USING SmartBASIC 2.0

by Wa6Msx

Basic 2.0 differs considerably from it's predecessors. Many if not all of Basic 1's faults have been identified and eliminated, and several new features have been added.

The aggravating "growing" DATA and REM LINE Bug has been fixed! And in fact old BASIC 1.0 files with sloppy DATA and REM lines can be instantly cleaned up by Booting Basic 2.0 and then LOADing and SAVEing the old file.

The utility of sequential Tape or Disc Files has been considerably improved by fixing the "Backup" Bug. A Backup Copy of saved files is NO LONGER MADE after APPENDing them. All that's necessary is that the APPENDED FILE be the LAST PROGRAM on the Tape or Disc. Then there will be no more rapidly filled up tapes with useless copies, and the SEQUENTIAL FILE feature of Basic can be used directly without resorting to Machine Language routines to overcome BASIC 1's Limitations. Also, DELETEing Programs in Basic 2.0 now actually FREES that space on the Tape or Disc for OTHER PROGRAMS. To be reliable though Programs should be DELETEd beginning with the LAST PROGRAM SAVED and then inward toward the FIRST Program. To DELETE all programs on a tape such as either BASIC tape that can't otherwise be INITiated, just be sure that the LAST PROGRAM DELETED is the FIRST PROGRAM on the Tape or Disc. That will result in a virtually INITiated Basic Tape or Disc with the Full Program Space Available for new programs.

The Line Buffer has been enlarged to now allow a line length up to 255 Characters for direct compatibility with MicroSoft Basic. Line Editing is easier too. The Insert and Delete Keys now work in Basic 2 as well as SmartWriter and the Clear Key Erases the Entire Line. There is another change that really IS an improvement, but takes some getting used to. When editing previously written lines if you should make an Error, the ERROR is ACCEPTED and the Original Line is LOST. This line is then printed on the screen with the "Sad Face" character proceeding it. (This Sad Face as well as the Syntax Error "^" and run time Error Messages are always accompanied by the Bell Sound). It must then be re-written. Sad Face lines will stay in the listing until changed, and may even be SAVEd with the program, but will Break the program when Run, if between sequential lines. This differs considerably from Basic 1.0 where any errors in editing were REJECTED and the original line was returned UNCHANGED. It's good insurance to copy lines before editing them. You'll then have an unchanged line to fall back on if you make an error and lose the line.

There is virtually NO imposed POKE LIMIT in Basic 2.0, you can poke up to FEBF (65215). POKEs are rejected above that limit only to prevent Poking into the DCB's which could be catastrophic. Basic 1.0 Programs with the old POKES TO 16149 and 16150, used to re-set it's Poke Limit SHOULD NOT BE USED with Basic 2.0. These pokes are in a different part of the Basic 2.0 machine language and will blow-up the Basic Boot and require re-booting. This damage may not be immediately apparent on all functions and I suspect this is one of the reasons for the reports that Basic 2.0 seems "incomplete".

This is true with the old Basic 1.0 COLOR POKES as well. DON'T use a program written for Basic 1.0 that changes the Screen Colors. These programs MUST be modified first.

The NEW COLOR POKES ARE:

POKE 17184 <0-15> for the Border
POKE 17240 <Normal Text *16 + Screen>
POKE 17251 <Inverse Text*16 + Inverse Background>

I have written a 6 Block program for the Public Domain called 1-2 Color that may be used unmodified with either Basic to set all the Basic Text-Screen Color Pokes. It identifies which Basic is booted and pokes the correct addresses.

The Basic 1.0 Graphic Color Pokes SHOULD NOT be used with Basic 2.0 as well. I have not identified the New Graphic Pokes yet (April 86). Perhaps someone else has this information.

