occurrence of the word "today" is:

**Ftoday$$** where "$" is the <ESC> key.

If the search is successful the "edit pointer" is positioned at the character immediately following "today". If not, the command gives the error message "CANNOT FIND".

To search for the next occurrence of the same word, you could give the same command again. However, since VEDIT PLUS always remembers the last used search string, you can give the simpler command:

F$$ Search with last used search string.

If you immediately want to search for the "nth" occurrence of "today" use the command form "**n**F*string*". For example:

7Ftoday$$ Search for the seventh occurrence of "today".

All searches are normally forwards, toward the end of the file. However, you can also search backwards toward the beginning of the file. The command form "**-F***string*" performs a backwards (reverse) search. For example:

-Fhouse$$ Search backward for the nearest occurrence of "house".

One difference with the reverse search is that if the search is successful the "edit pointer" is positioned at the first character of "house", i.e. at the "h".

As a matter of personal preference, you may want to use "*explicit delimiters*" to mark the beginning and the end of the search string. To use this option, you must precede the command with the "@"

modifier. The previous examples then become:

@F/today/

7@F/today/

-@F/house/

Notice that the <ESC>'s are no longer needed; you can use RETURN instead. The delimiter is shown as a "/", but any other character can be used. Note that the "-" must precede any other command modifiers.

When searching for a word, you usually don't care if the first letter is capitalized, or even if the entire word is capitalized. This presents no problem when searching since VEDIT PLUS normally equates upper and lower case letters. However, when needed, there is an option to make a distinction between upper and lower case letters. This option is selected with the command "**EP 5 0**".

A following topic "Command Modifiers" covers other options for the "**F**" command. The Programming Guide also covers more advanced search topics.

Replacing

The "**S**" (substitute) command is used to search for text and replace it with new text. For example, the command to replace the next occurrence of "today" with "not today" is:

**Stoday$not today$$** where "$" is the <ESC> key.

The "**S**" command can be thought of as an extension of the "**F**" command - the search string is followed by the replacement "*text string*". Although there is no reverse substitute command, all other options of the "**F**" command apply. For example:

@S/today/not today/ using "explicit delimiters".

4@S/today/not today/ replace the next four occurrences of "today" with "not today".

A common use of the substitute command is to replace all occurrences of a word (perhaps a misspelled one) with another word(s). You could precede the "**S**" command with a big number, but it is preferable to use the special number "**#**":

#Stoday$not today$$ replace all occurrences of "today" with "not today".

Another use for the "S" command is to delete all occurrences of some particular text. For example the command to find and delete all occurrences of the word "junk" is:

#Sjunk$$	Find and delete all occurrences of "junk", i.e. "junk" is being replaced with nothing.
#@S/junk//	Equivalent command.

Pattern Matching

"*Pattern matching*" is a powerful searching feature that greatly extends the types of editing you can do. It can be used with the "F", "S" and "EM" commands and with the **[FIND]** and **[REPLACE]** functions.

Pattern matching makes it possible to search not only for particular characters, but also for types of characters such as "any digit", or for characters that meet special conditions such as "occurring at the beginning of a line". You can even search for any five letter word beginning in "t" and ending in "n".

These sophisticated searches are performed by using "*pattern matching codes*" within the search string. Each pattern matching code consists of the special character "¦" followed by another character - typically a mnemonic letter.

NOTE: Although the mnemonic letter may be entered in upper or lower case, for purposes of clarity all examples show these letters in upper case. In case your keyboard does not have the "¦" character, you can customize VEDIT PLUS to use a different character (see Installation Task 8.2).

Here are a few examples of search commands using pattern matching:

F¦D¦D$$	Search for two consecutive digits.
F¦D¦D¦N¦D$$	Search for next two digit number. will not match a three digit number
F¦<note$$	Search for the word "note" appear at the beginning of a line.
Ft¦A¦A¦An$$	Search for any five letter w beginning in "t" and ending in "n".

The complete set of pattern matching codes are:

¦A Matches any alphabetic letter, upper or lower case.
¦B Matches a blank - a single space or a tab.
¦C Matches any control character - a character with a value of 0 - 31 (decimal).
¦D Matches any numeric digit - "0" through "9".
¦F Matches any alphanumeric character - a letter or a digit.
¦L Matches any line terminator - Line Feed, Form Feed or End Of File. Also matches the <CR> <LF> pair.
¦M Matches multiple characters - zero, one or more characters so that the string following the "¦M" is satisfied.
¦N Matches any character which is NOT the following single character or pattern matching code.
¦Pr Use the contents of text register 'r' as a "pattern set".
¦Rr Use the contents of text register 'r' as a "search string".
¦S Matches any separator - a character which is not a letter or digit.
¦T Matches selected separators (terminators).
¦U Matches any upper case letter.
¦V Matches any lower case letter.
¦W Matches "white space" - one or more spaces and/or tabs.
¦X Matches any single character.
¦Y Matches zero, one or more characters until the following character or pattern matching code.
¦< Matches the beginning of line (zero length match).
¦> Matches the end of line (zero length match).
¦¦ Matches a "¦" - this is the literal "¦". Actually, any undefined pattern matching code will match a literal "¦".

A commonly used pattern matching code is "¦S" which matches a "*separator*" - any character which is not a letter or a numeric digit. Notice that a simple search for the word "and" would result in matches by "sand", "Andres" and others. A search for " and " would be better, but would fail if the "and" appeared at the beginning or end of a line. ¦S" will match a space, punctuation, **RETURN** or any other character which is not a letter or a digit. Therefore, the preferred command to search for the word "and" is:

F¦Sand¦S$$ Best search command for the word "and".

The code "¦T" is similar to "¦S", but matches fewer characters - it only matches Space, Tab, ";", ":", ",", "'",'"', <CR> and <LF>. This is primarily useful for programming language constructs where characters such as "$" and "_" may be part of labels.

The code "¦X" is the search "*wildcard*" - it matches any single character.

