

EXTENDED BASIC GAME DEVELOPERS PACKAGE

“Juwel”

by Harry Wilhelm

09/29/2024

For all its strengths, Extended BASIC has two major weaknesses. It cannot utilize the full power of the 9918A video processor, and programs generally run slowly. This package contains two stand alone applications that can also be used together to address these shortcomings, making it possible to produce arcade quality games with XB.

XB256

XB256 is a collection of assembly language subroutines for Extended BASIC that unlocks the graphics and sound capabilities of the TI-99/4A computer. No knowledge of assembly language is required to use XB256.

XB256 lets you select from two independent screens. Screen1 is the screen normally used by Extended BASIC and it is accessed with the usual XB statements. Screen2 lets you define 256 characters, more than twice the 112 available in XB. Additionally, you can use up to 28 double sized sprites using the character definitions available to Screen1. You can toggle between the two screens as desired and preserve the graphics on each screen. When using Screen2 there are assembly subroutines that replace CHAR, CHARPAT, COLOR, and CHARSET. All other screen access in Screen2 is by the usual Extended BASIC statements such as PRINT, SPRITE, ACCEPT, etc.

Scrolling routines allow you to scroll screen characters left, right, up, or down. You can specify a window area for scrolling and leave the rest of the screen unchanged. Other routines let you scroll smoothly one pixel at a time to the left, right, up or down. Plus there is a text crawl that gives an effect similar to the STAR WARS title screen.

There are subroutines that let you highlight text, set the sprite early clock, print in any direction on the screen using all 32 columns, read from or write to the VDP ram, write compressed strings or sound tables to VDP ram, and play a sound list. A disk catalog subroutine has been added.

A utility (COMPRESS) is included that lets you save selected areas of VDP memory as compressed strings that can be merged with your program. This lets you save character definitions, sound tables, screen images, etc. in a more compact form that can be loaded virtually instantaneously.

There are two utilities (SLCOMPILER and SLCONVERT) that convert the CALL SOUNDS in an XB program to a sound table that contains music and sound effects in a form that can be loaded directly into VDP memory. This is much more compact, and your XB program can do other tasks while a sound list plays. After a sound table is loaded into VDP RAM you can play any sound list in it using CALL LINK(“PLAY”,address) When a sound list has started it will play automatically while your XB program does other things. Also, there is a second player that can play a different sound list simultaneously with the first. This way you can have background music playing and add sound effects without interrupting the background music.

With XB256 a fast paced arcade style game would not be possible without compiling, but a tantalizing possibility would be to have an autoloading large adventure game much longer than 24K using multiple program segments. XB256 can provide much richer graphics and use sound tables. Compressed data statements can load new graphics almost instantly. Rename XB256 to LOAD, put it into DSK1 where it autoloading, starts XB256 and then runs the initial segment of the program. "Who's behind the Mexican UFO's?" comes to mind as an example, although that used Missing Link and bit mapped graphics.

EXTENDED BASIC COMPILER

The Extended BASIC compiler lets you take advantage of the simple program development offered by Extended BASIC, then make an end run around the speed limitations. The goal was to implement Extended BASIC as fully as possible within the time limits of the programmer and the memory limits of the machine. There *are* limitations and you will probably need to adjust your programming style a bit, but in general, all the major features of XB are supported. You can concentrate on writing and testing a program in the XB256/Extended BASIC environment. When the program has been perfected it can then be compiled into an equivalent code that will run at a speed approaching assembly language. The average Extended BASIC program will run at least 30 times faster after being compiled, and some operations can run up to 70 times faster.

The compiler has been expanded to include all the XB256 subroutines. Compiling a program that uses XB256 is no different than compiling a conventional XB program. Write the program in XB or in XB256, test it until it is perfected, then use the compiler to get a huge performance increase. Some disk access has been added to the compiler – you can have up to three Display, Variable type files open at a time. Some recent improvements are that XB style IF/THEN/ELSE statements, named subprograms, and nested arrays can now be used. The compiler still has some limitations due to the use of integer arithmetic, no trig functions, etc. But if you work within its limitations you can create arcade quality games.

If you are developing a program using XB256 or XB that you intend to compile for increased speed, then you should keep in mind the limitations of the compiler. It is much easier to write code that takes those limitations into account rather than having to rewrite the code and debug it twice. Refer to the compiler manual *before* you start writing so you don't get stuck having to make revisions.

Although XB256 and COMPILER256 are designed to complement each other, remember that they are stand alone utilities. Programs developed using XB256 do not have to be compiled. If the execution time is adequate they will run fine in XB, and you would have access to floating point math, full disk access, etc. Similarly, XB programs do not have to use XB256; they can still be compiled for the increased speed. Standard TI BASIC programs can be also be compiled.

