SmartBASIC 2.0 "Review"

by George A. Havach

NOTES ON SMARTBASIC 2.0

Though never finally released by Coleco (who terminated the project shortly before discontinuing further Adam production and development), a revised, still incomplete (no enhanced sound or graphics!!) version of SmartBASIC dubbed 2.0 has found its way into the "Adam underground" in what amounts to a beta-testing form. Having obtained a working copy of my own, I've spent some hours checking it out and comparing its features with those of the V1.0 we've all come to know and love/hate. The following somewhat scattered notes summarize my discoveries to date.

(1) SmartBASIC II occupies 49 blocks on the disk, in contrast to SB I's mere 28K. The file size is so unexpectedly large because SB II COMES WITH ITS OWN OPERATING SYSTEM ON BOARD--true Revision 7 of EOS (different from the ROM-chip version built into the Adam)--which is overlaid onto the RAM-resident version at startup, beginning at address E4B8H on up. According to Joel K. Lagerquist, author of the revisions to SmartBASIC that were incorporated into V2.0, this revision comes equipped with all the fixes to EOS' file-handling functions that enable SB II to work "much faster" with data files opened by SmartBASIC programs. Thus, SB II is apparently capable of correctly opening and read/writing random-access files, whereas SB I seems full of bugs in this department, in spite of what the Coleco manual (pp. C9-C10) would lead you to believe. Furthermore, SB II takes up almost 500 bytes LESS runtime memory than SB I: a FREe space of 26401 [after NEW] versus 25954 bytes.

(2) The cold-start loader ('BOOT' block 0) is all new code and includes a RAM test for the presence of the 64K Memory Expander. Also, SB II will boot from ANY drive and set that as the active one, whereas SB I defaults to data-pack drive 1 unless patched otherwise (byte 2 of block 18: 4 = disk drive 1, 8 = data-pack drive 1).

(3) The default value of HIMEM (53632) is identical in the two versions, but LOMEM is lower in SB II: 26960 versus 27407 (the same difference as for FRE(0)). Also, the system-parameter area (see SmartBASIC Guide, p. C-23) has been moved: for example, the "LOMEM setting" is at 16095 in SB I, but at 1594 in SB II. More interesting, LOMEM and HIMEM can now be simultaneously reset; in SB I, you can change EITHER one, but then trying to change the other gives you an "Out of Memory Error."

(4) Both the REM and DATA statements have been fixed for the bug in SB I that keeps inserting a space after them on a program line each time the program is SAVEd or LOADed. In fact, SB II specifically strips out any extra leading spaces when LISTing. This single feature might make SB II worth having, just to be able to save programs within a minimum of file space.

(5) The cursor-display routine has been rewritten in SB II, so now the underline cursor

character DISAPPEARS while either executing an immediate command or running a program, reappearing only on return to command mode. This makes for a "cleaner" screen display, but it's no longer possible to change or blank the cursor character with a single POKE (to 16953 in SB I).

(6) Return of any error message by SB II is accompanied by a "beep" (CHR$(7)). Furthermore, a touch of humor--and genius--has been added to the way numbered program lines are accepted (or not) by the interpreter. Previously, any line containing a syntax ("Illegal Command") error was simply rejected by SB I. Now, the offending line will still be normally redisplayed with a caret pointing to where the error occurred, but in addition, SB II prints a "happy face" (CHR$(2)) as the first character after the line number. And if you LIST the program, any erroneous lines are RETAINED IN THE PROGRAM, but their presence is unmistakably signaled by a "beep" during the listing! This means a potentially great savings of time and retyping while programming: buggy lines now can be corrected AT ANY TIME DURING PROGRAM DEVELOPMENT, simply and easily by running the cursor over the line (skipping over the "happy face" with a CONTROL-right arrow), making the needed changes, and then hitting RETURN to reenter the line and cancel the incorrect version. The program, of course, will not run properly until all these troublesome lines are fixed.

(7) Four of the special-function keys ("hardkeys") are software-labeled to function for line editing in SB II: CLEAR works like CONTROL-X (cancels a line input), INSERT like CONTROL-N (opens up a space leftward of the cursor), DELETE like CONTROL-O (closes up the space beneath the cursor), and PRINT like CONTROL-P (dumps the screen to the printer). The screen dump, however, is now INTERRUPTABLE WITH A CONTROL-C, and if the CONTROL-C is issued during a screen dump activated from within a program (e.g., by PRINT CHR$(16)), control passes to the next step without a "Break."

(8) Line length (actually, the size of the line buffer) in SB II is expanded to 255 characters, versus SB I's 127.

(9) Use is made of an RST 38 interrupt vector that involves updating the spinner (the spool on the Super Action Controller), evidently with game design in mind.

(10) The BREAK and NOBREAK commands (tokens 64 and 65) have been deleted from SB II, but STORE, RECALL, and SHLOAD have not, and they're just as NON-FUNCTIONAL as in SB I!

