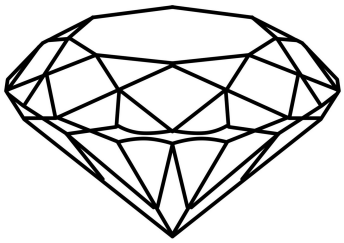
Texas Instruments

Extended BASIC 2.8 G.E.M.



by Harry Wilhelm
(based on Extended Basic v2.7 by Tony Knerr)

07/12/21

Extended BASIC 2.8 Graphics Enhancement Module.

The original XB G.E.M.module was developed to see how much power could be packed into TI Extended BASIC using just the 5 XB groms and the standard XB roms. Three graphics utilities totaling about 18K of code were compressed and squeezed into about 12K of free space in the XB groms. There is virtually no room left in the standard size roms and groms, and further development within these limits is not feasible.

These limitations go away when you bank switch additional ROM pages in the cartridge. This made it possible to add many significant enhancements. Starting with Tony Knerr's XB 2.7, I added 15 – 8K pages of ROM memory and a number of new CALLs and other refinements to the XB 2.7 groms. The result is Extended BASIC v2.8 G.E.M.

NEW FEATURES IN XB 2.8

Some of the existing CALLs have been modified:

CALL CHAR can now define any of the characters from 30 to 159. A string up to 255 bytes long can be used as the pattern identifier. This lets you define up to 16 characters with a single CALL CHAR.

CALL CHARPAT can now return a character pattern for characters from 30 to 159.

CALL COLOR can now define the color of all the character sets from 0 to 31. Although you cannot change the pattern of the characters below 30 and above 159, by setting the foreground and background to the same color you can have an 8x8 block of one solid color. There are 15 unused character sets, so you can have blocks of all 15 colors without sacrificing *any* of the characters from 30 to 159. These 8x8 blocks can be used for borders and other graphics.

CALL COLOR(ALL,foreground,background) will set all the characters to the selected colors.

These changes to the limit checks makes it possible for XB 2.7 G.E.M. to run TI BASIC programs using character sets 15 and 16, and usually with a considerable increase in speed.

CALL HCHAR and CALL VCHAR are *much* faster.

CALL INIT – after loading the assembly support routines INIT also clears the interrupt hook at >83C4. Unlike XB, CALL INIT is performed at XB startup.

CALL LOAD(filename) now uses the fast assembly language loader from the minimemory cartridge. This is about 20x faster than the normal loader used by other Extended BASICs. You can now load compressed code, use REFs and most of the other features missing from the normal XB loader. ***Do not AORG code to memory from >2008 to >20F7.*** This ram is temporarily used by the loader. When the loader is finished these addresses are then reset to the normal values provided by CALL INIT. Error messages are the same as in XB: DATA ERROR is used for CHECKSUM ERROR, and UNRECOGNIZED CHARACTER is used for ILLEGAL TAG.

CALL HELP now displays a detailed help menu with brief descriptions of all the new calls in XB2.7 and XB2.8. Use the space bar to advance to the next page of the help menu. The third screen has tables showing color codes and character sets. Enter a font number between 1 and 60 to display that font. Press space bar and you can press any key to see the ASCII code for that key. Press space bar and you get a number converter. Enter a decimal, hexadecimal or binary number. Press space bar to exit.

CALL VERSION(X) returns a floating point number. The latest version is 2.820210712 (2.8+year+month+day)

Added features providing a more satisfactory programming experience.

FONTS – 60 fonts are contained in the cartridge roms. A font can be loaded to the normal XB location with CALL FONT(number). The graphics packages have new CALL LINKs to load a font to screen 2 in XB256, to the bit mapped screen of The Missing Link, or to the text mode screens in T40XB or T80XB.

RND now uses the same code as the TI BASIC random number generator. This is about 5x faster than the XB random number generator. When you use a random number seed (i.e. RANDOMIZE 99), the random numbers generated are the same as in TI BASIC. There is another very useful modification: normally a LOAD program will always produce the same random numbers when it autoruns at XB startup. This has been changed so that now a LOAD program will actually produce random numbers.

There are changes to the line editor to make programming easier. You can enter long XB lines with up to 12 rows of code instead of the usual 5. The cursor repeat speed is faster. When editing a program line the cursor appears in the usual place, but you can go left and change the line number. The left/right/up/down keys function normally. Or you can hold down Ctrl and use the up/down/right/left keys to easily navigate a lengthy program line. Right/left work normally and up/down will go up or down one row in the line being edited. If you are on the last row, Ctrl+down goes to the end of the line. If you are on the first row, Ctrl+up goes to the beginning of the line. Fctn 3 (clear) erases the line from the cursor to the end of the line.

Instead of the * READY * message, XB 2.8 GEM gives the following messages at startup and when a program ends: * Extended Basic v2.8 G.E.M. if you have selected any XB option on the menu.

