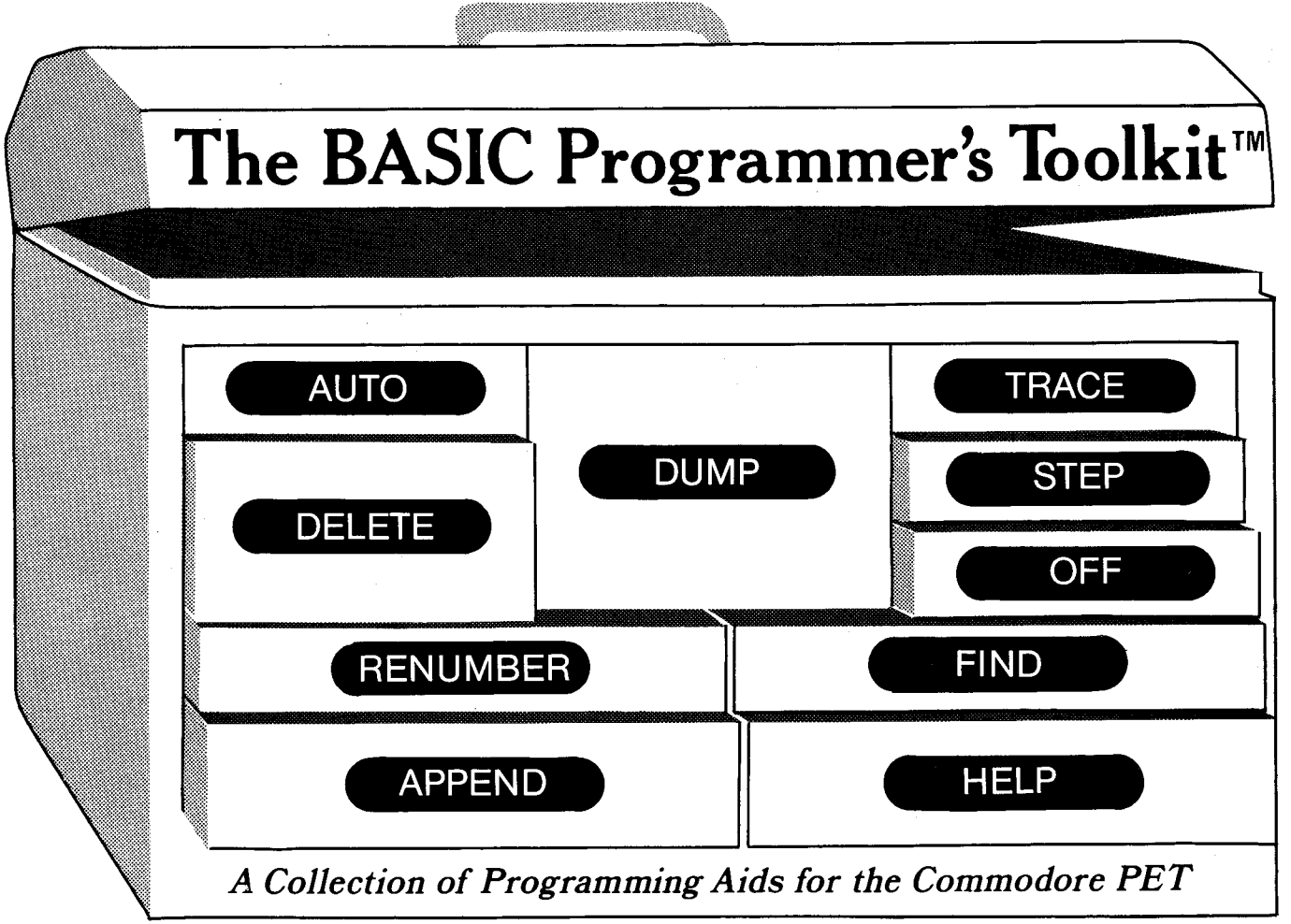

The BASIC Programmer's Toolkit™
AUTO
DELETE
DUMP
TRACE
STEP
OFF
RENUMBER
FIND
APPEND
HELP
A Collection of Programming Aids for the Commodore PET

THE BASIC PROGRAMMER'S TOOLKIT

A
Collection of Programming Aids
for the Commodore PET

USER'S GUIDE

Chuck Bond is the creator and main implementor of the BASIC Programmer's Toolkit.

This user's guide was originally written by Gregory Yob.

Revision 1 August 1979

What is it?

Your new BASIC Programmer's Toolkit[tm] is a machine language program which is provided in a 2 Kilobyte ROM. When this is installed in your PET, and the Toolkit's program activated, your PET's BASIC has ten new and very useful commands

AUTO	Provides new line numbers when you are entering BASIC program lines.
RENUMBER	Renumbers your BASIC program, including all GOTOs and GOSUBs.
DELETE	Removes groups of BASIC program lines.
FIND	Locates and displays the BASIC program lines that contain a specified string.
APPEND	Adds a previously SAVEd program to the one currently in your PET.
DUMP	Displays the names and values of all the variables used by your program (excluding arrays).
HELP	If your program stops due to an error, HELP displays the offending line and shows where the PET detected the error.
TRACE	As a program runs, the last six line numbers being executed are shown in the upper right corner of the PET's screen.
STEP	Executes one BASIC line and stops. Pressing SHIFT executes the next line. The line number is displayed in the upper right corner of the screen.
OFF	Turns TRACE or STEP off.

Getting Started

INSTALLATION

There are several versions of the Commodore PET -- models 2001-4, 2001-8, 2001-16, and 2001-32, and the PET's main circuit board has changed in many ways. For our purposes, though, there are only two kinds of PET, the "old" and the "new". The "old" PET has a small keyboard and the tape unit on the front, and on the right side there is a Memory Expansion Port. This port consists of PC fingers to which a connector can be attached. The "new" PET has a large keyboard, a separate tape unit which connects to the tape socket in the rear of the PET, and the Memory Expansion Port is made up of many pins that extend upwards from the main circuit board.

