

HANDY INFORMATION + PEEKS AND POKES

by Guy Cousineau

Following are some handy PEEKs and POKEs which can help you get around some of SMARTBASIC's limitations. They have been organized by category to simplify your search for the correct one. Before dealing with specifics, we should dispense a few definitions.

When dealing with colours, we often refer to the absolute colour table which goes like this:

0	transparent	8	med red
1	black	9	light red
2	med green	10	dark yellow
3	light green	11	light yellow
4	dark blue	12	dark green
5	light blue	13	magenta
6	dark red	14	gray
7	cyan	15	white

Note that colour 0 is often referred to as black but it is not; it lets the border colour, (or backdrop), shine through. When dealing with the default SMARTBASIC screen colour, it gives the appearance of black since the border is black. See the notes on TEXT colours below and try changing only the border colour. You will see that it changes the entire screen colour since characters have a transparent "off" colour.

Several of the memory addresses illustrated show the direct result of a command like COLOR or SCALE. They have been included since they offer you a choice of how you want to change or use a value. For example, you can't ask "what is the default colour value?". Furthermore, ML programmers may make use of some of these addresses since it is hard to CALL the COLOR command from a machine language routine.

TEXT

17059	border colour	0-15
17115	normal character colour	16*set+clear
17126	inverse character colour	16*set+clear
158	FLASH speed	
17291	speed of cursor flash	
	(poke 17000,1 disables cursor flash)	
16129	SPRBD value	
1145	number of prompts (2 max)	
1146	first prompt character	
1147	second prompt character	



16953	ASCII of cursor
16995	ASCII of character under cursor
16054	ASCII of blank space
17001	current vertical cursor position
17002	current horizontal cursor position

All character colours are specified with the set(on) colour and the clear(off) colour. Thus to get white characters on clear background you use 16*15 + 0 (240 decimal). For colour changes to take effect, you must issue a TEXT command.

GR

18607	border colour	0-15
18633	default colour of drawing plane	17*colour
18711	character colour (as text)	
16776	current draw(pen) colour	0-15

HGR and HGR2

25431	border colour	0-15
25471	default colour of drawing plane	17*colour
25568	character colour (HGR only)	
16777	current HPLLOT (pen) colour	
16763	current x-axis position	
16764	current y-axis position	
16765	current SCALE value	

You can use the current x-y coordinates to check where a DRAW command has left the pen. It can be moved directly by POKING rather than a DRAW AT x,y command. I have also discovered a bug in HPLLOT which prevents plotting on row 159.

This can be fixed with POKE 25940,160.

MARGINS

	TEXT	SCROLL
LINES	17198	16993
COLUMNS	17199	16994
TOP MARGIN	17201	16958
LEFT MARGIN	17202	16956
RIGHT MARGIN		16957
BOTTOM MARGIN		16959
START SCROLL		16995
LEFT SCROLL		16996

The values in the TEXT column reflect the implemented defaults when you issue a TEXT command. The scroll and bottom margins are calculated by the TEXT command. The values in the scroll column can be user modified at any time; their effect is instantaneous and temporary (until the next TEXT command). While scrolling windows are in operation, it is still possible to use VTAB HTAB to place the cursor outside the window. You can create good special effect with this technique. The scroll values work in TEXT GR and HGR. Remember that the total of 16958 and 16993 must always be 24. Unless you want to create a special effect, the START SCROLL should be the same as TOP MARGIN and LEFT SCROLL same as LEFT MARGIN.

SCREEN FORMATTING

The LIST command sometimes adds annoying spaces which are supposed to enhance the readability of programs. You can remove these by POKEing ZER0es into the following addresses:

13357 after semi-colon
13349 after a comma
13423 after COMMAND word
16148 before a line number in LIST

If you have a 40-column patch, you may have noticed that HTAB is still restricted to column 32. A partial fix for this involves resetting the HTAB margin to 64 with:

POKE 26198,63

Unfortunately, this will allow illegal HTAB values; use some restraint here. Furthermore, the HTAB routine will perform a screen wrap whenever the value is above 31. It is necessary to re-specify VTAB AFTER HTAB for this patch to work correctly.