The code "¦N" is a negation, similar to our usage of "Not". The

string "¦Na" (think of it as not "a") therefore matches any character except "a". The command:

```
Fexam¦Ns$$
```

will find occurrences of "exam", "examiner", but not "exams".

Most pattern matching codes match a single character; however, some will match zero, one or even many characters. The code "¦W" matches "white space" - one or more spaces and/or tab characters. For example, the command to replace all white space at the beginning of a line with a single tab character is:

```
#S¦<¦W$<TAB>$$
```

The codes "¦<" and "¦>" match zero characters; they only ensure that the entire search string matches at the beginning or end of the line, respectively. Remember when using the "S" command and **[REPLACE]** function that ALL text characters that are matched by the search string are replaced by the new text.

The code "¦M" is useful for finding text where the beginning and end are defined, but the middle does not matter. Besides being useful in searches, the "¦M" code can be used to delete large blocks of text. For example, the following command would delete this paragraph:

```
@S/The code¦Mparagraph://          End with a RETURN.
```

In assembly language programming, any text following a ";" character is considered a comment. Instructions are often followed by a few tabs (to align the comments), the ";" and the comment. The following command will delete the tabs (and/or spaces) and the comment which follows any instruction. Lines which are entirely comments are left unchanged.

```
#@S/¦W;¦M<RETURN>
/<RETURN>
/                              This command strips comments.
```

Often the "¦M" code is too unrestricted. For example, the search string "¦Sa¦Mtion¦S" will match words beginning in "a" and ending in "tion". However, it will also match the next word beginning in "a" followed by any text until it finds a word ending in "tion".

The code "¦Y" also can match multiple characters, but is more restrictive than "¦M". The code "¦M" matches ever more characters until the rest of the search string is satisfied, or the end of the file is reached. Once that portion of the search string in front of the "¦M" is matched, it is never searched for again; there is no

need. On the other hand, "¦Y" matches ever more characters only until the very next character (or pattern) matches. If the rest of the search string then fails, the entire search string is re-searched.

For example, we want to search for the following two lines:

```
MOV BL,DL          ;There could be anything at the end of the line
MOV BH,DH
```

We want to be certain that the second line immediately follows the first line. As indicated, the critical part of the first line could be followed by unknown text. The command to find these two lines is:

```
@F/MOV BL,DL¦Y¦LMOV BH,DH/
```

Notice that substituting "¦M" for "¦Y" would not perform the same function - we could no longer be sure that the second line immediately followed the first line.

Using Text Registers in Search Strings

The contents of a text register or edit buffer can be used as part of a search string. The register contents are accessed with the pattern matching code "¦Rr" where 'r' is the name of the register to use. This makes it possible to have "variable" search strings, or a search string consisting of more than 30 characters.

¦Rr	Use the contents of text register 'r' in this position in the search string (or filename).

For example, assume the following text register contents:

```
Register 1 contains:  "Nice"
Register 2 contains:  "a walk"
```

Then the following three search commands are all equivalent. Note that explicit delimiters (terminators) are used:

```
@F/Nice night for a walk/

@F/¦R1 night for a walk/

@F/¦R1 night for ¦R2/
```

A text register used in a search string can also contain a user-definable *"pattern set"*. If the text matches any item in the

set, then the overall match is successful. Each item in the pattern set can itself be a search string. The items are separated from each other by commas ",". Commas themselves are represented by "¦,". The pattern match code "¦**Pr**" uses register '**r**' as a pattern set.

¦**Pr**	Use contents of register 'r' as a pattern set

For example, we want to search for occurrences of the animal names "CAT", "DOG", "LION" and "MOUSE". We can make these four items into a pattern set in register "4" with the command:

```
@RI4/CAT,DOG,LION,MOUSE/
```

The command to display all lines in the edit buffer which contain one of these four animal names is:

```
[@F/¦P4/ OTT]
```

The pattern set is very useful when searching for alternative words. Unfortunately, the pattern set executes quite slowly. This last example would run very slowly since the pattern set has to be checked for each character in the edit buffer.

As another example, we want to search for occurrences of the words "automation" and "automobile". We could of course place the entire words into the pattern set. As an alternative, we will just place the word endings into the pattern set:

```
@RI4/MATION,MOBILE/
```

The command to display all lines in the edit buffer which contain "automation" or "automobile" is:

```
[@F/auto¦P4/ OTT]
```

Since the pattern set is only checked when "auto" has already been found, this search operation will run very quickly.

COMMAND MODIFIERS

Three command modifiers can be used to modify the operation of some commands. These command modifiers are most commonly used with the search commands, but apply to other commands as well. The command modifiers are single characters which immediately precede the command and affect only that command. A command can have more than one modifier - it can even have all three.

In some instances, especially when writing macros, you will repeatedly use a modifier on all applicable commands. Since it would be tedious to have to include it each time, you can select to have any (or all) of the modifiers included by default. The modifier is then no longer needed. The default enabling of the three modifiers is controlled with three "ES" command switches. The initial value of the command switches is set during installation. The examples in this manual assume that they are all set to OFF.

Modifier	Meaning	Affected Commands
_ (underscore)	Perform global file operation	B, F, K, L, S, T, Z, EM, PR, RM
@	Use explicit text delimiter	F, S, I, EM, RI, RQ, XK, XQ, YT
:	Suppress error handling	F, L, S
:	Suppress <CR> <LF>	XT, YR, YW
:	Suppress <CTRL-C> checking	XK

Global File Operations

When editing files which are larger than what will fit into memory at one time, (generally about 50K), you need to use the "*global modifier*" when you want a command to operate on the entire file. Otherwise, most commands only operate on what is in memory. If VEDIT PLUS automatically allowed all commands to operate on the entire file, the entire editing process would slow down objectionably.

Many commands can be preceded with the "_" (underscore) modifier to make them "global" to the entire file. With the global modifier, VEDIT PLUS performs automatic disk buffering when necessary. You can think of the "_" modifier as a command option which makes the size of files less noticeable.

