Eventide
the next step

COMMAND FILE ROM
INSTRUCTION MANUAL

FIRST PRINTING: AUGUST 1982

INTRODUCTION

The Eventide COMMAND FILE ROM is a firmware program designed to give the Hewlett-Packard 9845 series of desktop computers the capability of utilizing a COMMAND FILE. Installing this ROM adds several BASIC statements and commands to the computer's repertoire designed to accomplish this aim.

ASSUMPTIONS

In writing this manual we make the assumption that you are familiar with the programming and operation of your 9845. Of course the H-P manuals should be consulted when necessary, as should the Eventide manuals for our other option ROM's.

THE COMMAND FILE CONCEPT

Perhaps the fastest way to understand command files (CF) is to consider what you do when you sit down at the computer. Assuming the power is turned on (we wouldn't want the CF to be able to do that!), you will typically load a program (LOAD "PROGRM:msus") and hit the RUN key. After the program has initialized and run for a while, it may request an input from you.

WHAT IS TODAY's DATE, SIR?

At this point, you are expected to enter the date in whatever format is expected and press <CONT>.

After the program chugs away for an additional while, it may ask something like:

WHAT FILES SHALL I PROCESS TODAY?

To which you reply, EXPR_5, EXPR_6 and again press <CONT>

Let's assume that this program takes your experimental data in the two files and does some statistical analyses and stores the results in two DATA files named RES5 and RES6. This analysis may require anything from 5 to 30 minutes, depending upon the data. After this program is finished, it will typically issue an enlightening message such as "FINISHED" and the run light will go off. Your next step will be to prepare a report on the results of the experiment. To do this you require another program called "ANLZEX", which reads files in the same format as RES5 and RES6 are printed on the disk or tape.

ANLZEX requests the file name and other information necessary to prepare the printed report on each experiment and to draw the graphs. You will typically run this program once for each RES file. Each run requires approximately 16 minutes.

The above scenario is a typical one for many operations. While the details vary, the problem is the same: YOU waste your time WAITING for the computer to do something. Note the times involved: if each process takes a few seconds, you don't mind waiting. If each process takes several hours or days, you don't bother waiting, you just go off and do something else. But, if each process takes several minutes, you twiddle your thumbs.

While you're twiddling, you might want to think what to do about the problem. The standard solution can be characterized as "one day I'll have to get around to combining those two programs!" Unfortunately that particular day is not around the corner (it's actually on a little island near Novaya Zemlya). And besides, the programs are long, not well documented, and won't fit in memory.

OK, you say, I'll chain a slave to the keyboard and have him type in the commands when the computer is ready. Unfortunately slaves have to be fed and do require sleep. This is expensive and wasteful of computer time.

What you really want is a computer-resident genie who will notice when the computer is waiting for an input, and tell it what you would have told it were you not busy talking to the boss trying to justify the purchase of more memory. And that, in a roundabout definition, is what a COMMAND FILE is, and does.

COMMAND FILE DEFINITION

More formally: A COMMAND FILE is a LIST OF KEYBOARD COMMANDS which are executed sequentially each time the computer is idle.

Idle, in this case, means, whenever the computer is not executing a program, or when it is waiting for user input, such as when an INPUT or LINPUT statement is being executed in a program.

A COMMAND FILE is a hybrid between the special function keys and a regular program. An SFK can perform any function. You define an SFK to be a certain sequence of key presses. Thereafter, pressing the SFK is the equivalent to pressing the same sequence of keys. A program, on the other hand, is a sequence of statements. To be a valid program line, the statements have to fulfill certain requirements: each must be SYNTACTICALLY VALID, and it must be PERMITTED.

10	A=B+C	is syntactically valid because it conforms to the rules of BASIC, and is permitted.
20	A,B,C	is not syntactically valid (no verb).
30	SCRATCH	is not permitted for the obvious reason that self-erasure is not an especially desirable characteristic for most programs.

Many statements are valid keyboard commands but invalid in programs. Line 20 is valid: it displays the values of variables A, B, and C. Line 30 is valid. It is obviously OK for the operator to erase the program.