Please follow the instructions appropriate to your PET. If you have some expansion memory, such as the Expandamem, contact your dealer for installation information.

TOOLKIT INSTALLATION FOR "OLD" PETS

Your Programmer's Toolkit is mounted on a 1½" by 5" PC card which has an edge connector for attaching to the Memory Expansion Port. A short wire with a small connector extends from the Toolkit PC card and attaches to the Cassette Port in the back of the PET.

As shown in the diagram, plug the Toolkit onto the Memory Expansion Port with the ICs on the PC card facing the PET, and the wire extending to the back. Then plug the wire's connector into the 2nd Cassette Port with the wire towards the corner of the PET. The connector on the end of the wire is polarized so that it cannot be plugged in upside down.

BE SURE THE PET'S POWER IS OFF WHEN YOU ATTACH OR REMOVE YOUR TOOLKIT!!!

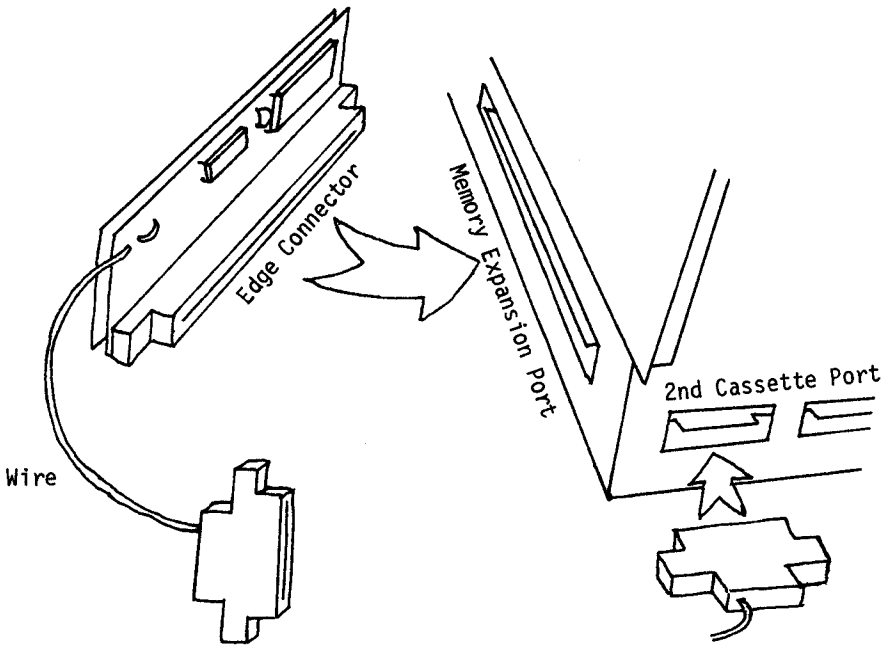


Figure 1. Installation for "OLD" PETS

TOOLKIT INSTALLATION FOR "NEW" PETS

For the new PET your Programmer's Toolkit consists only of the ROM IC with the Toolkit's program inside. If you have by some mischance got a Toolkit for the "Old" PET, check with your dealer. In most cases the chip from the "Old" PET version of the Toolkit cannot be used in the "New" PETs.

If you haven't opened your PET and looked inside, or if you have never installed or removed a 24 pin IC, YOU ARE STRONGLY ADVISED TO HAVE SOMEONE WHO HAS DONE THESE THINGS INSTALL THE TOOLKIT FOR YOU!

Turn the power off and remove the power cord from the wall socket. PLEASE DO THIS!

Unscrew the four screws on the bottom of the top half of the PET (about 5" back from the front of the PET on each side). Use the correct size phillips-head screwdriver to avoid damaging the screws.

SLOWLY open the PET's case by lifting the front. Inside will be several cables, some of which might be too short to allow full opening of the PET. Find these cables and gently disconnect them from the PET's main circuit board. When the case is fully open, set the bar (on the left side) so that the case will remain opened.

In the PET there is a row of 7 ROM sockets, with three vacancies on the right side. As shown in the diagram, install the Toolkit IC next to the PET's ROM ICs, with the notch and dot as shown. BE SURE THE NOTCH AND DOT ARE PLACED CORRECTLY!!

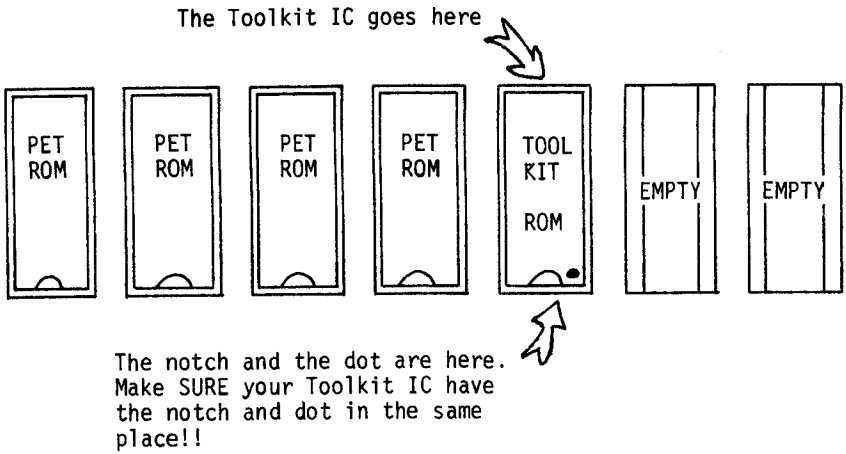


Figure 2. Installation for "NEW" PETS

SOME CAUTIONS ON INSTALLING ICs: 1) Touch the metal case of the PET with your hand before installing the Toolkit IC. This is to discharge any static electricity that might damage the ICs. 2) Remove the IC from the black plastic carrier that it has been shipped in. 3) CAREFULLY place the IC in the socket and make sure all the pins are aligned with the socket's receptacles. This may take a little cautious fiddling. 4) Press SLOWLY and FIRMLY on the IC until it slides completely into the socket. 5) Check for bent pins -- especially for a pin that has bent UNDER the Toolkit IC.