Common commands using the global modifier are "_F" and "_S" to search or replace to the end of the file instead of just to the end of the edit buffer, "_B" to move to the beginning of the file, and "_Z" to move to the end of the file.

Note that these global commands will operate even if you have disabled auto-buffering in the Visual Mode with the "ES 2" command switch.

You can select to have the global modifier enabled by default with the command switch:

ES 10 1 Enable global modifier by default.

Setting the global modifier switch will also make the Visual Mode [FIND] and [REPLACE] operate globally.

Text Strings and Explicit Delimiters

Several commands such as "F" and "S" are followed by a "*search string*", while others such as "I" and "YT" are followed by a "*text string*". For brevity's sake we shall refer to both as "text strings" - the difference is that search strings may contain pattern matching codes.

Since a text string can be of any length and contain any character, including **RETURN**, there has to be some way of indicating the end of the text string. This is done with a special character called the "*text delimiter*" which is normally the <ESC> character. Optionally, you can use the "@" modifier to have an "*explicit delimiter*" begin and end the text string. With this modifier, the character immediately following the command ("F", "I", etc.) is the delimiter. Any character can be the delimiter, but "/" is a good choice. Note that the text string itself cannot contain the explicit delimiter. In the following examples, the commands on the left side are equivalent to those on the right.

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

The explicit delimiter option can be made the default with the command switch:

ES 9 1

Although using this option requires more characters to be typed, many users find that it makes the commands more understandable. It eliminates the need for <ESC> to terminate any text strings and nearly eliminates the need for <ESC> entirely. It also allows the <ESC> character to be searched. For example, the following command

searches for the string "h<ESC><ESC>":

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

Note that <ESC> <ESC> therefore does not end a command when it appears between explicit delimiters. If you type <ESC> <ESC> or RETURN between explicit delimiters, the command prompt changes to "-" to remind you that VEDIT PLUS is still waiting for the delimiter. For example, to find "LABEL" at the beginning of a line, you can use the command:

```
F<RETURN>
LABEL<ESC><ESC>      Note: Prompt changes to "-" here.
```

The actual screen display just before typing the second <ESC> is:

```
COMMAND: F
-LABEL$
```

The "-" command prompt is normal for text strings which contain RETURN. However, if you get a "-" by mistake, press <CTRL-C> to abort the command.

NOTES:

1. The command "F$$" always searches for the last used string, even if explicit delimiters were used for the original command.

2. Commands which take filenames cannot use explicit delimiters.

Literal Character in Text Strings

The "*literal character*" <CTRL-Q> operates similar to the [NEXT CHAR LITERAL] in Visual Mode - the next character is treated literally and not interpreted. The literal character is only meaningful inside of text strings. This is the only way to search for characters such as <CTRL-U> and <CTRL-H> which are also used for line editing. It is also an alternative way to search for the <ESC> character. In the following examples, one command inserts a <CTRL-H> into the text and the second command searches for a <CTRL-H>: br

```
Iword<CTRL-Q><CTRL-H>$$      Insert a <CTRL-H>.

Fword<CTRL-Q><CTRL-H>$$      Search for a <CTRL-H>.
```

MS-DOS, CP/M and VEDIT PLUS all 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>**'s must be changed to **<CR> <LF>** pairs. One cannot simply search for a **<CR>** by pressing the **RETURN** key because it is expanded into **<CR> <LF>**, unless the **RETURN** is preceded with a **<CTRL-Q>**. Therefore, the command to change all lone **<CR>**'s to **<CR> <LF>** pairs is:

```
b#S<CTRL-Q><CR>$<CR>$$        Change <CR> to <CR><LF>.
```

COMMAND MODE FEATURES

Edit Parameters and Switches

The numerous parameters and switches controlled with the "EP" and "ES" commands give you tremendous flexibility in using VEDIT PLUS. Typing "EP" or "ES" and a **RETURN** displays the current parameter or switch settings.

EP Display the current values of all parameters.

If you forget the parameter or switch numbers, use the on-line help to see a list of the parameters or switches.

HES Display on-line help for "ES" command.

The initial values for the "EP" parameters and "ES" switches can be changed with the Installation program (See Installation Tasks 5 and 6). You will want to do this if you find yourself always changing a particular parameter or switch.

The "ET" command is used to change the tab positions. Typing "ET" and a **RETURN** displays the current tab positions:

ET Display the current tab positions.

Following the "ET" by a single number sets the tab positions uniformly, i.e. at every eight columns:

ET 8 Set tab positions at every eight columns.

Alternatively, the "ET" may be followed by two or more numbers specifying the tab positions. Up to 33 tab positions may be set. For example:

ET 10 18 35 Set tab positions at 10, 18 and 35.

NOTE: Since tab positions begin with column 1, setting them at every eighth column sets them to 9, 17, 25, 33, 41, 49, etc.

VEDIT PLUS maintains a separate set of "EP", "ES" and "ET" values for each edit buffer. This allows you, for example, to have word wrap enabled in one edit buffer, but not in another. Changing these values will affect the current and all subsequently created edit buffers (but not previously created edit buffers). Alternatively, you can use the command forms "-EP", "-ES" and "-ET" to change only the values for the current edit buffer.

Printing Text

Text can be printed from Visual Mode with the **[PRINT]** function, or from Command Mode with the "**PR**" command. "**PR**" takes a numeric argument identical to the "**T**" command to specify how many lines before or after the edit pointer are to be printed. For example:

40PR	Print the following 40 lines.
-5PR	Print the preceding five lines.
B #PR	Print entire edit buffer (file).

When text is printed all lines are offset from the left edge of the paper by a selectable "*printer margin*". You can also select how many lines are printed on each page and the physical number of lines per page - these numbers are typically 60 and 66 for 11 inch long pages. You can also select whether VEDIT PLUS advances to a new page by sending line-feeds (blank lines) or by sending a "form-feed" character. These four selections are made with the "**PP**" (print parameter) command. Typing "**PP**" and a **RETURN** displays the values of the current printing parameters:

PP	Display values of print parameters.