While a program rarely has need to execute "forbidden commands", the operator has this need frequently. The Special Function Keys, which memorize keystrokes and do no syntax-checking, are ideal for "forbidden commands." However, since they cannot, except under very limited circumstances, call each other, each SFK press can only execute one command.

Ideally, in order to simulate a complete operating session, one would like to be able to execute a number of SFK's automatically in sequence. This is another way of saying that one would like to be able to write an arbitrary list of commands, completely outside the structure of the BASIC program, which could be used to simulate the presence of the operator. Therefore, an alternative definition could be:

A COMMAND FILE is a list of commands, statements, operations, and keystrokes which are executed outside of and independently from the operation of a BASIC program.

And the final alternative definition goes like this:

A COMMAND FILE can do almost ANYTHING a user can do while sitting at the 9845 keyboard.

What this means to the user is that, as long as a sequence of operations can be defined in advance, it can be performed without human supervision. A long series of programs can be listed and cross-referenced. Even STOREd programs, KEY files, etc can be documented in this fashion. Programs can be EDITed by CF's. You can take an arbitrary list of STOREd programs and insert a copyright notice and serial# in line #2 if you so desire, even if it requires RENumbering the program to do so. Sequences such as those in the first example can be executed automatically by placing the names of the files and the -Continue key in the appropriate locations. Many more applications suggest themselves.

The CMD FILE ROM incorporates several BASIC extensions to assist you in developing and using these CF's. To create CF's, you may use EDIT CMD or LOG CMD. CF's may be renamed, stored, and loaded from the keyboard or from BASIC programs. CF's may even be written by BASIC programs. CF's may call each other or themselves. They may also incorporate up to 9 parameters passed from the calling routine.

Running a command file from the keyboard or from a program requires that the following statement be executed from a program or from the keyboard.

```
DO CMD "name",[parameter 1[,parameter 2...[parameter 9]]]
```

The balance of this manual will give the syntax and special instructions for using this new capability.

QUICK REFERENCE

For convenience, all of the CMD FILE BASIC extensions are listed here with brief descriptions. Most of the keywords are direct counterparts of their BASIC brethren.

EDIT CMD "command name"

This statement places the computer into a special editing mode in which CF's can be created. The "command name" must be 15 characters or less. Leading, trailing, and imbedded blanks count, and remain part of the CF name.

Most keys operate as they do in SFK definition (see H-P manual).

The <shift> CLEAR LINE key exits editing mode without storing changes. The STORE key exits the editing mode and preserves the file.

LIST CMD ["command name"]
LIST CMD # selectcode [,hpib] [;"command name"]

Lists a single command file or all CF's to the system printer, or to a specified printer.

SCRATCH CMD ["command name"]

Erases one or all command definitions from memory.

RE-NAME CMD "command name" TO "new name"

Allows renaming of command file definitions in memory.

STORE CMD "name:msus"

Creates a mass storage file "name" on a mass storage medium, and stores all memory-resident command files in a DATA file.

LOAD CMD "name:msus"

Reads command file definitions from the specified mass storage file, and adds them to those already in memory, except those with duplicate names, which are replaced.

DO CMD "name",[parameter 1[,parameter 2...[parameter 9]]]

Begins executing the command file. Parameters are any string expression which are substituted verbatim for the parameter declaration in the CF.

LOG CMD "command name" [,length]

Creates a buffer of <length> in bytes for storing commands as they are executed. When exiting the LOG CMD mode, this buffer is transferred to CF "command name." The default value for length is 1000 bytes, and the maximum value is approximately 32000 bytes, depending upon I/O processes pending and other CF's in memory.

When LOG CMD is active, an "L" appears immediately to the left of the "run light". The LOG CMD mode is exited by pressing <shift> CLEAR LINE.

INSTALLING THE ROM SET

Although we consistently refer to the product as a ROM, it is actually 2 ROM's, a low-order byte and a high-order byte. There are also additional chips (integrated circuits) which may have to be installed in their sockets. In this section only we will refer to the ROM SET when it is necessary to indicate more than one ROM.

If you are presently the owner of an Eventide WMAZ-4 memory board, and have purchased this ROM set as an add-in accessory, please retrieve your WMAZ-4 manual for the memory board installation procedure.

In addition to this instruction manual, and to any other manuals that came with the ROM set (if you ordered one of the combination products,) you will find a sheet that describes the ROM installation procedure.