An addition starting with “Isabella” is XB32K. This is a utility that lets you use all 32K of the memory expansion for an Extended BASIC program. The XB program must be in straight Extended BASIC without any assembly subroutines. This extra long XB program can be compiled if desired. *Using XB32K* in the document folder tells how to use this.

The “Jewel” package can now incorporate assembly support for a compiled program with some minor limitations. The assembly support routines must reside in low memory. Any numbers passed to or from the assembly routines must be integers. The file “XB Compiler.pdf” has a complete description of how to use assembly support in your compiled programs.

XB 2.9 G.E.M. can be used instead of TI Extended BASIC. (XB 2.9 *must* be used when making rom cartridges or multi cartridges.) XB programmers will appreciate the added editing capabilities. The assembly loader was borrowed from MiniMemory and it is 20x faster than the normal assembly loader used by XB. GEM adds many additional CALLs for the XB user, but the only ones recognized by the compiler are CALL PEEKV, CALL POKEV, CALL MOVE, CALL STCR, and CALL LDCR.

SETTING UP THE GAME DEVELOPER'S PACKAGE

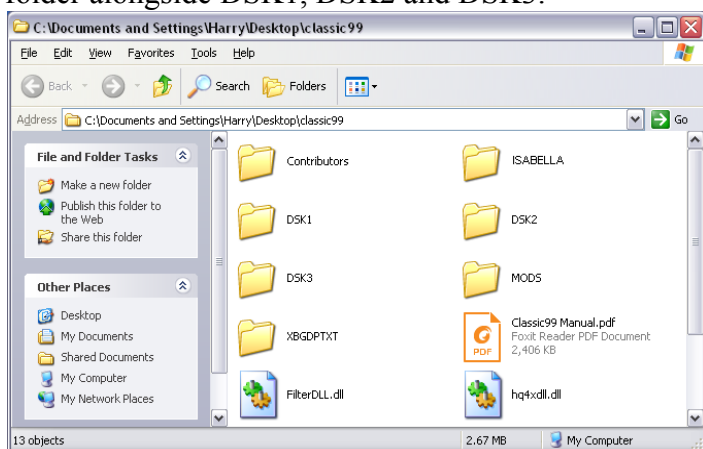
The package consists of one folder (JUWEL) with two sub-folders (DOCS and JUWEL99)

JUWEL contains the program files used by the Game Developer's Package. It is designed to be used with Mike Brent's Classic99 emulator which is an excellent tool for development work. It can read and write windows format text files so you can easily view compiled programs. Another advantage of CLASSIC99 is the ability to use CPU overdrive to greatly speed up the computer. Although Classic99 is used to create an XB256 or compiled program, the final product should be compatible with a real TI system or any other emulator.

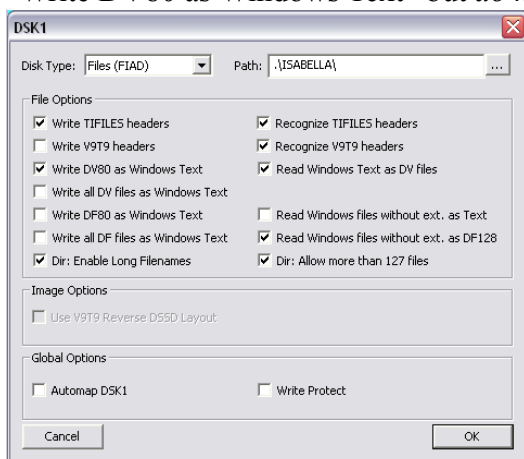
DOCS is a subfolder containing the documentation for the game developer's package.

Classic99 can be downloaded from Mike's web site <http://www.harmlesslion.com/> or from <http://atariage.com/forums/forum/119-ti-994a-programming/> under *TI-99/4A development resources*.

Classic99 is the preferred method for running the game developer's package. As received the classic99 folder comes with 3 disks configured: DSK1, DSK2 and DSK3. Put the folder JUWEL into the classic99 folder alongside DSK1, DSK2 and DSK3.



Then configure DSK1 as shown below. This tells Classic99 to use JUWEL as DSK1. Be sure to enable “Write DV80 as Windows Text” but *do not* enable “Write DF80 as Windows Text”. Press “OK”.



Write DV80 as Windows Text must be checked if you use Asm994a

Any other disks such as DSK2, DSK3, etc. that will be used with the compiler should also be set up as described above.

Very rarely XB256 will crash at startup. This seems to happen more frequently when using CPU overdrive. If this is a problem, enabling GRAM at >6000 allows XB256 to temporarily alter a pointer in XB. See page 9 for more information.(The latest version of XB 2.9 G.E.M. should not have this problem.)

