COLECO ADAM

SmartBasic 1.x

by Richard F. Drushel

Third Printing: February 8th, 1992

First Printing: August 1st, 1991
Second Printing: October 17th, 1991
Third Printing: February 8th, 1992

Facsimile PDF created from the original WordPerfect 5.1 DOS file
May 17th, 2010
by Richard F. Drushel

SmartBASIC 1.x User Manual

TABLE OF CONTENTS

WARRANTY.

The medium upon which SmartBASIC 1.x is supplied (either disk or digital data pack) is warranted to be free from defects at the time of sale, and for a period of ninety (90) days thereafter. During the warranty period, defective media will be replaced free of charge to registered users. I reserve the right to require that the ORIGINAL defective disk or tape be returned as a condition of the warranty replacement. *REPLACEMENT OF DEFECTIVE MEDIA IS THE SOLE RECOURSE UNDER THIS WARRANTY*.

Users must complete, cut out and mail the registration form below. This registration entitles the user to upgrades of the SmartBASIC 1.x software at reduced cost. User interest, availability of new hardware specifications, and bug reports will determine the form of any upgrades or changes to either the SmartBASIC 1.x software or the manual.

I have tried my best to insure that the information in the manual is correct. However, I cannot assume any responsibility for damage caused by the use of either the SmartBASIC 1.x software or the manual, including (but not limited to) lost files, lost data, damaged disks or data packs, and any financial loss caused by the same. The user must assume all risks associated with the use of SmartBASIC 1.x.

****

SmartBASIC 1.x REGISTRATION INFORMATION.

Serial Number_________________________Version Number_________________________

Date of Purchase____

Purchased From____

cut below this line and mail

****

SmartBASIC 1.x REGISTRATION FORM.

Serial Number_________________________Version Number_________________________

Name____

Address____

City, State, ZIP____

Phone (optional)____

Date of Purchase____

Purchased From____

Warranty returns and bug reports should be sent to:

Richard F. Drushel
3353 Mayfield Road
Cleveland Heights, Ohio 44118
(216) 397-0684

If you call, *PLEASE* make it between 8:00 P.M. and 12:00 midnight Eastern time! And please, I am unable to accept collect calls.

Since I am currently in the final stages of my Ph.D. research, the probability is high that within the next 18 months I will be moving--where I do not know. It is fortunate, therefore, that I am active on several computer bulletin boards. I will respond to E-mail at the following:

The Trading Post (A-NET)	(216) 791-4022	TP39 or RICH DRUSHEL
The Cleveland ADAM eXchange	(216) 883-9355	36
The Connection BBS (A-NET)	(518) 298-4294	TC30 or RICH DRUSHEL
ADAMlink of Utah (A-NET)	(801) 484-5114	AN62 or RICH DRUSHEL
The Cleveland FreeNet	(216) 368-8888	rfd
The Youngstown FreeNet	(216) 742-3072	ad194
Micro Innovations BBS	(703) 264-3908	75

I can also receive Internet and Bitnet E-mail from anywhere in the world at my Cleveland FreeNet addresses:

Internet:	rfd@po.CWRU.edu
Bitnet:	rfd%po.CWRU.edu@cunyvm

FORWARD.

On 25 December 1984, a Coleco ADAM computer appeared underneath my father's Christmas tree. ADAM was his first computer; he was a complete novice to word processors and BASIC programming languages (and to some extent video games). As a veteran BASIC programmer (MUBASIC for Digital's PDP-11/34 minicomputer, and MicroSoft BASIC for the IBM-PC), I was naturally curious about this SmartBASIC, which seemed to be a clone of AppleSoft for the Apple IIc/e. Going through the "SmartBASIC Programming Manual", we found a few discrepancies...FLASH didn't work...something seemed to be wrong with file I/O...column 1 didn't show on the TV screen...the HELLO program didn't run when we booted the system... Well, it was a hectic day; maybe with some quiet time to play with it Dad would figure it out...

After Coleco abandoned the ADAM, Dad joined NIAD. Every month there were lots of SmartBASIC programs. Lots of them had PEEKs and POKEs and CALLs...hmm, the manual didn't say much about these commands; well, try it and see what happens... Most of the programs bombed spectacularly. On my visits home from school, I spent long hours in front of the ADAM, trying to figure out why things didn't work they way they were supposed to. Reading a few NIAD issues, I saw the discussion about R59 versus R80 SmartWriters, about the later, "improved" version of SmartBASIC 1.0...^R revealed that Dad's ADAM was an R59, and that he had the original version of SmartBASIC 1.0. The memory map was different, so no wonder all those PEEKs, POKEs and CALLs didn't work!

By 1987, Dad was fed up with his ADAM. SmartFiler was useful, but SmartBASIC was still a mess, SmartWriter kept locking up, and it took too long to print out anything on that noisy printer. So...he junked it and bought a Tandy 1000TX. Rather than see his ADAM go into a dumpster, I took it. "Maybe I can get it to work," I said. And thereby hangs this tale...

After getting a revised SmartBASIC 1.0 and a new R80 system unit, I could program the ADAM after a fashion. But I kept trashing directories, and had only clumsy programs for editing blocks; *EVERYTHING* I wanted to do either had never been written or was some machine code routine from another program (that was invariably on the disk or tape that was trashed). I bought those two indispensable reference guides, "The Hacker's Guide to ADAM Volume 1" and "Volume 2" by Peter and Ben Hinkle, and slowly some of the fog began to lift. I wrote a disassembly program and started to dig into EOS-5. (This task was greatly aided by an IBM-based program I wrote to read ADAM disks and save the data in IBM format; 640K of RAM, 20MB hard disks and DOS editors were the only way to do it.) I commented the *ENTIRE* disassembly listing for EOS-5 (as DOS text), found a few bugs in the Hinkle descriptions, and was just beginning to think about how to publish it, when we moved. Once at our new house, the ADAMs never got unpacked, and they sat idle in the attic--I was too busy with my Ph.D. research in biology.

In January, 1990, I dug out my ADAMs again, and began to work on two vexing problems in Ben Hinkle's 40-column TEXT patch: the eventual scrambled fonts, and the ^P lockup. I fixed them, then thought, "Why not

FLASH and INVERSE, too?" So I did it. And then I allowed the control characters to echo on the screen. By this time I had produced a complete disassembly listing of SmartBASIC 1.0 and SmartBASIC 2.0, and was comparing the source code with Ben Hinkle's descriptions in "The Hacker's Guide to ADAM Volume 2." I began to consider adding TEXT40 to SmartBASIC as an option, instead of a substitution for 31-column TEXT. The idea of using the STORE, RECALL, and SHLOAD entries in the primary command/vector tables was unattractive, however, since everybody else used them for their own patches, and it would probably conflict. By mid-February, 1990, I had figured out how to search a *SECOND* command/vector table, which could bring the number of possible commands to 128. When TEXT40 worked from this second command/vector table, I realized that I had the skeleton in place to do a major rewrite of SmartBASIC 1.0, and add all the programming features I had always wanted whenever something went wrong!

I wrote out a list of things I wanted to do: block reads/writes, screen color control without POKEs, memory management without absolute addresses, ability to invoke EOS function calls without machine code subroutines POKEd in. By June, 1990, after a steady effort, most of my wish list had been implemented. But again circumstances dictated a hiatus from the project.

In October, 1990, I bought a Tandy 2800HD laptop for the purpose of writing my Ph.D. thesis. The internal 2400-baud modem was my gateway to the Cleveland FreeNet, whose ADAM sig introduced me to a living, coast-to-coast ADAM community (a great surprise). I discovered that there were double-sided disk drives, memory expanders, hard drives, clocks, and other kinds of hardware which had been produced for the ADAM after Coleco went bankrupt; and I also learned that various spaghetti patches were required to use this hardware within SmartBASIC. Wouldn't it be nice to implement drivers for *ALL* this hardware in my enhanced SmartBASIC, by now called SmartBASIC 1.x?

And so began the final phase of the project. Using loaned equipment, software, and technical information, I added support for bigger disk drives, RAM disks, and even hard disks; I wrote clock drivers, PEEK and POKE access to *ALL* of ADAM's memory spaces (ROMs and expansion RAMs); and I provided command-level access to serial ports. The very last thing, of course, was user documentation and utility programs to install and reconfigure SmartBASIC 1.x.

The chapters which follow describe how to use the SmartBASIC 1.x interpreter for programming on the Coleco ADAM computer. Believe me, it would have been *IMPOSSIBLE* to debug some of the later stages of the project without having the early commands like TEXT40, BLREAD, and BLWRITE, and the early functions HEX$ and MOD! I hope that you will find the SmartBASIC 1.x interpreter as useful in your own programming tasks as I do.

ACKNOWLEDGEMENTS: FIRST PRINTING.

SmartBASIC 1.x would never be what it is today without the assistance of other members of the ADAM community. I thank George Koczwara, The Cleveland ADAM eXchange BBS, for kind loans of his "Nibbles & Bits" collection, a 256K memory expander, and various software. Herman Mason, Jr., The Trading Post BBS, let me borrow RAM disk software and an Orphanware clock. Both George and Herman put their Mini Wini hard drive systems at my disposal. Alan Neeley, ADAMlink of Utah BBS, kindly lent a Dyno-Mite Digitizer/Clock, as well as several "ADAM Informant" issues containing technical information about the ADAMlink modem, the Orphanware serial card, and the internal SmartClock. George Harpster, head of our local users group, Cleveland B.A.S.I.C., surprised me with an Orphanware 80-column video unit (instrumental in the implementation of TEXT80). I am grateful to all the members of B.A.S.I.C., who have patiently endured demonstrations of SmartBASIC 1.x as it evolved month by month.

The Cleveland FreeNet, run by Case Western Reserve University, has my thanks for having an ADAM sig (run by Messrs. Koczwara, Mason & Harpster), because that was how I hooked up with them in the first place. FreeNet was also my means of acquiring some more ADAM hardware, this time for keeps, courtesy of Nick Poulos (ADAMlink modem) and Allen Tucholski (Orphanware serial card and 160K disk drive).

Steve Major, The ADAM Connection BBS, deserves mention as the source of my internal SmartClock. Steve and Alan Neeley both provided important technical information necessary to make SmartBASIC 1.x compatible with the PowerMate hard drive system.

Ron Collins, Bart "Zonker" Lynch, Alan Neeley, George Koczwara, and Herman Mason, Jr. were pressed into service as beta testers of SmartBASIC 1.x. Their comments (and bug reports) have been a big help as this project has drawn to a close.

Tony Patterson, the premier ADAM graphics programmer, made some valuable suggestions about graphical enhancements to SmartBASIC 1.x, and was the impetus for the current boot screen. He also provided me with PowerPaint, which I used to make the disk/tape labels.

A special "Thank You" goes to Herman Mason, Jr., for all the hours he has spent in chat with me on "The Trading Post" (mostly to the consternation of Zonker, trying to get through). Those may have been lost programming hours, but they have not been lost hours.

Last but not least, my wife, Joan, gets lots of hugs and kisses for keeping our two daughters, Christina and Elanor, out of the piles of disks and assembly listings and books.....well, most of the time.

Richard F. Drushel
Cleveland Heights, Ohio
1 August 1991

ACKNOWLEDGEMENTS: SECOND PRINTING.

Two months of actual usage have uncovered a few bugs in the original release of SmartBASIC 1.x. My thanks to Ron Mitchell for identifying problems with external terminal emulation file handling in the BOOT.SYS program, as well as the first actual testing of the routines to access the MIB2 serial/ parallel board. Special thanks are due to Alan Neeley, ADAMlink of Utah BBS, for his kind loan of an MIB2 board, enabling me to correct the bugs in the serial initialization and send routines.

SmartBASIC 1.x has proved successful beyond my wildest dreams. I thank Jim Notini of NIAD for his commercial support, as well as all those who have purchased SmartBASIC 1.x through NIAD. Their interest and feedback are shaping and making possible future enhancements to SmartBASIC 1.x.

Richard F. Drushel
Cleveland Heights, Ohio
17 October 1991

ACKNOWLEDGEMENTS: THIRD PRINTING.

A minor bug was found in the unasmhex sample program, which caused faulty disassembly of some SET and RES instructions. Also, I had neglected to rename the file in the HELLO program to SB1.x.20Y after making the MIB2 patches (when the interpreter was changed from 20X to 20Y). Both of these have been corrected. I thank Walt Nagel for pointing out an error in the description of EOS function 60. I am also grateful to Mark Gordon, Micro Innovations, for his kind gift of an MIB2 board; I now use it for parallel printing and as my TEXT80 terminal.

Richard F. Drushel
Cleveland Heights, Ohio
8 February 1992

STATEMENT OF ORIGINALITY.

While the finished SmartBASIC 1.x would not have been possible without the generous contributions of hardware and technical information from many kind people in the ADAM community, I wish to state here, for the record, that SmartBASIC 1.x is the product of my own efforts, intellectual and physical. I have not pirated anyone's proprietary software, but have used public domain routines and information (especially from "The Hacker's Guide to ADAM" series by Peter and Ben Hinkle) where available.

To lay to rest any doubts, I am *NOT* Dr. Solomon Swift. I have neither seen, nor met, nor spoken to, nor corresponded with Dr. Swift. SmartBASIC 1.x is *NOT* a reverse-engineered form of Dr. Swift's GoBASIC interpreter. The fact is, I never saw a demonstration of GoBASIC until SmartBASIC 1.x was nearly completed. In a couple instances, there are some convergences between GoBASIC and SmartBASIC 1.x (notably the CLS and BEEP commands), but these are most likely due to a common familiarity with MicroSoft BASIC. The internal architectures of GoBASIC and SmartBASIC 1.x are totally different. Comparison of the actual machine code routines above address 27407 reveals markedly different programming styles, proof of separate authorship.

I have visited and talked to *IN PERSON* the following members of the ADAM community: Herman Mason, Jr., George Koczwara, George Harpster, Ron Collins, Tony Patterson, and Barry Wilson. These people can affirm that I am indeed *NOT* Dr. Solomon Swift.

If additional proof of the originality of SmartBASIC 1.x is needed, I have all the hand-coded, hand-assembled, commented machine code routines for all the new commands and functions. I hope in the near future to be able to transfer these manuscripts to word processing files so that printed listings are available for advanced programmers or the merely skeptical or curious.

The stories I have been told about the GoDOS/GoBASIC fiasco have made me anticipate some skepticism toward SmartBASIC 1.x. "Ohhhhh no, not another GoDOS! Hands on your wallets, everybody!" Considering that the unfulfilled promises of GoDOS nearly wrecked the ADAM community (I am told), a jaundiced view toward my "one SmartBASIC accesses all" project is perhaps understandable.

I hope, however, as you read through the manual, try out the sample programs, and begin to write your own programs, that you realize I have not promised anything that isn't really *THERE*.

CHAPTER 1: WHAT IS SmartBASIC 1.x?

SmartBASIC 1.x is a major rewrite of the SmartBASIC 1.0 interpreter for the Coleco ADAM computer. SmartBASIC 1.x provides the programmer with 40 new commands and 38 functions, and enhances the operation of 7 existing commands and 6 functions. SmartBASIC 1.x allows the programmer *COMPLETE* access to the hardware and software capabilities of the ADAM, using simple, high-level commands--without arbitrary PEEKs, POKEs, or CALLs, and without the programmer needing to understand Z80 machine language.

SmartBASIC 1.x makes it easy to:

* read and write absolute disk/tape blocks
* change screen colors
* select 31, 40, or 80-column TEXT, complete with FLASH and INVERSE
* define a scrolling TEXT window smaller than the physical screen
* manage interpreter memory independent of absolute addresses
* print to a parallel printer
* initialize and communicate through serial ports
* access all system RAM, ROM, and expansion RAM
* invoke EOS operating system functions without machine language subroutines
* use larger disk drives, including hard drives
* format floppy disks for any size drive
* keep time and date with hardware and software clocks
* communicate directly with Z80 I/O ports

Additionally, SmartBASIC 1.x fixes some annoying bugs and oversights in SmartBASIC 1.0:

* parameters are range-checked for all commands and functions
* repeated error trapping does not crash the system
* ERRNUM returns rational error codes
* the keyboard line buffer is 254 characters long instead of 128
* control characters echo on the screen when typed
* INSERT and DELETE keys replace ^N and ^O
* the cursor is hidden unless waiting for typed input
* LIST does not pad everything with extra spaces
* LIST starts printing in column 2, so you can edit line numbers above 9999
* the parser error reprint of a bad line starts in column 2
* SmartBASIC 2.0-type parser errors--beep/sadface replacement of bad line
* POS and VPOS are 1-based instead of 0-based, and return absolute positions
* dummy arguments are not required for ERRNUM, POS, VPOS, FRE, and RND
* color commands and functions use TI color codes instead of AppleSoft
* RANDOMIZE command gives true random numbers (seeds optional)
* RESTORE, RESUME, and RETURN to any line number
* values set by SPEED, ROT, etc. can be found without PEEKs
* multiple programs can be MERGEd in memory

SmartBASIC 1.x provides shorthand synonyms for a few common commands:

* LOCATE line, column instead of VTAB line: HTAB column
* BEEP instead of PRINT CHR$(7);
* PUT x instead of PRINT CHR$(x);
* CLS instead of HOME

Lastly, SmartBASIC 1.x implements directly a few "miscellaneous" command and functions which previously could be achieved only by multi-statement, multi-line subroutines:

* WHILE...WEND loops
* HEX$(x) to return INT(x) as a hexadecimal string
* Z80-type bit test/set/clear functions
* Z80-type logical AND/OR/XOR/CPL functions
* MOD function to return remainder after integer division
* SPC$(x) function to return a string of x spaces
* STRING$(count,value) to make a string of count CHR$(value)'s.
* LINPUT command to get a whole line of text up to the CR, ignoring commas

All the enhancements of SmartBASIC 1.x take a little over 8K of extra RAM, leaving more than 17K of programming workspace. While this is, of course, less than SmartBASIC 1.0, the compactness of SmartBASIC 1.x commands and functions means that whole subroutines in older programs can be replaced by a statement or two.

Existing programs which may use some SmartBASIC 1.x reserved words as variable names can be safely loaded and edited, without loss of "incompatible" lines, thanks to the use of the SmartBASIC 2.0-type parser error handler. In SmartBASIC 1.0, if the disk/tape file has a syntax error in it, the offending line is *THROWN AWAY* and it cannot be edited. In SmartBASIC 2.0, parser errors which occur during a LOAD or keyboard entry are *SAVED* with a beep/sadface after the line number; such programs when RUN will stop at the offending line with a "Syntax Error." The beep/sadface is ignored by the line editor, so once the line is corrected, it disappears. I have implemented this type of parser in SmartBASIC 1.x; when an existing program is LOADed, any incompatibilities will be displayed when the program is RUN.

Any existing SmartBASIC 1.0 or SmartBASIC 2.0 program will work under SmartBASIC 1.x, *WITH THE FOLLOWING CAVEATS*:

* Programs which use PEEKs and POKEs to access interpreter parameters will probably work, *IF* the program was written for SmartBASIC 1.0. In a miracle of programming, I have left the commonly-used POKEs for SmartBASIC 1.0 screen color in the same place; but these should be changed to the appropriate COLOR(n)= statements, for compatibility with future versions of SmartBASIC 1.x *WHICH MAY NOT BE TIED TO THE 1.0 MEMORY MAP--HINT HINT*. POKEs to change the prompt are unnecessary, since the PROMPT= command will do it. POKEs to hide the cursor are unnecessary as well, since it is now displayed only when waiting for keyboard input. In general, SmartBASIC 1.x provides access to interpreter parameters independent of their actual location in memory.