At startup or when a program breaks or ends one of the following messages will also be displayed:
* XXB * if you selected XB 2.8 + XXB
* XB256 * when XB256 is loaded and active
* T40XB * when T40XB is loaded and active
* T80XB * when T80XB is loaded and active
* TMLGA * when TML Graphics Adventure is loaded and active

You would normally use CALL CAT(devicename) to get a disk catalog, but there is also an assembly language disk catalog program in XB256, T40XB, and T80XB. CALL LINK(“CAT”) will start the program. Press any key to pause or resume the listing.

CALL VPEEK, VPOKE and GPEEK have been renamed PEEKV, POKEV and PEEKG. This is consistent with the convention used in Editor/Assembler BASIC, MiniMemory BASIC and RXB.

When selecting an option from the main menu, you can hold down any key for about 2 seconds to defeat autoloading. This can be inverted so that when your finger is *off* a key XB *will not* autoload. See page 10 for information on how to customize XB 2.8 G.E.M.

You can temporarily change the edit mode colors and font with CALL LOAD(9456,colors,font) The color should be foreground*16+background-17. The font should be a number from 1-60. If you add 128 to the font number you can invert the autoload behavior so you have to hold down a key to RUN “DSK1.LOAD”. If you add 64 to the font number you can run force command on BYE. See page 10 for more information on how to customize XB 2.8 G.E.M.

NEW CALLS IN XB 2.8

(ALL OF THE NEW CALLS REQUIRE 32K AND USE ROUTINES IN CALL INIT)

CALL INITG
INITG does the CALL INIT routine, loads two additional support routines for GPLLNK and DSRLNK, then clears the interrupt hook at >83C4.
GPLLNK is at >24F4 and DSRLNK is at >253A.

CALL FONT(Number)
FONT is used to load one of the 60 fonts contained in the XB 2.8 G.E.M. roms. Patterns are loaded for ASCII 30 to 127. The number must be from 1 to 60 and can be a constant or a variable.

CALL LFONT(Filename)
LFONT is used to load a font from disk. Patterns are loaded for ASCII 30 to 127.

CALL SFONT(Filename)
SFONT is used to save a font to disk. The cursor pattern is moved to ASCII 127 and then the characters from 32 to 127 are saved in memory image (binary) format.

CALL SAVEIV("DSKn.PROGNAME").
Used to save a program to disk in IV254 format. This is the format used by XB when saving long programs to disk. CALL SAVEIV("DSK2.TEST3") will save the currently loaded XB program as TEST3 onto DSK2. If the program is very small, a line 1 is created to pad the program with a DATA statement, which lets you save even the shortest XB program as IV254. (Immediate mode)

CALL RUN(Filename)
RUN is used to run an XB256 program from a running program. It does the same thing as RUN "DSK1.PROGRAM" but you can use a string variable which is not permitted in XB. This could be useful in a menu program or when chaining programs together.

CALL RUNL1(Filename)
RUNL1 is used to run an XB256 program from a running program. It is similar to CALL RUN but there is a major difference. After it loads the program it runs the first line in the program, *but without doing a prescan or resetting any of the variables of the calling program.* By chaining programs together with RUNL1 the 24K limit disappears. You can divide a very long XB program into shorter sections and load the segments of program as needed, with instant start up of the chained programs.

More information on using RUN, RUNL1 and SAVEIV can be found in the section entitled AUTOLOADING AND CHAINING PROGRAMS, starting on page 12 below.

CALL FC
FC is used to start "Force Command". This will only work if you have TIPI. You will get the message I/O ERROR 00. if there is no TIPI. You can also change the default configuration of XB 2.8 G.E.M. so that BYE will start "Force Command". See page 10 for more information.

CHANGING GRAPHICS MODES

It is now possible to change graphics mode from the command line or from a running program. Also, you can optionally perform the CALL FILES operation at the same time. The CALLs are described below. If necessary, these will automatically move the stack to a new location. If the new graphics mode does not have enough stack space available you will get a memory full error.

CALL XB or CALL XB(n)
Selects the normal Extended BASIC graphics mode. CALL XB starts with the default 3 disk files. CALL XB(n) also does CALL FILES(n). N must be a number from 0 to 9

CALL XXB or CALL XXB(n)
Selects the normal Extended BASIC graphics mode and loads the XXB extensions. CALL XXB starts with the default 3 disk files. CALL XXB(n) also does CALL FILES(n). N must be a number from 0 to 9. Docs for XXB v1.5 are included in the DOCS folder.