Using the Game Developers Package

You must be using XB 2.9 G.E.M., TI Extended BASIC or another Extended BASIC such as RXB. To select Extended BASIC, click on Cartridge>Apps>Extended BASIC

Considerable effort has gone into revising the game developers package so it is as easy to use as possible, and there have been many performance improvements as well.

If you have never used the compiler it can be a bit intimidating. - 6 different files are used!

FILENAME	original XB or BASIC program
FILENAME-M	the same XB program saved in merge format
FILENAME.TXT	Assembly source code created by the compiler
FILENAME.OBJ	Assembly object code created by the assembler
FILENAME-E	Compiled program ready to run in EA5 format
FILENAME-X	Compiled program ready to run in an XB loader

Wow, that sure is a lot to keep track of, but don't let it scare you off. Before going into the details, let's give the game developer's package a test drive. HELLO is a simple XB demo program that is on DSK1 as part of the package. Let's see how easy it is to compile it.

Classic99 should be set up as described above. Then start up Classic99 and choose XB. The main menu appears. Press the space bar once for EXTENDED BASIC and press Enter. You are prompted OLD DSK. Make the filename OLD DSK1.HELLO and press enter. List the program and then run the program to see what it does and remind yourself how slow XB is. Break the program with F4. Go to Options>CPU throttling>CPU overdrive. Then type SAVE and press enter at each prompt. (22 times total – stop when the prompt is RUN). Reset the CPU speed to normal and then press enter one final time to run the program. Notice the speed improvement - about 30x faster! (In the docs, HELLO.GIF is an animated GIF showing this process. Appendix B in this manual is a listing of HELLO.

What just happened? Well, without having to know anything about assembly language you have just: Loaded an XB program, tested it, saved it, saved it in merge format, compiled the merge format XB program into assembly source code, assembled the source code into assembly object code, loaded the object code, saved the compiled program in EA5 format, saved the compiled program in an XB loader, and ran the compiled program! All by simply pressing the Enter key!

Now for the details. There is a lot of information below, and most of it is not all that important to the user. Basically, once you have saved an XB256 or Extended BASIC program you just have to keep pressing Enter until the program has been compiled.

The MENU program

The menu program is the heart of the game developer's package. It lets you easily select and load any component of the package. If you are holding down any key when the menu program starts it will CALL INIT, exit the loader, do a NEW, and return to the XB command line.

The programs loaded by the menu pass information to each other using a mailbox at >FFE8, an unused area of memory. Using the file name in the mailbox, autocomplete prompts are provided whenever an input is expected, so each program can seamlessly interact with the next. Saving, compiling, assembling, and loading are simply a matter of pressing Enter every time you are asked for a prompt, but you always have the option to change the filename if desired.



Navigate the menu with the up/down keys or the space bar. Press Enter to select an option. The options are:

XB256 – Loads XB256. If using XB 2.9 GEM it will load from the cartridge; other versions of XB will load from disk. Does a NEW, activates XB256 and goes to the XB command line. Detailed information on how to use XB256 can be found in the documents. Autocomplete will be active as described below.

EXTENDED BASIC – Does a NEW and goes to the XB command line.

Both the above options have an autocomplete routine running in the background. This routine assumes that you will want to load a program, edit and test it, then save it as a program, then as a merge file, and then return to the main menu. The center of the cursor has a light pixel when this routine is active.

At startup, the default prompt is OLD DSK. If a filename is in the mailbox the prompt is OLD DSKn.FILENAME. You can press Enter if you like the filename, or change the filename and press Enter, or press F4 to bypass the prompt. When you enter a filename here the XB/XB256 program is loaded and the filename is sent to the mailbox. You can turn off autocomplete at any time by pressing Escape or Fctn3. You know it is off when the cursor is solid black. There is no “undo” on the autocomplete routine. Once you press Enter, F4 or F9 you cannot return to that option.

Work on the program and test until you are satisfied. Type SAVE. When the autocomplete routine sees the four capital letters “SAVE” it will plug in the filename: SAVE DSKn.FILENAME or SAVE DSK if the mailbox is empty.

As before, press Enter if you like the filename, change it and press Enter, or press F3 or F4 to bypass the prompt. (This would be a good time to choose CPU overdrive in Classic99)

Then SAVE DSKn.FILENAME-M, MERGE is suggested. By now you know the drill: press Enter, change the name and press Enter, or press F4.

Then RUN DSK1.LOAD is suggested. Enter returns to the menu, or press F3 or F4 stay in XB256 or XB.

COMPILER – Runs the Compiler. Be sure to carefully read the documentation on the Compiler so you know its capabilities and limitations. When it runs, the compiler prompts are:

XB merge file to compile? If you saved an XB256 or XB program earlier that file will be suggested with a -M extension. Otherwise enter a filename.