When printing tabular data, you can use the COMMA to space out to 1/2 screen or 16 spaces. You can change the comma spacing to 8 with 2 POKES:

POKE 7879,15:POKE 7881,16

If you print a TAB character (CHR\$(9)), BASIC moves out to the next multiple of 4. You can change this to 8 with:

POKE 12329,7:POKE 12333,8

Both these format changes can really improve tabular printing, especially if you use the 40 column screen.

FILES ETC

Do you want a MERGE command? Follow these steps:

LOAD program 1
POKE 6356,201
LOAD program 2
POKE 6356,205

Want to be able to recover 'h' files? Simply:

POKE 20619,72.

You can modify the output of the CATALOG command with the following:

POKE 21298,0 shows usable blocks left, instead of unused
POKE 21370,8 shows actual size of file, rather than reserved size
POKE 21370,2 shows start block of file rather than length
POKE 21405,? where "?" is the ASCII value for the LOCKED FILE indicator

To correctly INIT a tape or disk with any size directory, simply POKE the desired values into these RAM locations prior to issuing the INIT command:

25308	contains the number of directory blocks
25305-25306	contains the size of medium
use	0,1 for tape
use	160,0 for SS disk
use	64,1 for DS disk
use	208,2 for 720K 3 1/2 inch disk

When saving a file to disk/tape, BASIC removes extra spaces and changes PRINT to "?". While this conserves storage space, it makes for an unpleasant display to some programmers if they subsequently use SMARTWRITER to print the file. This can be overridden with the following POKES made just prior to saving the file:

POKE 24100,0:POKE 24101,0:POKE 24102,0

After the file is saved, you should reset with

POKE 24100,50:POKE 24101,20:POKE 24102,63

OTHER PROGRAM CONTROLS AND INFORMATION

RAM LOCATIONS	WHAT YOU FIND THERE
260	revision # of BASIC, (260 is most current)
12185	Max length of program line (max 239 for immediate mode math calculations, or possibly as high as 255 otherwise)
12374	ASCII value of INSERT (^N)
12375	ASCII value of DELETE (^O)

16091-92 number of program lines

16095-96 current LONEM

16126-27 line number for ONERR

18307 ASCII of CLEAR (^L)

16134 ASCII value of break (^C)

16135 ASCII value of pause (^S)

16136 PAUSE flag (set zero to force pause)

16148 ASCII used to indent line

16149-50 POKE limit; put 255,255

16178 number of digits on float output

16641 current drive (1, 2 5, 6)

16821 current device (8, 24, 4, 5)

17008 current graphics mode

0=text 1=GR 2=HGR 3=HGR

17302 (and 18320, must change both), ASCII of "PRINT
SCREEN", (^P)

17950 bell frequency (1)

17954 bell frequency (2)

17958 volume of bell

17963 duration of bell

Consult an ASCII table for control and regular values:

KEY	UNSHIFT	SHIFT	CONTROL
BACKSPACE	8	184	8
TAB	9	185	9
2	50	64	0
[91	123	27
\	92	63	28
]	93	125	29
^	94	126	30
6	54	95	31
HONE	128	128	128
I	129	137	129
II	130	138	130
III	131	139	131
IV	132	140	132
V	133	141	133
VI	134	142	134
WILD	144	152	144
UNDO	145	153	145
MOVE/COPY	146	154	146
STORE/GET	147	154	146
INSERT	148	156	148
PRINT	149	157	149
CLEAR	150	158	150
DELETE	151	159	127*
UP	160	160	164
RIGHT	161	161	165
DOWN	162	162	166
LEFT	163	163	167
UP/RIGHT	168		
RIGHT/DOWN	169		
DOWN/LEFT	170		
LEFT/UP	171		
HONE/UP	172		
HONE/RIGHT	173		
HONE/DOWN	174		
HONE/LEFT	175		

PRINTER

You can underline characters by sending <character><backspace><underline> but BASIC counts this as 3 characters and thinks you have reached the right margin before you actually have. The trick is to tell BASIC that the printer has a wider margin with POKE 16176,255. Remember to reset the margin when you are done with POKE 16176,80.