If you have bent any pins, remove the IC carefully using a small blunt knife or a metal nail file. Using needlenosed pliers, straighten the pins, and try installation again. You can only do this once or twice before the pins break, so get it right the first time!!!

After the IC is installed and checked, reconnect any cables that were disconnected, close the PET, and replace the four screws.

STARTUP & CHECKOUT

Make sure you have installed your Toolkit correctly, turn on your PET, and then enter either

SYS 45056

or

SYS 11*4096

and press RETURN.

Your PET should reply with:

(C) 1979 PAICS

READY.

If this does not happen, TURN THE POWER OFF IMMEDIATELY and check your installation. If you have an "Old" PET, check that the 2nd Cassette connector is correctly placed and that the Toolkit board is completely seated onto the printed circuit board. "New" PET owners should check the position of their IC and the direction of the notch and dot.

If the Toolkit message still does not appear after you have checked your installation, contact your dealer for assistance.

After the Toolkit has been initialized with the SYS command, the new commands are available for use. The remainder of this manual describes the commands in detail.

The Commands

AUTO (First Line Number) , (Line Number Interval)

AUTO provides automatic line numbers when you are entering BASIC program lines. The first line number will be displayed, and when the BASIC line is entered via RETURN, the line number interval is added and the new line number will be displayed.

To leave AUTO mode, just press RETURN. If you don't provide a first line number or line number interval in the command, AUTO assumes that you meant AUTO 100,10. AUTO will remember the last line number and interval if a previous AUTO was executed.

EXAMPLES:

Type in NEW to remove any programs in your PET, and then type:

AUTO

The PET will respond with:

100 ▮ (The ▮ indicates the PET's cursor.)

The line number 100 will appear, and the cursor is in the right position for entry of a BASIC line. You may enter as many lines as you like, each terminated with RETURN. To leave AUTO mode, press RETURN without entering a line.

```
AUTO
 100 REM ONCE UPON A TIME
 110 REM A FAIRY PRINCESS
 120 REM MET MY PET COMPUTER.
 130
▮
```

You are now out of AUTO mode, and can do other things, like RUN. AUTO line numbering can be resumed where you left off just by typing AUTO again.

```
RUN

READY.
AUTO
 130
```

AUTO always remembers the last line number it gave to you.

If you want to start with a new line number, you can provide it on the AUTO command:

```
AUTO 456
 456 REM THEN SHE SMILED AT IT
 466
```

To change the line number interval requires that you provide both the first line number and the interval:

```
AUTO ,15

?SYNTAX ERROR                (It didn't work!)
READY.

AUTO 1000,15
1000
```

If you change only the first line number, AUTO remembers the old interval:

```
AUTO 2000
 2000 REM AND ASKED 'WHO ARE YOU?'
 2015
```

There are some situations in which AUTO will behave in an unexpected manner. The section on GOTCHAS describes these unusual cases.

RENUMBER (First Line Number) , (Line Number Interval)

RENUMBER renumbers the entire program currently in your PET. RENUMBER will change all line numbers, including those in IF - THEN, GOTO, GOSUB, ON-GOTO, ON-GOSUB, RUN, and LIST. References to non-existent line numbers are changed to 63999.

If no First Line Number or Line Number Interval are given, RENUMBER assumes you meant RENUMBER 100,10.

EXAMPLES:

Enter a small program on your PET:

```
NEW
10 REM A RENUMBERISH EXAMPLE
20 GOTO 10
30 GOSUB 1000
40 IF K=2 THEN 500
1000 REM SAMPLE SUBROUTINE
1002 RETURN

RENUMBER

READY.
LIST
 100 REM A RENUMBERISH EXAMPLE
 110 GOTO 100
 120 GOSUB 140
 130 IF K=2 THEN 63999
 140 REM SAMPLE SUBROUTINE
 150 RETURN
```

The GOTO in Line 20 and the GOSUB in Line 30 have been changed to reflect their new line numbers. Line 40 now jumps to 63999 since the program did not have a Line 500 in it

(Note: After you RENUMBER a program, you can then find all the illegal line number references by using FIND 63999.)

To start the line numbers with 5000 instead of 100, just use:

```
RENUMBER 5000
```

```
READY.
LIST

 5000 REM A RENUMBERISH EXAMPLE
 5010 GOTO 5000
  (etc...)
```

If a different line number interval is desired, you must provide the starting line number:

```
RENUMBER ,3

?SYNTAX ERROR

RENUMBER 300,3

READY.
LIST

 300 REM A RENUMBERISH EXAMPLE
 303 GOTO 300
  (etc...)
```

If the last new line number in your program is going to be larger than 63999, RENUMBER will tell you so:

```
RENUMBER 50000,10000
?OUT OF RANGE ERROR
```

No part of your program will have been renumbered if you get an ?OUT OF RANGE ERROR.

DELETE (Line Number) - (Line Number)

DELETE removes BASIC lines by specifying the line number range in the same way that LIST specified the BASIC lines to be displayed. For example, DELETE 100-200 will remove all lines from 100 to 200. DELETE -100 will remove all lines from 0 to 100. DELETE 100- removes lines 100 to the end of the program.

EXAMPLES:

```
NEW

10 REM ONE LINE FOR THIS TIME
20 REM ANOTHER TO FILL THE BILL
30 REM AGAIN TO SATISFY THE YEN
40 REM OK, KEEP IT THIS WAY
50 REM FLY ALONG, GET A LITTLE HIGH
60 REM DON'T STUMBLE WHEN YOU MUMBLE
DELETE 30-45

READY.
LIST
 10 REM ONE LINE FOR THIS TIME
 20 REM ANOTHER TO FILL THE BILL
 50 REM FLY ALONG, GET A LITTLE HIGH
 60 REM DON'T STUMBLE WHEN YOU MUMBLE
```

DELETE's line number range works like LIST -- there doesn't have to be a line at the numbers you specify.

