Southern Software

EXEC

Command List Processor for TRS-80®

southern software

CONTENTS Page

USES OF EXEC

EXEC is a disk-based command processor for TRS-80, Model I and III, or for Genie 1 and 2. It executes programs consisting of mixtures of commands, BASIC statements, and controls. These special programs will be called "Command Lists", or CLISTs, in the following documentation. They add a new dimension to the way in which you can program and control your computer. Their intended uses are:

1) General FILE Management. Repetitive operations on DOS files can be encapsulated into stored programs.

2) Test Case Execution. Any program product will normally be run against a substantial test case library. Interactive products, in particular, often involve tedious and time-consuming checking by hand. Such tests can be largely automated using CLISTs.

3) Timings and Benchmarks. CLISTs can automate performance testing, giving greater consistency, with much less manual effort.

4) Shop-Window Demonstrations. A CLIST can be written to drive a software or hardware product for demonstration in a shop, or at an exhibition, being run repeatedly, with intermixed comments, explanations, or sales information.

5) Tutorials. A software product can be shipped with one or more tutorial CLISTs which show the customer how the product should be run.

The TRSDOS DO command can be used to invoke BLD files created using the BUILD command. This function is comparable with EXEC, but EXEC gives the following advantages:

1) EXEC runs on Model I and Genie, whereas DO is available generally only on Model III.

2) The CLISTs are created in the same way as BASIC programs, stored with the ASCII option. So they can be listed, modified, or extended, unlike BLD files.

3) At invocation, parameters can be passed to a CLIST.

4) A CLIST will run correctly under the BASIC environment, not just under DOS.

5) CLISTs can prompt for and receive input during execution.

6) CLISTs can contain almost all BASIC statements, including file I/O, PRINT, IF THEN ELSE, and so on. Loops and conditional operations are supported.

7) When correct, a CLIST can be encapsulated with part of EXEC itself, to produce an module that can be invoked as a direct command, with parameters.

8) CLISTs can include remarks which are optionally displayed as commentary as the program executes.

EXEC does not unconditionally corrupt high memory, as DO execution does, but it IS necessary for the user to allocate space in memory for it to run. EXEC requires 1536 bytes normally, or 2560 bytes if commentary screens are used.

INSTALLING EXEC

The EXEC product is the module EXEC/CMD. This module, some sample programs, and installation and relocation utilities are supplied on the product disk which does NOT include a full operating system (such as TRSDOS). The disk is either 35-track (Model I), or 40-track (Model III), or it may be double-sided, with the two formats on the two sides. You can treat the product disk as a "DATA" disk, and if you have two or more drives, or you use a non-TRSDOS operating system, then you can COPY or CONVERT the files from it to your own system (or data) disk. You can even CONVERT the files from a 35-track disk to Model III 40-track, if necessary.

If you have only one drive, and only TRSDOS, then you should put the product disk in drive 0 and press reset (BOOT). Then follow the instructions to swap between the product disk, and your system disk, to install a utility called IMPORT/CMD on your system disk. Then invoke IMPORT, and follow its instructions to copy some or all the files from the product disk to your system disk.

Initially you need only COPY or IMPORT the files EXEC/CMD and SAMPLE/ASC. Then try out EXEC by typing, under DOS,

EXEC SAMPLE (enter)

This will invoke the module EXEC/CMD, which will in turn load and interpret the file SAMPLE as a "program" consisting of three DOS commands, which turn on the CLOCK, display the FREE disk space, and set AUTO to null.

To see what the SAMPLE CLIST looks like, enter BASIC in the normal way and type LOAD "SAMPLE/ASC". Then LIST the loaded program.

```
10 CLOCK (ON)
20 FREE
30 AUTO
```

As an exercise, add another command to the program, e.g. 40 CLOCK (OFF). Save this modified program with the ASCII option, i.e.

SAVE "SAMPLE/ASC",A

Now return to DOS, with CMD"S", and rerun the new SAMPLE CLIST. When you type EXEC SAMPLE, the extension ASC is assumed. You can supply an explicit extension if you wish, e.g. EXEC SAMPLE/ASC, or EXEC JOE/TXT. But the file must always be in ASCII format, not compressed BASIC.