All but the "physical number of lines" can also be selected with the **[PRINT]** function. The default print parameters are set during installation.

The "**PE**" (page eject) command advances the printer to the next page:

PE	Eject - advance printer to next page.

When writing a command macro to print labels, it may help to set the "physical number of lines" to the size of the labels - typically nine lines. Then use "**PE**" to advance to the next label.

If you set the "printer margin" to zero and the number of "printed lines" equal to the "physical number of lines" (typically 66), the text will be "dumped" to the printer. This is desirable when printing text which has already been formatted for a printer, such as the ".DOC" files created with V-PRINT.

The supplied macro PRINT.VDM prints the entire file and additionally prints the filename and page number at the top of each page. Examining this short macro is instructive for understanding the print commands better.

WordStar (tm) Files

WordStar (tm) files and files from other word processors often contain characters which have their "High" or "8th" bit set. These are often difficult to edit with VEDIT PLUS - the high bit characters are displayed as graphics characters on an IBM PC and in reverse video on a CRT terminal. Such files can be converted to normal text files with the "YS" command which "strips" the 8th bit. Examples of the command are:

B #_YS	Strip the 8th bit from every character in the file.
10YS	Strip the 8th bit from all characters in the next 10 lines.

If the paragraphs from the word processor are justified, you will find them easier to edit if you first have VEDIT PLUS "unjustify" them to remove the extra spaces between words. This is described under "Justification" in the Tutorial.

If you read a document created with VEDIT PLUS into WordStar, it will treat each text line as a new paragraph, due to the "hard carriage-returns" in the document. If this is annoying, you can use the following "macro" to convert a document, before saving it on disk, so that it is better suited for WordStar. Before running the macro, you should also "unjustify" the document.

The following macro converts a VEDIT PLUS document for use with WordStar. It changes all single carriage-returns into "soft carriage-returns" which have their high bit set. Multiple carriage-returns between paragraphs are left unchanged. It also ensures that each soft carriage-return is preceded by at least one space. (The Programming Guide guide describes macros in detail.)

```
B
[@_F/<RETURN>
/
@EM/<RETURN>
/
(.rv<>0)[
        (.c=^Z) [@YT/End of File reached/ JO ]
        -3C
        @EM/ /
        (.rv<>0) [ C 32EI ]
        -(.c+128)EI
        -(.c+128)EI
        ]
@_F/|N|C/
]
```

TEXT REGISTERS

The 36 text registers serve three primary purposes. One is for "cut and paste" operations, where they temporarily hold a block of text. The second is to hold sequences of commands which may be executed as "macros". The last is as additional edit buffers for simultaneously editing several files. In all cases, the registers are holding textual material; only the manner in which the text is used is different.

Text Register Commands

The text registers have additional flexibility in Command Mode. They can be loaded directly from disk or saved to disk and their contents can be displayed on the screen. All register commands are two letter commands beginning with "R". Except for "RU", each command is immediately followed by the name of the register.

Lines of text may be copied to a register with the "RC" command:

35RC5	Copy the next 35 lines to register 5.
-6RC+4	Append previous 6 lines to register 4.

A text register can be emptied with the "RE" command:

RE2	Empty register 2.

The "RG" (get register) command inserts the contents of the specified register at the edit pointer:

RG2	Insert (get) register 2 at the edit pointer.

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

RSZ B:SOMEFILE.TXT	Save contents of register Z in "SOMEFILE.TXT" on drive "B".

The "RL" command loads a register from a disk file. This is often used to load command macros from disk. Note that edit buffers cannot be loaded with the "RL" command.

RLY B:SOMEFILE.TXT	Load register Y from "SOMEFILE.TXT" on drive "B".

The contents of a text register can be displayed with the "RT" command. Press <CTRL-S> to stop and resume the screen display.

RT9	Type out contents of register 9.

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

RD9	Dump out contents of register 9.

The "RP" command prints the contents of a text register. This is useful for examining the contents of a text register. It also allows a disk file to be printed after first loading it into a text register.

RPZ	Print contents of register Z.

The "RU" command displays the number of characters contained in each of the text registers. The register name is preceded with a "*" to indicate which registers are edit buffers. It also displays the three memory usage numbers displayed by the "U" command.

Text Register Usage

With 36 text registers available, you may find yourself forgetting which register contains what. Although you can use any organizational scheme you wish, you may want to consider the following scheme for using the registers:

A - Y	Are used to contain command macros, the chosen letter being a mnemonic for the function the macro performs.
Z	This is the register used by the auto-startup VEDIT.INI and any auto-execution macros.
0	This is the default "cut and paste" register, since it is the easiest to use with the Visual mode [BLOCK] function.
4 - 9	Are used as additional "cut and paste" registers.
1 - 3	Are used or reserved as edit buffers, for editing additional files.

For simplicity's sake, the examples in this manual use "0" thru "9" as simple text registers for "cut and paste" and use "A" thru "Z" either as edit buffers or as registers containing command macros.

Using Text Registers in Filenames

Several commands such as "**EB**", "**EG**" and "**EW**" are followed by a '*filename*' which may include an optional drive designator, filename extension and MS-DOS pathname (or CP/M user number). The contents of a text register may be used in place of a normal filename. The register contents are accessed with the pattern matching code "**¦Rr**" where 'r' designates the register to use. "**¦Rr**" can also specify the DOS command to be executed with the "**OC**" command.

¦Rr	Use contents of text register 'r' for all or part of a '*filename*'.

The "**¦Rr**" can be used for the entire filename, or for just a part of it, such as the drive designator, pathname, filename extension, or CP/M user number. This makes it possible to specify a "variable" filename.