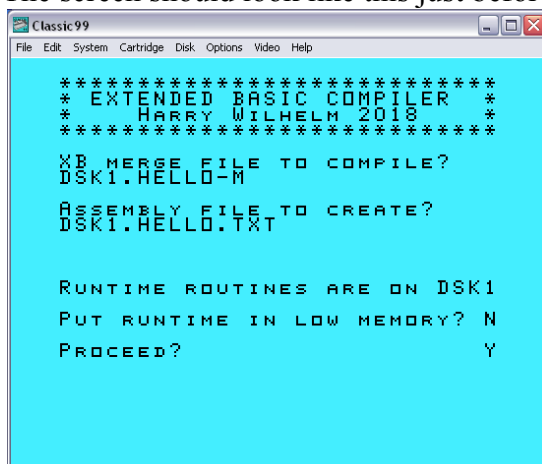
Assembly file to create? .TXT will replace -M in the file name you entered above. You can change the filename if you wish.

Using Asm994a? N If N then the followup question is **Runtime routines on DSK1**. If you have followed the setup directions they will be on DSK1. If they are on another disk then enter the disk number. If using Asm994a press Y and Enter.

Put runtime in low memory? N You have the option of putting the runtime routines in low memory.. As a rule you would only choose this option if the program is too large to fit in high memory. If there is enough room in high memory the entire process of compilation is simplified, and the XB loader only needs one file.

Proceed? Y Press Enter to compile or N and Enter to redo.

The screen should look like this just before you press Enter to proceed:



Or **ASSEMBLING WITH ASM994A? Y**
(Should only be Y if the program is too large to fit in high memory)

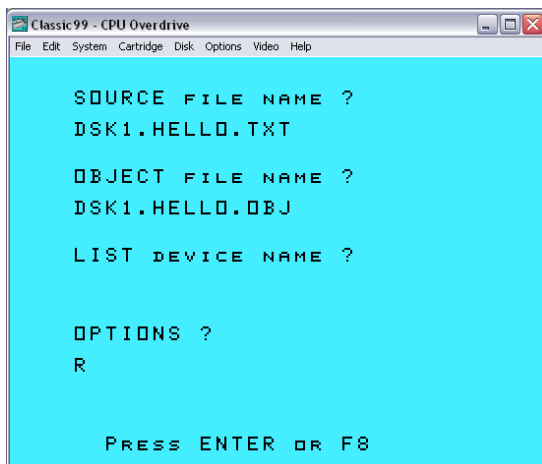
After you press <Enter> the compiler will analyze each line of the XB program and create an assembly language source code file.

When the compiler is finished the main menu will appear.

The compiler creates a file that is specific to the assembler option you choose. If you choose N for “Assembling with Asm994a” then you must use the TI assembler. To help guide you, the cursor in the menu points to “Assembler”

If you choose Y for “Assembling with Asm994a then you must use Asm994a. The cursor in the menu will point to “Loader” Remember to assemble with Asm994a before selecting the loader. Be sure you have copied the runtime routines and Asm994a.exe to your working disk, and made a shortcut pointing to the new location. The compiler manual describes how to use this cross compiler in more detail.

ASSEMBLER – Runs the Assembler. This has been modified from the Funnelweb assembler (which was adapted from the TI assembler) to be a stand alone program. If you have run the compiler before loading the assembler then the filenames generated by the compiler are filled in for you; otherwise you will have to fill them in yourself.



Press Enter to assemble or F8 if you need to redo the file names..

If the assembly process is successful you get this message:

0000 ERRORS

Press ENTER or QUIT

When the assembly process is complete, Enter will load and run the menu program. Some of the modified XB's such as RXB or XB2.7 may not return properly to the menu, or if they do there may be black letters on a black background. If this happens you should Quit, press a key and select the version of XB you wish to use. The contents of the mailbox are preserved if you do this.

There is a quirk in Classic99 that you should be aware of. When running the TI Assembler with CPU overdrive, when the assembly process has started there are no screen updates until the assembler is finished. You probably won't see the "ASSEMBLER EXECUTING" message. Don't assume there has been a crash, just wait until the assembler has finished its job. You will know this when you see "0000 ERRORS, Press ENTER or QUIT"

Under Options you can use RC to create a compressed object code file. This is more compact and loads a little faster. The loader for high memory runtime can handle compressed object code. You must use XB 2.9 G.E.M when loading only if you use compressed object code *and* place the runtime routines in low memory.

With XB 2.9 G.E.M. you can hold down any key to defeat autoloader. If you are using CPU overdrive in Classic99 you cannot get your finger off the enter key fast enough and so loading the menu will be blocked. Set CPU throttling to normal and tap the enter key quickly, or type RUN "DSK1.LOAD".