* A SmartBASIC 2.0 program which PEEKs and POKEs at interpreter parameters

will fail, due to the vast memory map differences between SmartBASIC 1.x and SmartBASIC 2.0.

* Programs which POKE in a machine code subroutine into the area between address 27407 (SmartBASIC 1.0's LOMEM) and SmartBASIC 1.x's LOMEM (given by the MEM(0) function) will *BOMB* spectacularly, since that workspace is used by SmartBASIC 1.x. Similarly, attempts to allocate disk buffers, etc. in this area will also cause problems. There are 2 alternatives: (1) recode the routine according to the principles described in Appendix 1: Writing Relocatable Z80 Assembly Language Routines, and move it to an area above MEM(0); or (2) translate the routine into equivalent SmartBASIC 1.x statements. Most machine code routines in the SmartBASIC programs I have seen are for disk/tape block reads/writes. A few make EOS function calls to check device status. Some are specialty programs which bank switch to a ROM and copy the contents back to RAM. BLREAD/BLWRITE, CALL EOS(n), and the enhanced PEEK(address,latch) and POKE(address,value,latch) commands are the proper equivalents in SmartBASIC 1.x. In general, only utility-type programs should present a significant difficulty in this regard.

* SmartBASIC 1.x does not alter the EOS RAM (57344-65535) in any way. The data area at 64864-65535 is common to both EOS-5 and EOS-7. PEEKs and POKEs to this data area will work. Beware of drivers for hard disks and RAM disks; these change the EOS code segment (usually overwriting sound routines).

* Programs with CALLs are a knotty problem. If the CALL is to a POKEd-in machine code routine, then the difficulties are as described above. CALLs to SmartBASIC 1.0 interpreter routines may or may not work, depending upon the routine CALLed and whether it is to an area of the interpreter which has been rewritten. CALLs to SmartBASIC 2.0 interpreter routines will fail because of memory map differences. Direct CALLs to EOS are rare, since almost all EOS calls require a register setup. CALL 64803 (read VDP register 8) has been used to restart non-maskable interrupts after disabling by a user routine. CALL 64743 (go to SmartWriter) is another common CALL. These will work, because SmartBASIC 1.x does not modify EOS in any way. Hard drive and RAMdisk users beware...the drivers modify EOS code.

* Programs which use HTAB and VTAB to specify cursor position, or use POS and VPOS to return the current cursor position, may require minor editing. All cursor position commands in SmartBASIC 1.x are 1-based, and access absolute locations on the screen. In SmartBASIC 1.0 and 2.0, HTAB/VTAB are 1-based, but POS/VPOS are 0-based; and screen coordinates are relative to the text window. For example, the GR/HGR text window occupies absolute screen lines 21 to 24. With the cursor at the top left corner of this window, VPOS(dummy) in SmartBASIC 1.0 returns 0, while VPOS in SmartBASIC 1.x returns 21. VTAB 4 in SmartBASIC 1.0 puts the cursor on screen line 24, while the same command in SmartBASIC 1.x gives an "Illegal Quantity" error. There is also no parameter checking for these commands in SmartBASIC 1.0, so attempts to HTAB 0 or VTAB 25 will fail in SmartBASIC 1.x. This is usually a problem only for programs which calculate the argument and don't consider the boundary cases.

SmartBASIC 1.x has been carefully designed to support most of the third-party hardware available for the ADAM. The only absolute hardware requirements, however, are an R80 ADAM system unit, a tape drive, and a TV. Earlier versions of the system unit, like the original R59, have design bugs which cause problems when bank-switching memory (required for hardware clocks and the enhanced PEEK/POKE commands) or connecting third-party hardware (my R59 locks up when I try to access an Orphanware serial card via the SERIAL command). To determine the version of your system unit, boot to ADAM's Electronic Typewriter, then type Control-R. A blue box will appear for SmartKey IV which says Rxx, where xx is a 2-digit number. SmartBASIC 1.x was developed on an R80 ADAM and tested on an R59. I have been told that there are R77s as well, though I have never seen one personally.

The following hardware is supported directly by SmartBASIC 1.x:

* Coleco ADAMlink modem (initialization only)
* Coleco 64K memory expander (and clones by Orphanware and OPA)
* Orphanware 256K and 512K memory expanders
* Micro Innovations 1024K memory expander
* Orphanware PIA2 parallel board (and clones)
* Eve/Orphanware RS-232C serial board (all 4 possible ports)
* Micro Innovations MIB2 dual serial/parallel board
* Eve SS/SC board (clock only)
* Orphanware clock board
* Trisyd SmartClocks ("no-slot" and Dyno-Mite digitizer versions)
* Eve/Orphanware 80-column video unit
* 320K, 360K, and 720K double-sided disk drives
* Non-256K tapes created by the MegaCopy tape formatter
* Mini Wini hard drive
* PowerMate hard drive

The following hardware is not supported directly by SmartBASIC 1.x, but may be accessed using the appropriate IN variable,port and OUT port,value commands. Existing programs which access these devices may require extensive rewriting to avoid memory conflicts:

* Coleco ADAMlink modem (dialing and data transfer)
* Coleco Autodialer
* Eve SS/SC board (speech synthesizer)
* MIDI interface board
* Dyno-Mite sound digitizer

Commercial RAMdisk driver software for expansion RAM does not conflict with SmartBASIC 1.x, since the drivers are installed to EOS RAM. However, RAMdisk software is usually part of a patched SmartBASIC 1.0 interpreter. SmartBASIC 1.x does not implement its own RAMdisk drivers; to access a RAM disk under SmartBASIC 1.x, you must first boot the patched SmartBASIC 1.0 (installing the RAMdisk drivers), then boot SmartBASIC 1.x. Details are presented in Chapter 2: Getting Started With SmartBASIC 1.x.

CHAPTER 2: GETTING STARTED WITH SmartBASIC 1.x.

SmartBASIC 1.x is supplied on disk or digital data pack. To run Smart-BASIC 1.x, put the disk or tape in any drive, and pull the reset switch. A boot screen will be displayed while the interpreter loads. (This takes a while, especially from tape drives, so do not be alarmed if nothing seems to be happening right away.) After printing a copyright message, SmartBASIC 1.x asks you to enter the time and date. At last, you are given the familiar] prompt. You are now in SmartBASIC 1.x!

Type CATALOG to verify that the following files are present:

```
   Volume: SB1.x

    *H  SB1.x.20Y     36  ;the SmartBASIC 1.x interpreter
    *0  BOOTSB1x20     1  ;boot block for hard drive systems (smiley face)
    *A  BOOT.SYS       5  ;reads CONFIG.SYS at startup and configures system
    *A  CONFIG.SYS     1  ;configuration data file; supplied with defaults
    *A  EDIT.SYS      18  ;configuration editor; creates/changes CONFIG.SYS
    *A  DFAULT.SYS     1  ;writes original default values to CONFIG.SYS
    *A  HELLO          1  ;boots SB1.x if you booted another BASICPGM first
    *A  cartwriter     3  ;game cartridge copy utility
    *A  clock          1  ;displays time and date using WHILE-WEND loops
    *A  format         1  ;format disks; same logic as internal FORMAT command
    *A  hgr2fun        1  ;shows use of 16 TI colors in hi-res graphics
    *A  memdump        1  ;dumps out any ADAM memory space (RAM or ROM)
    *A  squeezer       4  ;disk/tape squeeze utility
    *A  unasmhex      14  ;Z80 unassembler utility
```

Your first task should be to make a working copy of the master disk/tape. If you do not have a formatted disk at hand, you may make one by putting an unformatted disk in either disk drive and typing FORMAT drive where drive is either 5 (D5, disk 1) or 6 (D6, disk 2). You may use any public domain or commercial file copying program, such as File Manager to make backup copies, if you wish. However, in order to make the copies bootable, you will have to use the BLOCK COPY option to copy block 0 from the master to block 0 of the copy.

Next, you should customize SmartBASIC 1.x for the hardware you have attached to your system. Make a list of your hardware, being careful to note such items as port/baudrate/parity/stop for serial cards, disk drive sizes, and emulation for serial terminal. Then RUN EDIT.SYS, the SmartBASIC 1.x configuration editor program. Work your way through the menus, then save the new configuration. (See Chapter 3: Customizing SmartBASIC 1.x with EDIT.SYS for more information.) If you make a mistake, you can always abort out of EDIT.SYS without saving any changes, then restart.

You must reboot SmartBASIC 1.x in order to invoke the new configuration. Put the newly-configured disk/tape in any drive and pull the reset switch. SmartBASIC 1.x will now be tailored to *YOUR* system!

In SmartBASIC 1.0 and 2.0, any A-type program file named HELLO will be automatically run whenever the system boots up. In SmartBASIC 1.x, any A-type program file named HELLO1.x has the same property. You can thus write "turnkey" or menu programs, or use existing HELLO programs with SmartBASIC 1.x. Just be sure, however, that your old HELLO program is not full of PEEKs, POKEs, and CALLs which are incompatible with SmartBASIC 1.x...

SmartBASIC 1.x does not include RAMdisk drivers. If, however, a RAMdisk driver has already been installed into EOS, SmartBASIC 1.x can access it as D7. The Walters Software RAMdisk, for example, is part of a custom-patched BASICPGM (the binary SmartBASIC interpreter), and is invoked when that patched SmartBASIC is booted. To use the Walters-type RAMdisk from SmartBASIC 1.x, you must first boot the patched BASICPGM (pulling the reset switch with that disk/tape in the drive). When you are asked to initialize the RAMdisk, do so. You will then exit to a] SmartBASIC prompt. At this point, put in the SmartBASIC 1.x disk/tape and type RUN HELLO. This will BRUN the SmartBASIC 1.x interpreter, and go through the same startup sequence as if you booted directly from the master (although there is no "ADAM presents SMARTBASIC 1.x" boot screen). When you again reach the] prompt, D7 will access the RAMdisk.

You may use a file copying utility to make a self-booting SmartBASIC 1.x with RAMdisk by copying the SmartBASIC 1.x files onto a disk/tape which already has the RAMdisk-patched BASICPGM and standard SmartBASIC boot block 0 on it. When you pull the reset, BASICPGM will load, the patches will install the RAMdisk, and then the turnkey HELLO program will automatically load in SmartBASIC 1.x. Note that this will take twice as long as booting either the patched BASICPGM or SmartBASIC 1.x alone.

Hard drive users have a few kinks in the setup procedure. To install SmartBASIC 1.x on your hard drive, use File Manager to copy all the program files from the master disk/tape (or better yet, a backup of the master) to whatever EOS volume you want to keep them on. Next, you must copy the BOOTSB1x20 boot block file to EOS volume 0. This file has the smiley face (^B) as the filetype; as supplied, however, the system and read-only attribute bits are *NOT* set (this is so that the file will show up on a SmartBASIC CATALOG). You will have to set these attribute bits in order for the program to boot. At this point, reboot the hard drive (Shift-Undo), set the current EOS volume to the one where you copied the SmartBASIC 1.x program files, then boot the BOOTSB1.x file from volume 0. Once you get to the] prompt, D2 will reference the current EOS volume on the hard drive, but no others. (Note the] prompt instead of 2x> where x is the current hard drive volume.) Now you must RUN EDIT.SYS, select the proper hard drive type from the menu (Mini Wini or PowerMate), then save the new configuration. The next time you boot SmartBASIC 1.x, the proper hard drive patches will be installed, and you will be able to access all EOS volumes of your hard drive. You will also have the 2x> prompt.

Note: BOOT.SYS, EDIT.SYS, DFAULT.SYS, and HELLO are SmartBASIC 1.x program files. They are not copy or LIST protected in any way; you may LOAD them and LIST them to see what they do. Since they are system programs, however, NOBREAK statements are used at the beginning of each to prevent uncontrolled escapes with ^C. This is not to discourage the curious; it is simply to prevent system crashes at boot time or while copying files or while changing the configuration. Additionally, all system programs check to see that they are indeed running under the correct version of SmartBASIC 1.x. If they are run from SmartBASIC 1.0 or 2.0, or from a different version of the SmartBASIC 1.x interpreter than they were originally distributed with, an "INCORRECT SB1.x VERSION" error is given and the program aborts.

CHAPTER 3: CUSTOMIZING SmartBASIC 1.x WITH EDIT.SYS.

EDIT.SYS is a menu-driven program to change the configuration data stored in the CONFIG.SYS file. This file is read at startup by the BOOT.SYS program, and the data therein used to set important system parameters like the size of disk drives, which serial ports (if any) are to be turned on (and with what initialization parameters), whether you have a serial or parallel printer, the emulation to be used for the TEXT80 serial terminal, the type of real-time clock installed, etc. The reason for a separate configuration program, requiring input from you, as opposed to an auto-sensing, auto-configuring SmartBASIC 1.x, is simply SPACE. The machine code routines necessary to seek out and identify all possible types of hardware connected to your ADAM take up an enormous amount of RAM, RAM that is better put to use by YOU for YOUR own programs. Once you have your system configured properly, if you don't change your setup, you may never have to use EDIT.SYS again.

EDIT.SYS has many levels of menus, all SmartKey-driven. SmartKey VI, Escape, and ^C always return you to the previous menu level. The remaining SmartKeys are used to select configuration options. Type RUN EDIT.SYS at any] prompt, and you will see the main menu:

```
                   SMARTBASIC 1.x CONFIGURATION EDITOR

                        Master Configuration Menu

                           I  Hardware Clock
                          II  Screen Colors
                         III  Drive Parameters
                          IV  TEXT/Printers
                           V  Serial Ports

                          VI  Exit

                    Press the SmartKey of Your Choice
```

The menus and sub-menus are pretty self-explanatory; a summary of each main heading is given below, as a guide to where to go to select what:

I Hardware Clock. Select which kind of real-time clock you have. The default is none (system NMI clock *ONLY*)

II Screen Colors. Specify foreground and background colors in normal and inverse video for the TEXTxx modes; also border color, graphics screen color, and the default values for COLOR and HCOLOR.

III Drive Parameters. Describe the size of your disk/tape/RAM drives, select a hard drive type, specify the number of directory blocks to allocate when you INIT a disk/tape.

IV TEXT/Printers. Pick the default TEXT mode (31/40/80), the printer to be used for ^P screen dumps, and set the line printer width.

V Serial Ports. Turn serial ports on/off, set port initialization parameters, pick ports for serial printer and serial terminal, specify TEXT80 terminal emulation.

VI Exit. Gives you the choice of either reviewing your selections, saving the new setup, or aborting without changing anything.

Note: the file CONFIG.SYS *MUST* be present on the current drive. If it is missing, EDIT.SYS reports an error and continues using whatever configur-ation parameters are currently set in memory. If you then save the setup, a new CONFIG.SYS will be created; but it probably won't be the one which you were trying to edit in the first place.

IN CASE OF ERRORS. If you inadvertently saved a bad configuration, or if CONFIG.SYS is somehow trashed, all is not lost. Simply run the program DFAULT.SYS. This program, a regular SmartBASIC program (i.e., it will run under *ANY* version of SmartBASIC), rewrites the original "factory" default configuration values to CONFIG.SYS. So at least you can get back to where you originally started. An example: You have a PowerMate hard drive, but you picked Mini Wini by mistake and saved the configuration. When you try to reboot SmartBASIC 1.x, it hangs spectacularly (it will; the driver patches are totally different for the two hard drive types). You can't even get to the SmartBASIC 1.x prompt to RUN EDIT.SYS again. What do you do? Boot your original SmartBASIC 1.0 which was already patched to access the hard drive. Change to the volume where you have the SmartBASIC 1.x program files. Type RUN DFAULT.SYS. Now you can reboot SmartBASIC 1.x. You will have to rerun EDIT.SYS to make it work on all the hard drive volumes again, but you will at least have access to the particular volume where SmartBASIC 1.x is stored until you save the new configuration. For a disk/tape based system, just do a variation of the same thing. Boot regular SmartBASIC 1.0, put in the SmartBASIC 1.x disk/tape, RUN DFAULT.SYS, and you are back to the "factory" defaults. Rerun EDIT.SYS and save the correct configuration.

CHAPTER 4: SAMPLE PROGRAMS.

Seven sample programs are included with SmartBASIC 1.x. These programs illustrate the usage of the enhanced commands and functions. Some of them are are simple demos, but others are invaluable programmer utilities. A brief description of each is given below:

cartwriter Views the contents of any game or program cartridge as printable ASCII characters. Also saves any cartridge to disk or tape, as either a C-type file (compatible with COPYCART by MMSG) or an H-type file. Since the amount of ROM in a given cartridge is variable, you should use the view function first, to find where each cartridge ends (you will see nothing but .... in the block dump) before you save it.

clock Continuously displays the system time and date, using the WHILE-WEND programming loop structure. Hour, minute, second, day, weekday, month, and year all roll over.

format Formats floppy disks in any size disk drive, and initializes tapes. You specify the medium size, the number of directory blocks, and the volume label. This program duplicates the logic of the internal FORMAT drive command of SmartBASIC 1.x. It also demonstrates the use of CALL EOS(n) to invoke operating system functions without machine language programs.

hgr2fun Uses the default DRAW shape in HGR2 to show all 16 TI colors (instead of the 8 AppleSoft colors), available for the first time *WITHOUT POKES*. The shape bounces back and forth, changing color at random each time it comes to the screen borders. After a few minutes, all the color bleeding makes some interesting Jackson Pollock-type pictures. To get out of it, hit ^C and then type TEXT.

memdump Displays the contents of *ANY* ADAM memory space as printable ASCII characters. This program demonstrates the enhanced PEEK(address,latch) function. The program can easily be modified to print the contents in hexadecimal (HEX$(value)) rather than ASCII, and with some help from the cartwriter program, could even save the contents to disk or tape.

squeezer Squeezes unused space out of disks or tapes, putting it all in a contiguous block at the end. This program highlights the BLREAD and BLWRITE commands. To keep it simple, no buffering is done of the blocks (it reads one, then writes one) This means it will take a long time to squeeze down a tape! I do *NOT* recommend using this program on a hard drive, simply because if an error occurs, you will lose of the entire volume (at least 1024 blocks). (Other volumes will not be affected.)

unasmhex The most useful SmartBASIC program I have ever written, since it made SmartBASIC 1.x itself possible. Unassembles *ANY* ADAM memory space into Z80 assembly language instructions. Memory contents are given in hexadecimal, but all arguments and addresses are given in decimal. Target addresses are calculated for relative jump instructions and given in parentheses at the end of the statement line. If you change all occurrences of PEEK(address,latch) to PEEK(address), and save the program, you can use it under SmartBASIC 1.0 and 2.0 as well. I originally wrote it for SmartBASIC 1.0, and added the latch capability when it became available in SmartBASIC 1.x. It takes more room on disk than disass by Peter and Ben Hinkle (14K versus 8K), but unasmhex is much smaller in RAM (disass runs "Out of Memory" in SmartBASIC 1.x).