Other samples on the product disk are ACCEL4/ASC and TUTORIAL/ASC. Note that these, and the examples in the text, will only run under TRSDOS. This is because the other DOS's generally have an incompatible syntax for invocation of BASIC. If you EXECute ACCEL4/ASC, it will show an example SORT program running uncompiled, and then compiled by ACCEL4, and is the type of demonstration that could be shown at a software exhibition. TUTORIAL/ASC will show the type of CLIST that might be shipped with a software product.

PASSING PARAMETERS TO A CLIST

Suppose one of the repetitive operations you have to do is to create a new version of a program you are developing. The new version is built by DUMPing a core image. But before doing this you also want to preserve the previous version of the program as a backup, and you want to delete your previous backup. The sequence is typically:

```
KILL PROG/OLD
RENAME PROG/CMD to PROG/OLD
DUMP PROG/CMD (START=...,END=...,TRA=...)
```

Since you may have to do this operation for many different programs, you require to write a CLIST to which the string "PROG" can be passed as a parameter at execution. Let's also assume that the addresses in START, END, and TRA, are normally fixed, but you would like the option to override them if necessary. On Model I, under TRSDOS, the CLIST to do this would be:

```
10 KILL @1/OLD
20 RENAME @1/CMD to @1/OLD
30 DUMP @1/CMD (START=X'@2(7000)',END=X'@3(7200)',TRA=X'@2(7000)')
```

@1 stands for the string that is the first parameter in invocation. @2 and @3 are the second and third parameters. If @2 and/or @3 are null, then the string bracketed by <...> will be substituted instead, i.e. it's the default value. If this program is saved as UPDATE/ASC, say, then invocation (under DOS) will produce the following results:

```
EXEC UPDATE TEST
```

will execute as

```
KILL TEST/OLD
RENAME TEST/CMD TO TEST/OLD
DUMP TEST/CMD (START=X'7000',END=X'7200',TRA=X'7000')
```

"TEST" has been substituted for @1, while the default values "7000" and "7200" have replaced @2 and @3.

```
EXEC UPDATE TEST,,7300
```

will execute as

```
KILL TEST/OLD
RENAME TEST/CMD TO TEST/OLD
DUMP TEST/CMD (START=X'7000',END=X'7300',TRA=X'7000')
```

Note that parameters are separated by single blanks or commas, and that a null can be shown by ,, or simply by omitting trailing parameters altogether. Up to 9 parameters can be used.

PROMPTING FOR INPUT

EXEC uses the convention that @0, the "zeroth" parameter, means prompt for input as the program runs. Actually, EXEC switches input from the stored program to the keyboard, up till the next ENTER. So the input can be any string, and, unlike parameters 1 to 9, it can contain blanks and commas. Also each occurrence of @0 prompts for fresh input, rather than using the previously keyed value. So an alternative way of coding the previous example is:

```
10 KILL @1/OLD
20 RENAME @1/CMD TO @1/OLD
30 DUMP @1/CMD (START=X'@0(7000)',END=X'@0(7200)',TRA=X'@0(7000)')
```

Now the CLIST will pause three times, for the user to input the hex values, or null.

MEMORY ALLOCATION FOR EXEC

The CLISTs, (SAMPLE/ASC, UPDATE/ASC, etc.) are loaded by EXEC itself, through an internal buffer. So they can be any size. But EXEC itself occupies memory, just under X'600', or 1536 bytes. Finding this space is the user's responsibility, and where you put it depends largely on the purpose for which the EXEC program is intended. The shipped file EXEC/CMD is located at X'7000' to X'7600', and this is suitable for running DOS command sequences like the examples shown above. DOS commands normally use only memory below X'7000', although the utilities like BACKUP may use the whole of available memory, and would therefore destroy EXEC/CMD if they were included in an EXEC program. (These utilities normally reboot at completion, as well).