A small but significant improvement (as it seems to point up the level of effort put into the revision) can be seen by running 1-2 Color in both Basics. You'll notice the cursor is visible in Basic 1.0, even tho not necessary since no normal keyboard inputs are required. In Basic 2.0 the cursor is transparent unless waiting for input. The program could have poked a transparent character for the cursor in Basic 1.0, but if the program was aborted with Control C, without restoring the cursor, it would remain invisible. All this foolishness is unnecessary with Basic 2.0. AND ONCE AGAIN ANY Program that POKES 16953 to Alter the Cursor in Basic 1.0 MUST NOT BE RUN Before Deleting These Pokes!

The Printer will now start to Print from the Top of the Screen by Pressing the PRINT key on the Keyboard. It can also be STOPPED EARLY by Pressing CONTROL and C together. The Printer now Prints any INVERSE CHARACTERS in the text that have normal characters on the Daisy Wheel, as NORMAL CHARACTERS. Also the Printer can be made to print bi-directional, as with Basic 1.0, with CHR$(15). This feature was useless however on Basic 1.0 since the printer would lock up and continue printing the same 80 characters forever. This bug has been fixed in Basic 2.0 and the Printer will stop and return control to the program. There is also a NEW SCREEN PRINTING FUNCTION that requires a POKE. To display ANY of the 256 characters including "control characters" on the screen in the TEXT Mode, at the current cursor position, POKE its ASCII number into 16771 then CALL 16770. The following Line will show them all.

```
5 TEXT: FOR i=0 TO 255: POKE 16771,i: CALL 16770: PRINT " ";:NEXT.
```

The Basic 2.0 Source File or Program on the Disc/Tape takes up 49 BLOCKS. SmartBASIC 1.0 only required 28 Blocks. ALL this extra code includes a revision of the EOS. This NEW EOS Revision-7 is Loaded into RAM instead of the ROM Revision-5 resident in ADAM that was used with Basic 1.0. This is where the New File Handling Routines come from. Fortunately for the Pokers amongst us the EOS JUMP TABLE is at the same address in RAM and even most of the new routines are simply displaced by a few blocks. Caution is in order however. It's wise to investigate the area of ANY POKE used in your favorite Basic 1.0 Program before attempting to RUN it!

It is also worth noting that even with all this extra source code, Basic 2.0 requires 447 bytes (1/2 Block) LESS memory so that you have additional room for Basic or Machine Code programs even when in STDMEM. Basic 2.0 PRINT FRE(0) yields 26401 as opposed to Basic 1.0's 25954. (Each following NEW). HIMEM default is the same at 53632 but the extra space is revealed by Basic 2.0's default LOMEM of 26960 as opposed to Basic 1.0's 27407. Note too that now BOTH HIMEM AND LOMEM can be set.

Even Block ZERO didn't escape revision. This is the "Boot-STRAP LOADER" and now includes a test for the presence of the 64K Expansion Board as well as a Memory Test of it. Also code to reset the Default Drive to be the Current Drive (the Drive that you're Booting Basic 2.0 from). This means that unspecified Reads or Writes will be to IT and that HELLO will Self-RUN as Expected.

Basic 2.0 Also correctly INITilizes Discs with 160 Blocks. And BLOCKS REMAINING is now re-calculated each time CATALOG is requested, from the total of Those Programs NOT DELETED.

NEW COMMANDS

MERGE This command is used like LOAD but does not NEW the program space, so that favorite routines may be stored separately and MERGEd with any existing program. The MERGEd lines will replace any existing lines numbered the same, or will simply be added to the program if numbered differently.

EXTMEM This command accesses the 64K expansion board for additional basic program space. NOTE that when EXTMEM is entered the BASIC 2.0 Tape or Disc MUST BE IN THE CURRENT DRIVE. The Tape or Disc is then Accessed ONLY if the 64K Expansion was tested as Present when Basic 2.0 was Booted. If this is so the Screen will Blank while the Memory Map is reconfigured, then the Title "Coleco SmartBASIC 2.0" will reappear.

PRINT FRE(0) will confirm that your new Expanded Workspace is 90646. (90656 if proceeded by NEW). HIMEM And LOMEM can now be set anywhere in this new workspace but be careful with the values. Reports are that there is no Error Trapping on LOMEM and HIMEM values while in EXTMEM AND AN ILLEGAL ENTRY WILL CRASH BASIC. Note too that one drawback that comes with all this extra room is slower program execution.