The above sample programs (cartwriter, clock, format, hgr2fun, memdump, squeezer, and unasmhex) are hereby placed into the public domain. Modify them as you wish; if you make a useful improvement, I'd like to know about it.

SmartBASIC 1.x PROGRAMMING REFERENCE.

Related commands and functions are grouped together for easy reference. A complete alphabetical listing is given in the Index.

BIT AND LOGICAL OPERATIONS.***

x=BIT(value,bit) Tests whether bit (0-7) of value (0-255) is set (logical 1) or clear (logical 0). Returns either 1 (set) or 0 clear).

x=SET(value,bit) Returns the result of setting the specified bit (0-7) of value (0-255).

x=RES(value,bit) Returns the result of clearing the specified bit (0-7) of value (0-255).

x=AND(value1,value2) Returns (value1 AND value2). Both arguments must range 0-255.

x=OR(value1,value2) Returns (value1 OR value2). Both arguments must range 0-255.

x=XOR(value1,value2) Returns (value1 XOR value2). Both arguments must range 0-255.

x=CPL(value) Returns 1's-complement of value (0-255). This exchanges all ones and zeros (i.e., 11110000 becomes 00001111).

x=MOD(value1,value2) Returns the remainder after integer division of value1 by value2. Equivalent to:

```
value1-(INT(value1/value2)*value2)
```

None of these operations has any effect upon the values of the arguments (unlike the equivalent Z80 assembly language statements, where, for example, AND B is effectively A=(A AND B)).

x$=HEX$(value) Returns a string which is INT(value) expressed as a hexadecimal number. If value is 0-255, the string is 2 hex digits, with a leading zero if necessary (e.g., 23 decimal=17 hex; 12 decimal=0C hex). If value is 256-65535, the string is 4 hex digits, again with a leading zero if necessary (e.g., 256 decimal=0100 hex).

The bit and logical operations are very useful for interpreting tape and disk directories. They are much more efficient in terms of speed and space, compared to the equivalent multi-statement, multi-line subroutines necessary to implement them otherwise.

CLOCKS.***

TIME hh:mm:ss	Sets the system time: hours (0-23), minutes (0-59) and seconds (0-59). Sixtieths of seconds is always reset to zero by this command.
x$=TIME$	Returns system time as a string, e.g., "22:13:06".
x=TIME(arg)	Returns components of time as numbers:

0	sixtieth of a second	1	second	2	minute	3	hour

DATE mm\dd\yyyy[\ww]	Sets the system date: month (1-12), day (1-31), year (1983-2155), and optionally, weekday (1-7, 1=Sunday). All values except weekday are stored in EOS RAM at 64992-64994; hence this date will be stamped upon new files. If weekday is not specified, the current value is unchanged (default at boot time is 1=Sunday).
x$=DATE$	Returns system date as a string, e.g., "FRI-12-APR-1991".
x=DATE(arg)	Returns components of date as numbers:

0	day	1	month	2	year	3	weekday

The system clock uses the video NMI (non-maskable interrupt), which occurs 60 times per second. It has complete date (day and weekday), month and year rollover, correctly taking into account the lengths of different months. There is, however, no leap year handler--February is always 28 days long.

Due to an intractable bug in SmartBASIC, the video NMI is disabled in any of the graphics modes (GR, HGR, HGR2). Consequently, the NMI system clock comes to a halt as long as you are in a graphics mode. When you reenter a TEXTxx mode, the NMI system clock starts up again. Also, due to the design of the Z80 microprocessor, any device which uses the system bus (by exerting an active BUSREQ signal) also disables NMIs for as long as the device is actively using the bus. This means that tape and disk I/O will also temporarily stop the NMI system clock. So do not be alarmed if, after a 2-hour programming session, the system time is 5 or 10 minutes "behind."

CLREAD	Reads the time and date from the hardware clock specified by CLK, and sets the NMI system clock to that time and date.
CLWRITE	Writes the NMI system clock time and date to a hardware clock specified by CLK.
CLK=x	Sets the hardware clock to be used by CLREAD/CLWRITE. Currently supported values are:

0	no hardware clock installed	2	TriSyd SmartClock (Dyno-Mite digitizer cartridge)
1	TriSyd SmartClock (internal)	3	Eve SS-SC/Orphanware clocks

x=CLK	Returns the current clock device used by CLREAD/CLWRITE.

The default value is CLK=0 (no hardware clock). CLREAD and CLWRITE commands have no effect on the NMI system clock if CLK=0. If CLK is non-zero, and you do not have the appropriate hardware clock installed, then CLREAD and CLWRITE have unpredictable results. Generally, an "I/O Error" results, but sometimes the NMI system clock (CLREAD) is trashed. This is not fatal; simply use TIME and DATE to reset it. Be sure to use the weekday option with DATE, or garbage may remain. CLWRITEing to a non-existent hardware clock has no effect on the NMI system clock.

CLREAD correctly interprets hardware clocks operating either in the 12-hour or 24-hour modes; but CLWRITE always uses 24-hour mode. The leap year indicator of the Eve/Orphanware clocks is ignored.

Internally, the TriSyd and Eve/Orphanware clocks store the year as a binary-coded decimal (BCD) number 00 to 99. With the century rollover approaching, I have thought it useful to interpret the BCD year as follows: 83-99 are 1983-1999, and 0-82 are 2000-2082. This gives almost another 100 years of usefulness to the clock.

Regardless of the CLK value, TIME, DATE, TIME$ and DATE$ deal ONLY* with the NMI system clock; these do not access any hardware clocks.

CURSOR CONTROL.

LOCATE line,column	A more compact equivalent to VTAB line: HTAB column.
CLS	Same as HOME.

Both LOCATE and CLS are MicroSoft BASIC commands.

y=VPOS or **y=VPOS(dummy)**	Returns the current cursor line. The dummy argument required in SB1.0 and SB2.0 is optional.
x=POS or **x=POS(dummy)**	Returns the current cursor column. The dummy argument required in SB1.0 and SB2.0 is optional.

Both VPOS and POS are 1-based (unlike SB1.0 and SB2.0, where they are 0-based). In addition, VPOS and POS now return the *ABSOLUTE* screen coordinates, rather than *RELATIVE* screen coordinates. For example, the text window in GR and HGR (in y,x format) is (21,1) to (24,31). If the cursor is at the top left corner of that window, VPOS returns 21, *NOT* 0 like it would in SB1.0 and SB2.0. Similarly, to get the cursor to the top left corner, you must use LOCATE 21,1 (or VTAB 21: HTAB 1), not LOCATE 1,1--if you do, you will get an "Illegal Quantity Error." POS works the same way.

EOS FUNCTION CALLS.

CALL EOS(arg)	Directly access the operating system. arg is the number of the EOS function desired (0-100), which is simply the offset into the EOS jump table divided by 3. Parameters are passed and returned via the register variables AF,BC, DE,HL,IX, and IY. Call status is returned in the ZF and CF status variables (read only).
AF=value	Sets AF register for CALL EOS(arg). value is 0-65535.
x=AF	Returns value of AF register variable.
BC=value	Sets BC register for CALL EOS(arg). value is 0-65535.
x=BC	Returns value of BC register variable.
DE=value	Sets DE register for CALL EOS(arg). value is 0-65535.
x=DE	Returns value of DE register variable.
HL=value	Sets HL register for CALL EOS(arg). value is 0-65535.
x=HL	Returns value of HL register variable.
IX=value	Sets IX register for CALL EOS(arg). value is 0-65535.
x=IX	Returns value of IX register variable.
IY=value	Sets IY register for CALL EOS(arg). value is 0-65535.
x=IY	Returns value of IY register variable.
x=ZF	Returns value of ZF (zero flag) status (read only), 0 or 1.
x=CF	Returns value of CF (carry flag) status (read only), 0 or 1.

Note: the register variables will return negative (i.e., 2's complement) values if between 32768 and 65535. To see the positive form, add 65536.

These commands eliminate the need for most of the assembly language routines commonly used in SmartBASIC programs. Set up the register variables as you would for an assembly language routine, then CALL EOS(arg), then check the ZF status flag. Note that 2-byte integers are stored lobyte, hibyte in RAM, so that, for example, to set A=1 and clear all flags, AF=(1*256)+0 or 0100 hex.

EOS commonly sets the ZF if the function call was successful, and clears it if the call failed; if it failed, the A register has an error code. An easy way to extract this error code is INT(AF/256). Some of the more esoteric function calls use the CF to return the status of the call. For a complete listing of all EOS function calls, the register setups they require, and EOS error codes, see Appendix 2: EOS-5 Function Calls 0-100. For further technical information about EOS, see Appendix 3: EOS-5 Technical Notes.

```
     100 & CALL EOS demo
     101 & just a program fragment!
     110 LOMEM:MEM(0)+11+1024: & reserve buffers for filename and data
     120 file$="TESTA"+CHR$(3): & filename
     130 FOR n=1 TO LEN(file$)
     140 POKE MEM(0)+n-1,ASC(MID$(x$,n,1)): & put in filename
     150 NEXT
     160 GOSUB 1000: & create file
..........
     1000 & EOS create file
     1010 AF=4*256: &            LD A,4        ;disk 1
     1020 BC=0: &                LD BC,0       ;hiword of filesize in bytes
     1030 DE=size*1024: &        LD DE,size    ;loword of filesize in bytes
     1040 HL=MEM(0): &           LD HL,buffer  ;address of filename string
     1050 CALL EOS(51): &        CALL 64713    ;EOS create file
     1060 IF ZF=1 THEN RETURN: & RET Z         ;ok exit
     1070 GOTO 20000: &          JP error      ;error exit
..........
     20000 & error handler
     20010 BEEP: PRINT "ERROR ";INT(AF/256): & tell me what's wrong
     20020 LOMEM:MEM(0): & clean up
     20030 END
```

The register variables AF,BC,DE,HL,IX, and IY may be used as regular integer variables in SB1.x programs. It is recommended, however, that they be reserved for use with CALL EOS.

ERROR TRAPPING.

RESUME [line number] Returns to the main program after an error-handling routine was invoked via ONERR GOTO. If the optional line number is omitted, execution resumes at the statement at which the error occurred. If the line number is used, execution jumps to the first statement in the specified line.

x=ERRNUM or **x=ERRNUM(dummy)** Returns the code of the last error which occurred during program execution. The dummy argument is optional.

The error codes returned in SB1.0 and SB2.0 are irrational, presumably copied from AppleSoft. The interpreter, however, actually uses its own set, numbered 0 to 35, which is then mapped to the crazy codes. SB1.x uses this internal set. A few error codes unimplemented in SB1.0 and SB2.0 have been assigned to errors in SB1.x.

SB1.x code	error message	SB1.0/SB2.0 code
0	NEXT without FOR	0
1	Syntax	16
2	RETURN without GOSUB	22
3	Out of DATA	42
4	Illegal Quantity	53
5	Overflow	69
6	Out of Memory	77
7	Stack Overflow	77
8	Undefined Statement	90
9	Bad Subscript	107
10	Redimensioned Array	120
11	Divide by Zero	133
12	Type Mismatch	163
13	String Too Long	176
14	Formula Too Complex	-?-
15	Undefined Function	224
16	Incorrect Function Usage	-?-
17	Illegal Mode	-?-
18	INPUT Data	--unused--
19	Break	255
20	Break	255
21	Can't Continue	-?-
22	WEND without WHILE	--unused--
23	Range Error	2
24	Write Protected	-?-
25	End of Data	5
26	RESUME without ONERR	--unused--
27	File Not Found	7
28	I/O Error	8
29	No More Room	9
30	File Locked	10

31	Syntax Error	11
32	No Buffers Available	12
33	File Type Mismatch	13
34	Device Timeout	--unused--
35	--unused--	--unused--
36	Control Buffer Overflow	12

Error notes:

18 INPUT Data. The only time this error occurs is if you enter a string when INPUT is looking for a number, and you are inside an error-trapping routine (before a RESUME) which was enabled with ONERR GOTO, and an error has occurred. Normally, INPUT responds with "?Reenter" when you make this mistake, and you keep going 'til you get it right. If ONERR GOTO is enabled, any error causes a branch to the specified error-trapping routine; no message is issued for the error. Once inside an error-trapping routine, however, in order to prevent infinite loops, any subsequent errors which occur (before the RESUME) cause a program break and an error message. This is the situation which gives rise to error 18. This was not invented by me; the whole thing was already there in the interpreter, but with the error string set to null (so you would never see a message). All I did was to restore a message string.

19 Break. When ^C is pressed during program execution, an error 19 is issued. After cleaning up from the ^C, the error is changed to 20, and the program stops.

20 Break. STOP and END issue error 20 directly; this is to distinguish them from a ^C.

Error handling in SB1.x has been improved so that, if the commands are used correctly, an infinite number of errors can occur without blowing up the stack and hanging the system. For a demonstration, try running the following program under SB1.0 or SB2.0. (Take out all tapes and disks first!) See how many times it can RESUME before crashing. Then run it under SB1.x.

```
100 & error handling bug demo in SB1.0/SB2.0
101 & guaranteed to cause a system crash!
110 n=1: & initialize counter
120 ONERR GOTO 1000: & enable error trapping
130 x=4/0: & Division by Zero error
140 PRINT "You'll never get this far!": & you won't!
150 END: & only of your system!
1000 & error trapping routine
1010 PRINT n;" ";: & tell us how many times
1020 n=n+1: & one more time
1030 RESUME: & back for more grief
```

The state of the WHILE/WEND stack is preserved, as well as the status of any

FOR/NEXT loops. With RESUME line number, however, be sure not to branch into the middle of a FOR/NEXT loop or a WHILE/WEND loop, or the system will be corrupted.

Never issue CLRERR or another ONERR GOTO while in an error-trapping routine! Always RESUME somewhere first. Otherwise, the stack will be destroyed.

MEMORY MANAGEMENT.**

x=MEM(arg) Returns a memory usage address as a 2's-complement integer (-32768 to 32767).

arg	address
0	lowest possible LOMEM
1	current LOMEM
2	highest possible HIMEM after NEW
3	current HIMEM

MEM is provided for the implementation of relocatable assembly language subroutines and data transfer buffers (for BLREAD/BLWRITE). The programmer need not know the absolute addresses, but can access them as variables using arithmetic statements. Properly used, this allows subroutines and buffers to be allocated without conflict, regardless of the actual memory map of a particular version of the SB1.x interpreter. For example:

```
100 & sample program to copy disk 1 to disk 2
110 LOMEM:MEM(0)+1024: & reserves a 1K transfer buffer
120 FOR n=0 to 159
130 BLREAD 5,n,MEM(0): & reads block n from disk 1
140 BLWRITE 6,n,MEM(0):  & writes the data to disk 2 as block n
150 NEXT
160 LOMEM:MEM(0): & restore buffer space to free RAM
170 END
```

This example will run correctly, regardless of the absolute value of MEM(0). For examples of how to make totally relocatable Z80 assembly language subroutines, see Appendix 1: Writing Relocatable Z80 Assembly Routines.

x=FRE or **x=FRE(dummy)** Returns the amount of free program memory. The dummy argument required in SB1.0 and SB2.0 is optional.

x=ADDR(arg) Returns a pointer or vector address as a 2's-complement integer (-32768 to 32767).

arg	address
0	default shape table (the square)
1	current shape table (user-defined)
2	default USR routine (a RET instruction)
3	current USR routine (user-defined)
4	default NMI routine (a RET instruction)
5	current NMI routine (user-defined)
6	default TERM emulation pointer table (TTY, Heath 19, 6 External)
7	current TERM emulation pointer table (user defined)

ADDR(arg)=address Sets a pointer or vector address. address can either be an unsigned integer (0 to 65535) or a 2's-complement integer (-32768 to 32767). Because of the definitions of the ADDR function, even-numbered args are undefined for the ADDR command.

arg	address
1	current shape table
3	current USR routine
5	current NMI routine
7	current TERM emulation pointer table

DRAW and XDRAW draw the shape at ADDR(1). USR executes the assembly language routine at ADDR(3). With every NMI, the assembly language routine at ADDR(5) is executed. (This is useful for continuously reading and buffering incoming characters from a serial port, for instance.) All Z80 registers except those of the alternate set (AF', BC', DE', HL', IX', and IY') are PUSHed onto the stack prior to entry, and POPped off after the RET. The routine *MUST* end in a RET and *MUST* leave the stack in the same condition it was in on entry! A separate user stack need not be allocated; but if you do, be sure to save the old SP and restore it before you RET! There are no other restrictions on what the routine can do, including EOS function CALLs. ADDR(7) points to a 16-byte table of 2-byte pointers (lobyte, hibyte), each of which points to a terminal emulation data table for the appropriate value of TERM in TEXT80 (see Appendix 5: External Terminal Emulations). This function is provided for applications which might require multiple internal terminal emulations (such as a communications program).

When you are finished with your shape table or assembly language routine, it is good programming practice to restore ADDR to the default values. For example:

```
     100 & sample program using ADDR
     110 LOMEM:MEM(0)+100: & reserve space for a 100-byte shape table
     120 x=MEM(0): IF x<0 THEN x=x+65536: & make unsigned integer
     130 x$=",A"+STR$(x): & make address argument for BLOAD
     140 HTAB 1: PRINT CHR$(4);"BLOAD shapetable";x$: & load to address
     150 ADDR(1)=MEM(0): & set new shape table address
     1000 & rest of DRAW/XDRAW program here
........
     10000 & clean up and exit
     10010 ADDR(1)=ADDR(0): & restore default shape table address
     10020 LOMEM:MEM(0): & restore table space to free RAM
     10030 END
```

x=PEEK(address [,latch])   Returns the byte stored at a memory address.

POKE address,value [,latch]   Changes the contents of memory at address to value (0-255).

The optional latch parameter allows you to specify additional ADAM memory spaces:

latch	memory type	unique address
0	standard RAM (same as omitting latch)	0-65535
1-16	expansion RAM banks 1-16 (each 64K)	0-65535
17	cartridge ROM	32768-65535
18	OS-7 ROM	0-8191
19	EOS ROM	24576-32767
20	SmartWriter ROM	0-32767

Effectively, the latch parameter causes the specified memory to be bank-switched in for the PEEK or POKE, then the standard 64K RAM configuration is restored. If address is not within the unique space of the specified memory, the address accessed is the same as standard RAM. For example, PEEK(0,17) returns the same value as PEEK(0) or PEEK(0,0), because the cartridge ROM begins at 32768.

There are noticeable performance differences between TEXTxx mode and graphics modes, when a non-zero latch is used. Specifically, access is faster in the graphics modes. This is because, in order to avoid problems when bank-switching memory, SB1.x is dodging the non-maskable interrupts (NMIs) in TEXTxx modes. This care is not necessary in graphics modes, because NMIs are disabled.

On rare occasions, the system may still lock up during one of the bank switches. System electrical noise is probably the culprit; but the only solution is to pull the reset button. Original R59 ADAMs and hard drive systems seem to be more susceptible to this lockup than a stand-alone R80 ADAM. This is a hardware design problem.

POKEing to a ROM has no effect upon the contents of the ROM.

MISCELLANEOUS.

BEEP Equivalent to PRINT CHR$(7). A MicroSoft BASIC command.

x=HDRV Returns the current hard drive type (0=none, 1=Mini Wini, 2= PowerMate). This value is set at startup by BOOT.SYS based upon the configuration data in CONFIG.SYS. This function is provided for the benefit of assembly programmers who wish to write (or need to write) device-dependent routines to access hard drives.