A common use of an EXEC program is to LOAD one or more machine-code programs into high memory, and then to enter BASIC, protecting memory. The default location for EXEC, X'7000', is also suitable for this use, and you can even LOAD or RUN a BASIC program, provided that program is small (<5K bytes). E.g.

```
LOAD UTIL1/CIM
BASIC
@2(2)
@3(49152)
DEFUSR=49152-65536
IF '@1'()'' THEN RUN'@1/BAS'
```

However, if you want to use a CLIST to drive several operations in BASIC (e.g. to RUN a series of BASIC test cases) then X'7000' is not a suitable address for EXEC/CMD, since any large BASIC program will overwrite it. Instead you should run EXEC at the top of memory, in an area that must be PROTECTED, when you enter BASIC.

To run EXEC at the top of memory, or at any address other than X'7000', you must first RELOCATE it to that address. RELOCATE is a utility provided on the product disk. Since it modifies the file on which it works, make sure you have a backup of EXEC/CMD before using it, in case of disk error, etc. Install RELOCATE/CMD on your system and type

RELOCATE EXEC/CMD (enter)

The program will tell you the current location of EXEC, and also its length. Subtract this length (or simply use 1536) from your top-of-memory address to get the target location.

Type in this target address in response to the prompt, in decimal, or hexadecimal preceded by &, (or any lower address).

On a 48K machine, top-of-memory is 65536, minus 1536 gives 64000, and this value would be suitable as the relocation address. It must then be used in answer to MEMORY SIZE, when you enter BASIC. The following program will run under EXEC, bringing up BASIC, and running a test case, printing the TIME before and after execution:

```
BASIC
@2(2)
33(64000)          (protect EXEC in high memory)
PRINT TIME$
RUN"@1/BAS"
PRINT TIME$
```

RUNNING CLISTS IN THE BASIC ENVIRONMENT

Unlike DO processes, CLISTs will run in conjunction with BASIC, allowing you to initialise DEFUSR values, activate machine-code programs, time test cases, simulate user input, and so on. EXEC behaves as if it were the keyboard, delivering characters one at a time to DOS or BASIC. The CLIST does not load into the normal BASIC program area, so it does not get in the way of any BASIC program you want to run. Although a CLIST is a program, all the statements in it are executed in DIRECT mode (i.e. as if keyed directly). So you can use any BASIC language that would make sense in DIRECT mode. E.g. PRINT TIME$, or LPRINT TIME$ will display the current time on the screen or the printer. IF THEN ELSE can be used, on parameter strings, or on BASIC variables. INPUT, READ, or RETURN, however, would be rejected, since they are not legal in DIRECT mode in BASIC. RUN 100, GOTO 100, or GOSUB 100, will be treated as operations on line 100 of the current BASIC program. I.e. GOTO 100 will NOT branch to line 100 of the current CLIST. FOR NEXT can be used, but only within a single line, as is true for DIRECT mode. Variables can be set, and used later, but only within the same constraints as for keyboard commands. RUN or CLEAR will destroy all current values. Variable values can be read from files, but in this case the file number will have to be kept distinct from any used in a BASIC program, and running that program may CLOSE the direct file.

BRANCHING IN CLISTS

In order to allow loops in CLISTs EXEC supports a new command, the LINE command. LINE n means GOTO n in the EXEC program. If n is not found, then the program exits. In fact LINE 0 or LINE 1000 may be used as a way to terminate an EXEC program, where 0 or 1000 are non-existent lines.

Here is the previous example, extended to loop round prompting for the names of test cases to run:

```
10 BASIC
20 @1(2)
30 @2(64000)
40 DEFSTR T
50 TESTCASE="@0"                        Prompt for next test case
60 IF TESTCASE="" THEN LINE 1000        Quit, if finished
70 PRINT TIME$
80 RUN TESTCASE+"/BAS"
90 PRINT TIME$
100 LINE 40                             Loop round for more
```

Notes.

1) The use of a descriptive word like "TESTCASE" acts as a suitable prompt for the end user, since it will appear on the screen.

2) The value of TIME$ cannot be saved in a BASIC variable across the RUN statement, because RUN clears all variables. To compute the actual running time of the program, TIME$ would either have to be saved on a file, or better POKEd into an area of high memory and retrieved with PEEK after the RUN.