CROSS ASSEMBLERS

The assembler that comes with the Game Developer's Package is ported from the TI Editor/Assembler package. It works fine and is quite useable, but it does take some time to assemble, and error messages are very cryptic.

To speed up the development process you should learn how to use a cross assembler such as Asm994a. This is part of the Win994a package by Cory Burr, and is included in the XB game developers package. With Asm994a the assembly process happens almost instantaneously and error messages contain far more information. I have devoted a few pages in the compiler documentation describing how to use it.

LOADER – The loader loads the object code file the assembler created into memory. Then you have options to save in EA5 format, save in XB format, or RUN the program. You are prompted for the filename contained in the mailbox with .OBJ appended. Press Enter or change it as desired. The loader then reads the first line of the file to find out whether it is a valid file, how much memory remains, and whether the runtime routines are in high memory or low memory so it can use the appropriate loader.

If the runtime routines are in high memory the program is loaded using the assembly loader ported from the E/A cartridge. This is *much* faster than the standard XB loader and it can handle compressed object code. When the code is loaded you get a screen like the one shown below. Three prompts are automatically suggested in the XB command line. You can press F3 or F4 to bypass any of these options. (Autocomplete has no “undo.” Once you press Enter, F3 or F4 you cannot return to that option.)

LOAD COMPILED PROGRAM	
Enter filename to be loaded: DSK1.HELLO.OBJ	
Runtime in high memory 18898 bytes remaining	Amount of memory remaining
File is Loaded	
Press Enter (F4 will bypass)	
>CALL LINK("EA5","DSK1.HELLO-E")	Autoprompt to save in EA5 format
>SAVE DSK1.HELLO-X	Autoprompt to save in an XB loader
>RUN	

CALL LINK("EA5","DSKn.FILENAME-E") Press Enter to save the compiled program in EA5 format. Depending on the length, the EA5 program will be saved in from one to four sequential files (-E,-F,-G,-H).

SAVE DSKn.FILENAME-X Press Enter to save the compiled program embedded in an XB loader. This program will always be saved in one file (-X)

RUN Press Enter to run the compiled program. (If Classic99 is in CPU overdrive you should return to normal speed)

After test running the compiled program you will probably want to do some more work on the original XB program. Just “Quit” or do a “warm reset” in Classic99 and select XB again. The name of the file you are working on is still in the mailbox and will be suggested as the prompt when you select XB256 or XB.

When the runtime routines are in low memory a different loader must be used. When the code is loaded, you get a screen like the one shown below. There are prompts to save as EA5 and XB. You can use the suggested prompt, change the filename, or erase the file name. When you **Save as XB** two files are be created.. After pressing Enter for the **Save as XB** option, the cursor disappears for a few seconds. Do not press any keys until the cursor reappears on the **RUN** option. For the XB programs, the -X file must run first. It loads the runtime routines to low memory, then loads and runs the -Y file containing the program

Enter filename to be loaded: DSK1.HELLO.OBJ	
Runtime in low memory 24300 bytes remaining	
File is loaded.	
lowmemory 14394-16383 free	Free memory locations in low memory
Options for saving program Press Fctn 3+Enter to Bypass	
Save as EA5 DSK1.HELLO-E	Autoprompt to save in EA5 format
Save as XB SAVE DSK1.HELLO-X	Autoprompt for part 1 of XB loader program
SAVE DSK1.HELLO-Y	Autoprompt for part 2 of XB loader program
■UN	

This loader runs from VDP memory. If it breaks due to an error or Fctn 4, you should “Quit” or “warm reset” which will reset the pointers to recognize the 32K memory. It works fine in regular XB, but if you use XB 2.9 G.E.M. the program will be loaded much faster.

SAVE PROGRAM AS A CARTRIDGE

Saving as a cartridge requires XB2.9 G.E.M. and Classic99 version QI399.055 or later.

You also have the option of saving the program in a cartridge format, which many users prefer. You can save either as a grom cartridge or as a rom cartridge. A grom cartridge can be used on all TI-99/4a computers including V2.2 consoles. Grom cartridges have four 8K groms, while Rom cartridges have four 8K pages of rom. Both types of cartridge require the 32K expansion.

First the program must be loaded to memory. Normally you would create a cartridge after the program has been loaded during the compilation process as described above. After the options to **Save as EA5**, **Save as XB**, and **RUN**, the program will still be in memory. Or, if the program has already been saved in a previous session, **OLD DSK1.PROGRAM-X** will load it into memory.

To make a ROM cartridge: (Rom cartridges are more versatile but cannot run in a v2.2 console)

CALL XB(1)

CALL LOAD(-31868,0,0,0,0) This turns off the expansion memory. If you forget to do this, the program will stop and prompt you on what steps to take.