LINPUT ["prompt";] var$ [,var2$ ...] Line input into a string variable. All characters up to a carriage return are included; commas are *IGNORED* as data separators, but are included as part of the string. Each string must be less than 256 characters long. To LINPUT a list of strings, each one must be terminated by a carriage return. If the optional "prompt" is omitted, a "?" is used as a default. LINPUT may also be used to read text files from tape or disk, as long as each line ends in a carriage return and is not more than 255 characters long.

PROMPT=value Sets the prompt to CHR$(value). value must be 0-255.
x=PROMPT Returns the current ASCII code of the prompt.

Note: PROMPT has no effect in hard drive systems, since the hard drive patches use a different routine to display the current drive and volume with the > prompt.

PUT value Writes value (0-255) to the current PR device. This is the complement to GET. In file I/O, you can use this to put control codes into the file. You can actually use this to copy a binary file; however, before CLOSEing the new file, you *MUST* sent a null PRINT statement, or a carriage return, in order for SmartBASIC to correctly record the length of the file. This is due to intractable bugs in SmartBASIC's (not EOS's) file handling routines. Otherwise, PUT value is equivalent to PRINT CHR$(value);: (note the semicolon; PUT does not force a carriage return).

RANDOMIZE [seed1,seed2] Reseeds the random number generator. If the optional seeds are omitted, the current contents of the Z80 memory refresh register, R, are used as seeds. (This is a reasonably random number.) If explicit seed values are used, seed1 and seed2 must be 0-65535.

x=RND Returns the next random number in the sequence. Equivalent to RND(positive argument). This eliminates the need for a dummy argument in the most common use of RND.

x=ROT Returns current ROT.

x=SCALE Returns current SCALE.

x$=SPC$(arg) Returns a string consisting of arg spaces (1-255).

x=SPEED Returns current SPEED.

x$=STRING$(count,value) Returns a string consisting of count characters of ASCII code value. For example, STRING$(32,5) is equivalent to SPC$(5). The ASC function is helpful when you can't remember the ASCII code for a certain character, e.g., STRING$(ASC("*"),5).

x=VER(0) Returns the version number of SB1.x. Official release versions begin at 20; if a lower number appears on your machine, then either the interpreter has been corrupted (in RAM or on tape/ disk) or you have a beta test copy of SB1.x (hopefully not pirated).

x=VER(1) Returns the version number of EOS. The ROM version loaded when you reset your ADAM is 5. If you boot SB1.x from Disk Manager, however, the EOS version is 7. (Disk Manager and SB2.0 are the only Coleco programs which use EOS-7, which is supplied as a binary file and loaded into RAM, overwriting EOS-5.)

& Equivalent to REM. Explicitly implemented as REM, rather than as a USR-like function which ignores all arguments and is just a RET (as in SB1.0 and SB2.0).

PORT I/O.

IN variable,port Reads a byte from the specified Z80 I/O port and stores it in variable. port ranges from 0 to 255. variable must be numeric (real or integer).

OUT port,value Sends value out the specified port. value must be a numeric expression which evaluates to a number less than 256. Real values are truncated to integers.

These two commands are exact counterparts to the assembly language IN and OUT statements, thus removing another need for custom assembly language routines.

PRINTERS.

PR #0	Video screen or serial terminal (depending on TEXTxx).
PR #1	ADAM daisy-wheel printer (with PR #0 echo).
PR #2	Orphanware PIA2 port 64 parallel printer (with PR #0 echo).
PR #3	Serial printer (with PR#0 echo). The default port is 68, though it may be changed with the SER(0)= command. The port must be initialized with the SERIAL command.
PR #4-7	Currently the same as PR #0.

PRN #arg Selects which printer driver is used for ^P screen dumps. arg is the same as in PR #. The corresponding PR # driver does *NOT* have to be in effect. This means that you could ^P to a parallel printer (PRN #2) while being in screen mode (PR #0).

All printer output via either PR # or PRN # is checked as to whether the specified device is on-line or not. If the device does not respond in about 10 seconds, a "Device Timeout Error" results. Serial devices are also checked for parity, overrun and framing errors. If any of these occur, an "I/O Error" results. All printer errors cause the current PR # vector to be reset to PR #0 (screen), whether they are trapped with ONERR GOTO or not. If an "I/O Error" occurs on a serial device, it *MUST* be reinitialized with the SERIAL command in order to be accessed again.

PRWIDTH=value Set the column width of the printer.

x=WIDTH(0)	Returns current screen width (31, 40, 80)
x=WIDTH(1)	Returns current printer width.

Note: No corresponding WIDTH(arg)= command is provided, since the TEXTxx commands take care of that for the screen. Hence PRWIDTH= as a command without a corresponding function.

PROGRAM FLOW CONTROL.*

RESTORE [line number] Sets the DATA/READ pointer. If the optional line number argument is omitted, the READ pointer is set to the line of the first DATA statement in the program. With the line number argument, the READ pointer is set to that program line, and READ begins at the first DATA statement encountered thereafter. The line number referenced need not contain a DATA statement.

RETURN [line number] Transfers control back to the main program at the end of a subroutine invoked by GOSUB. If the optional line number is omitted, then the program resumes at the first statement after the GOSUB. With the line number argument, control passes to the first statement of the specified line. This is equivalent to POP: GOTO line number.

WHILE condition...
WEND This loop programming structure is more efficient than IF...GOTOs. On entry, the start of the loop is recorded, then subsequent statements are executed until a WEND is encountered. The entry condition is then tested. If the condition is true, control passes back to the beginning of the loop. If the condition is false, then the program continues with the next statement following the WEND. WHILE/WEND loops may be nested up to 10 levels; attempts to nest the 11th result in a "Stack Overflow Error."

```
100 & sample program illustrating WHILE/WEND
101 & shows a simple system clock without TIME$
110 hh=TIME(3): & hours
120 LOCATE 10,1: PRINT hh;: & show hours
130 WHILE hh=TIME(3): & hours loop
140      mm=TIME(2): & minutes
150      LOCATE 10,5: PRINT mm;: & show minutes
160      WHILE mm=TIME(2): & minutes loop
170           ss=TIME(1): & seconds
180           LOCATE 10,9: PRINT ss;: & show seconds
190           WHILE ss=TIME(1): & seconds loop
200           WEND: & end seconds loop
210      WEND: & end minutes loop
220 WEND:  & end hours loop
230 GOTO 110: & do it all over again
```

Programming note: There is a separate stack allocated for WHILE/WEND. For you hackers out there, do not consider this to be "free space!"

SCREEN COLORS.

COLOR(arg)=value	Sets the color of screen attributes.
x=COLOR(arg)	Returns the color of screen attributes.

arg		value		value	
0	COLOR	0	transparent	8	medium red
1	HCOLOR	1	black	9	light red
2	border	2	medium green	10	dark yellow
3	NORMAL foreground	3	light green	11	light yellow
4	NORMAL background	4	dark blue	12	dark green
5	INVERSE foreground	5	light blue	13	magenta
6	INVERSE background	6	dark red	14	grey
7	GR, HGR, HGR2 graphics screen	7	cyan	15	white

COLOR(0)= is the same as COLOR=. COLOR(1)= is the same as HCOLOR=. COLOR= and HCOLOR= are retained for compatibility with existing programs. It is recommended that future programs use the COLOR(arg)= syntax.

The AppleSoft color codes have been scrapped. SB1.x uses the Texas Instruments (TI) color codes required internally by the VDP. PEEKing and POKEing to set the screen colors is *ABSOLUTELY CONDEMNED*; however, since in the past these were widely used, those addresses have been left intact (in a miracle of programming, I might add).

Unlike SB1.0 and SB2.0, screen color attributes are *NOT* reset by the mode commands (TEXT, TEXT31, TEXT40, TEXT80, GR, HGR, HGR2). Screen colors remain in effect until changed by the appropriate COLOR(arg)= command.

COLOR(2)=, COLOR(5)= and COLOR(6)= have no visible effect in TEXT40, due to hardware limitations of the VDP. They remain set, however, and will be displayed in TEXT31, GR, HGR and HGR2.

SERIAL I/O.

SERIAL port,baudrate,stats Initializes a serial port.

port	baudrate	stats
1	300	0 (E71)
2	1200	1 (N81)
68	2400	
76	4800	
84	9600	
92	19200	
94		

Ports 1 and 2 are Micro Innovations MIB2 serial ports. Ports 68, 76, 84 and 92 are Eve/Orphanware serial ports. Port 94 is the ADAMlink modem. If port 94 is selected, the only valid option for baudrate is 300 (since the ADAMlink modem can only operate at 300 baud). While at the hardware level, other combinations of parity, character length and stop bits besides E71 and N81 are possible, there is no practical use for them, so they are not supported directly. If desired, they may be programmed using the proper IN and OUT commands; however, SB1.x will be unaware of the settings.

Technical note: The MIB2 serial ports 1 and 2 are *NOT* the actual hardware ports used. The designations "1" and "2" are logical names, consistent with the nomenclature used in the PowerMate documentation. The SmartBASIC 1.x interpreter maps these logical names to the appropriate hardware ports. The numbers of the Orphanware serial ports and the ADAMlink modem port, however, are the hardware ports.

Additional note: The MIB2 serial ports are *NOT* identical. Port 1 is wired as DCE (Data Communications Equipment) while port 2 is wired as DTE (Data Terminal Equipment). In practical terms, this means that port 1 is intended for use with a modem, and port 2 with a terminal. If you wish to use port 2 with a modem, or port 1 with a terminal, you must put a null modem connector in between the serial cable 25-pin connector and the modem/terminal 25-pin (or possibly 9-pin) connector. Null modems and 25-to-9 pin adaptors are widely available, e.g., from Radio Shack, for about $5.00 each.

SER(arg)=value	Sets the port for either PR #3 (arg=0) or the TEXT80 serial terminal (arg=1).
x=SER(arg)	Returns the serial port used for either PR #3 (arg=0) or the TEXT80 serial terminal (arg=1).
x=SER(port,parameter)	Returns either the baud rate (parameter=0) or the initialization statistics (parameter=1) for port.

Valid ports are 1, 2, 68, 76, 84, 92 and 94. Stats are returned as 0 (E71) or 1 (N81). If the returned baud rate is zero, then the port has not been initialized by the SERIAL command.

TAPE AND DISK I/O.

BLREAD drive,block,address	Reads a 1024-byte block from the drive and stores it, beginning at address.
BLWRITE drive, block,address	Writes 1024 bytes to the specified block of the drive, beginning at address.

drive	name	EOS device #	default block range
1	TAPE 1	8	0-255
2	TAPE 2 (or HARD DRIVE)	24	0-255 (tape)
3	Coleco hard drive prototype	9	0-255
4	Coleco hard drive prototype	25	0-255
5	DISK 1	4	0-159
6	DISK 2	5	0-159
7	RAM disk	26	0-63

These are extremely powerful, yet extremely *DANGEROUS* commands. They obviate one of the more common types of machine-code routine for the ADAM, and make tape/disk editing, directory untrashing, etc. very easy. However, you must be sure that transfer buffer has been reserved with LOMEM or HIMEM. Otherwise, writing to the buffer will corrupt the interpreter, possibly invoking garbage instructions which will trash your tapes or disks! *YOU HAVE BEEN WARNED*

DSIZE(arg)=x	Sets drive size parameters. For arg=0, x sets the number of blocks to be allocated to the directory of a tape/disk during INIT. The default value is 1. For arg=1,2,3,4,5,6, or 7, sets the volume size in blocks for drive arg during INIT to x. x may range from 1-65535, and is *ALWAYS* 1-based: 256 for tapes, 160 for single-sided disks, etc. These values are also used in parameter checking for BLREAD and BLWRITE, but in a 0-based form (since the block numbering is 0-based).
x=DSIZE(arg)	Returns drive size parameters. For arg=0, returns the number of directory blocks for INIT. For arg=1,2,3,4,5,6, or 7, returns the volume size in blocks for drive arg.

DSIZE allows you to implement the many third-party mass storage media which have been developed for the ADAM: 320/360/720K disk drives, hard drives, small-size tapes formatted using the MegaCopy device, and RAM disks using expansion memory.

RAM disks are tricky. The number of blocks available depends both upon the amount of expansion RAM you have, and the kind of proprietary software drivers you are using to make the RAM disk. In general, the more expansion memory, the less of it you are able to use as a RAM disk (1K blocks in each 64K bank must be reserved for inter-bank transfers; these cannot be allocated for files). You should consult your RAM disk documentation to see how many blocks are actually available. As a concrete example, one version of the Walters Software RAMdisk uses physical blocks 0-63 for the 64K memory expander, but the RAMdisk is only 63 blocks long, because block 0 is used as an inter-bank buffer. BLREADing block 0 returns the same block as BLREADing block 1. This means that, for INIT purposes, DSIZE(7) must be 63; but for BLREAD/BLWRITE, DSIZE(7) must be 64 (otherwise attempts to access block 63 will result in an "Illegal Quantity Error"). The default for DSIZE(7) is 64, for the 64K memory expander.

Hard drive users wishing to INIT a particular EOS volume may do so by setting DSIZE(2) to the appropriate size (1024, 2048 or whatever), then using INIT name,D2,Vn where Vn is the number of the desired volume. It does not matter that the various partitions may be of different sizes; as long as DSIZE(2) has the correct volume size, the INIT will be successful.

DSIZE parameters stay the same until you change them! So if you set DSIZE(0)=2, every drive you INIT will allocate 2 blocks to the directory.

FORMAT drive Formats the disk in drive and INITs it to the size currently given by DSIZE(drive).

The only valid values for drive are 5 (disk 1, D5) and 6 (disk 2, D6). FORMAT will *NOT* format tape drives, hard drives, or RAM disks. The default volume label "SB1.x" is used for the INIT; if you wish to change the label, or if you find that DSIZE(drive) had the wrong value (e.g., you wanted to format a 160K disk in a 320K drive, but forgot to set DSIZE(5) to 160, so you got a 320K disk instead), you can fix the DSIZE and then use the standard INIT command; the low-level formatting will still be okay.

FORMAT invokes the built-in formatting function stored in the disk drive controller EPROM. This causes the drive to low-level format the disk using all read/write heads available, regardless of the current value of DSIZE(drive). If you make a 160k disk (single-sided) in a 320K drive (double-sided), the disk is actually being formatted on both sides as a 320K. The INIT function of SmartBASIC 1.x is what makes it a 160K disk. Consequently, it is impossible to format flippy disks (2x160K) in a double-sided 320K drive: as soon as you flip the disk over, the FORMAT attempt destroys the data on the other side.

MERGE Modifies the LOAD command so that subsequent programs are combined with the program already in memory. In the case of identical line numbers, the LOADed program line replaces the line in memory.

NOMERGE Restores the default LOAD command, so that the program in memory is completely erased before the new program is LOADed.

MERGE remains in effect until cancelled by NOMERGE, and vice versa. To avoid problems, the following sequence is recommended:

```
     LOAD 1st.prog      ;source program
     MERGE              ;enable MERGE
     LOAD 2nd.prog      ;overlay 2nd program
     NOMERGE            ;restore original LOAD
```

TEXT MODES.

TEXT31	Selects 31-column video text mode (the default).
TEXT40	Selects 40-column video text mode.
TEXT80	Selects 80-column serial terminal text mode.
TEXT	Selects either 31-, 40- or 80-column text mode, depending upon whether TEXT31, TEXT40 or TEXT80 was the last text mode command.

For ease in adapting existing programs to SB1.x, TEXT31/40/80 needs only to be given once. Subsequent TEXT commands will invoke the last width set; graphics commands do not affect the text width. So you don't have to hunt down and change every TEXT in the program. Put the desired TEXTxx at the beginning and every subsequent TEXT will be just like the TEXTxx.

FLASH and INVERSE work in TEXT40, but border and separate inverse colors cannot be displayed due to hardware limitations of the VDP.

TEXT80 uses the serial port given by SER(1), which must have been previously initialized to proper baud rate, etc. with the SERIAL command. If the specified serial port does not exist, or has not been initialized, TEXT80 returns to TEXT40 and reports "I/O Error." If the specified serial port is valid, then the TV screen blanks and goes into TEXT31 mode, and all screen printing is directed to the terminal. The particular terminal emulation used is specified by TERM. TEXT80 is intended for use with the ADAM keyboard *ONLY*. The extent to which FLASH and INVERSE are implemented in TEXT80 depends upon the capabilities of the specific terminal emulation.

You may move freely between all text and graphics modes, in any order. Regardless of whether TEXT80 is the active TEXT mode or not, all text displays in GR and HGR are in the 4-line window at the bottom of the graphics screen. For example, going to GR from TEXT80 causes the blank TV screen (TEXT80 mode) to become GR, and no printing or typing echoes to the serial terminal. (If you use a composite color/monochrome monitor for both TV and serial terminal outputs, you might find a video-source switch very handy.)

In order for TEXT80 to function correctly, you must configure your terminal (either by DIP switches or software commands) to the following:

1. Proper baud rate, parity, and stop bits (to match parameters in CONFIG.SYS).
2. No line feeds after carriage returns.
3. No forced carriage return/line feed after printing in column 80.

Double-spaced screen lines are a good indication that settings for (2) and (3) are incorrect. The editing keys will *NOT* function properly if the display screen does not match the internal screen character map. For the Orphanware 80CVU, set DIP switch 2, #8 to ON (no LF after CR) and DIP switch 3, #8 to ON (wraparound mode disabled). You will have to reset the unit for these switches to take effect.

WINDOW is not available in TEXT80.

TERM=x Specifies the terminal emulation to be used in TEXT80. Currently supported values are:

0	Dumb TTY	4	ADDS Regent 200
1	Heath 19	5	Falco (may work for TeleVideo)
2	ADM-3A	6	-unassigned-
3	Beehive	7	External (user-supplied)

x=TERM Returns the current TEXT80 serial terminal emulation.

TTY and Heath 19 emulations are stored internally in the SmartBASIC 1.x interpreter; they are always available simply setting TERM=0 or TERM=1. An internal data area is reserved for *ONE* additional emulation (one of 2-7), which must be specified in CONFIG.SYS (by using the EDIT.SYS program) and installed at startup. Values 2-7 for TERM all point to this internal data area. If an extra emulation is installed, BOOT.SYS will correctly set TERM to the proper value (e.g., 2 for ADM-3A), but if you change TERM=3 (Beehive), you still have the ADM-3A emulation. If no extra emulation is installed, TERM values 2-7 default to Dumb TTY.

TTY is *VERY* dumb; the only control codes supported are carriage return and line feed. The remaining emulations are all "smart"--full arrow key editing, INSERT, DELETE, ^L (clear screen), control-arrow keys, HOME, ^V (erase to end of line) and ^X (cancel line or erase to bottom of screen) are supported. Heath 19, ADM-3A, Beehive, and Falco support INVERSE video; ADDS Regent 200 does not. Although none of the terminals listed support FLASHing video, I have substituted the control codes for INVERSE video to work for FLASH as well, so that FLASH will at least give an inverse video display.

The External emulation is a user-created, BLOADable data file containing the control character information for an additional terminal emulation. For detailed information about the creating additional terminal emulations, see Appendix 5: External Terminal Emulations.