The steps are as follows:

1: Remove the memory board. (Refer to the WMAZ-4 manual.)
2: Insert the ROM's in the appropriate sockets. (Refer to the sheet mentioned above.)
3: Re-install the memory board.

If you have ordered the ROM set in conjunction with the memory board, it will already be installed in the appropriate sockets. In this case, again refer to the WMAZ-4 manual for information on how to install the memory board.

CHECKOUT

After installation, turn on the computer and try the tests suggested in the WMAZ-4 manual to confirm that the board is recognized. If all the tests pass, try the following.

Key in the following sequence:

<clear line> SCRATCH CMD <execute>

If the keyboard input line clears, it indicates that the ROM has been recognized. If you get an error indication, it means that the ROM has not been recognized. If this happens, carefully review your installation procedure, and then call Eventide for assistance if necessary.

(If the computer remains in the MEMORY TEST mode, it means that you probably exchanged the LSB and MSB ROM's. This will not cause any damage, but must be corrected.)

If you have gotten this far with no errors or problems, your computer should have a fully functional COMMAND FILE facility.

CREATING COMMAND FILES

Command files may be created in three ways:

1: By using the LOG CMD statement
2: By using the EDIT CMD statement
3: By constructing them with BASIC programs

Which method you use is primarily a matter of convenience. For fairly short CF's, you will probably want to use EDIT. For CF's that involve very time-consuming sequences, you will also probably want to use EDIT. For CF's that use a large number of quickly-completed commands, you will probably want to use LOG.

If you do use LOG, you will probably want to EDIT the resulting file in any event, since the LOG procedure necessarily cannot log every key stroke. The reasons for this will be made clear later.

Constructing CF's with BASIC programs should probably be reserved for extremely long and regular CF's. Just because it can be done doesn't mean that it must.

EDITING COMMAND FILES

Before reading this section, you may want to re-read chapter 13 in the 9845 operating manual regarding SFK (Special Function Key) editing.

The syntax for the EDIT CMD statement is:

```
      EDIT CMD "command name"
```

"command name" is a quoted string of up to 15 characters. Note that leading, trailing and imbedded blanks (spaces) are significant! " CMD1" is not equivalent to "CMD1." When a CF is being edited or listed, the name will appear within quotes to show the positions of any leading or trailing blanks.

If there is presently a CF with the same name, it will appear on the screen ready for editing. If no such file exists, it will be created, and the name will appear at the top of the screen, just as the SFK number does during SFK editing.

The major differences between SFK and CMD editing are in the actions of the cursor movement keys and in the fact that keys that would normally terminate an SFK definition appear as a new line in the CF without terminating it. A "line" in a CF is any combination of typing key strokes or any SINGLE named key. Thus, if you type in ABC<execute>DEF<cont>HIJ<clear> the CF will appear as follows:

```
ABC
-Execute
DEF
-Continue
HIJ
-Clear
```

Unlike in the case of SFK editing, the cursor movement keys still work. Specifically, the roll keys, insert line, delete line, arrow, home, and similar keys still function essentially as they do in the program editing mode. This makes it somewhat easier to manipulate the typically much larger CF's. You can still insert these keys into the CF by pressing <control><cursor key>. This will cause the number of the key to appear on its own line in the CF.

INSERTING PARAMETERS WHILE EDITING

The utility of the CF is greatly increased by the ability to use PARAMETERS. Inserting a parameter in the CF allows one to supply certain information each time the CF is run, without requiring any editing. Hearkening to our initial example, let's say you have a program which, after 5 minutes, requests a file name. Your CF looks something like this:

```
LOAD "EXPROG"
-Run
20/7/82
-Continue
EXPR_5
-Continue
```

This file will first load "EXPROG" and run it. When the program asks for the date, 20/7/82 will be supplied, and the <continue> key will be "pressed." Five minutes later the file name will be requested, and the CF will supply EXPR_5 as the file name. If you want to run the CF tomorrow, you will have to edit it and change 20/7/82 to 21/7/82 and EXPR_5 to, say EXPR_6. On each subsequent run, you will have to edit the CF and change these two items.