CALL XB256 or CALL XB256(n)
Loads and activates the XB256 graphics extensions. CALL XB256 starts with the default 3 disk files. CALL XB256(n) also does CALL FILES(n). N must be a number from 0 to 9

CALL t40XB or CALL T40XB(n)
Loads and activates the T40XB graphics extensions that provide access to the 40 column text mode. CALL T40XB starts with the default 3 disk files. CALL T40XB(n) also does CALL FILES(n). N must be a number from 0 to 9

CALL T80XB or CALL T80XB(n)
Loads and activates the T80XB graphics extensions that provide access to the 80 column text mode. CALL T80XB starts with the default 3 disk files. CALL T80XB(n) also does CALL FILES(n). N must be a number from 0 to 9

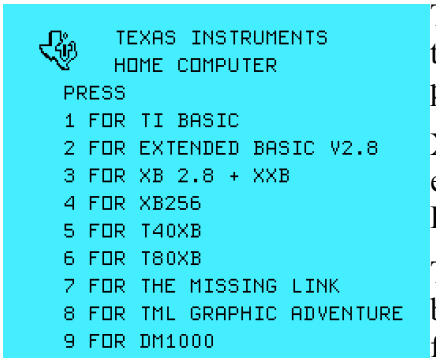
CALL TML or CALL TML(c,n)
Loads and activates The Missing Link graphics extensions that provide access to the bit mapped graphics mode. CALL TML starts with the default 16 color mode and 1 disk file. CALL TML(c,n) lets you select the color mode and also does CALL FILES(n). C is 1 for 16 colors or 2 for 2 colors; N must be a number from 0 to 3

CALL TMLGA or CALL TMLGA(c,n)
Loads and activates The Missing Link Graphics Adventure. These extensions are a customized version of The Missing Link configured for creating text and graphics adventures. CALL TMLGA starts with the default 16 color mode and 1 disk file. CALL TMLGA(c,n) lets you select the color mode and also does CALL FILES(n). C is 1 for 16 colors or 2 for 2 colors. N must be a number from 0 to 3

If a program requires a particular graphics mode you can select that mode in the first line of the program. Lets say your program requires XB256. You can add this line: 1 CALL XB256 and this way ensure that XB256 is loaded when the program starts.

These calls make it possible to go between any of the special graphics modes (XB256, T40XB, T80XB, TML, and TMLGA) from within a running program. If you just want to toggle between one of these modes and the standard XB mode, XB256, T40XB, T80XB, TML can already toggle between the two modes *without losing the graphics screen.*

The Graphics Packages



The main purpose of XB 2.8 G.E.M. is to provide graphics packages that unshackle the crippling limits on the TMS9918A video display processor imposed by TI XB.

XXB and the five graphics packages all require the 32k memory expansion to run. If it is not detected, the message "32K NOT DETECTED" appears. Press any key to go to Extended BASIC 2.8.

The brief descriptions below describe any differences from the disk based versions. Refer to their respective manuals in the documents folder for detailed information on how to use them.

XXB 1.5 by Barry Traver. This was part of XB 2.7. XXB is one of the more complete XB extension packages. It provides many additions to XB.

XB256 – this is part of the XB game developers package. You can select from two independent screens. Screen1 is the screen normally used by Extended BASIC. It is accessed with the usual XB graphics statements. Screen2 lets you independently define 256 characters, more than twice the 112 normally available in XB. Along with these graphics you can have up to 28 double sized sprites using the patterns available in Screen1. You can toggle between the two screens as desired, preserving the graphics in each screen. Screen2 has assembly equivalents that replace CHAR, CHARPAT, COLOR, and CHARSET. Except for these subroutines, all screen access in Screen2 is by the usual Extended BASIC statements such as PRINT, SPRITE, ACCEPT, etc. You can scroll screen characters left, right, up, or down. By specifing a window area you can scroll just the characters in the window. Other routines let you scroll smoothly one pixel at a time to the left, right, up or down, or do a "text crawl" like the title scene of "Star Wars. You can hilight text, play sound lists, print text using 32 columns and much, much more.

The autoprompt feature is off by default. This is only useful to those who use the entire game developers package (XB256, compiler, loader, etc.) If desired it can be activated with CALL LINK("XB256A"). The font used in screen 1 is whatever font used by the XB editor. Screen 2 uses font 1. The option to modify grom at >6000 is no longer needed and is disabled. The sound list compilers can be found in the disk based version of XB256.

T40XB – T40XB lets you select from two independent screens. G32 is the default screen when a program starts running.. This is the 32 column graphics mode screen normally used by Extended BASIC. It is accessed using the usual XB graphics statements. T40 is the 40 column text screen which displays 24 rows of 40 columns.. You can toggle between the two screens as desired, and the graphics on each screen are preserved. When using the T40 screen there are assembly equivalents that replace PRINT, CLEAR, COLOR, INPUT, CHAR, HCHAR, VCHAR plus routines that will scroll the screen and invert text on the screen.

The font used in graphics mode (G32) is the font used by the XB editor. Text mode (T40) uses font 1. . The option to modify grom at >6000 is no longer needed and is disabled.

T80XB – This is virtually identical to T40XB except it uses the 80 column text mode provided by some display devices or Classic99

The Missing Link – allows easy and versatile access to bit mapped graphics. You can plot points, lines, circles and boxes. Turtle graphics can be used with none of the ink and color restrictions found in LOGO. Up to 32 moving sprites can be placed on the screen. Text can be input from or displayed to the screen. Word wrap is used when displaying text. The character size can be changed, permitting up to 32 rows by 60 columns on the screen. Different sized text can be displayed simultaneously on the same screen.