NOTE: ASCII characters below 32 and above 127 are handled by the emulator routine. If the character is not one of the "emulatables", it is changed to a space (32) when it is printed on the terminal. This is for place-holding purposes when editing a program which has embedded Coleco graphics characters (like the sadface for parser error lines). If you pass the cursor over one of these "spaces", however, a space is substituted for the original control character in the program. This is not a problem for fixing parser errors (the sadface is ignored anyway when you edit the line), but for programs which have explicit ^Ds inside quote marks, it may cause some difficulties. LISTing, LOADing, MERGEing, and SAVEing a program with internal control characters is NOT AFFECTED by TEXT80. Only if you edit one such line. Similarly, control characters typed in at the keyboard while in TEXT80 are translated to spaces on the screen (unlike TEXT31 or TEXT40, where you can see them as graphics characters).

TEXT WINDOWS.**

WINDOW y1,x1 TO y2,x2   Defines a scrolling text window in TEXT31 and TEXT40 *ONLY*. y1,x1 are the line and column of the upper left corner of the window. y2,x2 are the line and column of the lower right corner of the window.

WINDOW is *NOT* available in TEXT80.

It is possible to LOCATE/HTAB/VTAB outside the boundaries of the window in order to PRINT or PUT. Any characters displayed outside the window will not be scrolled or erased by CLS/HOME/^L. The window remains in effect until a screen mode command is issued.

If your program stops with the cursor outside the window, you may notice a few strange things on the screen. You may move the cursor anywhere you like with the arrow keys, but once you get back inside the window, you cannot use the arrows to move out of it again. TEXT or any other screen mode command will reset it. I'm *PRETTY* sure you can't hang the system by playing around outside the window, but...

The safest practice is, when PRINTing outside the window, to always PRINT with the semicolon to suppress the automatic carriage return. Then make sure you LOCATE/HTAB/VTAB back inside the window. Try to avoid carriage returns outside the window.

APPENDIX 1: WRITING RELOCATABLE Z80 ASSEMBLY ROUTINES.

A primary goal of SmartBASIC 1.x has been to free the programmer from the need to know the absolute addresses of interpreter-specific parameters, such as the current LOMEM and HIMEM, the location of the default shape table, etc., and instead to allow the programmer to make use of this information through high-level commands. For example, it is more universal to use x=SPEED than x=PEEK(16129), especially since 16129 works only in SmartBASIC 1.0--in SmartBASIC 2.0 it is 1628.

User-supplied assembly language routines, however, present a problem. While the MEM functions can be used to make the *START* of a code block relocatable (e.g., LOMEM:MEM(0)+100 to reserve 100 bytes, then CALL MEM(0) to access it), and relative jumps (e.g., JR +25) and indexing commands (e.g., LD (IX+5),A) can be employed, most routines still have absolute addresses in them. This is unavoidable because the Z80 (unlike, for instance, the 8086 family) has no way to specify a CALL or LD address relative to the current instruction. Z80 code is thus not completely relocatable (able to be loaded anywhere in RAM and still execute correctly).

Z80 routines supplied as part of a SmartBASIC program, however, can be "helped" so that they are *IN EFFECT* relocatable. To show how this may be accomplished, consider the following assembly program:

```
;Sample Program:  Deletes a file, then prints a message on the screen
;
DELETE     EQU     64737     ;EOS delete file
PRINT      EQU     12110     ;SmartBASIC 1.0 print table
DISK1      EQU     4         ;device number of disk 1
;
START:     LD A,DISK1        ;device number                   62,4
           LD HL,FILE        ;filename string address         33,xxx,xxx
           CALL DELETE       ;delete the file                 205,225,252
           JR NZ,ERROR       ;failed                          32,7
           LD HL,STR1        ;ok; get message string address  33,xxx,xxx
EXIT:      CALL PRINT        ;print the message               205,78,47
           RET               ;back to SmartBASIC              201
;
ERROR:     LD HL,STR2        ;error message string address    33,xxx,xxx
           JR EXIT           ;print it and exit               24,247
;
FILE:      DB     "filename99A",3
STR1:      DB     14,"File Deleted",13,10
STR2:      DB     21,"Error Deleting File",13,10
```

The values of DELETE, PRINT, FILE, STR1, and STR2 are absolute; the values of EXIT and ERROR are relative. Furthermore, DELETE and PRINT must be the same value wherever the program is loaded, while FILE, STR1, and STR2 will change depending upon the load address. Note that, wherever the program is loaded, the offset of FILE, STR1, and STR2 relative to the start of the routine *IS AN ABSOLUTE NUMBER*:

```
FILE:     offset 22
STR1:     offset 34
STR2:     offset 49
```

This provides the key to making the routine fully relocatable. First, reserve space for the routine:

```
100 LOMEM:MEM(0)+71: start=MEM(0)
```

The value of MEM(0) will be the start address for the routine. Now POKE the machine code and data into RAM:

```
110 FOR n=start to start+70:  READ x: POKE n,x: NEXT
```

Wherever there occur absolute addresses which need to be made relocatable, replace these with zeros in the DATA statements (just as place holders):

```
120 DATA 62,4,33,0,0,205,225,252,32,7,33,0,0,205,78,47,201,33,0,0,24,247
..........rest of data omitted
```

Now you must calculate the "missing" addresses. Referring to the assembled machine code, note that:

label	value	occurrence
FILE	start+22	4th-5th bytes
STR1	start+34	12th-13th bytes
STR2	start+49	19th-20th bytes

Finally, translate this into SmartBASIC statements:

```
150 file=start+22
160 str1=start+34
170 str2=start+49
180 POKE start+4,MOD(file,256): POKE start+5,INT(file/256)
190 POKE start+12,MOD(str1,256): POKE start+13,INT(str1/256)
200 POKE start+19,MOD(str2,256): POKE start+20,INT(str2/256)
```

To access the routine, simply CALL start.

Essentially, this entire procedure is a mini linker. It duplicates the conversion of object code to the final executable memory image. As long as it is done accurately, the resulting machine code should run correctly regardless of what the current LOMEM is. Of course, LOMEM should not be changed while the routine is in use. (Remember that LOMEM erases all current variables.)

The example provided is somewhat frivolous, since CALL EOS(59) after appropriate setup of AF and HL could be used to bypass assembly language altogether. However, it clearly demonstrates the method. It is *STRONGLY RECOMMENDED* that *ALL* user-supplied assembly language routines in

SmartBASIC 1.x be written to be relocatable, using the memory usage parameters returned by the MEM function as base addresses. *UNDER NO CIRCUMSTANCES SHOULD ABSOLUTE ADDRESSES BE USED IN LOMEM AND HIMEM COMMANDS*.

APPENDIX 2: EOS-5 FUNCTIONS 0-100.

Function names are taken (with modification) from "The Hacker's Guide to ADAM Volume I" by Peter and Ben Hinkle (1986). Function descriptions are the result of my own disassembly of EOS-5 and EOS-7, and are reproduced from "The EOS-5 Technical Reference" by Richard F. Drushel (1991). For additional information and abbreviations, see Appendix 3: EOS-5 Technical Notes.

number	description	jump table address
0	EOS START/INITIALIZATION	64560
1	CONSOLE DISPLAY OF NON-CONTROL CHAR (in A)	64563
2	CONSOLE INITIALIZATION	64566
3	DISPLAY CHAR (in A) ON SCREEN (CTRL OR NOT)	64569
4	DELAY AFTER HARD RESET	64572
5	END PRINT BUFFER (at HL)	64575
6	END PRINT CHARACTER (in A)	64578
7	END READ 1 BLOCK	64581
8	END READ CHARACTER DEVICE (in A)	64584
9	END READ KEYBOARD	64587
10	END WRITE 1 BLOCK	64590
11	END WRITE CHARACTER DEVICE (in A)	64593
12	FIND DCB	64596
13	GET DCB ADDRESS (in IY)	64599
14	GET PCB ADDRESS (in IY)	64602
15	HARD INITIALIZATION (COLD BOOT)	64605
16	HARD RESET ADAMnet	64608
17	PRINT BUFFER (at HL)	64611
18	PRINT CHARACTER (in A)	64614
19	READ 1 BLOCK	64617
20	READ KEYBOARD	64620
21	READ KEYBOARD STATUS BYTE	64623
22	READ PRINTER STATUS BYTE	64626
23	READ DEVICE (in A) STATUS BYTE	64629
24	READ TAPE STATUS BYTE	64632
25	RELOCATE PCB (to HL)	64635
26	REQUEST DEVICE (in A) STATUS	64638
27	REQUEST KEYBOARD STATUS	64641
28	REQUEST PRINTER STATUS	64644
29	REQUEST TAPE STATUS	64647
30	SCAN ADAMnet FOR DEVICES	64650
31	SOFT INITIALIZATION (WARM BOOT)	64653
32	SOFT RESET DEVICE (in A)	64656
33	SOFT RESET KEYBOARD	64659
34	SOFT RESET PRINTER	64662
35	SOFT RESET TAPE	64665
36	START PRINT BUFFER (at HL)	64668
37	START PRINT CHARACTER (in A)	64671
38	START READ 1 BLOCK	64674
39	START READ CHARACTER DEVICE (in A)	64677
40	START READ KEYBOARD	64680

41	START WRITE 1 BLOCK	64683
42	START WRITE CHARACTER DEVICE (in A)	64686
43	SYNCHRONIZE Z80A AND MASTER 6801 CLOCKS	64689
44	WRITE 1 BLOCK	64692
45	WRITE CHARACTER DEVICE (in A)	64695
46	INITIALIZE FILE MANAGER	64698
47	INITIALIZE DIRECTORY	64701
48	OPEN FILE	64704
49	CLOSE FILE	64707
50	RESET FILE	64710
51	CREATE FILE	64713
52	FIND FILE (WITH TYPE)	64716
53	UPDATE DIRECTORY ENTRY	64719
54	READ FILE	64722
55	WRITE FILE	64725
56	SET CURRENT DATE	64728
57	GET CURRENT DATE	64731
58	RENAME FILE	64734
59	DELETE FILE	64737
60	READ DEVICE (in A) NODE TYPE	64740
61	GO TO SmartWriter	64743
62	READ EOS [not implemented in EOS-5]	64746
63	TRIM FILE	64749
64	CHECK IF FILE IS OPEN	64752
65	READ BLOCK	64755
66	WRITE BLOCK	64758
67	CHECK FILE I/O MODE	64761
68	READ DIRECTORY FOR FILE	64764
69	FIND FILE (NO TYPE)	64767
70	POSITION FILE [not implemented in EOS-5]	64770
71	EOS1 [not implemented in EOS-5]	64773
72	EOS2 [not implemented in EOS-5]	64776
73	EOS3 [not implemented in EOS-5]	64779
74	INCORRECT EOS VERSION ERROR	64782
75	GET I/O PORTS FROM OS-7	64785
76	BANK SWITCH MEMORY (to A)	64788
77	PUT ASCII CHARACTER PATTERN TO VDP	64791
78	WRITE VRAM	64794
79	READ VRAM	64797
80	WRITE VDP REGISTER 0-7	64800
81	READ VDP REGISTER 8	64803
82	FILL VRAM WITH 1 CHARACTER (in A)	64806
83	INITIALIZE VRAM TABLE	64809
84	PUT TABLE TO VRAM	64812
85	GET TABLE FROM VRAM	64815
86	CALCULATE OFFSET INTO SPRITE ATTRIB TABLE	64818
87	POINT TO PATTERN POSITION	64821
88	LOAD ASCII CHARACTER SET FROM ROM TO VDP	64824
89	WRITE VRAM SPRITE ATTRIBUTE TABLE	64827
90	READ GAME CONTROLLERS	64830
91	UPDATE SPINNER 1 AND 2	64833

92	DECREMENT LOW NIBBLE OF (HL)	64836
93	DECREMENT HIGH NIBBLE OF (HL)	64839
94	HIGH NIBBLE OF (HL) TO LOW NIBBLE	64842
95	ADD A TO WORD AT HL	64845
96	SOUND INITIALIZATION	64848
97	SOUND OFF	64851
98	START SONG	64854
99	SOUND	64857
100	END SPECIAL EFFECTS NOTE	64860

****
EOS Function 0: EOS START/INITIALIZATION.

This routine is jumped to by a powerup boot program located in page 0 of the SmartWriter ROM. (When the Z80A CPU is reset, the program counter is forced to 0000, and execution begins there. ADAM is hardwired so that this accesses the SmartWriter ROM.) On entry, 8K of EOS have already been copied from the EOS ROM to the upper 32K of RAM (starting at 57344). The EOS ROM is still switched in as the lower 32K. Function 0 sets up the EOS stack, moves up the EOS data tables by 1 byte (why I don't know), sets the revision number byte to 5, gets the I/O ports from OS-7, turns off the sound, fills all 16K of VRAM with zeroes, then bank switches the lower 32K to RAM. After further setup, it tries to load a bootstrap program from either disk (1, then 2) or tape (1, then 2). If a boot is successfully loaded, the routine jumps to it at 51200; otherwise, the routine jumps to Function 61 (go to SmartWriter). Note that tape 1 and tape 2 share the same DCB, and the node type byte of the tape DCB shows whether tape 2 is available (hex 03) or not (hex 33).

****
EOS Function 1: CONSOLE DISPLAY OF NON-CONTROL CHARACTER (in A).

On entry, A=character to display. Performs line wraparound and screen scroll up if necessary. Note: This routine is not used by SmartBASIC, though its own routine is almost identical.

****
EOS Function 2: CONSOLE INITIALIZATION.

On entry, B=number of columns (X) for screen, C=number of lines (Y), D=column of upper left corner (X-min), E=line of upper left column (Y-min), HL=VRAM address of name table. Note: This routine is not used by SmartBASIC, though its own routine is almost identical.

****
EOS Function 3: DISPLAY CHARACTER OR CONTROL CHARACTER ON SCREEN.

On entry, character to display is in A. If the character is ^\ (28), D=column, E=line to quickmove the cursor to. Note: This routine is not used by SmartBASIC, though its own routine is almost identical. The control character handling routines, however, are different.

****
EOS Function 4: DELAY AFTER HARD RESET.

This do-nothing loop seem unnecessarily complicated, but its complexity is probably due to its history. The necessary delay time was probably determined empirically, and this programming structure allows loops of varying lengths to be constructed simply by changing B and DE.

****
EOS Function 5: END PRINT BUFFER (at HL).

On exit, if end was successful, ZF=1. If not, ZF=0 and CF reflects various error conditions. CF=0 if not done printing, CF=1 if done but I/O error (code in A).

****
EOS Function 6: END PRINT CHARACTER (in A).

On exit, if end was successful, ZF=1. If not, ZF=0 and CF reflects various error conditions. CF=0 if not done printing, CF=1 if done but I/O error (code in A).

****
EOS Function 7: END READ 1 BLOCK.

On entry, A=device number. On exit, CF=1 if the I/O attempt has ended, with ZF=1 for ended successfully, ZF=0 for I/O error. If I/O is still in progress, CF=0. Error codes in A may be 1=NON-EXISTENT DEVICE, 3= I/O NOT DONE. (Exit from bit 7 clear test has A=0, which usually means OK.)

****
EOS Function 8: END READ CHARACTER DEVICE (in A).

On entry, A=device number. On exit, CF=1 if the I/O attempt has ended, with ZF=1 for ended successfully, ZF=0 for I/O error. If I/O is still in progress, CF=0. Error codes in A may be 1=NON-EXISTENT DEVICE, 3= I/O NOT DONE. (Exit from bit 7 clear test has A=0, which usually means OK.)

****
EOS Function 9: END READ KEYBOARD.

On exit, if end was successful, ZF=1 and A=character typed. Otherwise, ZF=0 and CF reflects various error conditions. CF=0 if not done reading, CF=1 if done but I/O error (code in A).

****
EOS Function 10: END WRITE 1 BLOCK.

On entry, A=device number. On exit, CF=1 if the I/O attempt has ended, with ZF=1 if successful, ZF=0 for I/O error. If I/O is still in progress, CF=0. Error codes in A may be 1=NON-EXISTENT DEVICE, 3=I/O NOT DONE. (Exit from bit 7 clear test has A=0, which usually means OK.)

****
EOS Function 11: END WRITE CHARACTER DEVICE (in A).

On entry, A=device number. On exit, CF=1 if the I/O attempt has ended, with ZF=1 for ended successfully, ZF=0 for I/O error. If I/O is still in progress, CF=0. Error codes in A may be 1=NON-EXISTENT DEVICE, 3= I/O NOT DONE. (Exit from bit 7 clear test has A=0, which usually means OK.)

****
EOS Functions 12/13: FIND/GET DCB ADDRESS (in IY).

On entry, A=device number. On exit, if the device exists, ZF=1, A=device number and IY=DCB address. Otherwise, ZF=0 and A=1 (NON-EXISTENT DEVICE error).

****
EOS Function 14: GET PCB ADDRESS (in IY).

On exit, IY=current PCB address.

****
EOS Function 15: HARD INITIALIZATION (COLD BOOT).
Current PCB address is set to 65216. After a hard reset of ADAMnet, the old PCB and DCBs are wiped out by being moved up 1 byte from 65216-65535 to 65217-65536. The Z80A and master 6801 clocks are synchronized, and ADAMnet is scanned for devices, creating new DCBs as active devices are found.
****
EOS Function 16: HARD RESET ADAMnet.
Sends 15 out the reset port (63), waits a bit, then sends 0.
****
EOS Function 17: PRINT BUFFER (at HL).
On entry, HL=address of a print buffer terminated by hex 03. The logical buffer may be of any length, but only 16 characters at a time may be printed (physical length). On exit, if printing ended successfully, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.
****
EOS Function 18: PRINT CHARACTER (in A).
On entry, A=character to send to the line printer. On exit, if the print was successful, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.
****
EOS Function 19: READ 1 BLOCK.
On entry, A=device number, BCDE=block number to read (BC=hiword, DE= loword), HL=data transfer address (DTA). On exit, ZF=1 if read was successful, ZF=0 if not.
****
EOS Function 20: READ KEYBOARD.
On exit, if read was successful, ZF=1 and A=character typed. If not, ZF=0 and A=error code.
****
EOS Function 21: READ KEYBOARD STATUS BYTE.
****
EOS Function 22: READ PRINTER STATUS BYTE.
****
EOS Function 23: READ DEVICE (in A) STATUS BYTE.
On entry, A=device number. On exit, if the device does not exist, ZF=0. Otherwise, ZF=1 and A=status byte from DCB.
****
EOS Function 24: READ TAPE STATUS BYTE.
****
EOS Function 25: RELOCATE PCB (to HL).
On entry, HL=new address of PCB.
****
EOS Function 26: REQUEST DEVICE (in A) STATUS.
On entry, A=device number. On exit, if the device exists, A=status returned by ADAMnet, with ZF=1 if A=128, ZF=0 otherwise. If the device does not exist, ZF=0 and A=1 (NON-EXISTENT DEVICE error).
****
EOS Function 27: REQUEST KEYBOARD STATUS.
On exit, A=status. ZF=1 if A=128, otherwise ZF=0.

****
EOS Function 28: REQUEST PRINTER STATUS.

On exit, A=status. ZF=1 if A=128, otherwise ZF=0.

****
EOS Function 29: REQUEST TAPE STATUS.

On exit, A=status. ZF=1 if A=128, otherwise ZF=0.