A more convenient way to do this is to use parameters. Substitute -Parameter 1 for the date, and -Parameter 2 for the file name. The new CF looks like this:

```
LOAD "EXPROG"
-Run
-Parameter 1
-Continue
-Parameter 2
-Continue
```

When the CF is run, you can substitute the date for Parameter 1 and the file name for Parameter 2 in the DO CMD statement. Thus, you will never have to edit the CF.

Parameters are entered into the CF by simultaneously pressing the Control key and any numeric key between 1 and 9 inclusive. Like other special keys, each parameter has its own line in the CF. A parameter may be any string expression. However, when calling CF's, the quote marks around the parameter ("<parameter>") will be stripped. If the parameter actually must be a quoted string, you can either explicitly place the quotes in the CF, or make the parameter string expression contain the quote marks. The two examples below are equivalent.

```
CMD "XMPL1"                                   CMD "XMPL2"
  LOAD "                                        LOAD
  -Parameter                                    -Parameter 1
  "                                             -Execute
  -Execute
```

followed by...   followed by...

```
DO CMD "XMPL1","EXPROG"     DO CMD "XMPL2",CHR$(34)&"EXPROG"&CHR$(34)
```

Clearly, it will be preferable to place the quote marks in the CF if the CF is to be used at all frequently.

INSERTING SPECIAL FUNCTION KEYS (SFK's) WHILE EDITING

SFK's can be inserted in a CF. Doing so is precisely equivalent to inserting the individual keystrokes that make up the SFK. When an SFK is encountered during CF execution, in effect the CF "presses" the key and then continues its own execution. If the SFK ends with a key such as CONTinue, the CONTinue will be executed as well.

It is important to note that SFK definitions can change under program, user, or CF control. Thus an SFK appearing in a CF may not perform the expected actions. You can take advantage of this fact to use an SFK file to act, in effect, as a set of parameters. If you do not plan to use this feature in a given CF, it would be wise to include a LOAD KEY command in your CF to make sure the correct set of keys are resident.

UNUSUAL APPEARING KEY FUNCTIONS

Certain keys will appear to act in a somewhat odd fashion. They are:

CLEAR LINE and CLR=>END

Each of these keys will clear a line of typing keys in a CF, but will have no effect on named keys such as -Execute.

Starting out with the following CF:

```
-Execute
GHIJKL
-Execute
```

Place the editing cursor under any of the letters GHIJKL:

If you then press CLEAR=>END, the letter under which the cursor resides, and all letters further on, will be erased without affecting either -Execute. Pressing CLEAR LINE will erase GHIJKL without affecting either execute. Placing the cursor on the lower -Execute and hitting CLEAR=>END will have no effect. Placing the cursor on the lower -Execute and pressing CLEAR LINE will erase GHIJKL without affecting either -Execute.

These keys are distinguished from DEL LN and DEL CHR, both of which will delete command keys such as execute. To summarize, CLEAR LINE and CLEAR=>END have no effect on named keys.

EXITING THE EDITING MODE

The CF editor may be exited in one of three ways:

1: STORE	Depressing STORE automatically saves the current definition of the CF being edited. Any previous definition with the same name is erased.
2: <shift> CLEAR LINE	Depressing this key combination discards any changes in the CF being edited, and retains the old version with the same name, if any.
3: RESET (<control> STOP)	Same effect as <shift> CLEAR LINE. Not recommended for casual use.

If you should run out of memory while editing a CF, the computer will BEEP, but you will receive no other error indication. No further command lines can be entered if this happens.

CREATING COMMAND FILES BY USING LOG CMD

If you have used your 9845 for long enough, you'll probably find most of your keyboard commands are issued by your fingers with little intervention by the brain. You will normally load a program and automatically hit the RUN key without thinking about it. If you were asked to write down the complete sequence of manual operations, you probably couldn't do it without serious thought. LOG CMD is a shortcut to the creation of CF's. In effect, it makes the list of keyboard commands for you, with certain limitations.