DELETE won't work without some line numbers. This prevents the loss of the entire program by mistake. Use NEW to delete the entire program

```
DELETE

?SYNTAX ERROR

DELETE 50

READY.
LIST

 10 REM ONE LINE FOR THIS TIME
 20 REM ANOTHER TO FILL THE BILL
 60 REM DON'T STUMBLE WHEN YOU MUMBLE.
```

APPEND (Program Name)

APPEND will load a previously saved program from cassette tape and add it to the end of the program already in your PET's memory. The APPEND command does not interleave or overwrite the program in the PET.

The program name works in the same way that LOAD does. You can provide either "name" or a string containing "name" and the PET will search the tape until it finds a program with the same initial characters as "name". You may also specify cassette unit 1 or 2, exactly as with the LOAD command.

EXAMPLES:

Enter this small program and SAVE it:

```
200 REM THIS IS PROGRAM ONE          (Be sure to NEW
210 REM TO SHOW HOW APPEND            to remove any
220 REM WORKS ON YOUR PET             other program.)
SAVE "FIRST PROGRAM"
```

Now remove this program and enter another one:

```
NEW

READY.
100 REM THIS IS ANOTHER PROGRAM
110 REM TO CONTINUE THE EXAMPLE.
```

Rewind your FIRST PROGRAM tape, and now add it via APPEND:

```
APPEND

PRESS PLAY ON TAPE #1
OK

SEARCHING
FOUND FIRST PROGRAM
APPENDING
```

The PET looks for a program just like LOAD does, and when the program is found, APPENDING appears to let you know what's going on.

```
LIST

 100 REM THIS IS ANOTHER PROGRAM
 120 REM TO CONTINUE THE EXAMPLE.
 200 REM THIS IS PROGRAM ONE
 210 REM TO SHOW HOW APPEND
 220 REM WORKS ON YOUR PET.
```

You can repeat APPEND as often as you like until memory is full. Another APPEND of FIRST PROGRAM results in:

```
LIST

 100 REM THIS IS ANOTHER PROGRAM
 110 REM TO CONTINUE THE EXAMPLE.
 200 REM THIS IS PROGRAM ONE
 210 REM TO SHOW HOW APPEND
 220 REM WORKS ON YOUR PET.
 200 REM THIS IS PROGRAM ONE
 210 REM TO SHOW HOW APPEND
 220 REM WORKS ON YOUR PET.
```

APPEND simply adds the program on tape to the end of the current program. It does not insert or overwrite lines within the current program.

If you have several routines stored on a tape, you can select the ones you want by mentioning their names -- just like LOAD does. Suppose your tape contains:

```
APPLE
PEAR
APPLICATION
PEACH
```

APPEND "APPLE" will find the first program -- so will the use of APPEND "APP". APPEND "PEAC" finds the program PEACH.

Since APPEND disregards line numbers, it is your responsibility to make sure your APPENDed programs have increasing line numbers. When BASIC executes IF - THEN or GOTO / GOSUB, the line numbers are searched from the start of the program until a line number equal to or larger than the jump's number is found. This means that the program won't be able to find out-of-order line numbers and you will see an ?UNDEF'D STATEMENT ERROR.

F I N D (BASIC code) , (Line Number) - (Line Number)

F I N D "(string)" , (Line Number) - (Line Number)

FIND locates and displays all lines that contain a specified fragment of BASIC code or a quoted string constant. The line numbers select the lines to be searched in the same way that LIST selects lines to be displayed on the screen. If the line numbers are omitted, the entire program is searched.

If the item being searched for is not a string surrounded by quotation marks, then the BASIC program itself is searched, excluding any quoted strings. If the item being searched is surrounded by quotes, then only quoted strings within the program will be searched. For example, FIND A will find all occurrences of the variable A such as A=3.1415, and FIND "A" will find the character A inside strings such as PRINT "MOVE AGAIN".

Note that BASIC programs are stored internally as "chunks" of text called tokens, and not just as sequences of characters. Each BASIC keyword, such as PRINT and THEN, is a separate token, as are special characters such as = and +. When FIND looks at BASIC text, tokens must match entirely in order for the search to be successful. Thus in order to find, say, all PRINT statements in a program, you must type FIND PRINT, and not just FIND PRI. This often works to your advantage, however, because you can also type FIND I to find all occurrences of the variable I without also finding all PRINT statements. Of course, you can also look for sequences of tokens, as in FIND I = or FIND IF X=0.

When you use FIND in a large program, more than a screen full of lines might be listed. Just as for LIST, you may press RVS to slow the display, and STOP to stop it entirely.

EXAMPLES:

Here is a short program to use as an example:

```
10 PRINT" [clear screen][4 cursor down][4 cursor right]
     THIS IS A SILLY"
20 PRINT" [4 cursor right] EXAMPLE FOR YOUR"
30 PRINT" [4 cursor right] SERIOUS CONSIDERATION."
40 FOR J=1 TO 1000: NEXT J
50 INPUT" [5 cursor down] TRY IT AGAIN";A$
60 IF A$="YES" THEN 10
70 IF A$="NO" THEN END
80 PRINT" [clear screen] ":GOTO 40
```

Let's try a few BASIC code searches:

```
FIND IF
 60 IF A$="YES" THEN 10
 70 IF A$="NO" THEN END

READY.
```

Notice that FIND prints the entire line if the search is successful. You can then use the screen editor to change the lines as needed and reenter them.

The line numbers can be used to search portions of your program. These work in exactly the same way that LIST does.

```
FIND A$,70-
 70 IF A$="NO" THEN END
```

Remember to use quotes to search inside strings. For example:

```
FIND FOR
 40 FOR J=1 TO 1000: NEXT J

FIND "FOR"
 20 PRINT" [4 cursor down] EXAMPLE FOR YOUR"
```

The FOR without quotes looked for tokens, and the search ignored the FOR inside the quotes in line 20. The "FOR" with quotes searched in the quoted parts of the program only.