3) If the test program requires keyboard input, then either this must be incorporated into the EXEC program itself, or better, the input should be switched "permanently" to the keyboard, using @$ as described later.

COMMENTARY IN CLISTS

Remarks in a CLIST have an important use as "commentary" in demonstrations or tutorials, and EXEC supports a variety of ways in which they can be used.

Lines starting with REM act as true remarks, i.e. you can include them in a CLIST to remind you what the program does. They can only be used while running in the BASIC environment, not under DOS, and they are displayed and ignored, exactly as REMarks are in direct mode.

Remarks using the single quote format are interpreted in a special way by EXEC. They can be used under DOS, as well as BASIC. Normally, the comment text is placed not on the screen, but in a separate comment buffer, equivalent to a second unseen screen. Whenever the program flow pauses for keyed input, the user can press the CLEAR key to swap between the real screen, and the comment screen. Whenever he types in real input the real screen will be restored, and the input shown on that, as appropriate. The net effect is that the real screen shows what the user would see if he were typing in the CLIST by hand (plus his own input). The commentary screen is always available for help, explanation, or instruction, but as a separate process.

The main options available are:

1)Text on the commentary screen can remain invisible until the user presses CLEAR.

2)The commentary screen can be forced into view, by a control code, @!, in the CLIST.

3)The duration of visibility of commentary can be controlled either as a fixed delay, or by pause-until-ENTER.

```
10 BASIC
20 @1(2)
30 @2(49152)
40 'THIS IS A COMMENT          Put on the comment screen, not yet visible
50 TESTCASE$="@@"               Pause for input, CLEAR will show comment
60 @!                          Force comment screen to be visible, wait for ENTER
70 etc
```

Note that the second commentary screen, if used, has to be saved in memory by EXEC, and that the memory used for this immediately FOLLOWS the module EXEC/CMD, in core. This requires a further X'400' bytes to be set aside following EXEC, when you relocate EXEC, and when you set up protected memory. I.e. the total size of EXEC will be just under X'A00', or 2560 bytes, in this case. So before you try out the use of commentary screens, go back and RELOCATE EXEC/CMD to make this space available. You will normally want have different versions of EXEC available, since CLISTs for your own use will not use commentary screens, but those for tutorials will.

Another form of comment is distinguished by a starting single quote, immediately followed by a plus symbol. This uses the main screen itself, clearing it before and after the comment. The comment is displayed in LARGE CHARACTERs suitable for demonstrations. Typically this will be run with controlled delays, to give time to read the comment, and the actual program, but without interaction by the end-user. No extra buffer space is needed for this comment.

INVOKING CLISTS WHEN IN BASIC

All the examples shown so far have shown a program that would be invoked from DOS, e.g. by EXEC UPDATE param1 param2. If you are already in the BASIC environment, then you can invoke EXEC by e.g.

CMD"I","EXEC UPDATE param1,param2"

CMD"I" has the effect that the program (i.e. EXEC) is invoked under the DOS environment, not BASIC, and so the first line of the EXEC program must be a DOS command, not a BASIC statement. I.e. ALL EXEC programs must be written as if they start in the DOS environment, whether or not they are invoked by

EXEC clist or

CMD"I","EXEC clist"

Of course an EXEC program can start with the DOS command BASIC * (i.e. re-enter BASIC retaining the existing program). This will maintain the number of files, and memory protection limit, but under TRSDOS, at least, DEFUSR values need to be restored.