You enter the LOG CMD mode by typing:

```
LOG CMD "command name" [,length]   <execute>
```

Length is a parameter governing the maximum number of bytes that may be used by the CF. Its default value is 1000 bytes. After hitting EXECUTE, the following happens:

1: A temporary buffer is created of <length> bytes.
2: The letter "L" appears in the system display line immediately to the left of the "run light." This L appears to the right of the keyboard mode display, so TYPWTR will appear as TYPWTRL and SPACE DEP will appear as SPACE DEPL.
3: Typed lines and certain key presses are saved in the buffer until the LOG CMD mode is exited.

To be saved, a line must be terminated by any of the following keys:

EXECUTE	STORE	CONTINUE	STEP

The following key presses are also saved individually:

RUN	PAUSE	STOP

SFK's are saved as their equivalent key strokes providing they terminate with any of the above keys, or if the key strokes are in the keyboard input line when any of the above keys is pressed.

Certain other keys, which would normally be saved in CF EDITING mode, are NOT SAVED in LOG CMD mode. Keys not saved are CLEAR SCREEN, DEL LINE, INS LINE, BACKSPACE, and others involved in cursor movement and display control. If you want these keys to be included in the CF, they must be added during an editing operation. (The rationale for this is simple...you certainly don't want your CF full of typing corrections!)

The same comments with respect to unsaved key presses apply to SFK's.

If an SFK is defined as:	It will be LOGged as:
-Clear line	
MASS STORAGE IS ":T15"	MASS STORAGE IS ":T15"
-Execute	-Execute

This is usually of no consequence because the keyboard input line is automatically cleared by the previous command execution.

EXITING THE LOG CMD MODE

You normally exit the LOG CMD mode by hitting <shift> CLEAR LINE. When this is done, the following events occur:

1: The logging operation ends and the "L" is turned off.
2: The completed buffer is stored as a command file with the name specified in the LOG CMD operation. (Any other CF with the same name is deleted.)
3: Any unused buffer space is released.

It is also possible to exit the LOG CMD mode by executing SCRATCH A or by hitting RESET (<control> STOP). RESET discards the LOG CMD buffer without affecting any data in memory (although, see the H-P manual for the other effects of RESET). SCRATCH A discards the buffer, all CF's, and, of course, everything else.

If you run out of buffer space before terminating the LOG CMD operation, the computer will beep each time it would have stored a command line, and the line will not be stored. Lines stored up to that point will, however, be retained and will remain usable.

Only one LOG CMD operation can be active at a time. If you try to do more, the 9845 will beep at you.

CREATING COMMAND FILES WITH BASIC PROGRAMS

Because CF's are stored and retrieved from BASIC DATA files, it is possible to generate CF's from BASIC programs. Each CF is stored as two strings:

String 1: The CF NAME, in 1 to 15 characters.
String 2: The CF CONTENTS, in 1 to however many characters are required.

Multiple CF's are stored in one file in groups of 2 strings as above for each CF.

Function keys are stored in the same manner as they are read by the ON KBD command: Regular typing keys show up as an ASCII character, system keys as a key number preceded by CHR$(255). Parameters are inserted as:

```
CHR$(255)&CHR$(55)    -Parameter 1
CHR$(255)&CHR$(56)    -Parameter 2
...
CHR$(255)&CHR$(63)    -Parameter 9
```

Using the tables in the 9845 owners manual, you can determine the codes for any of the system keys.

An example program to write a simple CF follows:

```
COMMAND FILE:              CMD "XMPL"
                             LOAD "PROG"
                             -Execute

PROGRAM:    10    DIM Name$[16],Cmd$[200]                   ! DIM strings
            20    Name$="XMPL"                              ! Assign CF name
            30    Cmd$="LOAD "&CHR$(34)&"PROG"&CHR$(34)! LOAD "PROG" and
            40    Cmd$=Cmd$&CHR$(255)&CHR$(21)              ! EXECUTE TO Cmd$
            50    CREATE "CMDXMP",1                         ! CREATE DATA FILE
            60    ASSIGN #1 TO "CMDXMP"                     ! AND PRINT
            70    PRINT#1;Name$,Cmd$                        ! CF DATA IN IT
            80    ASSIGN #1 TO *                            ! CLOSE FILE
            90    LOAD CMD "CMDXMP"                         ! LOAD NEW CMD FILE
            100   END
```

USING COMMAND FILES

Once you have created a CF, it may be executed (run) using the following statement:

```
DO CMD "command name" [,parameter 1 [,parameter 2...[,parameter 9 ]]]
```

"command name" must be the name of a valid CF in memory.

