EXTENDED BASIC GAME DEVELOPERS PACKAGE
“Isabella”

by Harry Wilhelm

3/12/2019

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 hilight 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 autoloads, 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 to “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.

SETTING UP THE GAME DEVELOPER'S PACKAGE

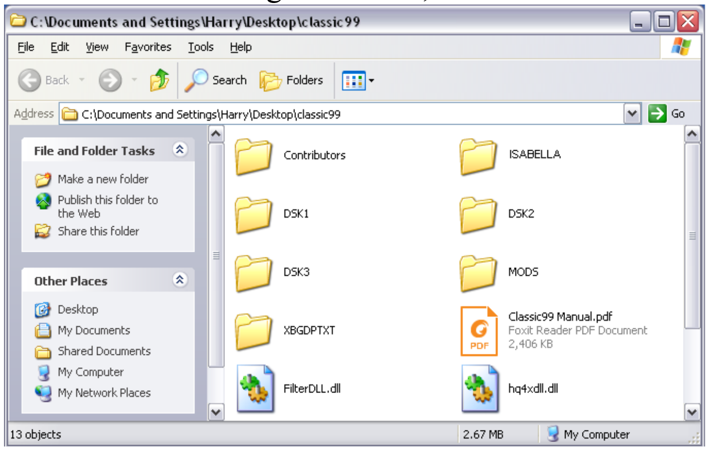
The package consists of one folder (ISABELLA) with two sub-folders (DOCS and ISABELLA99)

ISABELLA 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.

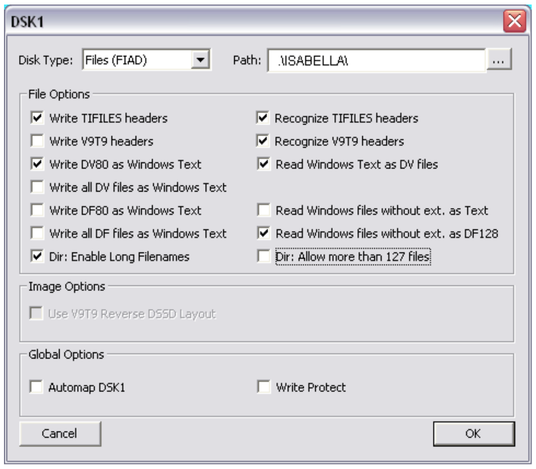
DOCS is a subfolder containing the documentation for the game developer's package.

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 ISABELLA into the classic99 folder alongside DSK1, DSK2 and DSK3.



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



DSK1

Disk Type: Files (FIAD)   Path: .\ISABELLA\

File Options

- ✓ Write TIFILES headers
- ☐ Write V9T9 headers
- ✓ Write DV80 as Windows Text
- ☐ Write all DV files as Windows Text
- ☐ Write DF80 as Windows Text
- ☐ Write all DF files as Windows Text
- ✓ Dir: Enable Long Filenames
- ✓ Recognize TIFILES headers
- ✓ Recognize V9T9 headers
- ✓ Read Windows Text as DV files
- ☐ Read Windows files without ext. as Text
- ✓ Read Windows files without ext. as DF128
- ☐ Dir: Allow more than 127 files

Image Options

- ☐ Use V9T9 Reverse DSSD Layout

Global Options

- ☐ Automap DSK1
- ☐ Write Protect

Cancel   OK

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.

Using the Game Developers Package

You must be using 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.

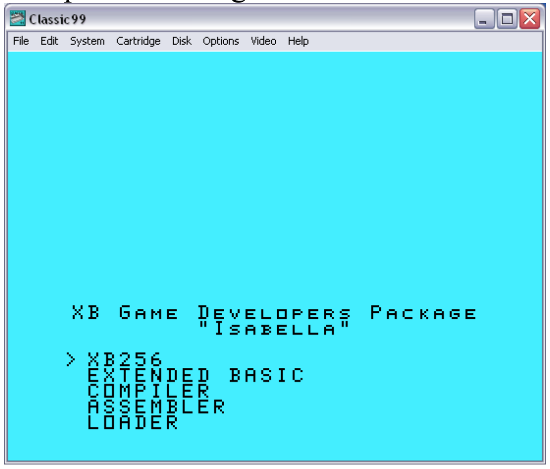
What just happened? Well, without having to know anything about assembly language you have just: Loaded an XB program, tested it, saved it, saved it in merge format, compiled the merge format XB program into assembly source code, assembled the source code into assembly object code, loaded the object 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.



```
XB GAME DEVELOPERS PACKAGE
         "ISABELLA"

> XB256
  EXTENDED BASIC
  COMPILER
  ASSEMBLER
  LOADER
```

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

XB256 – Loads XB256, 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.