When you select The Missing Link from the main menu, a submenu lets you configure TML. You can choose 16 or 2 colors and 0-3 disk files open. The "Texas" cursor has been replaced with the standard rectangular cursor. The default font in bit-mapped mode is font 1. CALL LINK("FONTA",N) will load a font from the cartridge. CALL LINK("LFONTA",filename) or ("SFONTA",filename) will load or save a font used by the bit mapped mode.

Since TML is loaded from the cartridge, it is possible to save programs to cassette. With 16 colors you can save a casette program up to 2048 bytes long. With 2 colors the program can be up to 5724 bytes long. You can trick XB into saving or loading even longer cassette programs. (up to 10176 bytes in 16 color or up to 13928 bytes in 2 color.) First type:
CALL LOAD(-31888,63)
Save or load the cassette program. Then type:
CALL LOAD(-31888,31) when ready to run the program.

The Missing Link Graphic Adventure – This is version of TML was modified so that bit mapped graphics pictures can be added to text adventures. It can load pictures to the top third of the screen while having text on the lower third. The default font uses 5x7 characters for a 48 column screen. The lower third of the screen will scroll automatically as you input commands to the computer. The turtle graphics routines have been eliminated to make room for the new routines.

Disk Manager 1000 – Press 9 to run DM1000 v7.0 This is not useful for those who use emulators, but is very handy if you are running a real TI-99/4a.

EXTENDED BASIC VERSION 2.7

Documentation by Tony Knerr with modifications for XB 2.8 by Harry Wilhelm

There are several new commands in Extended Basic v2.7 which support floppy disk formatting from the command line of Extended Basic. It is only a limited implementation, as I have very few bytes of program space left in the gram banks. The new commands are: CALL SSSD, CALL DSSD, CALL DSDD, and CALL DSQD.

They have been tested with the TI, Corcomp, and Myarc Floppy disk controllers and the Myarc Hard and floppy disk controller. The disk is formatted as it would be using Disk Manager 1000 or Disk Utilities and choosing N(o) when prompted to verify. The only sectors checked are 0 and 1 as these sectors are written to at the end of the formatting process.

Extended Basic v2.7 is compatible with TI Extended Basic with the following differences:

The default colors in edit mode are the standard black on cyan, although this can be customized as described on page 10. Regardless of the colors in edit mode, when a program runs the colors revert to the standard black on cyan.

The default character set is one with true lower case. This can be customized as described on page 10.

The command CALL CHARSET has been modified to restore both upper and lower case characters to the original TI character set and also resets screen, foreground, and background colors to Black on Cyan.

The CALL INIT command executes a little faster and clears the interrupt hook at >83C4.

By holding down any key after selection of XB v2.7, the user can bypass the auto-search for DSK1.LOAD and go directly to edit mode. This can be inverted as described on page 10.

The following CALLs are resident in version 2.7 (and 2.8):

CALL ALLSET in XB2.7 is replaced by CALL FONT(7). Used to reset the characters 32 through 126 to their original TI definitions. Works like CALL CHARSET but does not affect screen, foreground, or background colors.

CALL BYE resets to the title screen from a running program.

CALL BEEP sounds the familiar accept tone.

CALL CAT("pathname.") catalogs a device while in command mode. Has been tested with TI, Corcomp, and Myarc floppy controllers as well as the Myarc HFDC and Horizon Ramdisk. The pathname must end in a period. Directories on the HFDC of up to a total length of 11 characters are acceptable. Example – CALL CAT("WDS1.ABCDE.") is the longest pathname that will work.

CALL CHARA1 is replaced by CALL FONT(2). Restores the resident XB v2.7 character set if it has been redefined.

CALL CHIME generates a chime sound.

CALL CLSALL closes all open files.

CALL COLOR(ALL,foreground,background) replaces CALL COLR13, COLR15, COLR17, COLR18, COLR19, COLR1A, COLR1B, COLR1F, COLRF1, COLRF2, COLRF4, COLRF6, and COLRFC.

CALL DSDD formats a 1280 sector disk in drive 1 if you are using a Myarc floppy controller with a 40 track Eprom. A 1440 sector disk will be formatted if using a Corcomp floppy controller, a Myarc floppy controller with an 80 track Eprom, or a Myarc hard and floppy disk controller. You will be prompted to insert a disk in drive 1 and press enter to continue. Pressing any other key aborts the formatting process. Using this command with a TI controller will only be a waste of your time as the floppy controller will go through the whole formatting process only to give you an error at the end. Don't say I didn't warn you!

CALL DSQD formats a 2880 sector disk in drive 2 if you are using a Myarc floppy controller with an 80 track Eprom or a Myarc HFDC properly configured for 80 tracks. You will be prompted to insert a disk in drive 2 and press enter to continue. Pressing any other key aborts the formatting process. Using this command with a TI or /Corcomp controller will only be a waste of your time as the floppy controller will go through the whole formatting process only to give you an error at the end, and may damage your disk and/or drive. Don't say I didn't warn you!