Parameters, if included, must be valid string expressions. Any string expression will be accepted, whether it be a literal (quoted string) or an expression containing string variables.

ERROR MESSAGES:

If "command name" does not correspond to a valid CF name, error 56, "File name is undefined", will be issued.

If a parameter is not a valid string expression, an IMPROPER EXPRESSION error will result. Note that if a parameter evaluates to an expression that is invalid in context, any error message may result depending upon the parameter's use. For instance, "A/0" is perfectly valid as a file name, but will result in ERROR 31 (Division by zero) if the CF tries to execute the expression.

CALLING CF'S FROM OTHER CF'S (NESTING)

CF's can be nested without limit, subject to the same memory limitations mentioned below. When a CF calls another CF, the calling CF is suspended until all statements in the called CF are exhausted.

CALLING CF'S FROM THEMSELVES (RECURSION)

CF's may call themselves recursively without limit, subject to certain memory limitations. Each time a CF calls itself, an additional quantity of memory is allocated. Eventually the computer will report ERROR 2 (Out of memory) and CF execution will abort. The memory available for CF's is not the total available to the 9845, but only that available to I/O processes, typically 30K bytes.

All CF's that call themselves are not necessarily recursive. If a CF calls itself as its final step, it in effect loops, and can continue indefinitely.

NON-RECURSIVE CF

```
CMD "NON-REC"
  LOAD "PROG"
  -Execute
  LIST
  -Execute
  DO CMD "NON-REC"
  -Execute
```

RECURSIVE CF

```
CMD "RECURSIVE"
  PRINT "RECURSION LEVEL";A,"MEMORY";AVM
  -Execute
  A=A+1
  -Execute
  DO CMD "RECURSIVE"
  -Execute
  PRINT "HELP ME! I'M STUCK"
  -Execute
```

The non-recursive example will continuously list "PROG." The recursive program will continuously print the recursion level and the amount of memory available. (AVM is a statement in the Eventide PUP PLUS ROM. If you don't have this ROM, delete the portion of the line after the variable A.) Eventually the recursive example will terminate with an ERROR 2 without ever having printed "HELP ME! I'M STUCK."

CALLING CF'S FROM PROGRAMS

Normally DO CMD is executed from the keyboard. It may be executed from a program, but doing so produces what may appear to be anomalous results:

1: If a DO CMD statement is executed from a program, it will not begin executing until the program becomes idle, STOPs, or ENDs. (Just as a CF executed from the keyboard will not begin until the program reaches an idle state.)

2: If multiple DO CMD statements are executed from a program, they will be performed in REVERSE ORDER after the program reaches idle.

To prevent confusion, we recommend that you only place DO CMD statements at the end of a program, and only call CF's from the keyboard or from other CF's.

TERMINATING COMMAND FILE EXECUTION

Normally, a CF will terminate execution when its last statement is exhausted. It will also terminate if BASIC reports an error or if an operator physically presses the STOP key (or RESETs the computer).

Note that errors trapped (by ON ERROR, ON END, etc.), will not stop CF execution. If your major CF application is to run the computer unattended for long periods, it would be a good idea to add routines to trap predictable errors such as the possibility that peripherals might be temporarily down due to power failure. Or, for instance, if your program generates long printouts, you might want to trap a PRINTER OUT OF PAPER error and redirect your I/O to disk. (No, the COMMAND FILE cannot replace the printer paper.)

REMOVING COMMAND FILE DEFINITIONS

It is possible to remove CF's from memory using the following command:

```
SCRATCH CMD ["command name"]
```

If the command is executed without the optional name parameter, all CF's in memory will be erased. If the name is used, only that CF will be erased, if present.

If an attempt is made to SCRATCH non-existent CF's, the command will nonetheless execute and no error message will be issued

If a SCRATCH CMD statement is executed from a running CF, the CF will be erased. However, because an executing CF is saved in a special buffer, it will continue to execute even though the CF is no longer present!

However, once the buffered CF is completed, it will disappear, and the version previously in memory, having been SCRATCHED, will no longer be extant it any usable form.