RUN "DSK1.MAKECART8" or **OLD DSK1.MAKECART8** then **RUN**

The loader program should be self explanatory. You are asked for three things:

```
Program name on menu? XB256 DEMO
Initialize VDP? Y
Name of file? DSK4.256DEMO-8.BIN
Creating cartridge...
```

This will be the name that appears as option 2 on the menu.
Usually this is Y, but you may want N with chained cartridges
The cartridge will be saved as 256DEMO-8.BIN
(A ROM cartridge ends with -8.BIN)

There is one other option that you will only see if your compiled program uses *The Missing Link*, and if at least 12K of memory is available:

Using TML-add picture? Y/N

If you press Y then you are prompted to load a bit mapped picture file. Do not append the -P to the file name, the program does this for you. The picture will be embedded in an unused portion of memory, out of the way of the compiled program. You can display the picture by including the following line of code in the XB source program:

100 CALL MOVE(3,-24530,8192,6144):: CALL MOVE(3,-18386,0,6144) !Move pattern and colors to VDP ram.

To make a GROM cartridge: (GROM cartridges will run in a v2.2 console, but cannot be chained)

CALL LOAD(-31868,0,0,0,0) This turns off the expansion memory. If you forget to do this, the program will stop and prompt you on what steps to take.

RUN "DSK1.MAKECARTG" or **OLD DSK1.MAKECARTG** then **RUN**

The loader program should be self explanatory. You are asked for two things:

```
PROGRAM NAME ON MENU? XB256 DEMO
NAME OF FILE? DSK2.?X.256DEMO-G.BIN
Creating cartridge...
```

This will be the name that appears as option 2 on the menu.
The cartridge will be saved as 256DEMO-G.BIN
(A GROM cartridge ends with -G.BIN)

CHAINING CARTRIDGES

Creating a chained cartridge requires XB2.9 G.E.M. and Classic99 version QI399.055 or later.

A versatile feature of Extended BASIC is the ability for one XB program to run another XB program using RUN "DSKn.PROGRAM". The first program could be a menu program that lets you select different programs to run. Or you can get around the 24K program length limit by dividing a long program into smaller segments. Each segment of code can run any of the other segments, and in this way you can create a program of nearly unlimited size. The folder MULTITUTOR contains a tutorial showing this process.

If you are running compiled programs from XB you can chain programs together in a similar way. This process is described on page 10 of *XB Compiler.pdf*. This works because a compiled program is just a very large assembly language subroutine. When the program ends, control returns to XB, which can then load and run another compiled program.

If you are running a compiled program in E/A5 format, the compiler has no provision for running another E/A5 program.

However, it *is* possible to chain together programs that are saved in cartridge format. When you use MAKECART8 as described above to make a rom cartridge, it will contain four 8K banks of memory. But a rom cartridge is not limited to just 32K of memory. It can have many 8K banks, totaling 512K and even more. With all that memory available, there is plenty of room to combine individual cartridges into a large combination cartridge, where any of the cartridges can load and run any of the others.

As always, the first step is to get the various segments of the program so they work properly in Extended BASIC.. You will use RUN "DSK1.PROGRAM" as usual, but with one difference. The file names should all be the same except for the last letter. The name of the first program should end in "A", the second program should end in "B", and so on. The names could be as simple as DSK1.A, DSK1.B, DSK1.C... Or, if it works better for you, DSK1.PARTA, or DSK1.PROGRAMA, etc.

Once this is working in XB, you do not need to make any changes when compiling. When XB comes to RUN "DSK1.PARTC", it will load and run PARTC. When the compiled program comes to that line, it looks at the last letter of the file name. It finds a "C", so it will load and run the third cartridge in the combination cartridge. If it finds a "D" it runs the fourth cartridge, etc.

Once the various programs are compiled, the next step is to convert each program to a cartridge using MAKECART8 as described above. The cartridges should have the same name as the XB files with -8.BIN appended. e.g. DSK1.A-8.BIN or DSK1.PARTA-8.BIN. You have a choice of initializing the VDP to retain the vdp data. For the first cartridge (with suffix A) you should always initialize the VDP. Other cartridges in the chain can be initialized or not, depending on what is needed. This step must be done in MAKECART8 and they should all be saved on the same disk drive.

Once all the carts in the chain have been created, it is time to combine them into the combination cartridge. **RUN "DSK1.CHAINCARTS"** or **OLD DSK1.CHAINCARTS** then **RUN**

The prompts should be self explanatory.

Name of first cart in chain?
DSK4.PARTA-8.BIN

The program checks to see if the file exists and if there is an A

Name to put on the menu?
PROGRAM NAME

This will be the name that appears as option 2 on the menu.