Numerous methods can be used to store the desired filename in a text register. The topic "Interactive Input and Output" in the Programming Guide describes how you can interactively enter a filename from the keyboard.

Examples:

EB ¦R4	Use the contents of register 4 as the full name of a file to be opened for editing.
ER datafile.¦R3	Use the contents of register 3 as the filename extension with the filename "datafile", which is to be opened for reading. Note that the "." is part of the "**ER**" command. Alternatively, it could be the first character in register 3.
EL ¦R5:help.txt	List the entire file "help.txt" with register 5 containing the drive designator. Note that register 5 must contain a single letter in the range "A" - "P" or an error will result.

When specifying a line range for the "**EG**" and "**EL**" commands, the "**[**" is considered the last character of the filename. The line number range cannot be specified with the "**¦R**" pattern. Instead, the numeric registers can be used to specify a variable line range. See the topic "Numeric Registers" in the Programming Guide.

Comparing Two Files or Blocks of Text

The supplied macro "COMPARE.VDM" can perform a very sophisticated comparison of two files. However, you often only need to know how much of two files match or if two blocks of text are equivalent. This can be done with the "**RMr**" command, which is more fully described in the Programming Guide.

It is very easy to quickly compare two files. First open the files for editing in two edit buffers and position the edit pointer (cursor) at the beginning of each file. Then from one edit buffer give the command "**_RMr**" where 'r' is the name of the second edit buffer. Go into Visual Mode. If the cursor is at the end of the file the files match, otherwise the cursor in each edit buffer is at the point of their first mismatch. If desired you can manually re-align both edit pointers and repeat the "**_RMr**" command to find the next mismatch.

You can also compare a block of text (here called the "template") against the text in your current edit buffer. For example, you may want to know if a previously written subroutine is identical to a subroutine in your current file. First copy the block of text (template) to a text register. Then position the cursor at the beginning of the text to be compared in the edit buffer. Give the command "**RMr**" command where 'r' is the name of the text register. Go into Visual Mode. The cursor will have moved past all characters which matched those in the text register (template).

VISUAL MODE TECHNICAL TOPICS

Screen Display

On CRT terminals the cursor is produced by the terminal and VEDIT PLUS cannot change its appearance. However, on the IBM PC (and other memory mapped systems) VEDIT PLUS produces its own cursor and its appearance is user selectable. You can choose an underline character, a solid block or a blinking block. Even the blink rate is selectable. This is strictly a matter of personal preference. The options are more fully described under the "EP" command and Task 5 of the Installation.

VEDIT PLUS's interruptable screen updating allows the screen to be updated in the fastest way possible when you are performing rapid screen changes. This is primarily applicable to CRT terminals, whose screen cannot be updated "instantaneously" as can a memory mapped screen. With CRT terminals, 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 press another **[PAGE DOWN]** while the screen is updating, VEDIT PLUS interrupts the unwanted update and restarts to display the most current screen. VEDIT PLUS therefore, does not necessarily update the screen in the order in which you perform edit changes. It skips the intermediate screen displays and goes directly to the current screen display.

The leftmost column of the screen is reserved for continuation characters. Due to technical reasons, the rightmost column of most CRT terminals is not used at all.

On CRT terminals the line and column numbers on the status line are not updated immediately following every cursor movement, but only after you stop typing for about 1/2 of a second. This reduces annoying screen flicker.

Lower and Upper Case Conversion

This topic is primarily applicable to programmers.

Several modes are available for converting between lower and upper case letters as they are typed on the keyboard. (These modes do not affect upper and lower case letters which are already in the text.)

The "EP 4" command parameter allows four options for converting from lower to upper case in Visual Mode:

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

Mode "1" is similar to the "Caps Lock" on a keyboard, the 26 lower case letters are converted to upper case.

Modes "2" and "3" are specifically designed for assembly language programming. In Mode "2" lower case letters are converted to upper case if they occur to the left of a special character, called the "*conditional conversion 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 PLUS 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. The "conditional convert character" may be changed with the "EP 5" command parameter.

Mode "3" is almost identical to Mode "2"; instead of converting lower case to upper case, it reverses the case of letters appearing before the ";". This mode makes it easier to enter lower case literals into a program.

EP 4 2	Switch to LC/UC conversion Mode 2 to simplify editing assembly language programs. Pertains only to Visual Mode.

Upper and lower case letters can also be unconditionally reversed; i.e., lower case are converted to upper case and upper case are converted to lower case. This was originally designed for the Radio Shack TRS-80 Model I, whose keyboard normally produces upper case letters and lower case with the Shift key. This reversal is done immediately when a keyboard character is received and before any resulting lower case letter is converted to upper case as

described above. The letters are also reversed for the Command Mode. This mode may also be handy in the case where most text is to be entered in upper case, but where an occasional lower case character is also needed. This mode is selected with the command:

ES 7 1	Reverse all upper and lower case letters, including Command Mode.

End of Lines

Each text line is assumed to end in a **<CR> <LF>** pair as is required for other MS-DOS and CP/M programs and the **<LF>** is the true terminator of text lines. Pressing **RETURN** inserts a **<CR> <LF>** pair at the cursor position. Pressing **[DELETE]** at the end of a line deletes both the **<CR>** and the **<LF>**. Although VEDIT PLUS, in Visual Mode, never creates 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 **<CR>** will be displayed in the normal control character convention as "**^M**". Such lines may be corrected by deleting the offending lone **<CR>** or **<LF>** with **[DELETE]** and then inserting the **<CR> <LF>** pair with **RETURN**.

High Bit (Bit 8) Character Support

The IBM PC and other computers (NEC APC) have graphic and special characters which have their "High" or 8th bit set, i.e. they have a numeric value between 128 and 255. VEDIT PLUS can be configured to properly display these characters.

Since some machines, particularly CRT terminals, do not support special characters, VEDIT PLUS can alternately be configured to display High bit characters by stripping their High bit and displaying the resulting character in reverse video.