STDMEM This command returns you to the normal Basic Map. Once Again the BASIC 2.0 TAPE OR DISC MUST BE IN THE CURRENT DRIVE WHEN STDMEM IS INITIATED. The Tape/Disc is Accessed Again and the Basic Map is configured as "normal". Anything in the Workspace is LOST. PRINT FRE(0) reports 26391 (26401 if proceeded by NEW). (It might prove simpler just to ReBoot the tape or disc from the start)

COMMANDS PDL(1) and PDL(2) in BASIC 2.0 have been INTERCHANGED as referenced to BASIC 1.0. PDL(2) now yields the VERTICAL POSITION for Controller #1 (0-256) and PDL(1) now yields the HORIZONTAL POSITION for Controller #2. ALSO PDL(15) For the #1 controller Or PDL(14) For the #2 Controller now returns an Update on the Position of the SPINNER, used in the SUPER ACTION CONTROLLER.

BASIC 2.0 SPRITES

Sprites as implemented in Basic 2.0, are really SHAPES each consisting of 32 Bytes of Bit-Mapped Memory and can be put anywhere on the screen in ANY of the Display Modes and EACH in any of the 15 HCOLORS! EACH sprite is defined on its OWN SCREEN PLANE, easily visualized as 32 layers of transparencies. That's why each can have it's own color AND that they CAN overlap. The Basic 2.0 Source tape/disc comes with Sprite 1 and 2 already defined. Their 64 bytes are stored in page 0 at 00C0 (192) thru 00FF. Sprites that you define may be stored here or anywhere else in free memory space like any other machine code. Poke 16786 (Low Byte) and 16787 (Hi Byte) to tell the sprite routine where YOUR Sprite Table begins. Before you can use the Default Sprites or any others you must also poke 16788 with a 1. This sets the Sprite Flag and tells the routine that Sprites HAVE been defined.

In "normal" Graphics, Basic 2.0 XDRAWS with Transparent to assume the Background Color (instead of simply Black in Basic 1.0). The Sprite routine uses THIS XDRAW to AUTOMATICALLY Erase That particular Sprite if you Draw it somewhere else on the screen. Needless to say this makes for extremely simple animation routines. To see the two Sprites provided enter this line. 5 TEXT: POKE 16788,1: HCOLOR = 3: DRAW 1 AT 120,70: DRAW 2 AT 120,120. I'm sure they'll be a pleasant surprise.

For more surprises try the following short program.

```
4  REM BASIC 2.0 SPRITE DEMO
5  TEXT:POKE 16788, 1
10 VTAB 3:HTAB 5: PRINT "Remember Text Can ALSO"
15 HTAB 8: PRINT "Be On The Screen"
```

```
20 a=4: b=236: c=1: d=1
25 HCOLOR =3:FOR i=a TO b STEP c:DRAW d AT i, 75:GOSUB 35:NEXT:IF b=4 THEN 40
30 a=236: b=4: c=-1: d=2:GOTO 25
35 FOR t=100 TO 0 STEP -1:NEXT: e=RND(-e):DEF FN f(e)=INT(1+e*RND(1)):
   HCOLOR = FN f(15):RETURN
40 VTAB 6:HTAB 4: PRINT "Wanna See It Again?  y/n ":GET a$
45 IF a$="y" OR a$="Y" THEN VTAB 6: PRINT:GOTO 20
```