As an example, suppose you have a utility called SQUEEZE, which reduces the size of BASIC programs by removing blanks and remarks. You want to write a CLIST, called SQUEEZE/ASC, which saves the original BASIC, and then squeezes and saves the compressed BASIC, printing out memory used before and after. You would load and edit your BASIC program, and then invoke the CLIST from the BASIC environment with:

```
CMD"I",'EXEC SQUEEZE MYPROG'
```

where SQUEEZE/ASC would be:

```
BASIC *                 Re-enter BASIC immediately
SAVE "@1/BAS"           Save original version of MYPROG
PRINT 38298-MEM         Display original program size
CMD"I",'SQUEEZE"        Invoke SQUEEZE utility
PRINT 38298-MEM         Display new program size
SAVE "@1/SQU"           Save compressed version
```

ENCAPSULATING CLISTS

For your own use the most efficient way of using EXEC is to hold one copy of EXEC/CMD on disk, along with the separate CLISTs, such as UPDATE/ASC, SQUEEZE/ASC, etc. However, if you prepare a CLIST that you want to make available to a third party, either as a gift, or for sale, then you should follow the instructions below to produce an encapsulated "command" which ties the module EXEC/CMD to the program it will interpret. There are three reasons for doing this:

1)It makes the program easier to invoke. E.g. instead of typing EXEC UPDATE PROG1 the end-user only has to type UPDATE PROG1.

2)It is more efficient, both in disk space and in invocation speed. (It's LESS efficient in disk space, if you have many CLISTs, because it involves storing several copies of EXEC.)

3)It avoids copyright infringement. If you were to give away, or sell, UPDATE/ASC in its original form, then you would have also to provide the module EXEC/CMD, and this would contrary to the copyright concession under which Southern Software programs are distributed. But if you encapsulate UPDATE with EXEC, then, although you are still selling or giving away a part of EXEC, this will be ignored by Southern Software, and no royalties will be claimed.

For this discussion assume that EXEC loads at X'7000' to X'75FF'. When you run EXEC it uses its first 256 bytes (X'100') as an input buffer, destroying its own initialisation code in the process. When execution has completed, the buffer will be intact, in core. So provided your CLIST, (e.g. UPDATE/ASC) is less than 256 bytes long, it will be completely contained within EXEC's memory area, after EXEC has completed.

From DOS, DUMP the core-image of EXEC, including the 256-byte buffer, but set the module entry point (the TRAnsfer address) to the start of EXEC plus 256, i.e. immediately following the buffer. Under TRSDOS the DUMP command would be e.g.

```
Model I      DUMP UPDATE/CMD (START=X'7000',END=X'75FF',TRA=X'7100')
Model III    DUMP UPDATE/CMD (START=07000,END=075FF,TRA=07100)
```

Note that the syntax of DUMP varies between the different DOS's, so check the one for your DOS.

This new module, UPDATE/CMD, can be invoked under DOS simply by typing UPDATE followed by the parameters, if any, or from under BASIC by CMD"I","UPDATE param1 etc". You dont need to ship UPDATE/ASC, since it's encapsulated in UPDATE/CMD.

If your CLIST is longer than 256 bytes, this simple form of encapsulation wont work. In this case run EXEC against your product CLIST, as above, and DUMP the core-image, but set the TRAnsfer address to 256+5 bytes from the start of EXEC. The resulting module will open and read the CLIST used during encapsulation. You dont need to DUMP the buffer area, either, although of course that area of memory will be used, when the command runs. You must ship the CLIST with the command. Under TRSDOS the DUMP commands would be:

```
Model I      DUMP UPDATE/CMD (START=X'7105',END=X'75FF',TRA=X'7105')
Model III    DUMP UPDATE/CMD (START=07105,END=075FF,TRA=07105)
```

COMPLETE LIST OF EXEC FUNCTIONS

The discussion so far has hopefully enabled you to write useful CLISTs for a number of applications. This section gives a complete summary of all the different functions available through EXEC, along with reasons why you might need to use a particular function. Many of the functions are only necessary if you write CLISTs for commercial, third party use.

1) CLIST format. A CLIST is a "BASIC" program which must be saved in ASCII (uncompressed) format. If it has the extension /ASC, then the extension can be omitted in the invocation. But other extensions can be used, if specified explicitly.

2) Invocation format. A CLIST called X/ASC can be invoked by:

```
EXEC X param1 param2...                 (under DOS) or
CMD"I","EXEC X param1 param2..."        (under BASIC).
```

When encapsulated into a command, X/CMD, the invocation is:

```
X param1 param2...
CMD"I","X param1 param2..."
```

The parameters are strings separated by single blanks or commas. Double blank or double comma implies a null parameter.