You can search for cursor movements and graphics characters by putting them inside quotation marks:

```
FIND "[cursor down][cursor right]"
 10 PRINT" [clear screen][4 cursor down][4 cursor right]
     THIS IS A SILLY"
```

Sometimes it is important to distinguish lines from other similar lines. To do so, just FIND with a more complex program segment; it need not be a single item.

```
FIND THEN 10
 60 IF A$="YES" THEN 10
```

There are two lines with the keyword THEN, so using the item THEN 10 made the search more precise. When making complex items, be careful with blanks.

```
FIND THEN10

READY
```

Your item must match the blanks in the program lines as well as the non-blank parts.

D U M P

DUMP displays all of the non-array variables present in the PET's memory. The variables are displayed in the form:

(variable) = (value)

which permits the use of the screen editor to change their values.

When a program has many variables in it, the DUMP display can be "frozen" with the SHIFT key. STOP will exit the DUMP entirely.

The variables are displayed in the order that they were created.

EXAMPLES:

Execute the following direct statements on your PET:

```
CLR
X=3                          (This example ignores the
Y=2                           PET's reply of READY)
A$="HELLO OUT THERE"
C%=256
```

To see these variables, use DUMP:

```
DUMP
X=3
Y=2
A$="HELLO OUT THERE
C%=256
```

Notice that the variables are displayed in the order that these were created. DUMP will always display the current value for each variable:

```
Y=123456987
Z=555666777
DUMP
X=3
Y=123456987
A$="HELLO OUT THERE"
C%=256
Z=555666777
```

Try using the screen editor to change these values, like making A$ become "SOMETHING ELSE NOW". When you DUMP again, the values will be changed.

When a lot of variables are in the PET, the screen will become filled and the first variables will scroll off the top of the screen. For example:

```
CLR
A=1:B=2:C=3:D=4:E=5:F=6:G=7:H=8:I=9:J=10
K=11:L=12:M=13:N=14:O=15:P=16:Q=17:R=18
S=19:T=20:U=21:V=22:W=23:X=24:Y=25:Z=26
DUMP

.....
W=23
X=24
Y=25
Z=26

READY.
```

Variables A - D scrolled off the top of the screen. Try DUMP again, and press the SHIFT key a moment later. The display will stop, and remain "frozen" as long as SHIFT remains depressed. By releasing SHIFT for short moments you can look at a DUMP in groups of 2 or 3 variables.

If you press STOP, the DUMP will be aborted:

```
DUMP
A=1
B=2                    (Press STOP immediately after DUMP!)
C=3
D=4

BREAK
READY
```

You can combine SHIFT and STOP to let the display move to the variables that interest you. Use SHIFT to find the variable, and then press STOP to terminate the DUMP.

The Programmer's Toolkit does not DUMP array variables for two reasons. First, DUMPing arrays is much more difficult to do in machine language, especially with 2 and 3 dimensional arrays. Second, in most cases only a few array elements are

of interest when debugging a program, and the rest just clutter up the screen. If you want to display an array, use FOR-NEXT in a direct command:

```
FOR J=Ø TO 2Ø:PRINT A(J):NEXT
```

DUMP provides a very convenient way to display the value of string variables that contain cursor-motion characters. Since the string values are displayed in quotes, the cursor-motion characters are printed as their reverse-graphic equivalents just as they are entered in a program, and they may be easily changed using the screen editor. In contrast, printing the value of a string containing cursor-motion characters causes the cursor motion to occur, which makes it difficult to see what the string actually contains.

HELP

When a program is RUN and an error is encountered, the PET will print an error message and the line number. The HELP command will print the line on the screen and indicate where the PET found an error by printing the "bad" spot in reversed field.

HELP must be used immediately after an error, or else the PET "forgets" the location of the error. (HELP will do nothing if any other command is used first.)

If a program is interrupted via the STOP key, HELP will display the line that includes the last successfully completed statement. The reversed field indicator will be at the end of the last completed statement.

EXAMPLES:

Here is a program with some bugs in it:

```
10 X=10/0
20 Y=(((((((15)))))))
30 Z=12345678912345678912345678912345678912345678 9
```

```
RUN

?DIVISION BY ZERO ERROR IN 10
HELP
 10 X=10/0
```

(These examples ignore the PET's READY. message.)

The zero is in reverse field (indicated here with an underline) which is where the PET decided something was wrong.

```
GOTO 20

?SYNTAX ERROR IN 20
HELP
 20 Y=((((((15))))))))
```

If you carefully count the parentheses, the reversed field indicator is on the 6th right parenthesis. Since there were 6 left parentheses going in, the 7th right parenthesis should have made the error. In most cases, HELP puts the indicator at the character before the suspected error.

Try GOTO 30 and see where the PET gives up.

Lines displayed by HELP are easily changed and reentered by using the screen editor as usual.

The PET easily forgets where errors were because that information is kept in temporary storage. If any command at all is done before HELP, the HELP line is forgotten.

```
RUN

?DIVISION BY ZERO ERROR IN  10
PRINT X
 0

HELP

READY
```

The PRINT X made the PET forget the location of the error, and HELP isn't much help now.

TRACE

The TRACE command turns on a "tracer" which will display the currently executed line number when a program is RUN. The last six line numbers are displayed in the upper right corner of the screen in a reverse-field "window". As lines are executed, their numbers scroll up the "window" with the most recently executed line's number at the bottom.

Pressing SHIFT when a program is running will slow the TRACE down to about 2 lines per second. Even without the SHIFT key, though, TRACE slows the running of a program considerably.

EXAMPLES:

Enter this program into your PET:

```
NEW
10 PRINT"[clear screen]";
20 X=1
30 PRINT"[home cursor]"X
40 X=X+1
50 GOTO 30
```

Now trace it:

```
TRACE

READY.
RUN
```

The screen will clear, and a reverse-field "window" appears in the upper right corner. In the upper left, a number is shown which counts upward. A typical display might look like this:

```
13                                  #50
                                    #30
                                    #40
                                    #50
                                    #30
                                    #40
```

(On the PET, the numbers starting with # will be in reverse-field -- black on white.)