After you've tried the above program then try the following one. This Program replaces the existing 2 sprites with 2 new ones. The original ones are erased so if you want to use them again you'll have to re-boot Basic 2.0

```
3 REM NEW SPRITES DEMO FOR BASIC 2.0 ONLY
6 DATA 139,217,169,169,137,137,139,0,139,137,137,249,137,137,139,0,165,37,41,
  49,41,37,165,0,161,33,33,33,33,33,189,0
7 DATA 209,0,0,226,0,0,229,0,47,40,72,143,72,40,47,0,85,0,0,170,0,0,85,0,120,
  68,68,12,0,80,72,68,0
8 FOR x=192 TO 255:READ d:POKE x, d:NEXT
9 TEXT:POKE 16788, 1
10 VTAB 3:HTAB 5: PRINT "Remember Text Can ALSO"
15 HTAB 8: PRINT "Be On The Screen"
20 a=4: b=220: c=1
25 HCOLOR =3:FOR i=a TO b STEP c:DRAW 1 AT i, 75:DRAW 2 AT i+16, 75:
   GOSUB 35:NEXT:IF b=4 THEN 40
30 a=220: b=4: c=-1:GOTO 25
35 FOR t=100 TO 0 STEP -1:NEXT:RETURN
40 VTAB 6:HTAB 4: PRINT "Wanna See It Again?  y/n ":GET a$
45 IF a$="y" OR a$="Y" THEN VTAB 6: PRINT :GOTO 20
```

MAKING YOUR OWN SPRITES

If you have SmartLOGO then you can very quickly define your own sprites using the Logo SHAPE EDITOR. LOGO Shapes are defined in exactly the same way as SmartBasic 2.0 Sprites. This is described fully in Chapter 6 of the Logo Reference Manual. The simple procedure is: Boot the SmartLogo Program and answer NO to the Demos. Then Enter after the Logo Prompt ES 32 (RTN).

LogoShape 32 is normally Blank so an empty 256-Block Box 16 blocks High by 16 Blocks wide will fill the screen. (The Turtle Shape, actually another Sprite will be visible in the center of the screen but will not affect the following procedure. If it's objectionable then it can be HIDDEN first following instructions detailed in the Logo Manual).

To FILL a Block move the Logo Cursor to that block and Press HOME. Pressing Home when the cursor is already in a FILLed Block will ERASE that Block and return it to Transparent. Remember that even though Black is used to FILL blocks in the Logo Editor these may be later defined as any Color. If you ran program #2 above then you saw that Sprites Can Also Be configured to look like Text. These Text Characters should each be 7 blocks Hi by 5 or less Blocks wide with a space under neath and to the right. This will allow a sprite to contain 2 lines by 3 characters each. For 2 Lines of Text several sprites can be defined and placed side by side.

After you have your first Sprite Defined the way you want it Press SMARTKEY [VI]. This will Save its Shape in the Workspace (Its not really necessary to save it on the tape). The Editor will disappear and when the Logo Cursor returns Enter MAKE "SPRITE GETSH 32 (RTN). And then Enter PR :SPRITE (RTN). The 32 decimal numbers that define a Sprite will be listed in their correct order. Just copy them onto a sheet of paper. If you want to define more sprites you can use Shape 32 over or any other Of the 60 LogoShapes. Any changes that you make will not be permanent unless you follow the procedure to make them so detailed in the Logo Instructions. After you have the list of 32 numbers you can copy these into Program #2 above in place of one of the DATA Lines. When you run the program your Sprite will be displayed.

BIT-MAPPING YOUR OWN SPRITES

If you don't have the Logo Program or if you just want to learn how to define your own sprites by plotting a bit-map try the following simple procedure.

On a sheet of graph paper enclose a 256 block square, 16 Blocks high By 16 Blocks wide. Starting at the TOP RIGHT corner, outside the square along the top line, from Right to Left write above each column: 1,2,4,8,16,32,64,128,1,2,4,8,16,32,64,128. Now starting at the Top Left Corner, outside the square from Top to Bottom, Number each of the 16 Rows 1 thru 16. Now on the RIGHT Side, outside the square from Top To Bottom, number these same rows 17 thru 32. Finally, in the Center of the Square Draw a Vertical Line from Top To Bottom, starting between the numbers 1 and 128 along the Top.