You can reset the printer with CALL 64662. This clears any outstanding characters and resets the print head to column 1.

Sometimes you may wish to turn off the screen echo while sending data to the printer. This is done with:

POKE 12043,201

After printing, turn screen echo back on with:

POKE 12043,245

KEYBOARD

Have you ever run a MENU DRIVEN program which does not work because you accidentally hit the LOCK key? Prevent this from happening by unlocking the keyboard from your program:

p=PEEK(65220):POKE 65220,2:POKE 65220,p.

The keyboard can also be reset with CALL 64659.

Following is a list of keyboard codes for the special keys.

HANDY CALLS

Following is a description of handy SmartBASIC system calls. These may be particularly useful for machine language programs which want to make use of some of SMARTBASIC's routines to reduce the size of an ML utility.

26163 FLASH on
26141 INVERSE on

26151 NORMAL on
11090 Clear screen and home cursor

18112 SCROLL SCREEN. Leaves cursor in present position but scrolls screen up one line.

64572 WAIT 33 microseconds
This is part of the cold boot routine which waits for the system to reset. The wait value is just barely a "blink", but can be changed to a full 1/2 second with:

POKE 63844,255:POKE 63845,255

You can further increase to several 1/2 seconds with

POKE 63842,x

where x is the number of 1/2 seconds desired.

17289 GET CHARACTER
Flashes cursor and waits for a keypress; character is returned in the accumulator

11865 CHECK FOR LETTER
This routine will check for a-z and A-Z without doing case conversion. The carry flag is set if character in accumulator is OK.

11885 CHECK FOR NUMBER
Checks that value in accumulator is in the range of ASCII 0 to ASCII 9. If the carry flag is set on return, simply subtract ASCII 0 to convert to an absolute number.

12110 PRINT MESSAGE
Prints length-encoded message pointed to by HL.

11194 PRINT CHARACTER
Prints character in accumulator to the current output device interpreting control codes like BELL or BACKSPACE.

12128 PRINT <CR>
Sends <CR><LF> to current output device.

12420 PRINT TO SCREEN
Prints character to screen even if FILE WRITE is active. This can be handy to echo messages to screen during I/O.

7066 HL*DE
Multiplies 2 integer values. The result is returned in HL and the carry flag set if there is overflow.

54272 LAST CATALOG READ
PEEK into this area to view the complete directory from block 1, and uncover information that the CATALOG program did not report. This presumes you are familiar with the directory

structure.

55296 LAST BLOCK READ/WRTTEN
The last 1024 bytes read or written to tape/disk will be stored here. You can PEEK into here to analyze what was read.

63599 WARM BOOT
This will perform the same function as pulling the reset switch. If no disk or tape is present, it will fall into the exit to SmartWRITER.

63606 BOOT DISK ONE
This alternate boot will wait for a disk to be inserted in disk drive 1 and proceed to boot it.

64148 EXIT TO SMARTWRITER
Although several addresses have been quoted, this is the correct one. ("CALL 64743" will send the Z80 microprocessor to the SmartWRITER jump table address, from which it will Jump to 64148).

64809 SPRITES OFF
This handy routine will clear any sprites which may be left on the screen if a program using them exits without clearing them.