VEDIT PLUS is normally configured to read and decode all 8 bits from the keyboard. However, with some CRT terminals, the High bit is used as a "parity" bit which should be ignored. In this case, VEDIT PLUS can alternately be configured to ignore the High bit on keyboard input.

VEDIT PLUS's edit functions are accessed by typing control characters or function keys. The latter either send a special High bit character or an Escape sequence. Those control/function keys which are not assigned to an edit function are normally ignored in Visual Mode. However, for applications where special characters are to be entered directly into the text, VEDIT PLUS can be

configured to enter unused function/control keys into the text. Escape sequences are inserted as the corresponding character with its High bit set. Only those control sequences which are not used in the keyboard layout can be entered. Other characters can be entered with the **[NEXT CHAR LITERAL]** function or the "**EI**" command.

All of these options pertaining to High bit characters are controlled by the "**EP 8 n**" command. The default value is set during installation. The "**EP 8 n**" parameter is actually three parameters. The first bit enables High bit keyboard characters, the second bit enables the display of High bit characters, and the third bit enables insertion of unused control/function keys. There are eight possible values, 0 - 7, each representing a combination of three bit values. The eight values and their meaning are:

VALUE	ALLOW HIGH BIT ON KEYBOARD INPUT	GRAPHICS CHARACTERS OR REVERSE VIDEO	INSERT UNUSED CONTROL /FUNCTION KEYS
0	NO	REVERSE	NO
1	YES	REVERSE	NO
2	NO	GRAPHICS	NO
3	YES	GRAPHICS	NO
4	NO	REVERSE	YES
5	YES	REVERSE	YES
6	NO	GRAPHICS	YES
7	YES	GRAPHICS	YES

The normal value for the IBM PC, NEC APC and other systems with graphics character is a "3" - allow 8 bits on keyboard input, display High bit characters as graphics characters and ignore unused control/function keys. The normal value for CRT terminals is "1" - allow 8 bits on keyboard input, display High bit characters in reverse video (if possible) and ignore unused control/function keys.

NOTE: Whether High bit characters are allowed on keyboard input has NO EFFECT on High bit characters already in the text. Such characters are left unmodified when they are read from disk or written to disk. To strip High bit characters in the text file (for example WordStar files) use the "**YS**" command.

FILE AND MEMORY MANAGEMENT

This section covers the somewhat more technical topics of file and memory management by VEDIT PLUS. It explains the automatic disk buffering used to handle large files and how to override it. It covers "Disk Write Error Recovery" - what to do if you accidentally run out of disk space while using VEDIT PLUS. Last, it explains how memory beyond 64K is allocated to the edit buffers and text registers.

For most applications, it is not essential that you have a detailed knowledge of how VEDIT PLUS manages large files and memory. The topic "Basic Editing Concepts" in the Introduction covers the fundamentals of file management.

Automatic Disk Buffering

"*Auto-buffering*" is any file reading or writing which VEDIT PLUS performs automatically, without the user giving explicit read or write commands. The simplest auto-buffering called "*auto-read*" involves reading the input file into the edit buffer, such as when the editor is invoked with an existing file. "*Auto-write*" involves writing from the edit buffer to the output file, such as when the editor is exited.

VEDIT PLUS can also perform more sophisticated disk buffering when editing 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 automatic disk buffering.

If the edit buffer fills up during the course of editing, Visual Mode will attempt to write out 1K byte sections from the beginning of the edit buffer to the output file. 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 PLUS displays the FULL message on the status line. More text cannot be inserted until manual or automatic disk buffering is then performed.

While VEDIT PLUS performs auto-buffering in Visual Mode to simplify editing, it only performs auto-buffering in Command Mode for selected commands. This gives you precise control over the reading and writing of files, especially with command macros. For special purposes, you can also fully or partially disable auto-buffering in Visual Mode. This option is controlled with the command switch "ES 2".

NOTE: The Visual Mode **[GOTO]** function, particularly **[GOTO]**-Begin and **[GOTO]**-End always work as intended, even if auto-buffering is disabled with the "ES 2" command switch.

Forward Disk Buffering

When VEDIT PLUS edits a file it reads text from the input file into the edit 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 edit buffer for editing. When editing is complete, the edit buffer is written to the output file. Things become more complicated when editing files which are too large to fit into memory all at one time. Only a portion of the input file is then initially read into the edit buffer for editing. In order to edit the rest of the file, some of the edit 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 helps to consider the displayed screen a "window" into the edit buffer. This "window" may be readily moved anywhere within the edit buffer with the **[PAGE UP]**, **[PAGE DOWN]** and other cursor movement functions. Furthermore, the edit buffer may be considered a "window" into the file. Moving this edit 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*".

Forward disk buffering is performed in Visual Mode whenever the user reaches the end of the edit buffer, which is not yet the end of the file. VEDIT PLUS then attempts to read more of the input file and, if necessary, write text to the output file. The minimum amount to be read from the input file, loosely called a "*File Page*" (typically 8 Kbytes), is determined during installation (Task 9.5). If this much free memory is not available, a "File Page" is written from the beginning of the edit buffer to the output file. VEDIT PLUS then reads more from the input file until the memory is "nearly" full. "Nearly" is defined as leaving the number of bytes free that were specified during installation (Task 9.4).

Forward disk buffering is only done automatically in Visual Mode if it was enabled during installation or with the "ES 2" command switch. It should normally be enabled.

Backward Disk Buffering

Backward disk buffering augments the forward disk buffering to further simplify the editing of large files. When editing a large file you may want to edit some text which has already passed through the edit buffer and has been written to the output file. Without backward disk buffering, you would have to start another edit session.

Backward disk buffering reads text from the output file back into the beginning of the edit buffer for further editing. However, before reading text back from the output file, it needs to make space free in the edit buffer. VEDIT PLUS does this by writing text from the end of the edit buffer out to a temporary disk file. (The file has a name extension of ".R" for the main edit buffer and ".rR$" for the corresponding edit buffer named 'r'.) Subsequently, the forward disk buffering will first read from this temporary file before it reads more from the input file. The backward disk buffering also operates with text the size of a "File Page".