(11) The new MERGE command works beautifully, in precisely the same way as in Microsoft BASIC: "Format: MERGE <filename>[,Dn]. Merges a specified disk file into the program currently in memory....If any lines in the disk file have the same line numbers as lines in the program in memory, the lines from the file on disk will replace the corresponding lines in memory. (MERGEing may be thought of as 'inserting' the program lines on disk into the program in memory.)" This single addition greatly facilitates "modular" programming, and makes SB II particularly valuable for larger

writing projects.

(12) The INIT command in SB II assigns the correct number of "Blocks Free" when used to blank out the directory and reinitialize a disk, whereas SB I treats a disk the same as a data pack, and so 'CATALOG' shows an impossible "253 Blocks Free"!

(13) While SB I sets an upper limit to use of the POKE command of 54160 (D390H, the start of the FCB-header area), SB II raises this limit nearly to the top of RAM, to 65216 (FEC0H, the beginning of the DCB/FCB area). You can override this limit by POKEing 255 to addresses 16149 and 16150 in SB I, or to 1648 and 1649 in SB II; then you can POKE anywhere in memory--at your own risk!

(14) In general, special POKEs into the BASIC interpreter of SB I--and programs that use them--will NOT work in SB II, since it has been completely recompiled. In many cases, however, corresponding locations can be found with a little investigation. Among the more useful are the POKEs to change the screen and character colors (SB I addresses in parentheses):
17184 (17059) border color (0-15)
17240 (17115) normal-character (high nibble) and display-screen (low nibble) colors
17251 (17126) inverse-character (high nibble) and block-background (low nibble) colors

(15) The two new commands STDMEM and EXTMEM have been added to SB II to permit switching between standard memory (64K RAM plus 16K VRAM; also the default configuration) and extended memory, which uses the 64K Memory Expander (if present) in much the same manner as SmartWriter uses this extra RAM to expand its workspace, i.e., through a bank-switching arrangement. To load the bank-switched configuration (which takes up much of that extra space in the SB II disk file), enter "EXTMEM" WITH THE SOURCE DISK STILL IN THE DRIVE. The disk will spin, the screen will blank, and after a few seconds the title screen will reappear. To get some idea of the size of your new workspace, enter "PRINT FRE(0)"; the value 90646 is displayed [90656 after "NEW"]. Compare this with the value of 26391 [26401 after "NEW"] in SB II's standard memory configuration. To reload the standard configuration (and clear the workspace), enter "STDMEM", again with the source disk in the drive; the screen will blank, the disk will spin, and the title screen will reappear. The MEMORY MAP seems to be quite different in these two configurations; things really get moved around, so watch out! For example, LOMEM and HIMEM don't seem to have the same error checking in EXTMEM, and so entering an illegal value can cause an immediate system crash! Also, programs will generally RUN MORE SLOWLY (about half as fast) in EXTMEM than in STDMEM, because of all the extra CPU time spent in bank switching. That's the way it goes!

(16) High-resolution graphics have supposedly been fixed in SB II to eliminate the problem of "video bleed" when HPLOTing or DRAWing in adjacent pixels, but the few simple tests I've run indicate that the color-bleed problem is AS BAD AS, IF NOT

WORSE THAN, IN SB I. Nonetheless, THE BIG NEWS IS THAT SB II IS CAPABLE OF HANDLING SPRITES! It does this through the "DRAW" and "XDRAW" commands--and these work IN ALL THREE SCREEN-DISPLAY MODES: TEXT, GR, and HGR/HGR2. To activate this function, first POKE a nonzero value into address 16788 (the shape/sprite flag; default value is 0); addresses 16786 and 16787 hold the pointer to the starting address of the sprite table, low byte first (the default table is at 192 decimal). A sprite table can be located anywhere in memory you choose, with its address POKEd into these two bytes. There are TWO default sprites already provided; you can view them by the following interaction:

```
POKE 16788, 1: HOME
HCOLOR = 3: DRAW 1 AT 100, 50: DRAW 2 AT 100, 100
```

This will display sprite 1 as a sailboat and sprite 2 as a communications satellite! The format for creating your own sprites is EXACTLY THE SAME AS FOR SHAPES IN SMARTLOGO, as described in the SmartLogo Reference Manual (pp. 72-74); each sprite is a 32-byte string of hexcodes. A maximum of 32 sprites is allowable (the limit for simultaneous display by the video chip), numbered 1 through 32.

(17) SB II has a built-in function for displaying ANY character on the screen at the current cursor position (in TEXT mode), even the control-character symbols below ASCII 32. To utilize this function, POKE the ASCII code into address 16771, then CALL 16770. This function interprets the code as a displayable character, NOT a control code, so you can print even the characters equivalent to a linefeed (10, a Greek "pi") or carriage return (13, a left-pointing arrowhead). Try this short program:

```
10 HOME: FOR char = 1 to 127: POKE 16771, char: CALL 16770: PRINT " "; :
NEXT
```

which will show a staggered screen display of the entire Adam character set, from ASCII 1 (a centered dot) through 127 (a flashing cursor).