****
EOS Function 30: SCAN ADAMnet FOR DEVICES.

All DCBs are shifted up by 1 byte in memory (presumably inactivating them). The device count in the PCB is zeroed. ADAMnet is scanned for devices 1-15. If the device is active (status=128), a new 21-byte DCB is allocated, and the PCB device count is incremented. Otherwise, scanning continues. On exit, PCB byte 3 has the number of active devices, and the DCBs follow consecutively. Note: tape 1 (device 8) and tape 2 (device 24) share the same DCB.

****
EOS Function 31: SOFT INITIALIZATION (WARM BOOT).

On entry, HL=new PCB address. This routine is like Function 15 (cold boot) with two exceptions. The new PCB address is supplied on entry, not automatically set to 65216. It also moves 318 bytes worth of PCB and DCBs, not 319. The significance of this latter difference is not clear.

****
EOS Function 32: SOFT RESET DEVICE (in A).

On entry, A=device number. On exit, A=128 and ZF=1 if device is reset and ready for use. ZF=0 if device doesn't exist or is busy.

****
EOS Function 33: SOFT RESET KEYBOARD.

****
EOS Function 34: SOFT RESET PRINTER.

****
EOS Function 35: SOFT RESET TAPE.

****
EOS Function 36: START PRINT BUFFER (at HL).

On entry, HL=address of a print buffer terminated by hex 03. The logical buffer may be of any length, but only 16 characters at a time may be printed (physical length). If start was OK, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.

****
EOS Function 37: START PRINT CHARACTER (in A).

On entry, A=character to print. On exit, if start was successful, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.

****
EOS Function 38: START READ 1 BLOCK.

On entry, A=device number, BCDE=block number to read (BC=hiword, DE= loword), HL=data transfer address (DTA). On exit, ZF=1 if start was successful, ZF=0 and A=error code (1=NON-EXISTENT DEVICE, 2=DEVICE NOT READY) if not.

****
EOS Function 39: START READ CHARACTER DEVICE (in A).

On entry, A=device number, DE=buffer start address, BC=buffer length.

On exit, ZF=1 and A=0 if start was OK. Otherwise, ZF=0 and A=error code (1,2).

****

EOS Function 40: START READ KEYBOARD.

On exit, if start was successful, ZF=1 and A=1 (device number). Otherwise, ZF=0 and A=error code (1=NON-EXISTENT DEVICE, 2=DEVICE NOT READY).

****

EOS Function 41: START WRITE 1 BLOCK.

On entry, A=device number, BCDE=block number to write (BC=hiword, DE=loword), HL=data transfer address (DTA). On exit, ZF=1 if start was successful, ZF=0 and A=error code if not.

****

EOS Function 42: START WRITE CHARACTER DEVICE (in A).

On entry, A=device number, HL=buffer start address, BC=buffer length. On exit, ZF=1 and A=0 if start was OK. Otherwise, ZF=0 and A=error code (1,2).

****

EOS Function 43: SYNCHRONIZE Z80A and MASTER 6801 CLOCKS.

On exit, ZF=1 and A=0 if synch was OK. If synch failed, ZF=0 and A=18 (Z80A NOT SYNCHRONIZED) or A=19 (MASTER 6801 NOT SYNCHRONIZED).

****

EOS Function 44: WRITE 1 BLOCK.

On entry, A=device number, BCDE=block number to write (BC=hiword, DE=loword), HL=data transfer address (DTA). On exit, ZF=1 if write was successful, ZF=0 and A=error code if not.

****

EOS Function 45: WRITE CHARACTER DEVICE (in A).

On entry, A=device number, HL=buffer start address, BC=buffer length. On exit, ZF=1 and A=0 if write was successful, ZF=0 and A=error code (1,2,3) if not.

****

EOS Function 46: INITIALIZE FILE MANAGER.

On entry, DE=address of DTA0, HL=address of FCB0. On exit, the I/O mode byte (24) of FCB0, FCB1 and FCB2 is set to zero.

****

EOS Function 47: INITIALIZE DIRECTORY.

On entry, A=device number, C=number of directory blocks to initialize, DE=length of volume in blocks, HL=address of new volume name string. If the name is longer than 12 characters, it is truncated. On exit, if the initialization was successful, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.

****

EOS Function 48: OPEN FILE.

On entry, A=device number, HL=address of filename string, B=I/O mode. I/O mode decodes as follows: 1=read, 2=write, 3=random (read/write), 4=execute. The file must have already been created with Fn51 (create file). On exit, if the open was successful, ZF=1 and A=file number. If the file was not opened for write alone, the DTA contains the first block of the file. If the opened file is only 1 block long, bit 7 of the FCB I/O mode byte (24) is set. If the open was unsuccessful, ZF=0 and A=error code.

****
EOS Function 49: CLOSE FILE.

On entry, A=file number. On exit, if the close was successful, ZF=1 and A=0. If the file was opened for writing, the contents of the DTA are written to the file, and the directory entry updated. If the close was unsuccessful, ZF=0 and A=error code.

****
EOS Function 50: RESET FILE.

On entry, A=file number. On exit, if reset was successful, ZF=1 and A=0. The current DTA is written to the file (if opened for write), and then the first block of the file is read into the DTA (if not opened for write alone). If reset failed, ZF=0 and A=error code.

****
EOS Function 51: CREATE FILE.

On entry, A=device number, BCDE=length of file in bytes (BC=hiword, DE= loword), HL=address of filename string. If BCDE=0, the file will not attempt to reuse deleted file space, thus allowing for maximum file size. On exit, if create was successful, an entry for the file is added to the directory, "BLOCKS LEFT" is updated, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.

****
EOS Function 52: FIND FILE (WITH TYPE).

On entry, A=device number, DE=address of filename string (10 characters max for name, then filetype byte <A,a,H,h>, then hex 03), HL=address of 23-byte buffer to contain the directory entry (no date bytes). The routine reads the directory and looks for the first match to the file name string. Unlike Fn69, both the 10-character filenames and the file type bytes must match. On exit, if a match was found, the buffer at HL contains the directory entry, BCDE=start block of file, and ZF=1. Otherwise, ZF=0 and A=error code.

****
EOS Function 53: UPDATE DIRECTORY ENTRY.

On entry, HL=address of 23-byte buffer containing a directory entry (no date bytes), DE=address of filename string, and FCB0 is set up to read the directory block(s). On exit, if the file already exists, the entry is updated, A=0 and ZF=1. Otherwise, ZF=0 and A=error code.

****
EOS Function 54: READ FILE.

On entry, A=file number, BC=number of bytes to read from the file, HL= address of read buffer to receive the data (not the same as file manager DTA). The file must already have been opened by Fn48 (open file). On exit, if the read was successful, ZF=1, A=0, BC=same as entry, and FCB bytes 33-34 point to the end of the read buffer. Otherwise, ZF=0, A= error code. If A=9 (BAD FILE NUMBER) or A=10 (INPUT PAST END), BC=number of bytes actually read from the file; for other errors, BC is unknown.

****
EOS Function 55: WRITE FILE.

On entry, A=file number, BC=number of bytes to write to the file, HL= address of write buffer to send the data (not the same as file manager DTA). The file must already have been opened by Fn48 (open file). On exit, if the write was successful, ZF=1 and A=0. Otherwise, ZF=0 and

A=error code.
****
EOS Function 56: SET CURRENT DATE.
On entry, B=current day, C=current month, D=current year.
****
EOS Function 57: GET CURRENT DATE.
On exit, B=current day, C=current month, D=current year. If no date has been set (0/0/0), ZF=0 and A=4 (NO DATE SET error). Otherwise, ZF=1 and A=0.
****
EOS Function 58: RENAME FILE.
On entry, A=device number, DE=address of old filename string, HL=address of new filename string. On exit, if the rename was successful, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.
****
EOS Function 59: DELETE FILE.
On entry, A=device number, HL=address of filename string for file to delete. On exit, if the delete was successful, ZF=1 and A=0. Otherwise, ZF=0 and A=error code.
****
EOS Function 60: READ DEVICE (in A) NODE TYPE.
On entry, A=device number. On exit, if the device exists, ZF=1 and A= node type byte. If the device doesn't exist, ZF=0 and A=1 (NON-EXISTENT DEVICE error).

The node type byte contains ADAMnet status information for each device. Unfortunately, there are 2 physical devices mapped to each node, with the high nibble and low nibble of the node type byte showing the status of the 2 devices, respectively. Devices which share DCBs also share node type bytes, hence tape 1 and tape 2 are shared, but disk 1 and disk 2 are separate. For tape 1, disk 1 and disk 2, the low nibble contains the status information; for tape 2, the high nibble. Returned values of the nibbles decode as follows:

0	No Error (everything is OK)
1	CRC Error (block corrupt; data failed cyclic redundancy check)
2	Missing Block (attempt to access past physical end of medium)
3	Missing Media (not in drive or drive door open)
4	Missing Drive (not connected or not turned on)
5	Write-Protected (write-protect tab covered)
6	Drive Error (controller or seek failure)

Why this is called the "node type" byte is anybody's guess. The name comes from "The Hacker's Guide to ADAM Vol.I."
****
EOS Function 61: GO TO SmartWriter.
Bank switches to the SmartWriter ROM, then jumps to the first byte of code at address 256.
****
EOS Function 62: READ EOS.
Not implemented in EOS-5; it is only a RET. Presumably this routine was intended to read in a fresh copy of the current operating system from the

EOS ROM (which can hold up to 4 different 8K EOS versions). A user program not needing EOS routines could use the upper 8K of RAM for itself, then restore EOS when done. Perhaps.

****

EOS Function 63: TRIM FILE.

On entry, A=device number, DE=address of filename to trim. On exit, any excess blocks allocated to the file (but not actually used by it) are deallocated. If the next directory entry is BLOCKS LEFT, the free blocks are allocated for use by subsequent files. Otherwise, the space is wasted.

****

EOS Function 64: CHECK IF FILE IS OPEN.

On entry, HL=address of file name string. On exit, if the file is open, ZF=1, A=0 and B=lower 3 bits of the I/O mode byte from the FCB (read, write, execute). Otherwise, ZF=0 and A=5 (FILE NOT OPEN error).

****

EOS Function 65: READ BLOCK.

On entry, A=device number, BCDE=block to read (BC=hiword, DE=loword), HL= data transfer address (DTA). On exit, if the read was successful, ZF=1. If not, ZF=0 and A=22 (I/O ERROR) or other error code. Oddly, this routine reads the block, checks the device status, then rereads the block. This cannot be for data integrity, as no verify operation is performed on the data (e.g., load same block into 2 places and compare them). The extra reading time is not noticeable from disk drives, but probably is significant for the tape drives. Why the routine does this I don't know. By comparison, Fn66 (write block) writes the block once, followed by a status check.

****

EOS Function 66: WRITE BLOCK.

On entry, A=device number, BCDE=block to write (BC=hiword, DE=loword), HL=data transfer address (DTA). On exit, if the write was successful, ZF=1. If not, ZF=0 and A=22 (I/O ERROR). Unlike Fn65 (read block), this routine only writes the block once.

****

EOS Function 67: CHECK FILE I/O MODE.

On entry, IX=address of FCB, HL=address of directory entry. On exit, IX and HL have their entry values. This routine determines if the attribute of the file will permit the I/O type requested. If the mode check was OK, ZF=1 and A=0. Otherwise, ZF=0 and A=error code (17=BAD I/O MODE, 20=FILE ACCESS DENIED).

****

EOS Function 68: READ DIRECTORY FOR FILE.

On entry, A=device number, HL=address of filename string. On exit, if file was found, ZF=1, A=0, BCDE=start block of file, and FCB0 bytes 33-34 contain the address of the matching entry in DTA0. Otherwise, ZF=0 and A=error code.

****

EOS Function 69: FIND FILE (NO TYPE).

On entry, A=device number, DE=address of filename string (10 characters max for name, then filetype byte <A,a,H,h>, then hex 03), HL=address of 23-byte buffer to contain the directory entry (no date bytes). The routine reads the directory and looks for the first match to the file name string. The 10-character filenames must match, but the filetype bytes need not. On exit, if a match was found, the buffer at HL contains the directory entry, BCDE=start block of file, and ZF=1. Otherwise, ZF=0 and A=error code.

****

EOS Function 70: POSITION FILE.

Not implemented in EOS-5. Used in EOS-7 to move the read/write pointer in a random-access file. SmartBASIC 1.0 provides its own routine to do this; SmartBASIC 2.0 requires the EOS-7 routine.

****

EOS Function 71: EOS1.

Not implemented in EOS-5. In EOS-7, it is a consolidated master block I/O routine, part of the space-saving rewrite to add a third 1024-byte buffer.

****

EOS Function 72: EOS2.

Not implemented in EOS-5. In EOS-7, it is a block I/O subroutine.

****

EOS Function 73: EOS3.

Not implemented in either EOS-5 or EOS-7. What it was supposed to do is anybody's guess.

****

EOS Function 74: INCORRECT EOS VERSION ERROR.

EOS-5 leaves several of its functions unimplemented. This might not be true in some later, suped-up version of EOS. Programs written to utilize these extra functions would bomb if run under earlier, incompatible versions of EOS. Consequently, jump table entries for routines which never got off the drawing board in EOS-5 point here to return an error code (A=23). This allows the incompatible program to terminate nicely with an error message, rather than just locking up the system when the call to a non-existent routine sends the program counter off to never-never land. Under EOS-5, Fn70 (position file), Fn71 (EOS1), Fn72 (EOS2), Fn73 (EOS3) remain unimplemented, and are redirected here. On exit, ZF=0 and A=23 (INCORRECT EOS VERSION error).

****

EOS Function 75: GET I/O PORTS FROM OS-7.

The values are stored in RAM as follows:

64551	memory switch port	127
64552	ADAMnet reset port	63
64553	VDP control port	191
64554	VDP data port	190
64555	game controller 1 port	252
64556	game controller 2 port	255
64557	strobe set port	128
64558	strobe reset port	192
64559	sound port	255

****

EOS Function 76: BANK SWITCH MEMORY (to A).

On entry, A=memory configuration (0-15). Memory configurations decode as follows:

	lower 32K	upper 32K
0	SmartWriter or EOS	RAM
1	RAM	RAM
2	expansion RAM	RAM
3	OS-7 plus 24K RAM	RAM
4	SmartWriter or EOS	expansion ROM
5	RAM	expansion ROM
6	expansion RAM	expansion ROM
7	OS-7 plus 24K RAM	expansion ROM
8	SmartWriter or EOS	expansion RAM
9	RAM	expansion RAM
10	expansion RAM	expansion RAM
11	OS-7 plus 24K RAM	expansion RAM
12	SmartWriter or EOS	cartridge ROM
13	RAM	cartridge ROM
14	expansion RAM	cartridge ROM
15	OS-7 plus 24K RAM	cartridge ROM

In order to select the SmartWriter ROM, an OUT (63),0 must be executed first. To select the EOS ROM, use OUT (63),2.
****
EOS Function 77: PUT ASCII CHARACTER PATTERN TO VDP.

On entry, HL=code of first character pattern to load, BC=number of patterns, DE=VRAM address of pattern generator table.
****
EOS Function 78: WRITE VRAM.

On entry, DE=VRAM target address to write, HL=RAM source address of data, BC=number of bytes to write.
****
EOS Function 79: READ VRAM.

On entry, DE=VRAM address to read, HL=RAM target address to receive data, BC=number of bytes to read.
****
EOS Function 80: WRITE VDP REGISTER 0-7.

On entry, B=register number to write (0-7), C=data byte to send. If register written was 0 or 1, the data byte sent is stored in RAM at 64865 (0) or 64866 (1).
****
EOS Function 81: READ VDP REGISTER 8.

The value is stored at 64867.
****
EOS Function 82: FILL VRAM WITH 1 CHARACTER (in A).

On entry, A=character to fill, DE=number of times to fill, HL=VRAM address to write.
****
EOS Function 83: INITIALIZE VRAM TABLE.

On entry, A=code for which table to initialize: (0) sprite attribute table, (1) sprite generator table, (2) pattern name table, (3) pattern generator table, (4) color table. HL=VRAM address of table.
****
EOS Function 84: PUT TABLE TO VRAM.

On entry, A=table code (see Fn83), HL=table address in RAM, DE=entry number in table, IY=number of entries to be moved.

****

EOS Function 85: GET TABLE FROM VRAM.

On entry, A=table code (see Fn83), HL=table address in RAM, DE=entry number in table, IY=number of entries to be moved.

****

EOS Function 86: CALCULATE OFFSET INTO SPRITE ATTRIBUTE TABLE.

On entry, D=Y-coordinate of pattern position, E=X-coordinate. D and E are signed 8-bit numbers (-128 to +127). On exit, DE=(Y*32)+X.

****

EOS Function 87: POINT TO PATTERN POSITION.

On entry, DE=signed 16-bit number (X-coordinate or Y-coordinate of pattern). On exit, DE ranges from -128 to +127.

****

EOS Function 88: LOAD ASCII CHARACTER SET FROM ROM TO VDP.

Retrieves the bit patterns for ASCII characters 0-127 stored in the SmartWriter ROM.

****

EOS Function 89: WRITE VRAM SPRITE ATTRIBUTE TABLE.

On entry, A=number of sprites to write, HL=address of sprite order table in RAM, DE=address of RAM copy of sprite attribute table.

****

EOS Function 90: READ GAME CONTROLLERS.

On entry, IX=address of 10-byte RAM table to hold controller data (joy-stick, left button, right button, decoded keypad, spinner for player 2, followed by player 1), A=weird code for which controller(s) to read:

BITS 1,0:	00	none
	01	controller 2
	10	controller 1
	11	controller 2, then controller 1
BIT 7:	1	add old EOS spinner value to old in RAM data table
	0	don't update RAM spinner

NOTE: This routine must be called TWICE in succession with the same controller code in order to update the RAM table ONCE.

****

EOS Function 91: UPDATE SPINNER 1 AND 2.

Reads the game controllers, incrementing or decrementing the spinner counters in the EOS game controller data table, depending on which way the spinner was spun. Presumably this would be called by an interrupt routine.

****

EOS Function 92: DECREMENT LOW NIBBLE OF (HL).

On exit, the low nibble of (HL) is decremented, A=new value of low nibble, and ZF=1 if it is now zero.

****

EOS Function 93: DECREMENT HIGH NIBBLE OF (HL).

On exit, the high nibble of (HL) is decremented, A=new value of high nibble, and ZF=1 if it is now zero. This function is not used anywhere in EOS-5.

****

EOS Function 94: HIGH NIBBLE OF (HL) TO LOW NIBBLE.

On exit, the high nibble of (HL) is moved to the low nibble, with the original high nibble unchanged.

****

EOS Function 95: ADD A TO WORD AT HL.

On entry, A=data to add, (HL)=lobyte of word, (HL+1)=hibyte of word. On exit, A is added to word, and HL still points to the lobyte.

****

EOS Function 96: SOUND INITIALIZATION.

On entry, B=number of voices to initialize (1-4), HL=address of song table with the following format:

```
   address of noise note table (lobyte, hibyte)
   address of noise output table (lobyte, hibyte)
   ...same for voices 1,2,3.
```

On exit, the current note of each voice in the output table and the saved sound control byte are set to 255, and the EOS voice table pointers are set to 58342 (address of another 255). The sound control byte is used to prevent identical data from being sent to the noise channel (which would cause an audible click and "ruin" white noise).