Press the SHIFT key and notice how the display slows down. Now about 2 lines per second appear in the "window".

To stop the program, press the STOP key. STOP will work with SHIFT depressed also.

Try pressing RETURN when you have stopped this program with TRACE. You will get a ?SYNTAX ERROR. Whenever the RETURN key is pressed, the PET scans the entire line -- and when you stopped this program, the cursor was in the 5th line on the screen. On the right, the "window" is still there, and the PET took the #(line number) as your entry.

If you leave TRACE with the cursor in the top 6 lines of the screen, press [cursor down] or [clear screen] to remove the "window" first.

Let's try another program:

```
NEW
10 PRINT"[home cursor]"X:X=X+1:GOTO 10
RUN
```

When this is TRACEd, only line 10 appears in the window -- and only on the bottom line. The TRACE does not fill the window with the line number if the program is in a loop on one line. This was done intentionally since many programs have "GET" loops like this one:

```
222 GET A$: IF A$="" THEN 222
```

TRACE will let you see what happened before the loop by not repeating the 222 in the window.

S T E P

The STEP command activates the "tracer" and then executes one BASIC line and stops. (The line number is displayed in the "window" in the upper right corner of the PET's screen.) To execute the next line, simply press SHIFT.

If you hold SHIFT down, the program will continue to execute lines until SHIFT is not depressed. To exit a program, press the STOP key; this will work regardless of whether the SHIFT key is pressed.

After RUN, SHIFT must be pressed to execute the first line. If a loop is included in a program line, like:

```
10 GOTO 10
```

the line will be executed indefinitely. Press STOP to leave the program.

EXAMPLES:

Here is a small program to demonstrate STEP:

```
NEW
10 PRINT"FIRST THING"
20 PRINT"SECOND ITEM"
30 PRINT"THIRD OBJECT"
40 GOTO 10
STEP

READY.
RUN
```

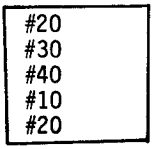
At this point, the "window" will appear in the upper right corner of the PET's screen with line #10 at the bottom. Note that line #10 has <u>not</u> been executed yet.

Tap the SHIFT key briefly. FIRST THING will appear, and the "window" now contains #10 and #20. Each time you press SHIFT, STEP will execute one BASIC line and display the line number of the next statement that will be executed.

Hold the SHIFT key down. Now the "window" will scroll the line numbers in the same way that TRACE does, and the screen will show:

```
FIRST THING
SECOND ITEM
THIRD OBJECT
FIRST THING

(etc)
```



If you press STOP, the program will end, and this works with or without SHIFT depressed. If the cursor is at the bottom of the screen when STOP is pressed, only the last three lines of the "window" will be visible because the PET has to scroll the screen to display the BREAK IN ### and READY. messages.

STEP works like TRACE if a one-line loop is encountered:

```
NEW
10 PRINT"[home cursor]"X:X=X+1:GOTO10
RUN
```

STEP is waiting for you to press SHIFT. Press SHIFT and release it.

The "window" remains the same, with only the #10 shown. However, an increasing number appears near the upper left corner as the increasing X is displayed. When STEP sees a one-line loop, the loop is executed until STOP is pressed. The "window" behaves like TRACE, and does not fill up with repetitions of the same line number.

O F F

OFF turns off the "tracer" which TRACE and STEP activate. After OFF, a running program will no longer display the "window" or wait for the SHIFT key before executing the next line.

EXAMPLES:

Take the example program for STEP and RUN it with STEP turned on. Press the STOP key, and then type OFF.

Now RUN the program again -- the "window" is gone, and the program RUNs as usual.

Gotchas!

Your Programmer's Toolkit has been designed to give you the most powerful and effective extensions to your PET's BASIC for a reasonable price. The Toolkit takes advantage of the PET's ROM subroutines wherever possible, and as a result, the PET's "way of doing things" is reflected in the Toolkit.

By now, you know that a computer does what it is told to do, and that is not always what you meant for it to do. There are many situations where the Toolkit commands will act in a "strange" way -- that is, the Toolkit's action will not be what you expected.

These situations have been called "Gotchas!", and this part describes the Gotchas! that we know about. Take the time to try these situations out, and then the Toolkit won't have nasty surprises for you in the middle of an important program.

INSTALLATION

The old model PETs have the Toolkit installed on the Memory Expansion Port on the right side of the PET. A short wire and connector are provided which supplies +5 volt power from the 2nd Cassette Port. Be sure both connectors are installed properly.

The new PET's Toolkits are installed on one of the PET's ROM sockets on the Main Logic Board. Be sure that you are using the right socket and that Pin 1 of your Toolkit is in Pin 1 of the socket.

PETs with expansion memories that use the Memory Expansion Port will require differing arrangements. Expandamem users need a small ROM board (available from your dealer) which fits on the Expandamem's expansion sockets.

Contact your dealer if you have any questions concerning the installation of your Toolkit.

MEMORY

The Programmer's Toolkit requires ROM space for the machine language program, and a small amount of RAM to remember variable date. The memory space used is:

$ Ø3EØ - $ Ø3FF for RAM
$ BØØØ - $ B7FF for ROM

The RAM is in the upper part of the 2nd Cassette Buffer, and with the exception of APPEND, use of the 2nd Cassette in BASIC programs with the Toolkit may have unpredictable results.

The ROM is placed above the Screen RAM and will not interfere with most PET users, including those with memory expansion RAM. If you have a Computhink Disk, a conflict exists. The PC board which comes with the "old PET" version of the Toolkit has an extra socket which can be used for any B2716-compatible ROM or PROM; it is wired for addresses starting at $9000.

Initialization

The SYS 45056 (or SYS11*4096) command is necessary to initialize the Toolkit and activate its commands. You need to do it again only if the PET has been turned off, or reset with a switch, or if a machine language routine has been used to reset the PET software.

SOME GENERAL FEATURES

Immediate Mode Only

Your Toolkit's commands will work as direct statements only. If you include a Toolkit command in a BASIC program, you will get a ?SYNTAX ERROR.