CALL DSSD formats a 720 sector disk in drive 1. You will be prompted to insert a disk in drive 1 and press enter to continue. Pressing any other key aborts the formatting process.

CALL CRASH generates a crash sound.

CALL PEEKG(address, numeric variable list) – readsthe contents of addresses in GROMS. The function is similar to CALL PEEK.(This was CALL GPEEK in XB2.7)

CALL HELP has been modified to display help screens showing the commands in XB 2.8. You can view a table of color codes, character sets, display any font, find ascii codes and convert numbers.

CALL HONK generates generate the familiar error tone.

CALL LRGCPS is replaced by CALL FONT(13). Loads the large capitals character set from the TI title screen.

CALL MLOAD("filename", mode) loads a program image file (E/A option 5) into memory and run it if MODE=1. Example - CALL MLOAD("DSK1.UTIL1",1) will load and run an E/A option 5 file named UTIL1 from floppy drive #1.

CALL MOVE(mode, start address, target address, # of bytes) moves a block of memory. The four modes available are: 1 - from VDP RAM to VDP RAM, 2 - from VDP RAM to CPU RAM, 3 - from CPU RAM to VDP RAM, 4 - from CPU RAM to CPU RAM

CALL MSAVE("filename", start address, # of bytes) saves a portion of memory in program image format.

CALL NEW is the same as the NEW command except that it can be used within a running program.

CALL NYANYA generates the NYANYA sound our children love to do so often.

CALL QUIT is the same as CALL BYE, that is, it will return to the title screen from a running program.

CALL QUITOF disables the function of the QUIT key (FCTN =).

CALL QUITON restores the function of the QUIT key (FCTN =) after a CALL QUITOF.

CALL RESTORE(numeric variable) is the same as the RESTORE statment except that a variable may be used. Example - 100 A=500 :: CALL RESTORE(A)

CALL SCREENOF turns of the screen.

CALL SCREENON turns on the screen again after a CALL SCREENOF.

CALL SPROF stops the motion of all sprites.

CALL SPRON restarts the motion of all sprites after a CALL SPROF.

CALL SSSD formats a 360 sector disk in drive 1. You will be prompted to insert a disk in drive 1 and press enter to continue. Pressing any other key aborts the formatting process.

CALL PEEKV(address, numeric variable list) – reads the contents of addresses in VDP RAM. The function is similar to CALL PEEK. (This was CALL VPEEK in XB2.7)

CALL POKEV(address, numeric variable list) – writes values to VDP RAM. The function is similar to CALL LOAD. (This was CALL VPOKE in XB2.7)

CALL WAIT(duration) causes a delay. DURATION must be a value from 0 to 16382. A value of 60 equals one second of delay.

CALL XB returns to the normal XB graphics mode. CALL XB(n) will returns to XB with 0 to 9 disk files available. (modified for XB2.8 G.E.M.)

CALL XXB returns to the normal XB graphics mode and loads Barry Traver's XXB version 1.5 into low memory. All XB v2.7 and XXB v1.5 commands will then be available to the user, making for a very powerful Extended Basic environment. Docs for XXB v1.5 are included in the DOCS folder. CALL XXB(n) loads XXB with 0 to 9 disk files available. (modified for XB2.8 G.E.M.)

CUSTOMIZING XB 2.8 G.E.M.

TEMPORARILY CHANGE EDIT MODE COLORS AND FONT

The two bytes at cpu memory 9456 contain the colors and font used in the edit mode. At startup they are loaded with the default values 23,1. (>17,>01 for black on cyan and font 1) You can change these with CALL LOAD(9456,colors,font) The color should be foreground*16+background-17. (Neither color can be transparent and they cannot be the same.) The font should be a number from 1-60.

Normally when selecting an option from the main menu you can hold down a key for about 2 seconds to defeat autoloading of DSK1.LOAD. This can be inverted by adding 128 (>80) to the font number. Then if your finger is *off* a key XB *will not* autoload. If you do want to autoload you can hold a key down for about 2 seconds and XB will try to autoload DSK1.LOAD. TIPI users can add 64 (>40)to the font number. Then BYE will RUN "TIPI.FC.FCMDXB" and go directly to Force Command instead of going to the usual start screen.

Example: You want dark blue characters on a gray background, font 17 and autoload deactivated. Color is 5*16+15-17 which is 78. Font = 17+128 which is 145. From the immediate mode you can CALL LOAD(9456,78,145). Or you can make this a 1 line program saved as LOAD. When XB is chosen autoload is active. It loads your preferences for colors and font and inverts the autoload behavior.