****

EOS Function 97: SOUND OFF.

On exit, the three voices and the noise channel are turned off.

****

EOS Function 98: START SONG.

On entry, B=number of the song to start. On exit, the first note of that song is ready to play.

****

EOS Function 99: SOUND.

On entry, all necessary noise and voice tables must have been properly set up and initialized. Steps through each table once, playing the current note, updating as necessary.

****

EOS Function 100: END SPECIAL EFFECTS NOTE.

On entry, IX=output table address, HL=address of next special effects note, DE=?.

****

APPENDIX 3: EOS-5 TECHNICAL NOTES.

The following information is provided for the benefit of programmers who wish to use CALL EOS for operating system functions. This information was compiled from "The Hacker's Guide to ADAM Vol. I" by Peter and Ben Hinkle (1986), and from "The EOS-5 Technical Reference" by Richard F. Drushel (1991). See Appendix 2: EOS-5 Function Calls 0-100 for a summary of the EOS-5 operating system functions.

PROCESSOR CONTROL BLOCK (PCB).**

The PCB is a 4-byte block, usually at address 65216 but relocatable, used by the master 6801 ADAMnet controller to keep track of devices.

offset	meaning
0	processor status/request byte
1-2	PCB start address (lobyte, hibyte)
3	number of active devices (number of DCBs)

Reading byte 0 returns status information from ADAMnet; the meaning of individual status bits is uncertain. Writing to byte 0 requests the following operations:

data	function
1	synchronize the Z80 clock
2	synchronize the master 6801 clock
3	relocate PCB

DEVICE CONTROL BLOCKS (DCBs).**

DCBs are 21-byte blocks, beginning at address 65220, used by EOS to communicate with ADAMnet devices. Fifteen DCBs are allocated; the number of active DCBs, however, depends upon how many valid devices were attached to ADAMnet at system startup. Active devices are allocated DCBs in order of primary device number, 1 through 15. (Device numbers may be greater than 15, however. See tape DCB discussion.)

offset	meaning
0	device status/request byte
1-2	buffer start address (lobyte, hibyte)
3-4	buffer length (lobyte, hibyte)
5-8	block number accessed (loword, hiword in lobyte, hibyte format)
9	high nibble of device number
10-15	always zero (unknown purpose)
16	device number
17-18	maximum block length
19	device type (0 for character device, 1 for block device)
20	node type (returns ADAMnet drive error codes--see Function 60)

Reading byte 0 returns status information from ADAMnet; the meaning of individual status bits is uncertain. Writing to byte 0 requests the following operations:

data	function
1	return current status
2	soft reset
3	write
4	read

ADAMnet device numbers decode as follows:

number	device	type
0	master 6801 ADAMnet controller	-
1	keyboard	0
2	ADAM printer	0
4	disk drive 1	1
5	disk drive 2	1
8	tape drive 1	1
9	ADAM hard drive EOS partition volume 1	1
13	ADAM parallel interface	0
14	ADAM serial interface	0
15	--unknown--	0
24	tape drive 2	1
25	ADAM hard drive EOS partition volume 2	1
26	expansion RAM disk drive	-

Device Notes:

0	The master 6801 uses the PCB as its DCB.
8,24	Tape 1 and tape 2 share the same DCB.
9,25	According to Ron Collins, two volumes of the 2.5 megabyte EOS partition of a planned 5 megabyte ADAM hard disk drive, never built. These correspond to D3 and D4 in SmartBASIC. Like tape 1 and tape 2, they would have shared the same DCB.
13	The prototype ADAM parallel interface, never released. SmartBASIC 2.0 has routines to access it as PR #4.
14	The prototype ADAM serial interface, never released. SmartBASIC 2.0 has routines to access it as IN #2 and PR #2.
15	This always appears as the last DCB, and always with a not-ready status. What it is for is anybody's guess.
26	Allocated as an expansion RAM disk drive, but never used by Coleco. Third-party developers (e.g., Walters Software) have implemented it, using modified EOS drivers, but *NOT* as a true ADAMnet device.

EOS DIRECTORY STRUCTURE.**

Each entry consists of 26 bytes. Directories begin at block 1, and may be up to 255 blocks long. Entries for files have the following format:

bytes 0-11	Filename specification. A valid filename consists of up to 10 characters, followed by a type byte, terminated with hex 03. The type byte can be A, a, H, or h (ASCII or hex, lowercase for backups). Executable binary files are usually type hex 02, which appears as a smiley face.
12	File attribute. Set bits in the attribute byte map as follows: 0 not a file (used for BLOCKS LEFT) 1 execute protected (can't be opened for execution) 2 deleted file 3 system file (hidden from SmartBASIC CATALOG) 4 user file 5 read-protected 6 write-protected (read only) 7 delete protected
13-16	Start block. Stored loword, hiword.
17-18	Allocated length. Includes any unused "holes" which may be tacked on at the end.
19-20	Used length. Actual length of file, discounting unused "holes" at the end.
21-22	Number of bytes used in last allocated block of file. Thus, to compute file length in bytes, (allocated-used-1)*1024+lastbyte.
23	Creation year. There is no set rule for how to represent the year. One common method is (year-1900); thus 1987 would be hex 57. This wastes the values hex 00-52 (since ADAM appeared in 1983), unless these are interpreted as 2000+value (as in SB1.x).
24	Creation month. Range hex 01-0C.
25	Creation day. Range hex 01-1F.

Four special entries are found in the directory of every EOS disk. The first three are always:

VOLUME.	bytes 0-11	volume name
	12	attribute=hex 80 (delete protected)
	13-16	hex 55AA00FF directory check for EOS format
	17-18	disk size in blocks
BOOT.	bytes 0-11	BOOT
	12	attribute=hex 88 (delete protected, system file)
	17-18	allocated length=1 block
	19-20	used length=1 block
	21-22	lastbyte=0 (means 1024, i.e., the whole block)
DIRECTORY.	bytes 0-11	DIRECTORY
	12	attribute=hex C8 (delete and write protected, system file)
	13-16	start block (default=1)
	17-18	maximum size of directory in blocks

19-20	current size of directory in blocks

The last entry of every directory in EOS-5 is:

BLOCKS LEFT.	bytes 0-11	BLOCKS LEFT
	12	attribute=hex 01 (not a file)
	13-16	first free block in largest contiguous cluster of free blocks at the end of the storage medium
	17-18	total number of free blocks (contiguous or not)
	19-20	used length always 0
	23-25	EOS version date=hex 570711 This date does not conform to the pattern described above. In "The Hacker's Guide To ADAM Vol. II", Ben Hinkle suggests that this should be read as 7/11/1957, perhaps the birthdate of one of the programmers.

Note: EOS-7 does not use BLOCKS LEFT to keep track of free blocks.

EOS FILE CONTROL BLOCK (FCB) STRUCTURE.*

EOS sets up areas of RAM to store data about open files, called file control blocks (FCBs). EOS-5 has 3 FCBs: 1 for the system, and 2 for user files (3 in EOS-7). FCBs are contiguous 35-byte blocks set up as follows:

bytes 0-22	from EOS directory entry (no date bytes)
23	I/O mode byte. The set bits are mapped as follows: 7 current block in buffer is the last block of the file 6 data in buffer is waiting to be written to the file. Makes EOS write the data before loading a new block into the buffer. 5 file won't reuse deleted file space 4 unused 3 unused 2 open for execute 1 open for write 0 open for read Bits 5,2,1,0 are set by the caller; bits 7,6 are set and used internally by EOS. Files can be opened for read, write, read-write or execute; an error results if read, write and execute are simultaneously set.
25-28	current block for I/O. Loword, hiword.
29-32	last block of file. Loword, hiword.
33-34	address of I/O buffer (DTA). This area is also used in EOS-7 by Fn70 (POSITION FILE) and Fn50 (RESET FILE) to store the byte offset into the current DTA.

EOS ERROR CODES.**

These are returned in A with ZF=0 upon return after an error. There is no official Coleco nomenclature for these error codes, since there was never any software access to EOS. I have selected likely names based upon what seems to be going wrong when they are issued. In some cases, I have provided alternate names which may be more meaningful or explanatory.

code	meaning
0	[No Error]
1	Non-Existent Device
2	Device Not Ready
3	I/O Not Done (or I/O Not Over)
4	No Date Set
5	No More Directory (or File Not Found) [Fn68]
	File Not Open [Fn64]

Due to a seeming typographical error by a programmer, two different errors are allocated to code 5, while code 15 is unused, not even mentioned. SmartBASIC 1.0 interprets EOS error 5 as "File Not Found", so this suggests that "File Not Open" should have been assigned to error 15.

6	File Already Exists
7	Too Many Open Files [SmartBASIC calls this "No Buffers Available"]
8	Match Not Found
9	Bad File Number
10	I/O Past End
11	File Too Big
12	Directory Full
13	No More Room (or Disk Full)

SmartBASIC does not differentiate between errors 12 and 13, lumping them together under "No More Room".

14	Bad File Name
15	[unused; see error 5]
16	File Locked (or File Write Protected)
17	Bad I/O Mode
18	Z80A Not Synchronized
19	Master 6801 Not Synchronized
20	File Access Denied
21	[unknown, unimplemented]
22	I/O Error
23	Incorrect EOS Version
24	Non-EOS Volume

Appendix 4: External Terminal Emulations.

If you have a serial terminal not currently supported under SmartBASIC 1.x, you can create an emulation data file specific for your terminal, and install it at startup as an External Emulation. The internal format for all SmartBASIC 1.x terminal emulation data tables is as follows:

```
byte 0:     xx     offset to 12          (clear screen and home cursor)
            xx     offset to 22          (erase to right of screen)
            xx     offset to 24          (erase to bottom of screen)
            xx     offset to 28          (absolute cursor move x,y)
            xx     offset to 128         (home cursor)
            xx     offset to 151         (delete character to right of cursor)
            xx     offset to INVERSE ON
            xx     offset to INVERSE OFF
            xx     offset to FLASH ON
            xx     offset to FLASH OFF
byte 10:    --     data for 12
            --     data for 22
            --     data for 24
            --     data for 28
            --     data for 128
            --     data for 151
            --     data for INVERSE ON
            --     data for INVERSE OFF
            --     data for FLASH ON
            --     data for FLASH OFF
```

Offset is the number of bytes from the *BEGINNING OF THE TABLE* to the data for the particular control character or function. Data is in the format number of data bytes, data. If the terminal is incapable of a given control character function, set the length byte to 0 and do not supply any data bytes.

The location of each terminal emulation data table in RAM is stored in a 16-byte table of 2-byte addresses (lobyte, hibyte). The default table is pointed to by ADDR(6). The table currently in use is pointed to by ADDR(7), and may be changed by changing ADDR(7) to point at a user-supplied table anywhere in memory. At startup, however, both ADDR(6) and ADDR(7) point to the same default table. The value of TERM*2 is the offset into this table, pointing at the address of the data for the TERMth emulation:

```
             lobyte    hibyte
ADDR(7):       xx        xx       ;address of TERM=0 emulation
               xx        xx       ;1
               xx        xx       ;2
               xx        xx       ;3
               xx        xx       ;4
               xx        xx       ;5
               xx        xx       ;6
               xx        xx       ;7
```

In the default table, there are only three unique addresses. The first entry points to the internal Dumb TTY emulation data, the second to the internal Heath 19, and the third through eighth to a reserved area where emulations 2-7 are loaded by BOOT.SYS at startup, *IF* TERM=2, 3, 4, 5, 6, or 7 has been specified in CONFIG.SYS. If not, this area contains a duplicate of the Dumb TTY emulation data. This is done to conserve space.

In a user-supplied table, ADDR(7) not equal to ADDR(6), all eight entries may have unique addresses, and point to eight different emulation data tables. The programmer must supply all the necessary data, and insure that the pointer table entry addresses are correct.

To create an External Emulation installable by BOOT.SYS at startup, first create the data table in the proper format. Two examples given below are for SmartBASIC 1.x's internal Dumb TTY and Heath 19 emulations:

	Dumb TTY		CHR$(n)	Heath 19	
byte 0:	10		12	10	;offsets
	10		22	13	
	10		24	16	
	10		28	19	
	10		128	22	
	10		151	25	
	10		INVERSE ON	28	
	10		INVERSE OFF	31	
	10		FLASH ON	34	
	10		FLASH OFF	37	
byte 10:	0	;none		2,27,69	;data
				2,27,75	
				2,27,74	
				2,27,89	;x,y supplied by SB1.x
				2,27,72	
				2,27,78	
				2,27,112	
				2,27,113	
				2,27,112	;no FLASH, but uses
				2,27,113	;INVERSE ON and OFF

Next, find the address of the emulation data pointer table. For the default table, the value of ADDR(6) has the address. Since emulations 2-7 in the default setup all have the same pointer address, you need only look at the third entry (instead of the eighth, TERM=7 is eighth entry). Get this pointer address, and *THAT* is where you can POKE the emulation data table into memory. Without recourse to the actual, version-dependent addresses, here is how you find where to start POKEing:

```
   100 base=ADDR(6): lobyte=PEEK(base+4): hibyte=PEEK(base+5)
   110 address=lobyte+256*hibyte:  IF address<0 THEN address=address+65536
```

Once the data is in memory, simply BSAVE it, beginning at address, with length equal to how many bytes are in the table:

```
   500 HTAB 1: PRINT CHR$(4);"BSAVE ";name$;",A";STR$(address);",L";
       STR$(length)
```

Of course, you should test your emulation data before saving it. Set TERM=7 and go to TEXT80 and make sure that the arrow keys, etc., all work as they are supposed to.

Next, RUN EDIT.SYS and select External Emulation from the proper menu. At the prompt, enter the name of the emulation data file you BSAVEd. After saving the new configuration, reboot SmartBASIC 1.x. TEXT80 should now correctly access the external emulation.

NOTE: Only terminals which have Heath 19-type absolute cursor positioning can be used successfully with SmartBASIC 1.x. The Heath-type coding is:

```
<ESCAPE> <SOME CHARACTER SEQUENCE> <line+31> <column+31>
```

There can be as many control characters as necessary to specify the absolute cursor positioning mode, but the actual positioning must use the +31 format.

APPENDIX 5: LIST OF RESERVED WORDS.

The following is a list of reserved words in SmartBASIC 1.x. These may not be used as regular variable names, although there is no prohibition upon a variable name containing a reserved word as *PART* of the name. For example, window=5 is an illegal statement (WINDOW is a reserved word), whereas window9=5 is acceptable.

Note: the words RENUM, SOUND, SPDEFINE, SPDRAW and SPXDRAW are reserved for possible use in future versions of SmartBASIC 1.x.

&	DE	INT	POP	SPDRAW
?	DEF	INVERSE	POS	SPEED
ABS	DEL	IX	POSITION	SPXDRAW
ADDR	DELETE	IY	PR	SQR
AF	DIM	LEFT	PRINT	STEP
AND	DRAW	LEN	PRN	STOP
APPEND	DSIZE	LET	PROMPT	STORE
ASC	END	LINPUT	PRWIDTH	STR
AT	EOS	LIST	PUT	STRING
ATN	ERRNUM	LOAD	RANDOMIZE	TAB
BC	EXP	LOCATE	READ	TAN
BEEP	FLASH	LOCK	RECALL	TERM
BIT	FN	LOG	RECOVER	TEXT
BLOAD	FOR	LOMEM	REM	TEXT31
BLREAD	FORMAT	MEM	RENAME	TEXT40
BLWRITE	FP	MERGE	RENUM	TEXT80
BREAK	FRE	MID	RES	THEN
BRUN	GET	MOD	RESTORE	TIME
BSAVE	GOSUB	MON	RESUME	TO
CALL	GOTO	NEW	RETURN	TRACE
CATALOG	GR	NEXT	RIGHT	UNLOCK
CF	HCOLOR	NOBREAK	RND	USR
CHR	HDRV	NOMERGE	ROT	VAL
CLEAR	HEX	NOMON	RUN	VER
CLK	HGR	NORMAL	SAVE	VLIN
CLOSE	HGR2	NOT	SCALE	VPOS
CLREAD	HIMEM	NOTRACE	SCRN	VTAB
CLRERR	HL	ON	SER	WAIT
CLS	HLIN	ONERR	SERIAL	WEND
CLWRITE	HOME	OPEN	SET	WHILE
COLOR	HPLOT	OR	SGN	WIDTH
CONT	HTAB	OUT	SHLOAD	WINDOW
COS	IF	PDL	SIN	WRITE
CPL	IN	PEEK	SOUND	XDRAW
DATA	INIT	PLOT	SPC	XOR
DATE	INPUT	POKE	SPDEFINE	ZF

Appendix 6: Memory Maps.

On the following pages are detailed memory maps for the Coleco ADAM computer running the SmartBASIC 1.x interpreter under the EOS-5 operating system.

The first map is of the 64K RAM directly addressable by the Z80 micro-processor. Where possible, segment boundaries are given in terms of global symbols rather than absolute addresses. In cases where both an absolute and a symbolic address are given, it is imperative that the programmer employ the symbolic reference, to maintain portability between, and compatibility with, current and future versions of the interpreter and/or EOS. While published accounts of the SmartBASIC 1.0 memory map give the absolute addresses of many pointers to interpreter parameters (such as the location of the tokenized program, line number table, start of string space, etc.), programmers should *AVOID* using these addresses.

The remaining maps show VRAM use by SmartBASIC 1.x in the 6 possible video modes. Unlike the microprocessor memory maps, these maps show segment boundaries in page form, rather than byte address form. (One page equals 256 bytes, thus page 1 is address 256.) Byte offsets into a page are shown in the form +offset. Hence, 31+128 means page 31, byte offset 128. For detailed information about ADAM's video chip, see either "TMS9918A/28A/29A Video Display Processors Data Manual" or "Video Display Processors Programmer's Guide", both by Texas Instruments (1984).