64851 SOUND OFF
This one is very handy if a program crashes while a sound is in progress.

64885 LAST KEY PRESSED
This is not actually a routine, but is in fact the location where the Master 6801 saves the keypress in RAM. The Z80 picks it up from this location when the programmer desires to read it. It is useful for reading the keyboard "on the fly" in Basic, so that the user doesn't have to wait until some operation is complete before he is permitted to make a keypress. Or you can check this value until it changes, (as Basic does with the GET command); or wait until a desired value appears.

BOS SYSTEM CALLS

For you machine language enthusiasts, the following is a summary of the most useful operating system calls along with the values that must be in the indicated Z80 registers when the CALL is made. These routines can be used to set up your own machine language utilities which will run independently from SMARTBASIC.

Many of these routines and a great deal of information about the Video Data Processor, and the audio chip is discussed more extensively in the "HACKERS HELPER" series of ADAM

books. Contact the author for more details.

64566 INITIALIZE CONSOLE

B= number of columns

C= number of lines

D= home column

E= home row

HL=pointer to pattern table

This routine does the same kind of work that the TEXT command does in BASIC. It initializes a scrolling region and defines the cursor within that region.

64569 PRINT CHARACTER ON SCREEN, called "CONS_OUT"

A =character

DE=Moves cursor to new xy coordinate, (in DE), if A=28. If A= other screen control character, performs the function requested. If A=arrow ASCII, HOME ASCII, then processes cursor accordingly. Otherwise prints character in A on the screen at the present cursor position.

64614 PRINT CHARACTER ON PRINTER

A=character

64620 READ KEYBOARD

This routine needs no setup. It returns the keyboard character (if any) in A.

64755 READ BLOCK

A =device

BC=0

DE=block number

HL=RAM address in which to store that which is read.

This routine will read 1K from the specified device into the buffer pointed to by HL. If the ZERO flag is set on return everything is fine. Otherwise, the error code is returned in A. All other registers are preserved.

64758 WRITE BLOCK

A =device

BC=0

DE=block number

HL=RAM address

Write 1K of data from address in HL to device; similar to above.

64794 WRITE VRAM

BC=number of bytes to write

DE=address in VRAM

HL=source address in RAM

This routine takes care of the complex job of transferring data to Video Memory. You can use it to update characters, graphics, sprites, etc.

64797 READ VRAM

BC=number of bytes to read

DE=address in VRAM

As above reads data from Video Memory. Use it to analyze what is there.

64800 WRITE VDP REGISTER

B=register to write to

C=data to send

Use this routine to set up VDP registers for the various graphics modes and sprite attributes. Following are the VDP register specifics

- 1 bit 7 always 1=large VRAM, as in ADAM
bit 6 0 blanks display, 1 enables
bit 5 1 enables interrupts
bit 4 text mode flag
bit 3 multicolor mode flag
bit 2 always 0
bit 1 small or large sprites
bit 0 double-sized sprite flag
- 2 (pattern name table)/400H
This table is 24 rows by 32 columns or 768 bytes.
- 3 (address of colour table)/40H
32 bytes reflecting the colour of each group of 8 characters. E.G. patterns 0 1 2 3 4 5 6 7 have the same colour.
- 4 (pattern generator table)/800H
This table has 8 bytes for each pattern for characters 0 to 128. Thus 2K is required for the table.
- 5 (sprite attributes)/80H
32 sprites by 4 bytes each requires 128 bytes.
- 6 (sprite pattern)/800H
32 sprites by 8 bytes each requires 256 bytes.
- 7 bits 7,6,5,4 character colour in TEXT mode
bits 3,2,1,0 border/background colour

64812 PUT VRAM

A =table number. (0=Sprite Attribute Table; 1=Sprite Generator, (or Sprite Pattern Table); 2= Pattern Name Table; 3=Pattern Generator Table, (or Pattern Table); 4=Pattern Color Table.