Toolkit Commands Only

You must make a Toolkit command the only item in a direct statement. You will get a ?SYNTAX ERROR if you try to put more than one command on a line, or if you try to mix Toolkit and BASIC statements on the same line.

The Toolkit will either ignore extra characters, or, in most cases, give you a ?SYNTAX ERROR. If you do one thing at a time with your Toolkit, there won't be any trouble.

PET Abbreviations Work

As most of you know, the PET will accept shortened versions of the BASIC keywords if the last character is shifted. For example, L[shift-O] will perform a LOAD, V[shift-E] will do a VERIFY, and so on.

Your Toolkit will accept abbreviated commands as well. Here is a list of Toolkit abbreviations. The underlined letter is shifted:

AU	AUTO
RE	RENUMBER
DE	DELETE
FI	FIND
AP	APPEND
DU	DUMP
HE	HELP
TR	TRACE
ST	STEP
OF	OFF

Other partial abbreviations will work -- for example, RE, REN, and RENUM, will all function correctly.

AUTO

When using AUTO, you often want to use the screen editor on previously entered lines and then return to AUTO. AUTO will remember the next line number if the line number of the edited line is less than the next line number that AUTO is to provide.

If a larger line number is edited, the AUTO's line number is changed to reflect this. Here is an example to get you started; there are many combinations -- try several to see how this works.

```
AUTO 100,10
 100 REM LINE ONE
 110 REM LINE TWO
 120
```

Now use the screen editor to change Line 100 to 200 and press RETURN. AUTO now provides the new line number, 210, where 110 used to be.

When leaving AUTO after entering lines in the middle of a program, be aware that pressing RETURN after a line number results in the deletion of a line! Use the DEL key to remove the line number, and then press RETURN on the blank line if you are in danger of losing a line.

AUTO 100,0 and AUTO 0,0 are acceptable to the Toolkit, though rather useless.

If the next line that AUTO is to provide is larger than 63999, you will get an ?OUT OF RANGE ERROR, and the last line you entered will be missing. For example,

```
AUTO 50000,10000
 50000 REM ONE
 60000 REM TWO
?OUT OF RANGE ERROR
READY.
LIST

 50000 REM ONE
READY.
```

RENUMBER

RENUMBER always renumbers the entire program in memory. If a line number is unreferenced, e.g., there is a GOTO xxx and the program does not have line xxx, the line number is changed to 63999. Use FIND 63999 to locate these references to missing lines.

If the largest line number after RENUMBER is to be more than 63999, RENUMBER gives an ?OUT OF RANGE ERROR and won't renumber the program.

RENUMBER 200,0 and RENUMBER 0,0 will work -- and leave you with a mess!

When RENUMBER converts small line numbers to large ones, and vice versa (e.g., 25 to 1000), the program has to be moved in the PET memory and the line linkage pointers changed. A

large program that is drastically renumbered will take some time to do. Programs that are extremely long (i.e., under 100 bytes free) may get an ?OUT OF MEMORY ERROR and not be renumbered correctly. One cure is to start with a fresh copy of the program and RENUMBER 1,1 to keep the line numbers short. But you probably won't have enough space for variables when you try to run the program anyway!

Numbers included in REM and in quoted strings won't be changed by RENUMBER. Be aware of this if you have hidden a statement like

REM GOTO 500 - CONDITIONAL FOR 16K PET

or similar nasty deeds.

Programs that modify themselves by printing statements on the screen and then stuffing the input buffer with RETURNs will probably not work correctly after RENUMBERing.

If you have APPENDed a program with out-of-order line numbers, RENUMBER can be used to make the code accessible by the program. Any jumps (GOTO, IF-THEN, etc.) will now be pointed to incorrect line numbers, so beware!

DELETE

DELETE without any line numbers, or DELETE -, will give a ?SYNTAX ERROR. This is intended to prevent loss of your program through an accidental DELETE. (To remove a program, use the NEW command.)

If DELETE is given out-of-order line numbers, it will give a ?SYNTAX ERROR if a line exists in the given (but backwards) range. If there are no lines, DELETE does nothing.

FIND

The line number range for FIND operates like LIST. FIND without a line number range looks at the entire program. If line numbers are given out-of-order, FIND behaves like DELETE.

If the search item is not in quote marks, it is tokenized, and the search proceeds through the unquoted parts of the BASIC program only. If the search item is surrounded by quotes, only the quoted parts of the BASIC program are searched.

Remember that FIND without a quoted string looks for matching text that has been broken into tokens. Since all of the BASIC program text has been tokenized, FIND PRINT will look for and find all PRINT statements. However, REM statements are not tokenized; PRINT inside a REM is stored as a sequence of five characters instead of a single token. The result is that FIND PRINT will not find a statement like REM PRINT RESULTS, because it is looking only for the token PRINT. One way out of this difficulty is to search only for partial keywords in REM statements; as in: FIND RINT.

FIND FOO "BAZ" will tokenize FOO and ignore "BAZ". A similar fate awaits FIND "FOO" BAZ, with the search being for "FOO". FIND searches only the part of the line after the line number, and will not find the line number itself.

APPEND

APPEND works for cassettes #1 and #2 only. APPEND will not recognize IEEE device numbers, and will not work for disk.

If the program to be appended would exceed the available memory, APPEND will abort. The program size is on the tape header, so partially appended programs will not result.

Tapes are searched for program names in the same way that LOAD does.

DUMP

DUMP will show the variables whenever BASIC still has them. Note that editing any line in the program causes all the variables to be discarded.

HELP

HELP is very transient, and must be the first command after stopping a program, or nothing will be displayed.

IF YOU DISCOVER A NEW GOTCHA!

PAICs would like to know about any bugs, etc. concerning the Programmer's Toolkit. Please write us with the required details to duplicate the bug. However, please don't phone. Though phone conversations are nice for the psyche, they take time away from making new and neater products, and, phone calls tend to be forgotten when it is time to fix or improve a product. So, please write instead.

The reverse field marker is often one character before the offending part.