Although backward disk buffering works just as automatically and invisibly as forward disk buffering, it must be used with a little more care, especially if you are not using a hard disk. Since it requires an additional temporary file, you are more likely to run out of disk space. Although VEDIT PLUS always lets you recover from running out of disk space, it is more complicated if you are using backward disk buffering.

Since editing a large file requires both an input file and an output file, the maximum file size which may be edited is 1/2 a disk. (Reading from the input file does not free up disk space.) If the input and output files are on different drives, the maximum file size is a full disk. Due to the additional temporary file needed for backward disk buffering, the maximum file size (in the worst case) is reduced to 1/3 a disk. The temporary file is always on the current (logged-in) drive. (With a three drive system you could safely edit a file one disk in length, by making the current, the input file and output file drives all different.) These file size limitations arise because in the worst case VEDIT PLUS needs to create a temporary file which is nearly as large as the output file, which is generally as large as the input file.

If you only have floppy disks, you may want to customize VEDIT PLUS with only automatic forward disk buffering enabled in the Visual Mode. You can also control the auto-buffering with the "ES" command:

ES 2 2	Enable automatic forward and backward disk buffering in Visual Mode.
ES 2 1	Enable only automatic forward disk buffering in Visual Mode.
ES 2 0	Disable all automatic disk buffering in Visual Mode.

Remember that the "**ES 2**" command switch does not affect Command Mode nor the **[GOTO]-Begin** and **[GOTO]-End** functions.

If you are near the end of a very large file and wish to begin editing from the beginning again, it may be faster to edit the file over again by using **[FILE]-New** or a combination of the "**EY**" and "**EB**" commands.

If you use backward disk buffering and run out of disk space, you can still recover without losing any edited text. The procedure is described below under "Disk Write Error Recovery".

To calculate if you have enough disk space to edit a file, use the MS-DOS "DIR" command (or CP/M "STAT") before invoking VEDIT PLUS. It is always best be to sure that there is enough free disk space before editing a file. 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.

Disk Buffering In Command Mode

Auto-buffering is only performed in Command Mode when the "global modifier" is used on selected commands ("**_F**", "**_B**", etc.) or when the default global option is selected with the command switch "**ES 10 1**". (The commands "**EA**", "**EX**" and "**EY**" always perform as much buffering as needed to save the entire file!)

The "**ER**" command opens a file for reading, but does not actually read anything in. The file can be read with the "**A**" command. Similarly, the "**EW**" command opens a file for writing, but does not write anything out. Text can be written out with the "**W**" command. Forward disk buffering in Command Mode, therefore, can be done with 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 edit buffer is nearly full. The "**EY**"

command performs all the reading and writing to finish editing and saving a file without leaving the editor.

As described earlier, backward disk buffering writes text from the end of the edit buffer to the temporary ".R" file and then reads back text written earlier to the output file. The "-W" and "-A" commands give you control over backward disk buffering 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 PLUS's file handling. Generally, the easy to use "_B", "_Z", "_L", "_T" and "_F" commands can perform any additional disk buffering you will need in Command Mode.

You can also use the "EN" (need memory) command to make more memory space free for the edit buffer:

10000EN	Perform forward and/or backward disk buffering to make 10000 bytes of memory free for the edit buffer (if possible).

The "-nA" and "-0A" commands read back text which was written earlier to the output file. "-nA" reads back 'n' lines from the output file or until the edit buffer is completely full or the output file is empty. "-0A" reads back lines until the edit buffer is "nearly" full or the entire output file has been read back.

The command "-nW" writes text from the end of the edit buffer out to the temporary .R file. Its main purpose is to make more memory space available for performing the "-nA" command.

The "A" command and any auto-buffering always read the contents of the temporary ".R" file before reading more from the input file. You, therefore, do not need to explicitly remember whether or not there is any text in the ".R" file.

Disk Write Error Recovery

You may occasionally, but especially with floppy disks, run out of disk space and get the error "NO DISK SPACE"; or you may run out of directory space and get the error "NO DIR SPACE". Fortunately, VEDIT PLUS lets you 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 "EF" command write part of the file on one disk and the rest of the file on another disk. The following paragraphs describe these procedures in 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 you 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 PLUS are on drive A and the disk in drive B is blank, give the command:

VPLUS *infile.ext b:outfile.ext*

The simplest and most common recovery is to delete files from the disk which is full. Use the "**ED**" command to list the directory of the disk. If you find files which you can delete, you are all set. 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 such as any ".COM" or ".EXE" files. The file VPLUS.COM can also be deleted.

Never delete the ".$$$" and ".R" files from within VEDIT PLUS. (You can delete them after exiting VEDIT PLUS in the unlikely event they still exist.)

STOP

> If you are still reading this in order to learn more about VEDIT PLUS, 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 make enough disk space free by deleting old files. You then have several alternatives. If a different disk drive has enough free space you can close the current output file (with the "**EF**" command) and create a second output file on the other drive. For example:

EF	Close the current output file.
EWA:PART2	Create an output file on another drive.
EX	Save the rest of the file and exit.

Then use VEDIT PLUS (or CP/M PIP or MS-DOS COPY) 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 delete the input file or change disks. In either case you want to read as much of the input file as possible. Begin by issuing the command:

#A

Then look at the "FILE" message on the status line, if it reads "FILE" or "fILE" (last letters are capitalized) the entire input

file has been read into memory. If it reads "File" or "file" the entire input file has not been read and the procedure becomes more complicated.

If the entire input file has been read, it is often simplest to delete it from disk with the "**EK**" command. This will make enough space available for the rest of the output file. However, this assumes that you have a backup of the input file or no longer need it. For example, if the file you are editing is "LETTER.TXT", you could give the following commands:

EK LETTER.TXT	Delete the input file.
EX	Save the rest of the file and exit.