DE=starting index into the table

HL=user buffer

IX=number of entries to write

```

64815 GET VRAM
      A =table number
      DE=starting entry
      NL=user buffer
      IY=number of entries to read

```

```

0  sprite attributes
1  sprite pattern
2  pattern name table
3  pattern generator table
4  colour table

```

The only set up required for this routine is to set the VDP registers via the SET_REGISTER routine (64800), so the routine knows where to put the character definitions. It takes care of finding the character definitions from ROM and installs them in the VDP.

Whether using tapes or disks, the EOS treats the media as a sequential device. This means that all 'files' are written to contiguous blocks. While this approach does not take full advantage of available disk space, it reduces the amount of directory information required.

```

0          1          2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
|   file name      | | | |   |   |   |   |
|   ^ ^ ^ ^   ^   ^   ^   ^   ^
file type  - - / | |   |   |   |   |
end-of-name - - - / |   |   |   |   |
attributes  - - - - /   |   |   |   |
start block - - - - - /   |   |   |   |
reserved size - - - - - - /   |   |   |
used size  - - - - - - - /   |   |   |
used in last block - - - - - - - /   |
date       - - - - - - - - - - - /

```

"VOLUME", (the first), entry, which can have 11 characters. The file name, except the "VOLUME" entry name, is followed with the single letter filetype extension; its maximum position, the 11th, or position 10, is shown above. The file type is followed by a CHR\$(3), (seen as a "3" when loaded to RAM and PEEKed), which is the signal to the EOS that the end-of-name has been reached. The file type extension, or simply "filetype" as discussed in the EOS system, is located just before the end-of-name. All data from the end-of-name to position 11 is ignored and does not need to be blanked out.

BIT SET	Meaning
---------	---------

6--WRITE PROTECT: This bit prevents appending or deleting a file; it cannot be set from BASIC.

5--READ PROTECT: This bit prevents the OPENing, READing, CLOSing, and LOADing, of files from BASIC or SmartWriter. It can be useful for protecting programs or data that will be loaded in via READ BLOCK.

4--USER FILE: This bit must be set if the file is to be shown by the CATALOG command. It can however be overridden by setting bit 3. Regardless, setting the user bit allows normal opening, closing, renaming, etc.

3--SYSTEM FILE: Setting this bit disables the listing of the file by normal DIRECTORY or CATALOG functions. It has no other effect on file operations.

2--DELETED FILE: A file may be un-deleted by resetting this bit. Make sure, however, that the same file name has not been used elsewhere by an active file, or you may confuse the BIOS.

1--EXECUTE PROTECT: This bit prevents UNLOCKing a file but will not prevent LOCKing. Any file LOCKed while this bit is set will not be UNLOCKable using conventional means.

157

directory entries following it. See
BLOCKS LEFT ENTRY for more information.

The START BLOCK bytes identify the start of the file on the media. It uses 4 bytes which are placed in registers BCDE to address, in theory, a drive of over 4000 MB.

The RESERVED SIZE bytes shows the size of the file originally placed in the "HOLE", a term used by the original EOS programmer to define a "planned" region, or number of blocks, in which to save a file which the user wanted to save. A smaller file may reside in the same hole later, but the RESERVED SIZE or "HOLE" will always be the same amount of disk space. A reserved size may be up to 65535 bytes, (64K), since 2 bytes are used in the directory for this data. Note that the sum of the two quantities, (START BLOCK) + (RESERVED SIZE), must equal the START BLOCK value of the next directory entry for proper management.

The USED SIZE bytes show the actual length of the file. When a small file is placed in a big HOLE, these bytes tell the EOS how many K of file to actually load in when getting the file.

The USED IN LAST BLOCK bytes are needed since the EOS does not use an END-OF-FILE marker in ASCII files. When a file is saved, its exact length in full blocks is computed, and the number of the 'remainder bytes' in the last incomplete block is placed in these 2 bytes. The EOS will know, when re-loading the file, exactly how many bytes of the last block to actually read in. All information on the media beyond the "USED IN LAST BLOCK" pointer will be ignored by the EOS read file routines.