PERMANENTLY CHANGE COLORS AND FONT USING HEX EDITOR

You can change the default colors and font permanently by by changing the grom file XB28GEM_G.BIN. Use a hex editor such as *HxD hex editor* to change 6AF0 from 17 01 (black on cyan, font 1) to the desired colors and font. These values are the same as described above, only in hexadecimal. To get white letters on dark blue, font 3, suppress autoload and invoke force command on BYE, 6AF0 should be F4 C3. (The font number is >80 to suppress autoload +>40 to invoke force command on BYE +font number >3=>C3). Be sure to save the modified file.

Now that you can select any desired graphics mode directly from Extended BASIC, it is no longer essential to have the graphics modes at the start menu. You can unclutter the meny by removing entries 3 – 8. Use a hex editor to modify the file XB28GEM_G,BIN. Change 1C76 from 7C 5C to 6A 01. Be sure to save the modified file.

For XB256, T40XB, and T80XB, the default colors and fonts can be customized by using a hex editor to modify the rom file XB28GEM_8.BIN. A font from 1 to 60 may be used, but only a few will look good in the text mode. As usual, be sure to save the modified file.

For T40XB:

The default text mode color is at	>1AEBB	(default is >F4)
The default text mode font is at	>1AC27	(default is >01)

For T80XB:

The default text mode color is at	>1CEE3	(default is >F4)
The default text mode font is at	>1CC4F	(default is >01)

For XB256:

The default screen 2 color is at	>18D33	(default is >F4)
To change default screen 2 font	>189BC	must contain a memory address having the desired font number. Default is >2C24 which contains >0001. For font 2 this can be >209A which contains >0002. Any address in ram from >0000 to >3FFF can be used if it contains the desired font number.

Using the REF directive in an assembly language program

The new assembly loader in XB 2.8 G.E.M. can process REF directives. Instead of VMBW EQU >2024, your code can simply REF VMBW. Below is a table showing the labels that will be recognized by REF. *If you want to use REF GPLLNK or REF XMLLNK, be sure to CALL INITG.* This does the normal CALL INIT, then loads the additional code for GPLLNK and XMLLNK.

```
UTLTAB  EQU >7020       VMBW    EQU >2024       SMUL    EQU >0E8C
PAD     EQU >8300       VSBR    EQU >2028       SDIV    EQU >0FF8
GPLWS   EQU >83E0       VMBR    EQU >202C       CSN     EQU >11AE
SOUND   EQU >8400       VWTR    EQU >2030       CFI     EQU >12B8
VDPRD   EQU >8800       DSRLNK  EQU >253A       FCOMP   EQU >0D3A
VDPSTA  EQU >8802       LOADER  EQU >25AE       NEXT    EQU >0070
VDPWD   EQU >8C00       GPLLNK  EQU >24F4       COMPCT  EQU >0000
VDPWA   EQU >8C02       NUMASG  EQU >2008       GETSTR  EQU >0002
SPCHRD  EQU >9000       NUMREF  EQU >200C       MEMCHK  EQU >0004
SPCHWT  EQU >9400       STRASG  EQU >2010       CNS     EQU >0006
GRMRD   EQU >9800       STRREF  EQU >2014       VPUSH   EQU >000E
GRMRA   EQU >9802       ERR     EQU >2034       VPOP    EQU >0010
GRMWD   EQU >9C00       FADD    EQU >0D80       ASSGNV  EQU >0018
GRMWA   EQU >9C02       FSUB    EQU >0D7C       CIF     EQU >0020
SCAN    EQU >000E       FMUL    EQU >0E88       SCROLL  EQU >0026
XMLLNK  EQU >2018       FDIV    EQU >0FF4       VGWITE  EQU >0034
KSCAN   EQU >201C       SADD    EQU >0D84       GVWITE  EQU >0036
VSBW    EQU >2020       SSUB    EQU >0D74
```

A bug was found in Assm994a:

REF GPLWS
DATA GPLWS is fine, but
LWPI GPLWS does not assemble and load properly. It must be LWPI >83E0

GROM space available in XB 2.8 G.E.M.

Much code and data was moved to the cartridge ROMs and this freed up a considerable amount of memory in the five XB 2.8 groms. This could be used for future modifications and improvements to the GPL code. 10520 bytes of free grom are available in the following locations:

```
>9848->996F     >128 bytes
>9B48->9FDF     >498 bytes
>B9E8->BFFF     >618 bytes
>D588->D617     >90 bytes
>D7E0->DBFF     >420 bytes
>DEB0->DFFF     >150 bytes
>E400->FFFF     >1C00 bytes
TOTAL           >2DD8 bytes (11736)
```

AUTOLOADING AND CHAINING PROGRAMS

Let's say you have written a complex program that needs to be in two or more parts. You have put all the character definitions, into the first program and then want that program to run a second program while preserving the characters you just defined. Simple enough, just add RUN “DSKn.PARTTWO to the first program and it will load and run PARTTWO, or you can use CALL RUN(A$) if the disk name is a string variable. If you are using one of the graphics packages in XB 2.8 G.E.M. and want to preserve sound tables, the screen2 or text mode definitions and colors then PARTTWO **must** be in IV254 format.