3) Parameter substitution. @1 ... @9. The occurrence of @n, where n is a digit from 1 to 9, in the body of a CLIST will be replaced by the nth parameter string of invocation.

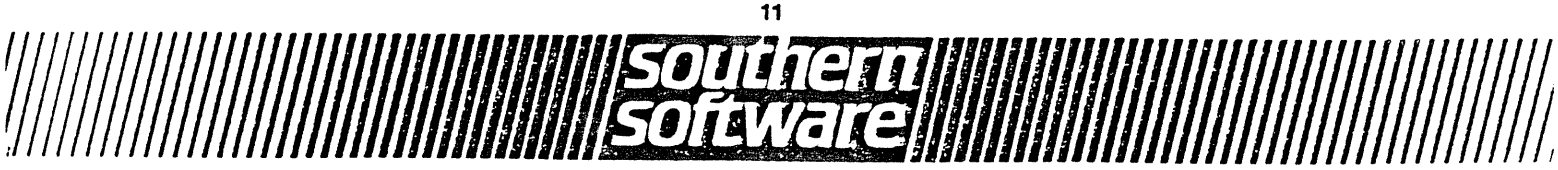
4) Input parameters. @0. If @0 appears in the body of a CLIST, then keyboard input is solicited at that point, and the keystrokes which are typed in, up to but not including the terminating line-end (ENTER), are substituted in place of @0.

5) Parameter defaults. <...>. If @n, including @0, is followed immediately by text within < > brackets, and if the corresponding parameter is null, then the text within < > is substituted for the parameter.

6) Continuous input. @$. If @$ appears in the body of a CLIST, then input is switched to the keyboard, up till, but not including, the next @ character typed. This is of value when running e.g. a batch of test cases for which the input is unpredictable. Use @$ after the RUN statement, and instruct the operator to give control back to the CLIST by typing @ at the end of the test case. @$ should also be used where run times are being measured. EXEC normally takes a significant amount of time, just ignoring unwanted keyboard polls. @$ avoids this overhead.

7) Branching in CLISTs. LINE n. Use the statement LINE n in a CLIST to branch to line number n. EXEC implements this by rereading the CLIST from the top (which could involve disk I/O) until line n is found. If line n does not exist, then the CLIST exits cleanly, without any error indication.

8) Illegal BASIC. @"...". A CLIST may contain arbitrary application input as well as commands, and some characters may be illegal in BASIC. Examples are lower-case text, a null line, or the ? symbol, which BASIC converts to PRINT. In these cases surround the text with @" ". BASIC will treat the enclosed text as literals, and EXEC will simply ignore the @" and closing ".



9) Slow Keystrokes. @+ and @-. In a demonstration or a tutorial it may be attractive to have the data characters appear slowly, as if keyed by hand. @+ in the CLIST will turn on a delay of about a third of a second per character, while @- will revert to normal, i.e. full speed.

10) Input prompt. @line-end. It may be that you want a tutorial to pause indefinitely, while the user reads the screen, and yet you dont want the user to enter any real input. @ on the end of a line, or on a line by itself, will prompt for input, but will ignore any keystrokes except for CLEAR, which swaps screens, and ENTER, which lets the CLIST continue.

11) CLIST REMarks. A line starting with REM will be ignored (in the BASIC environment only), and may be used as a remark in the CLIST.

12) Direct screen commentary. '+. A "remark" that starts with a single quote and a plus will be displayed in large characters (16 per line) on the real screen. The screen is cleared before and after the comment is displayed, and no control is given to the user. So normally you will need to add a delay (see below) before and after the comment. This is intended for shop-window, or exhibition use. For an effective large-character display each line should be centred horizontally, and the whole message should be centred on the screen vertically, using down-arrow graphics. (See ACCEL4 sample CLIST).