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. For example:

EF	Close the current output file.
EC	Insert a new disk and press **RETURN**.
EW PART2	Create a second output file.
EX	Save the rest of the file and exit.

You will then have to use VEDIT PLUS (or CP/M PIP or MS-DOS COPY) to merge your two output files back into one file.

The "**EC**" command checks that it is safe to change disks - it checks that no output file is open and that no ".R" file is being used for backward disk buffering. It also closes the input file to prevent reading garbage from a changed disk.

This procedure has several potential difficulties. If you were using backward disk buffering, the "**EC**" command may give 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. Remember that under MS-DOS you can delete files in other subdirectories too.

If you were unable to read the entire input file into the edit 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 PLUS. Then 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 you could not read the entire input file into memory and cannot delete it because you do not have a backup 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 PLUS 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.

Memory Management

VEDIT PLUS can simultaneously edit several files, each of unlimited length. Although you can edit multiple large files with only 128K of memory on an IBM PC (or 56K of memory on an 8080/Z80 machine), performance is better on IBM PC/8086/8088 machines with more memory. When enough memory is available, each file has its own reserved 64K segment of memory. Of this 64K, about 58K is available for text (the rest is used for internal purposes). The automatic disk buffering handles all files larger than 58K in the usual manner.

If you simultaneously edit more files than there are available 64K segments of memory, the associated edit buffers will share a memory segment. Normally the memory management pertaining to disk buffering and memory sharing is handled automatically, making file size nearly transparent, except for some additional disk access time.

When VEDIT PLUS is invoked it first tries to reserve a 64K memory segment for the text registers (and a few other internal buffers). It then tries to reserve another 64K of memory for the main edit buffer. If 64K of memory is not available, it will settle for a 48K memory segment. Therefore, with less than 112K of available memory (about 196K of total memory), all edit buffers and the text registers will have to share only 64K of memory. Each created edit buffer also attempts to reserve another 64K memory segment for itself. If at least 48K is not available, the new edit buffer will share memory with the most recently created edit buffer.

Consider a typical 256K IBM PC XT. The first 64K of memory is largely used up by the operating system and the VEDIT PLUS program. The text registers (which are not edit buffers) have their own 64K of memory. The main edit buffer also has its own 64K of memory. The second file you edit will also have its own reserved 64K of memory. It is not until you create a third edit buffer that any files will have to share memory. Realistically, you can edit six or seven large files simultaneously. With 512K of memory, you can

easily edit ten or more large files simultaneously.

NOTE:

> The following discussion applies mostly to 8080/Z80 machine users and to IBM PC/8086/8088 users who are simultaneously editing many large files.

Performance is best when each file has its own reserved segment of memory. Switching between edit buffers is then instant with no disk buffering performed by VEDIT PLUS. When edit buffers do share a memory segment, the "**EE**" command and **[WINDOW]-Switch** function will sometimes perform disk buffering to make more memory space free for the newly active edit buffer.

The "**EN**" (need memory) command makes more memory space free in the current edit buffer by buffering some of the file being edited back to disk. The "**EN**" command takes three forms:

0EN Buffers out to disk until the customized amount of "spare memory" (Installation Task 9.4) is again free. This is typically 4 - 8K bytes.

nEN Buffers out to disk until 'n' bytes of memory are free.

EN Buffers out to disk until the memory resident file is reduced in size to the customized size of a file buffering "page" (Task 9.5) plus 1000 bytes. This is typically 6 - 12K bytes.

The "**EN**" command does not buffer out any text which is within 2000 bytes of the current edit pointer. If you specify too large a value, "**EN**" will not be able to make the total requested amount of memory free, but will make as much free as possible. Use the "**U**" command to confirm how much memory actually is free.

If an "**EE**" command or **[WINDOW]-Switch** function switches to an edit buffer which uses a different memory segment from the current edit buffer, the operation occurs instantly without any file buffering. However, if the two edit buffers share a common memory segment and a large amount of the memory space is being used for the currently edited file, some of the file will be buffered out to disk to make more memory space free for the new edit buffer. In the case of shared memory, the "**EE**" command performs an "**EN**" to make more memory space free and can take each of the three forms available for the "**EN**" command. Additionally, the command "**-EE**" switches edit buffers without performing any disk buffering.

-EE1	Switch to edit buffer "1" without performing disk buffering to free more memory space.

When the "**EE**" command switches between edit buffers sharing memory, it buffers the old edit buffer out to disk, leaving just one "page" of text resident. With this in mind, you can compute how many large files can be edited in the last reserved memory segment. The size of the last reserved memory segment, typically 48K, divided by the buffering "page" size gives how many large files can be simultaneously edited without continuously running into full memory problems. You may want to take this into account when customizing the "page" size in Task 9.5. Specifically, you may want to make it smaller, although 4K bytes should be considered the minimum value.

Multi-Tasking Operating Systems

Some operating systems, such as Digital Research's MP/M and Concurrent, and Microsoft's XENIX, allow several programs to be run simultaneously on one computer system by one or more users. These operating systems must deal with the situation where one program attempts to access a file which is already in use by another program. In effect, the second program is denied access to the file, or "locked out". This process is called "*file locking*". For example, two users cannot simultaneously run VEDIT PLUS on the same file.

VEDIT PLUS detects when it is running under these multi-tasking operating systems and then works in conjunction with their file locking. Typically, if you try to access a file with VEDIT PLUS which is already in use by another program, the operating system will first issue you an error message. Then VEDIT PLUS will issue an additional error message "FILE NOT OPENED" to note that the file was not successfully accessed. You also cannot perform an "**EC**" (change disk) command on a disk which is in use by other programs. VEDIT PLUS ensures that files which it is working on, or will soon need to access, are locked from use by other programs. VEDIT PLUS will also release files as soon as it is done with them so that they may then be used by other programs. (It closes all input files as soon as the end of the file is reached.)