Name of chained cartridge?
DSK4.BIGCART-8.BIN

The chained cartridge will be saved as BIGCART-8.BIN

CHAINCARTS will start loading the individual carts in the order A,B,C,D, and continue until it comes to one that that does not exist. Then the chained cartridge is saved and the program ends. If necessary the last cart will be saved repeatedly to make a total of 1, 2, 4, 8, 16, or 32 carts in the combination cart.

IS THERE ENOUGH MEMORY?

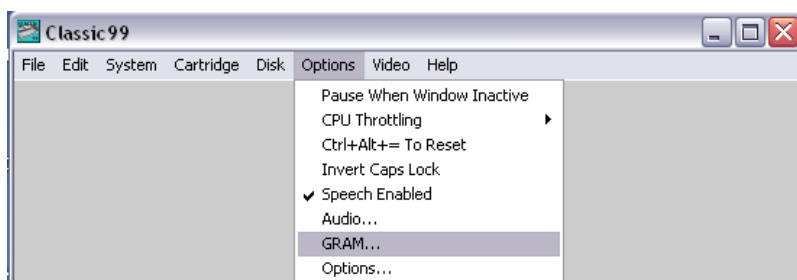
The loader reports how many bytes of memory are left when the compiled code is embedded in an XB loader. There are 56 additional bytes if you run the program as EA5. This remaining memory is used for string variables and the subroutine stack, and you must be sure there is enough room. Unfortunately, there is no hard and fast rule to know how much memory you need, and the compiler has no way to determine that. It mostly depends on how many string variables you use and how long they are, although subroutines and subprograms also use some memory. The compiler will do a “garbage collection” as necessary to clean out redundant string variables, just like XB does but faster. Most of the suggestions for saving stack space in the XB256 manual also apply to compiled code.

If you run short of memory, the compiler has an option to put the runtime routines into low memory. This frees up from 6K to 8K of memory. With the runtime routines in low memory, Extended BASIC will use the slow GPL assembly loader built into XB. If you are using XB 2.9 G.E.M. the files will load about 20x faster. When the runtime routines are in low memory, two files are created when saving as an XB program (-X and -Y). The program with -X must be run first. It loads the runtime routines to low memory, then loads and runs the -Y program from DSK1. You can modify the drive number if desired. This is not as tidy as having everything contained in a single program, so putting the runtime routines in low memory should only be done if the program is so large to load normally.

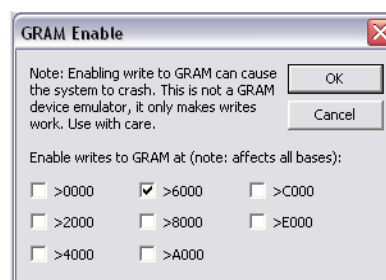
Enable GRAM at >6000

Very rarely XB256 will crash at startup. This seems to happen more frequently when using CPU overdrive, especially on newer, faster computers.. I believe that this is caused by the interrupt routine which forces XB to reserve more VDP ram than is normal. At CPU overdrive speeds the program “gets ahead” of the interrupt routine which confuses XB. If this is a problem for you, an option is to enable GRAM at >6000. At startup, XB256 tries to alter a pointer in the XB grom at >690D and >690E. This value can only be changed if the GRAM is enabled. The new value lets XB reserve the necessary vdp ram without having to use the interrupt routine and eliminates the stability issues when using CPU overdrive.

This step is entirely optional - if you do not enable gram XB256 will work as it always has.



Choose “Options” then “GRAM”



Check >6000, then OK

In the current version of Classic99, there is a minor bug in the GRAM enable screen. After you check >6000 and OK, if you bring up the GRAM enable screen again the check mark is gone. The GRAM is still enabled but there is no indication of that. Until you feel comfortable with this, when you run XB256 you can get into the debugger and display gram at >690D. If it is still >0958 then gram is not enabled.

The change to XB is not permanent, but it will remain in effect until you either change cartridges or close and restart Classic99. If you restart Classic99 you will have to re-enable the gram again.