64K Map Notes:

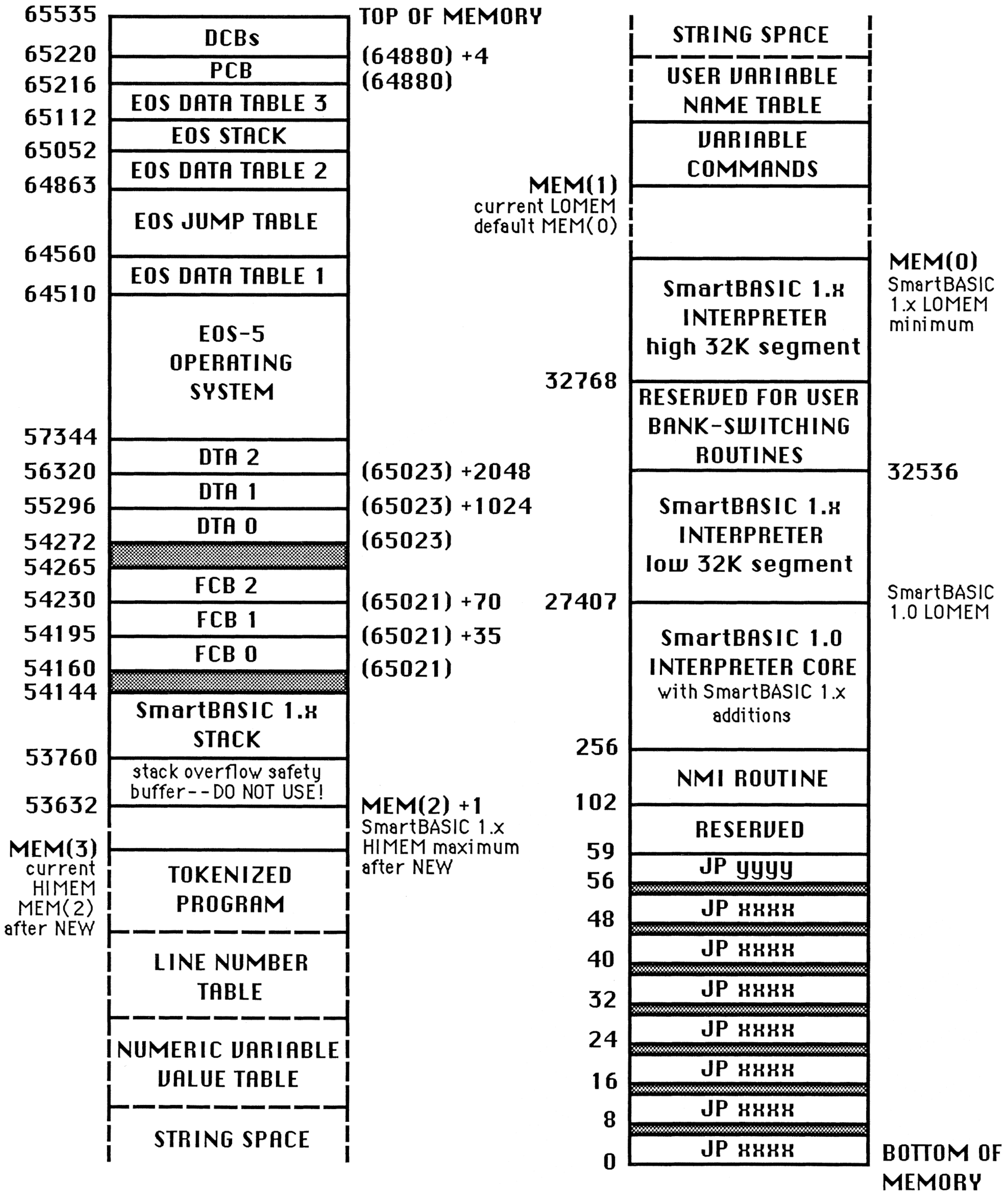
(1) The first 7 RST vectors contain 3-byte JP xxxx instructions, where xxxx is the address of a RET instruction.

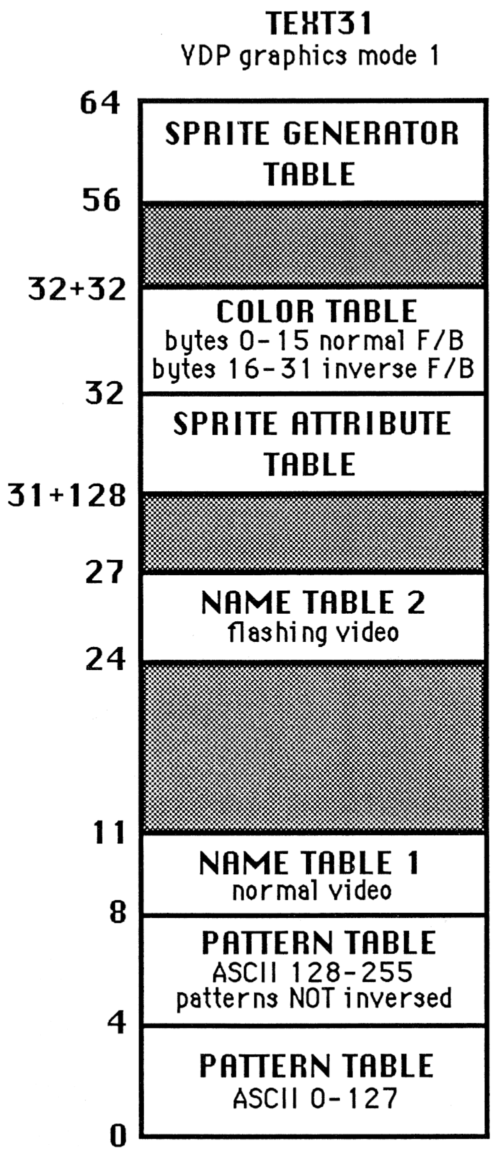
(2) The RST 38H vector contains a 3-byte JP yyyy instruction, where yyyy is the address of a RETI instruction.

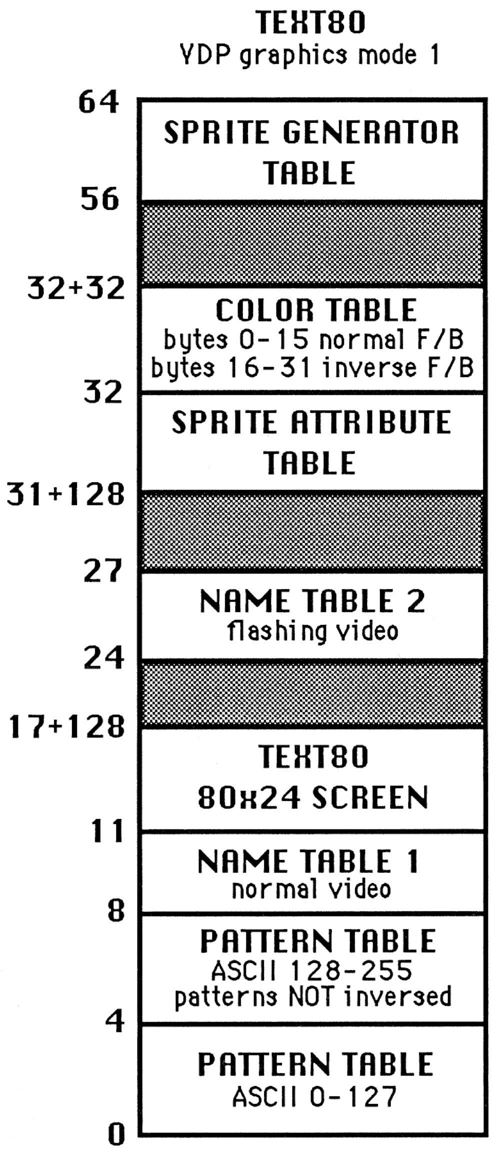
(3) Maskable interrupts are always *ENABLED* in SmartBASIC 1.x. During internal bank-switching routines (such as required for the enhanced PEEK and POKE commands), they are temporarily disabled (DI), but are always re-enabled (EI) when the normal RAM configuration is restored. Consequently, a single DI instruction in a user-supplied machine code routine can *NOT* guarantee that the maskable interrupts remain disabled. A hardware device requiring the maskable interrupt (such as the spinner on the Super-Action Game Controller, or the MIDI interface) should "enable/disable" the interrupt by changing the JP yyyy at address 57-58 to point to either the handler code or to a RETI instruction, respectively.

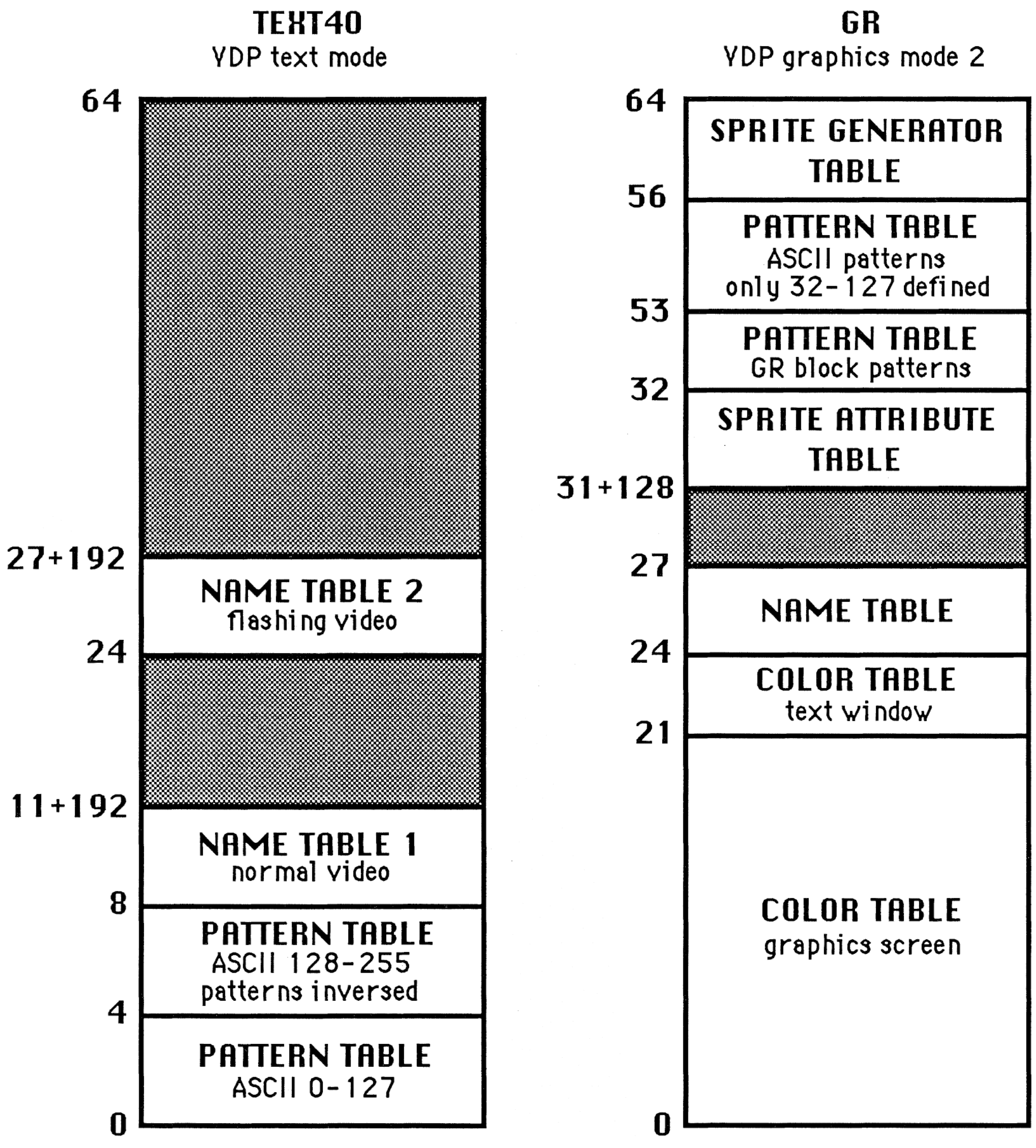
(4) User machine code routines may be placed in DTA2 (pointed to by PEEK(65023)+2048) as long as no more than *ONE* file is open at any one time. If the main program is large, this conserves program space for the SmartBASIC interpreter.

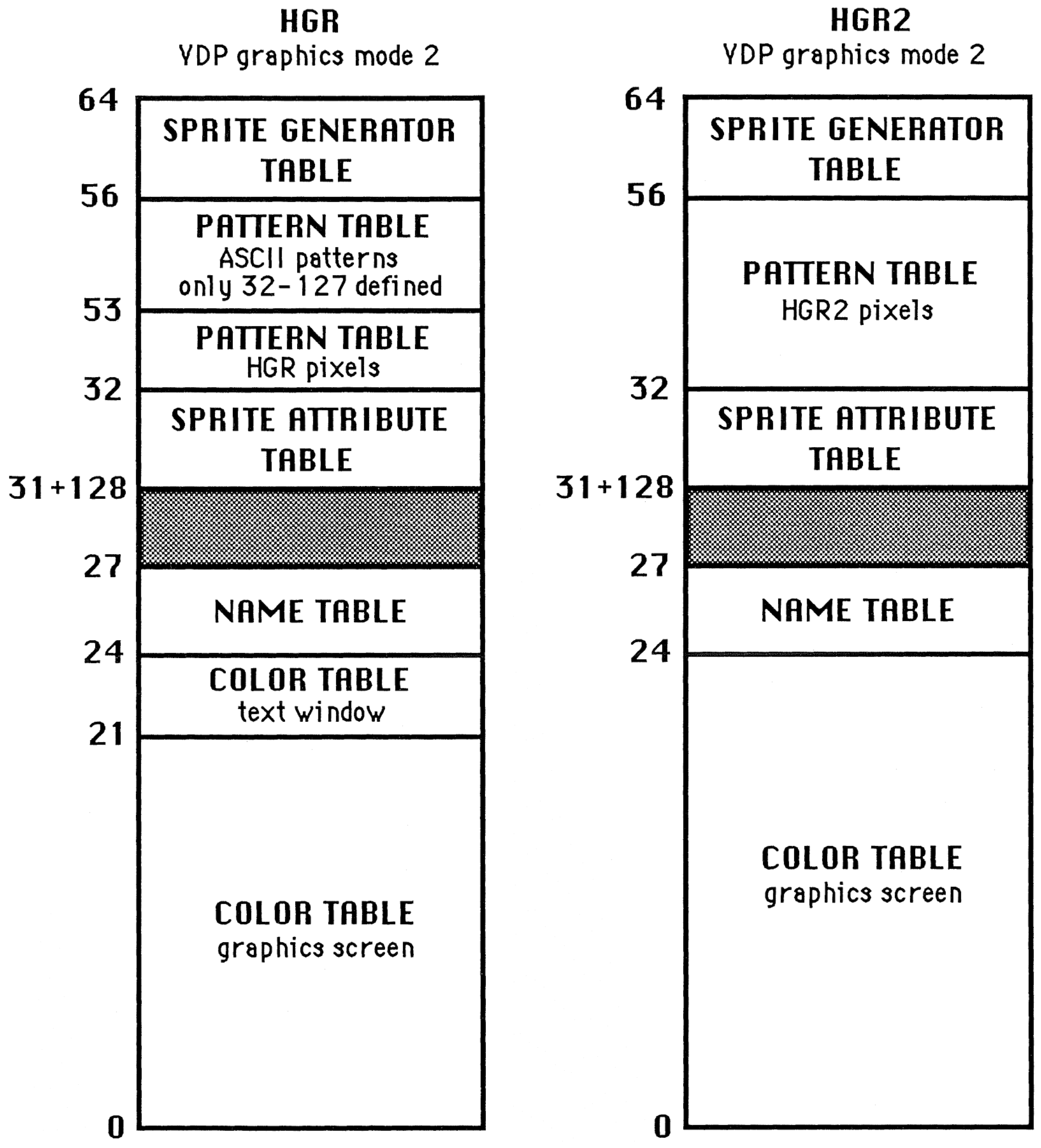
(5) A small area below address 32768 is reserved for user-supplied machine code routines which involve bank-switching out the upper 32K of RAM. This area is provided as a courtesy to programmers who, for performance reasons, choose not to use the enhanced PEEK command to access alternate upper 32K memory spaces. Sufficient space is allocated for the necessary code and for a buffer to hold retrieved data. Remember that switching out a bank of RAM makes all programs and data in that bank *INVISIBLE* to the Z80 until it is switched back in.


65535
TOP OF MEMORY
DCBs
65220
(64880) +4
PCB
65216
(64880)
EOS DATA TABLE 3
65112
EOS STACK
65052
EOS DATA TABLE 2
64863
EOS JUMP TABLE
64560
EOS DATA TABLE 1
64510
EOS-5 OPERATING SYSTEM
57344
DTA 2
56320
(65023) +2048
DTA 1
55296
(65023) +1024
DTA 0
54272
(65023)
54265
FCB 2
54230
(65021) +70
FCB 1
54195
(65021) +35
FCB 0
54160
(65021)
54144
SmartBASIC 1.x STACK
53760
stack overflow safety buffer--DO NOT USE!
53632
MEM(2) +1
SmartBASIC 1.x HIMEM maximum after NEW
MEM(3)
current HIMEM
MEM(2) after NEW
TOKENIZED PROGRAM
LINE NUMBER TABLE
NUMERIC VARIABLE VALUE TABLE
STRING SPACE
STRING SPACE
USER VARIABLE NAME TABLE
VARIABLE COMMANDS
MEM(1)
current LOMEM
default MEM(0)
MEM(0)
SmartBASIC 1.x LOMEM minimum
SmartBASIC 1.x INTERPRETER high 32K segment
32768
RESERVED FOR USER BANK-SWITCHING ROUTINES
32536
SmartBASIC 1.x INTERPRETER low 32K segment
27407
SmartBASIC 1.0 LOMEM
SmartBASIC 1.0 INTERPRETER CORE with SmartBASIC 1.x additions
256
NMI ROUTINE
102
RESERVED
59
JP yyyy
56
JP xxxx
48
JP xxxx
40
JP xxxx
32
JP xxxx
24
JP xxxx
16
JP xxxx
8
JP xxxx
0
BOTTOM OF MEMORY


TEXT31
VDP graphics mode 1
64
SPRITE GENERATOR TABLE
56
32+32
COLOR TABLE
bytes 0-15 normal F/B
bytes 16-31 inverse F/B
32
SPRITE ATTRIBUTE TABLE
31+128
27
NAME TABLE 2
flashing video
24
11
NAME TABLE 1
normal video
8
PATTERN TABLE
ASCII 128-255
patterns NOT inversed
4
PATTERN TABLE
ASCII 0-127
0


TEXT80
VDP graphics mode 1
64
SPRITE GENERATOR TABLE
56
32+32
COLOR TABLE
bytes 0-15 normal F/B
bytes 16-31 inverse F/B
32
SPRITE ATTRIBUTE TABLE
31+128
27
NAME TABLE 2
flashing video
24
17+128
TEXT80
80x24 SCREEN
11
NAME TABLE 1
normal video
8
PATTERN TABLE
ASCII 128-255
patterns NOT inversed
4
PATTERN TABLE
ASCII 0-127
0


TEXT40
VDP text mode
64
27+192
NAME TABLE 2
flashing video
24
11+192
NAME TABLE 1
normal video
8
PATTERN TABLE
ASCII 128-255
patterns inversed
4
PATTERN TABLE
ASCII 0-127
0
GR
VDP graphics mode 2
64
SPRITE GENERATOR TABLE
56
PATTERN TABLE
ASCII patterns
only 32-127 defined
53
PATTERN TABLE
GR block patterns
32
SPRITE ATTRIBUTE TABLE
31+128
27
NAME TABLE
24
COLOR TABLE
text window
21
COLOR TABLE
graphics screen
0


HGR
VDP graphics mode 2
64
SPRITE GENERATOR TABLE
56
PATTERN TABLE
ASCII patterns
only 32-127 defined
53
PATTERN TABLE
HGR pixels
32
SPRITE ATTRIBUTE TABLE
31+128
27
NAME TABLE
24
COLOR TABLE
text window
21
COLOR TABLE
graphics screen
0
HGR2
VDP graphics mode 2
64
SPRITE GENERATOR TABLE
56
PATTERN TABLE
HGR2 pixels
32
SPRITE ATTRIBUTE TABLE
31+128
27
NAME TABLE
24
COLOR TABLE
graphics screen
0

INDEX OF SmartBASIC 1.x COMMANDS AND FUNCTIONS.

COMMANDS.

FUNCTIONS.