When a program is stopped by STOP, or ends by itself, HELP will display the last executed line. The reverse field marker will be at the end of the last completed statement.

The reverse field marker will reverse the entire token in HELP.

If the first character in a line is in error, HELP will not display a reverse field marker.

In some cases, the source of the error might not be where the marker is; the marker is what the PET was looking at when the error was discovered. This is especially true in arithmetic expressions and with READ or INPUT or GET.

TRACE

If the cursor is in the upper six lines of the screen after TRACE has been used, any direct command will include the characters in the "window" when you press RETURN -- which won't work very well. Either clear the screen, or move the cursor below the window on the screen, and then enter your commands. Shift-RETURN will move the cursor down nicely.

STEP

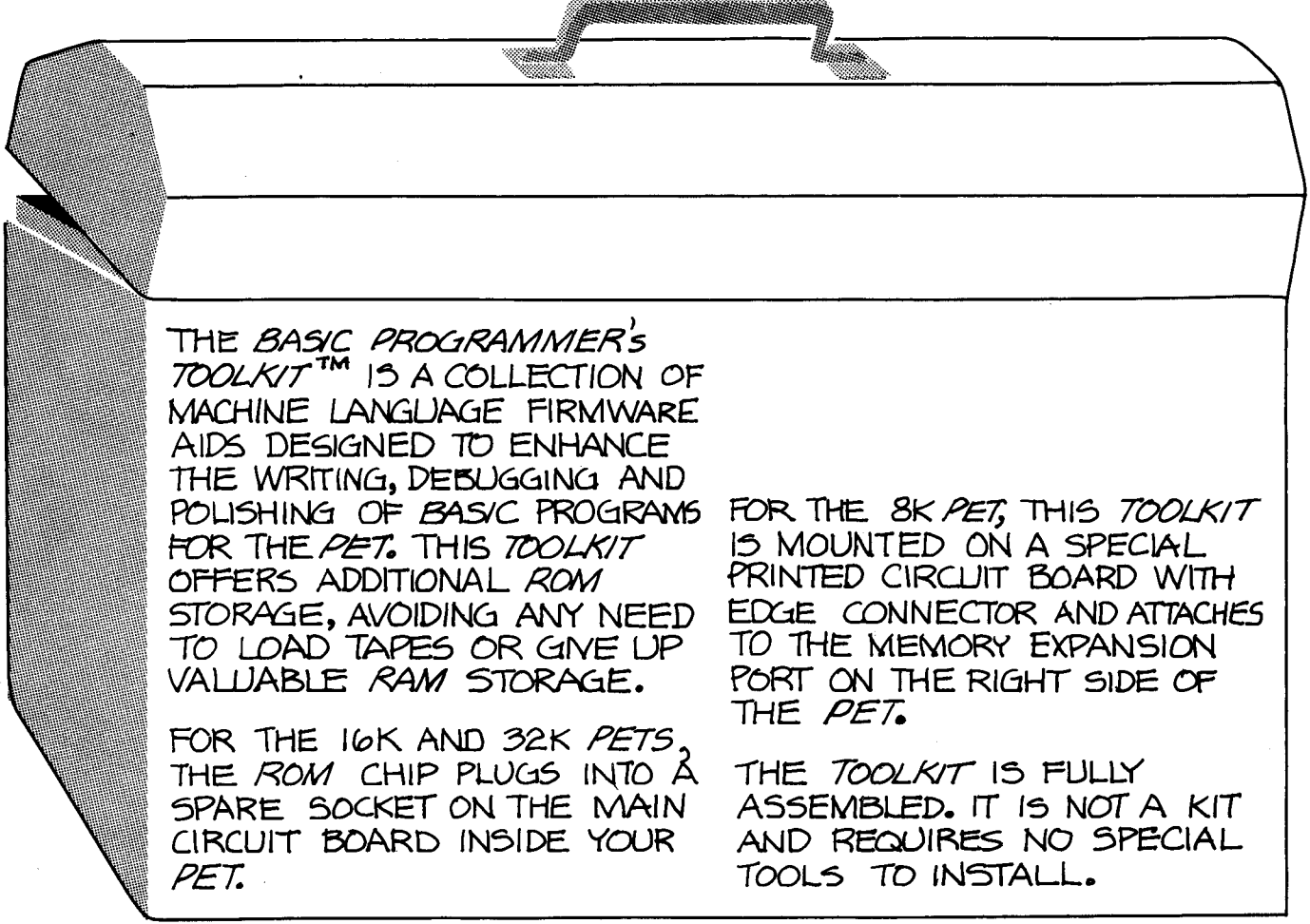
Same as TRACE.

OFF

No known GOTCHAS!

CAUTION FOR HACKERS

The Toolkit requires that the PET BASIC program be intact and correct regarding pointers, line numbers, etc. If you have mangled a program, the Toolkit might go bonkers.



THE *BASIC PROGRAMMER'S TOOLKIT*™ IS A COLLECTION OF MACHINE LANGUAGE FIRMWARE AIDS DESIGNED TO ENHANCE THE WRITING, DEBUGGING AND POLISHING OF *BASIC* PROGRAMS FOR THE *PET*. THIS *TOOLKIT* OFFERS ADDITIONAL *ROM* STORAGE, AVOIDING ANY NEED TO LOAD TAPES OR GIVE UP VALUABLE *RAM* STORAGE.

FOR THE 16K AND 32K *PETS*, THE *ROM* CHIP PLUGS INTO A SPARE SOCKET ON THE MAIN CIRCUIT BOARD INSIDE YOUR *PET*.

FOR THE 8K *PET*, THIS *TOOLKIT* IS MOUNTED ON A SPECIAL PRINTED CIRCUIT BOARD WITH EDGE CONNECTOR AND ATTACHES TO THE MEMORY EXPANSION PORT ON THE RIGHT SIDE OF THE *PET*.

THE *TOOLKIT* IS FULLY ASSEMBLED. IT IS NOT A KIT AND REQUIRES NO SPECIAL TOOLS TO INSTALL.