Internal, Variable 254 is the format used by Extended BASIC when saving or loading large programs that are longer than the available stack space. If the program is large enough it will be saved in this format automatically. If you don't know what format the program has been saved in, you can CALL CAT("pathname.") to catalog the disk and this will display the file types.

CALL SAVEIV(”DSKn.PROGNAME”)

XB 2.8 G.E.M. has a subprogram that will to force XB to save a program of any length in IV254 format. With your program loaded, in the immediate mode type:CALL SAVEIV(”DSKn.PROGNAME”) The program will be saved in IV254 format. If the program is very short a line 1 will automatically be added. This is a DATA statement that does nothing other than adding enough bytes to permit saving even the shortest programs in IV254 format. This will not happen if the program is larger than 256 bytes.
This should be compatible with a CF7 device, but has not been tested.

CALL RUN(A$)

RUN is used to run an XB program from a running XB program. The normal way to do this is with RUN “DSK1.PROGRAM”, which normally works fine. The only reason for using CALL RUN would be to run a file name contained in a string variable:

```
10 D$=”DSK1.PROGRAM”
20 CALL RUN(D$)          (XB cannot do this)
```

As long as there is no break in the programs you can chain programs together this way and when the new program runs the screen, colors, character definitions, sprites, etc. will be preserved. But variables are *not* preserved. When the new program runs, prescan resets all numeric variables to zero and string variables to a null string. If you want to pass variables from one program to the next you have to store them with CALL LOAD and retrieve them with CALL PEEK, or save them to disk and then read them back with the new program. Wouldn't it be nice if there was a way to preserve the variables and automatically pass them along to the new program? Well, there is.

CALL RUNL1(A$)

RUNL1 is used to run an XB program from a running program. It is similar to CALL RUN but with a major difference. After it loads the program it runs the first line, *but does not do a prescan or reset any of the variables of the calling program.* By chaining programs together with RUNL1 you can create a very long XB256 program, and the neat thing is that bypassing the prescan means super fast start up time and that all the variables are automatically carried over into the newly loaded program. You don't have to save them before exiting one program and retrieve them in the next program.

RUNL1 takes some liberties with the XB interpreter and to use it successfully, there are six rules that must be followed:

1 – The XB program that is first in the chain must be the longest. If RUNL1 tries to load a program that is longer than the first program, a "Memory Full" error will be issued. SIZE tells how many bytes are free and with that you can determine if the first program is the longest. If necessary, you can pad the first program with remarks to make it the longest program.

2 – Any program being loaded with RUNL1 must be in IV254 format. You can save a program of any length in this format by using CALL SAVEIV(filename). An I/O ERROR will be issued if RUNL1 tries to load a program that was saved in program format. If the first program in the chain is in IV254 format you can use RUNL1 to go back to it.

3 – All variables must all be contained in the first program. Prescan is only performed for the first program in the chain, which means that new variables cannot be introduced in a chained program. If the second program uses HCHAR and the variables X,Y,Z and A$, but they are not used in the first, the first program should include them this way:

```
10 GOTO 100::X,Y,Z,A$::CALL HCHAR
```

This forces prescan to reserve space for the four variables and HCHAR. This line doesn't give a syntax error because XB never gets to the second part of the line. When you get an unexpected “SYNTAX ERROR IN #”, be sure any variables in that line were included in the first program.

4 – If the chained programs use DATA statements, you must use RESTORE or RESTORE line-number. Otherwise the program cannot find the data and you will get a DATA ERROR message.

5 – SUB, SUBEND, SUBEXIT, and DEF can be used, subject to three rules: They must be included in all the programs. They must be the first lines entered into each program. They must be exactly the same in all the programs. Once they are included in the program you can merge program lines, paste XB in Classic99, enter lines of code from the keyboard, and even resequence. ***But do not make any changes to the SUBs or DEFs.*** as this will change their physical location in the program. This is described in greater detail below.

6 – Always do a fresh reload of the program if you want to edit a program that was loaded by RUNL1. When RUNL1 loads a program it moves the line number table so the program is the same length as the first program. This causes some hidden bytes to be temporarily included in the program and you don't want to make these permanent. These go away when you reload the program. After reloading the program you can safely make any necessary changes, but remember: no changes to SUBs or DEFs.

Programs that do not use SUBs or DEFs can be entered normally normally, but be sure to observe the six rules above.

Programs that use SUBs or DEFs require more attention to detail. Below is a step by step description showing how to include them in chained programs. For this example we create two programs in the chain called RL1TESTA and RL1TESTB and save them on DSK9.