The last 3 bytes of a directory entry are reserved for a creation date. There is a date in the EOS which appears to be the birth date of one of the programmers in 3 BCD numbers YY-MM-DD. There is no function in BASIC, SmartWriter, SmartPiler, or any other COLECO software that I know of that makes use of this date. You may reset the system date by poking the date directly into the EOS, (POKE 64992,year:POKE 64993,month:POKE 64994,day), and all files created that day will have that day's date. Furthermore, jumping to SmartWriter with a JP_WP instruction, or "CALL 64743" will also preserve the system date for directory entries made in the SmartWriter program. Turning off the computer will obviously erase the date entry, since it is stored by the POKES in dynamic RAM.

There are four special entries in the EOS directory which look like just like file entries but are used by the EOS to work with the directory and MEDIUM.

VOLUME ENTRY

The Volume entry is set by default to FIRST DIR. This is the name that will be reported by BASIC when a CATALOG command is issued. It is also the name passed by the INIT function in BASIC. Since the volume name does not need a file type, this entry may be up to 11 characters. The attribute byte consists of 80H + the number of media blocks reserved for the DIRECTORY (see DIRECTORY). The Next 4 bytes are 55H AAH 00H FFH; these are simply a series of check bytes for the EOS routines to use to see if a media contains a valid EOS directory, (as opposed to a CPM directory, for example, which EOS routines are not designed to interpret). Bytes 17 and 18 reflect the size of the medium, 255 for tapes, 160/320/720 for disks. The other bytes are non-significant and usually zero except for the date which may or may not be filled in as explained above.

BOOT ENTRY

This entry tells the directory how many K have been reserved for a BOOT block. It points by default to Block 0 with a length of 1K and 1024 bytes used. The exact use of this entry is unclear since all EOS media have this entry filled out in the same way. SMARTBASIC and other systems reserve another entry in the directory for a BASICPGM file which is read and loaded according to Machine Language instructions placed at RAM 51200 from the BOOT block. The BOOT entry should not be modified until further study of it is made, although at present, it seems to fill no purpose.

DIRECTORY ENTRY

This entry tells the EOS the size of the directory by the 'size used' information. All other data in this entry appears to be insignificant but should not be changed, just in case. Note that the VOLUME entry also has a directory size, but it only indicates the maximum, or reserved directory size. The DIRECTORY entry reveals how much has been used so that when EOS is ordered to search the directory for a file, for example, it doesn't search "reserved" directory blocks that haven't been "used" yet. To change a directory from 1 to 2K with a utility other than the INIT command with the appropriate POKES in Basic, mentioned above, both of these entries must be modified.

BLOCKS LEFT ENTRY

The Blocks Left entry indicates the number of blocks available from the last file listed in the directory to the end of the disk or tape. Although it is handled slightly differently by BASIC and BASIC II, it only requires 3

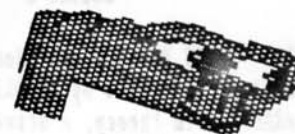
parameters:

- 1 Setting the SYSTEM FILE and BLOCKS LEFT bit in the attributes byte means that this is the "Blocks Left" entry, regardless of the "BLOCKS LEFT" file name. The ROS loaded with BASIC II does not even bother writing "BLOCKS LEFT" in the filename location of the BLOCKS LEFT entry. Neither the standard ROS nor the version used with SmartBASIC V2 looks at the name for any function, but both simply test the attribute byte for bit 0 set.
- 2 Setting the start block as for any other entry as the sum of previous-file-start and previous-file-size. In other words, if a new file is added it will assume this "start block" value, and this value will be changed in preparation for the adding of yet another new file.

Setting the reserved-size by subtracting the start block from DISK SIZE. If Blocks Left starts at 100, the reserved size should be set to 60 for a single sided disk (not 59).

This information has been gathered through my own investigations and from numerous other sources of ADAM information too numerous to enumerate.

Guy Cousineau



ANN

ADAM