What you now have is a graphic representation of the 32 bytes in memory that define a sprite arranged as they are displayed on the screen as two side by side 16 Byte columns. Each 1/2 line of 8 blocks is an 8 bit Byte. Blocks that you FILL assume the value of 1 and empty Blocks are 0. To easily convert these Binary Bytes to the decimal numbers required by Basic just look up at the top row of numbers. If a Block is FILLED, ADD This Number to the Total for this 1/2 line Byte (Numbered 1-32). Empty 1/2 lines are 0. Write these 32 numbers down in order and with a comma between each one and copy them into either of the Data Lines in the 2nd Program above inplace of the existing numbers. When you RUN the program you'll then see your own sprites displayed.

SmartBASIC V2.0
PEEKS, POKES and CALLS
Compiled by Sharon McFarlane/NIAD

Location	Function/Description	Default/Range
153	FLASH Speed (1=Slowest/255=Fastest)	12
259	PEEK to determine STDMEM=195 EXTMEM=210	-
1121	Number of Prompt Fonts (1-2)	1
1122	1st Left Line Margin Prompt	93
1123	2nd Left Line Margin Prompt	0
1594-95	LOMEM Pointer (lo/hi bytes)	0/0
1628	SPEED Value	255
1648	Highest Pokeable Address (lo byte)	144
	Value 255 increases poke limit to 65535	-
1649	Highest Pokeable Address (hi byte)	211
	Value 255 increases poke limit to 65535	-
11943	Value 208 corrects bug & allows	191
	removal of sprite once drawn (STDMEM)	-
12454	Value 208 corrects bug & allows	191
	removal of sprite once drawn (EXTMEM)	-
16771	Display any of 256 ASCII characters	-
	including Cntl. characters in TEXT Mode	-
	POKE ASCII value & then CALL 16770	-
16781	Value of Current Storage Device	-
	DISK#1=4 DISK#2=5 RAMDISK=205	-
	TAPE#1=8 TAPE#2=24	-
16783	Text Window Color in HGR Mode	1
16786	Pointer-Start of Sprite Design (lo byte)	192
16787	Pointer-Start of Sprite Design (hi byte)	0
16788	Displays pre-programmed Sprite #1	1
	Color MUST be defined first	-
16788	Displays pre-programmed Sprite #2	2
	Color MUST be defined first	-
16788	Leave Sprite Mode	0
16789	Flag for version of SBasic	0
	0=STDMEM Non-0=EXTMEM	-
16939	HOME Key / Substitute any Value	32

16957	Number of Lines to Clear (TEXT)	24
	A value of 20=clear 20 lines only	-
16957	Number of Lines to Clear (GR/HGR)	4
16958	Number of Columns to Clear	30
16959	Top Margin to Clear	0
	A value of 16=HOME/Cursor to Line 16	-
16960	Left Margin to Clear	1
17111	Current COLOR value (0-15)	255
17184	Background Color in TEXT Mode	0
17229	Value 200:TEXT corrects bug & allows	208
	sprites to appear in any sequence	-
17240	Text & Screen Color NORMAL TEXT Mode	240
17251	Text & Screen Color INVERSE TEXT Mode	15
17322	Start value # of Lines	23
17323	Start value # of Columns	30
17325	Start value of Top Margin	0
17326	Start value of Left Margin	1
17339	Value 227:CALL 17338 = Large Sprite	226
	Enlargement	-
17339	Value 226:CALL 17338 = Standard	226
	Sprite Size	-
17339	Value 225:CALL 17338 = Small Sprite	226
	Enlargement	-
17339	Value 224:CALL 17338 = Standard	226
	Small Sprite	-
17437	Value 0 stops Cursor blink	4
24695	Background Color in GR/HGR(2) Mode	1
24784	Text Window Color in GR Mode	17
24847	Text & Screen Color in GR/HGR(2) Mode	240
25360-78	Correct GR/HGR Color Table by inputing:	1
	FOR x = 0 to 15. POKE 25360 + x, x	-
	POKE 25378 + x, x. NEXT x	-
25992	Value 32 sets up for Large Sprites	32
25992	Value 8 sets up for Small Sprites	32