Start by placing the SUBs and DEFs in a program of their own:

```
10 DEF CUBE(X)=X*X*X
20 DEF FOURTH(X)=X*X*X*X
200 SUB SQUARE(X)
210 PRINT X*X
220 SUBEND
230 SUB ROOT(X)
240 PRINT SQR(X)
250 SUBEND
```

Now let's write the two programs RL1TESTA and RL1TESTB.

```
1 ! RL1TESTA - The initial program in chain
30 DIM SR(30)
40 CALL CLEAR :: CALL SCREEN(4)
42 PRINT "Now running RL1TESTA"
50 CALL CHAR(128,"0103070B0A1B2A2B2A2B4A4B4A7B0E0A0080C0A0A0B0A8A8A8A8A4A4A4BCE0A0")
60 CALL MAGNIFY(4):: CALL SPRITE(#1,128,5,100,120,-7,0)
62 X=11 :: PRINT :X :: CALL SQUARE(X):: CALL ROOT(X)
64 PRINT CUBE(X):: PRINT FOURTH(X)
66 Z=31 :: A=PI
70 FOR I=1 TO 30 :: SR(I)=SQR(I):: NEXT I
75 READ DT$ :: PRINT DT$
80 INPUT "ENTER A STRING: ":A$
90 CALL RUNL1("DSK9.RL1TESTB")
100 DATA "Data-line 100, 1st program"


1 ! RL1TESTB - 2nd part of demo
110 CALL CLEAR :: CALL SCREEN(12)
120 PRINT "Now running RL1TESTB"
122 FOR I=1 TO 30 :: PRINT SR(I),:: NEXT I :: PRINT
124 PRINT Z :: CALL SQUARE(Z):: CALL ROOT(Z)
126 PRINT CUBE(Z):: PRINT FOURTH(Z)
130 PRINT A$
140 RESTORE :: READ DT$ :: PRINT DT$;
150 GOTO 150
190 DATA "Data-line 190, 2nd program"
```

These two programs must be combined with the subs and defs. The easiest way is to use a text editor and Classic99 to paste into XB. (Edit>Paste XB) The order is important - the subs and defs ***must*** go first.

Start with NEW, then copy/paste the subs and defs into XB, then copy/paste RL1TESTA into XB, then CALL SAVEIV("DSK9.RL1TESTA").

Similarly, for the second program start with NEW, then copy/paste the subs and defs into XB, then copy/paste RL1TESTB into XB, then CALL SAVEIV("DSK9.RL1TESTB").

All done! Now you are ready to test. Enter RUN "DSK9.RL1TESTA"
(Some explanation of the program is in order so you know what is happening.)

(Description of RL1TESTA)
42 – prints "Now running RL1TESTA"
50,60 – creates a sprite and sets it in motion
62 sets X=11; prints X; CALL SQUARE(X); CALL ROOT(X) (The two SUBs print X^2,then SQR(x))
64 PRINT CUBE(X); PRINT FOURTH(X) (uses DEFs to print x^3 and x^4)
66 Z=31 :: A=PI (these values not used here, they will be used by the next program in chain)
70 fills the array SR(n) with square roots from 1 to 30 (used in next program in chain)
75 reads a string from a data statement and prints it
80 asks you to enter a string (the string is used in next program in chain)
90 loads and runs the program RL1TESTB

When RL1TESTB is run in line 90, XB doesn't know that a completely new program has been loaded. It assumes it is continuing with the original program and so all the variables are retained automatically, as demonstrated by the second program in the chain.

(Description of RL1TESTB)
The sprite created in the first program continues to move across the screen.
120 – prints “Now running RL1TESTB”
122 – prints the 30 square roots that were created in line 70 of RL1TESTA (shows that variables can be passed)
124 – prints Z; CALL SQUARE(Z); CALL ROOT(Z) (shows that Z is passed and that SUBs work properly)
126 – PRINT CUBE(Z); PRINT FOURTH(Z) (shows that DEFs work in the chained program)
130 – PRINT A$ (shows that string variables can b e passed to the second program)
140 – RESTORE; read string from DATA statement; print it. (DATA can be used in the chained program)
150 – wait for Fctn 4 to be pressed

You can add lines to these programs freely, but be *very careful* not to edit any of the lines with SUBs or DEFs. Even though the listing is the same, the physical location of the code will be moved, and the only way to restore the proper order is to start over. Be sure to make frequent backups!

A few more steps are required when you use a real TI-99/4a.

1 – Enter the program segment containing the SUBs and DEFs. Save this program to disk.
```
SAVE DSK9.RL1TESTSD.
```
2 – Enter the listing for RL1TESTA. Save to disk in merge format:
```
SAVE DSK9.RL1TESTAM,MERGE
```
3 – Enter the listing for RL1TESTB. Save to disk in merge format:
```
SAVE DSK9.RL1TESTBM,MERGE
```

Now combine the subs and defs with the main programs.

4 – Combine and save the first program:
```
OLD DSK9.RL1TESTSD
MERGE DSK9.RL1TESTAM
CALL SAVEIV(“DSK9.RL1TESTA”)
```
5 – Combine and save the second program:
```
OLD DSK9.RL1TESTSD
MERGE DSK9.RL1TESTBM
CALL SAVEIV(“DSK9.RL1TESTB”)
```

Now the programs can be tested with RUN “DSK9.RL1TESTA”