13) Commentary screen. Remarks that start with a single quote but no plus will be directed to a second, invisible, screen buffer. This resides in memory immediately after EXEC, and is X'400', i.e. 1024 bytes long. Space must be made available for it, if such comments are used. This form of commentary is intended to be used with detailed tutorials, and has the advantage that the true screen is not corrupted by the commentary, which may contain help information, instructions, or explanation.

14) Commentary continuation. @line-end. Each new single-quote comment clears the commentary screen by default. Since BASIC allows lines only to be 256 bytes long, this would restrict commentary to 256 bytes. But if you put a @ at the end of the CLIST comment line, then EXEC assumes the next line is a continuation comment, i.e. it adds it to the screen without clearing it. The continuation line should start with a single quote, but a plus is not needed.

15) CLEAR key as swap. Whenever the CLIST pauses for keyboard input, the user may press the CLEAR key to toggle back and forth between the real screen and the commentary screen. If no commentary screen exists, then CLEAR has no effect. When the user types real input, EXEC will automatically revert to the real screen. The CLEAR key cannot be used as application input, except in continuous input mode.

16) Forced swap. @!. The CLIST can force the commentary screen to become visible with @!. The CLIST then pauses till the user types ENTER. (CLEAR will reshow the real screen).

17) Pause and swap. @@. This is similar in effect to the above, except that an extra pause-till-enter is implied before the commentary screen is shown. This would be used when it was necessary to give the operator time to read the real screen before forcing the commentary screen.

18) Delay and swap. @###. The combination @# gives a delay of about a second before swapping to the commentary screen. Multiple # symbols give multiples of the delay. No operator intervention is necessary.

19) Delay. # in comment. The occurrence of # anywhere in a comment, will give a delay of about a second. If the # is at the beginning of the comment, then the delay occurs before the screen is cleared. This would be used with large-character displays, for instance, to ensure the user has time to read the screen. No operator intervention is necessary.

PITFALLS AND RESTRICTIONS

The ideal command list processor would allow any combinations of commands, statements, and applications to run, exactly as they would if driven by hand. Unfortunately that's not always achievable, although EXEC gets as near this aim as possible.

1) Memory used by EXEC. EXEC requires X'600' bytes to run, plus a further X'400' if the commentary screen is used. If your application already fills memory you will be unable to run it under EXEC.

2) Memory Clearing. On Model III all of RAM will be cleared by certain programs running with execute-only protection. In particular, the KILL command may clear memory, depending on the machine state. If this happens when running under EXEC, EXEC will be destroyed, and the machine will reboot, or hang. The DOS command CLEAR cannot be used under EXEC.

3) Memory-mapped input. There is a certain class of applications which processes input not by using the ROM next-key routine (X'0049') but by examining the bits in the memory area which maps the keyboard. These will not run under EXEC. This includes full-screen editors, and word-processors. Generally the reason for using memory-mapped input is because the application needs to treat special key combinations as shifts, control keys, etc.

4) Pausing a display. Commands like DIR, or LIST, that produce a continuous display of more than a screenful will not be pausable under EXEC, and will not generally be usable.

5) ASCII bug. Although you can produce a BASIC program with a line which starts with a digit, and SAVE it correctly in ASCII, it will not reload. Instead, surround the digit with @"...". E.g.

```
10 BASIC
20 @"2"
30 @"49152"
40 etc.
```

6) Timing inaccuracies. Although EXEC is recommended for driving benchmark timings, it will normally be necessary to use the continuous input option, @$, across the timed code, to avoid the effects of EXEC returning the keyboard poll.

7) Line-end in CLISTs. Some control codes include the line-end in their definition. These are the single-quote comments, and the @line-end pause prompt. The other control codes, such as @+, do not "absorb" any line-ends. This can be significant to you if your CLIST is driving an application that expects precise data. E.g.

```
20 PRINT X'Comment text
30 Y=1
```

is equivalent, programming terms, to PRINT XY=1, whereas

```
20 RUN"PROG/BAS"
30 @+
40 10,20
```

results in PROG receiving a null input line before the data 10,20.

8) Reserved characters. The @ and CLEAR keys are preempted by EXEC, and cannot be used as application data. (CLEAR can be used after @$). +, #, and @ cannot be used in comments.