LISTING COMMAND FILES

Command files may be listed to the system printer:

```
LIST CMD ["command name"]
```

If the optional name parameter is used, only that CF, if present, will be listed. Otherwise, all CF's will be listed. If the name parameter is used and there is no corresponding CF, no listing will be output.

Command files may be listed to a designated printer:

```
    LIST CMD # selectcode [,hpib] [;"command name"]
```

<Selectcode> corresponds to the address of the optional printer, and <hpib> corresponds to the HPIB (GPIB) address, if any. See the 9845 operating manual for further information on these concepts if necessary.

CHANGING THE NAMES OF COMMAND FILES

The names of CF's resident in memory may be changed. You might want to do this as part of the process of saving a file under one name before editing or modifying it and saving it under a new name.

To change a name, use the command:

```
    RE-NAME CMD  "old command name" TO "new command name"
```

NOTE: This command differs slightly from the mainframe RENAME command in that the keyword is hyphenated.

ERROR MESSAGES

If <old command name> doesn't exist, ERROR 56, File name is undefined, is issued.

If <new command name> already exists, ERROR 54, Duplicate file name, is issued.

SAVING AND LOADING COMMAND FILES ON MASS STORAGE

All command files in memory may be stored on any mass storage device (including RAM if you have the Eventide Memory Mass Storage ROM). This is done with the command:

```
    STORE CMD <name>[:msus]
```

For this statement, you may use any valid mass storage file specifier as defined in the H-P manuals. Note that mass storage directory file names may be a maximum of 6 characters.

To load a set of CF's, use the corresponding:

```
    LOAD CMD <name>[:msus]
```

Again, use any valid mass storage file specifier.

ERROR MESSAGES: LOAD CMD and STORE CMD

These commands are precisely analogous to the LOAD/STORE program commands. Any mass storage error can be encountered, such as missing/duplicate file names, defective or missing drives, etc. There is no error message specifically related to the fact that you are dealing with CF's instead of programs.

SPECIAL CONSIDERATIONS REGARDING CF'S AND MASS STORAGE

There is no specific TYPE of file that contains only CF's. Unlike the KEYS file that cannot be accessed by BASIC, CF's are STOREd to and LOADed from files of type DATA. As explained earlier, it is possible to write CF's from BASIC by placing the CF lines in string variables.

There is no inherent mechanism preventing one from accidentally trying to load a non-CF DATA file with a LOAD CMD statement. If the DATA file contains non-CF data, unpredictable results may occur. Attempting a program GET operation on a CF file will produce the same error messages as will trying to GET a string data file without proper line numbering.

COMMAND FILE LIMITATIONS

There is one major limitation on the usefulness of the CF. As stated in the introduction, CF's are activated whenever the computer is waiting for an input or otherwise idle (STOPped). Some programs are written using ON KBD, ON KEY or other statements that require an input but do not put the computer in any type of wait state.

Unfortunately, there is no consistent method of detecting when the 9845 is waiting for this type of input. Thus, CF statements will not be executed under such circumstances. If you wish to use programs containing any of the statements listed below, you must re-write that portion of the program.

Here are the statements that CF's cannot detect:

ON KEY   ON KBD   EDIT KEY   EDIT CMD   DIGITIZE   LETTER

Note that CF's CAN drive regular program editing.

Re-writing programs to eliminate this disability can be trivial or challenging. Some ON KBD statements are inserted to save a single keystroke. In others (such as text editors), it is hopeless even to try.

MISCELLANEOUS NOTES AND CF TRIVIA

Each command file definition in memory uses one of 128 available buffers. Hence a maximum of 128 command files may be loaded simultaneously. In practice, keeping more than 20 to 30 in memory should be avoided to prevent degradation of the 9845's I/O performance. Note that each assembly language module in memory requires one buffer as well.

SCRATCH A stops command file processing (and everything else!) if you want to clear the memory to the extent possible with a CF without stopping execution, perform SCRATCH C, SCRATCH P, and SCRATCH K.

STORE ALL and LOAD ALL preserve command file definitions, but NOT active CF processing. Also, a STORE ALL file created on a machine with a CF ROM cannot be LOAD ALL'ed on a machine without one. (This is generally true for ROM configuration differences.)