Appendix A - Files used in XB Game Developer's Package

LOAD	menu program
XB256 COMPRESS SLCOMPILER SLCONVERT	(all are used by XB256)
COMPILER ASSEMBLER L24 L32A and L32B L24AL1 and L24AL2 MAKECART8 MKCART8.OBJ MAKECARTG CHAINCARTS RUNTIME1-3.TXT RUNTIME4.TXT RUNTIME5.TXT RUNTIME6.TXT RUNTIME7.TXT RUNTIME8.TXT RUNTIME9.TXT RUNTIME10.TXT YRUNTIME1.TXT SINE255 SOUNDLIB.TXT UC2LC and LC2UC FIXAL T40XBC T80XBC TMLC XXBC MCOLORC	All below are used by the compiler) assembler ported from Editor/Assembler high memory loader low memory loader part A and B loader for XB+AL progs part 1 and 2 Make rom cartridge from compiled program assembly subs for MAKECART8 Make grom cartridge from compiled program Combine rom cartridges into a multi cartridge XB routines (3 runtime files) XB256 routines (runtime) StarWars scroll (runtime) CHSETD routines (runtime) disk access routines (runtime) speech support (runtime) assembly support (runtime) CALL MOVE, LDCR, STCR from XB 2.9 runtime routines for old BASIC compiler table of sine values crash and chime sounds from EA manual converts upper case to lower case or vice versa fix assembly subs for compatibility T40XB modified for compiler T80XB modified for compiler The Missing Link modified for compiler XXB modified for compiler Multicolor routines for compiler
XB32K.OBJ	Lets you write an XB program using all 32K
APERTURE 256DEMO 256DEMO2 HELLO 8QUEENS	(demo programs)
HMLOADER	High memory loader for A/L subroutines
Folder - TMLSOUNDPLAYER	Sound lists for TML – use in XB & compiled
Folder - FLICKERROUTINE	XB256 flicker routine – use in XB & compiled
Asm994a.exe	Asm994a assembler (windows program)

XB 2.9 G.E.M. folder

XB29GEMDOCS 60FONTS XB29GEM_G.BIN XB29GEM_8.BIN	Documents for XB 2.9 G.E.M. Sixty fonts GROM files for XB 2.9 G.E.M. ROM files for XB 2.9 G.E.M.
--	---

Appendix B – Customizing the menu program

Turn off autocomplete in XB and XB256

You can temporarily turn off autocomplete by pressing F3 at the OLD DSK prompt. If you find that you prefer to not use the autocomplete feature at all, it can be permanently turned off.

For Extended BASIC, remove the underlined/highlighted text in line 180 of LOAD

```
180 CALL LOAD(-7,3,2):: CALL LOAD(8192,255,152):: CALL LINK("X"):: CALL LINK("ACON")::  
END
```

For XB256, remove the underlined/highlighted A in line 10 of XB256 so it is CALL LINK("XB256")

```
10 CALL INIT :: CALL LOAD(8192,255,152):: CALL LINK("X"):: CALL LINK("XB256A"):: END
```

Setting the defaults in the menu program

You can change two of the start up defaults in the menu program. In line 100, D tells the compiler the disk number where the runtime routines will be found. A positive number will be used as the disk number prompt; the default is 1. If it is -1 then the compiler assumes you are using Asm994a.

```
100 CALL CLEAR :: CALL INIT :: CALL PEEK(-8,D,R):: IF D=0 THEN D=1
```

When the menu program first runs the default option is XB256, but you can make it Extended BASIC. In line 101, R sets the default row at startup. R=1 is XB256; R=2 is Extended BASIC.

```
101 IF R=0 THEN R=1 :: CALL LOAD(-8,D,R)!D=DSK#,R=1:XB256;R=2:XB
```

Boot tracking

Boot tracking was causing some erratic behavior even when running only on DSK1, so it has been disabled. Now you *must* run the Game Developer's Package from DSK1.

There are three more BASIC and XB256 programs on the disk that you can practice compiling.

8QUEENS – The classic chess problem where you put 8 queens on the board such that no queen can capture any other queen. This BASIC program was one of the first to be compiled.

256DEMO (for XB256)

256DEMO2 (for XB256) This uses compressed DATA statements to show the power and speed of that technique while running in XB

APERTURE by Adamantyr. I have modified APERTURE to be compatible with the compiler.

Unfortunately, my changes have made it so it will not run in TI BASIC, but it should run in RXB or XB256 G.E.M. if you want to try it out uncompiled.

Appendix C - Listing of “HELLO”

```
5 ! HELLO WORLD
10 A$=" Hello World!"
15 CALL CLEAR
20 FOR ROW=1 TO 24 :: FOR COL=1 TO 32
30 R=ROW :: C=COL
40 FOR I=1 TO LEN(A$)
50 CALL HCHAR(R,C,ASC(SEG$(A$,I,1)))
60 C=C+1 :: IF C<33 THEN 100 :: C=1 :: R=R+1 :: IF R<25 THEN 100 :: R=1
100 NEXT I
110 NEXT COL :: NEXT ROW
120 GOTO 20
```

Appendix D

Using the Game Developer's Package with a real TI-99/4a

I have not included the files needed to run on a real TI-99/4a. I can do that if there is any demand for this, but not until it has been tested throughly and found to work properly, I