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. At the first prompt, OLD DSK, you can turn off autocomplete by pressing F3. You know it is off when the cursor is solid black. There is no “undo” on the autocomplete routine.Once you press Enter, F3 or F4 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)

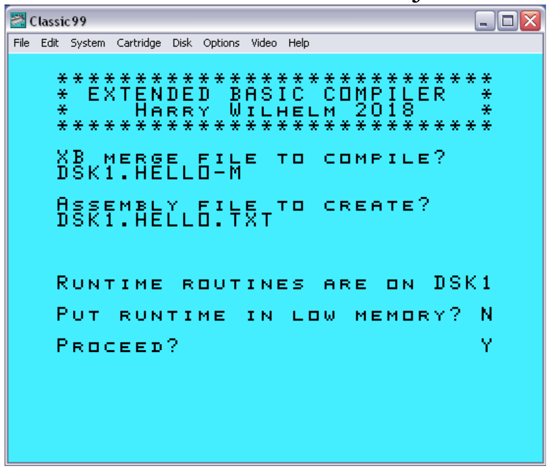
Then SAVE DSKn.FILENAME-M,MERGE is suggested. By now you know the drill: press Enter, change the name and press Enter, or press F4.

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 then press Y and Enter. If you are using Asm994A, be sure you have copied the runtime routines to your working disk as described in the compiler manual.
Put runtime in low memory? N This gives you the option of putting the runtime routines in low memory.. You should only choose this option of the program is too large to fit in high memory. If there is room for them in high memory the entire process of compilation is simplified. (After assembly, you will be loading the compiled program as described on page 8. If you choose N you must select 24K ram in the loader; if you choose Y you must select 32K in the loader. The pointer in the menu should help guide you.)
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:

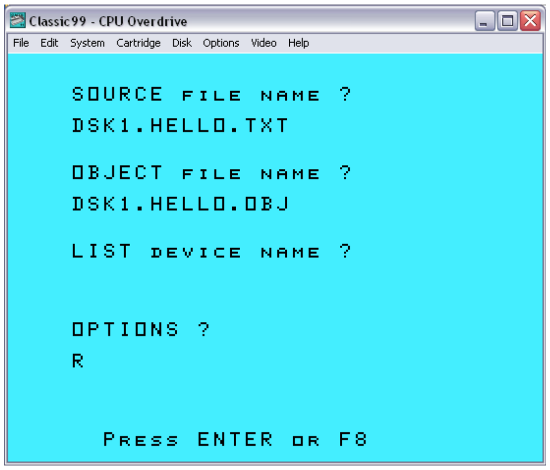


```
****************************
* EXTENDED BASIC COMPILER  *
*     HARRY WILHELM 2018    *
****************************

XB MERGE FILE TO COMPILE?
DSK1.HELLO-M

ASSEMBLY FILE TO CREATE?
DSK1.HELLO.TXT


RUNTIME ROUTINES ARE ON DSK1

PUT RUNTIME IN LOW MEMORY? N

PROCEED?                   Y
```

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.

ASSEMBLER – Runs the Assembler. This has been modified from the Funnelweb 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.


Classic99 - CPU Overdrive
File Edit System Cartridge Disk Options Video Help

```
SOURCE FILE NAME ?
DSK1.HELLO.TXT

OBJECT FILE NAME ?
DSK1.HELLO.OBJ

LIST DEVICE NAME ?


OPTIONS ?
R


  PRESS ENTER OR F8
```

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

If the assembly process goes well you get this message:

```
0000 ERRORS
Press ENTER or QUIT
```

After 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 RXB. 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, once 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"

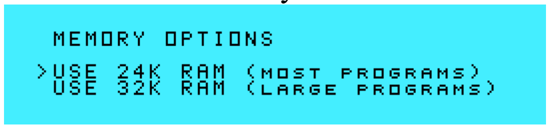
CROSS ASSEMBLERS

The assembler that comes with the Game Developer's Package is adapted from the assembler that was included in the TI Editor/Assembler package. It works fine and is quite useable, but it does take some time to assemble, especially for a large program.

If you want to speed up the development process you should take the time to learn how to use a cross assembler. Fred Kaal's XA99 is one possibility; xas99 by RalphB is another. I know nothing about either of these, so someone else would have to describe their use.

The one cross assembler I have first hand knowledge of is Asm994a, which is part of the Win994a package. Asm994a is my preferred method for assembling a compiled program because the assembly process happens almost instantaneously and error messages are much more descriptive. I have devoted a few pages in the compiler documentation to describe how to use Asm994a.

LOADER – When you select LOADER you get these options:

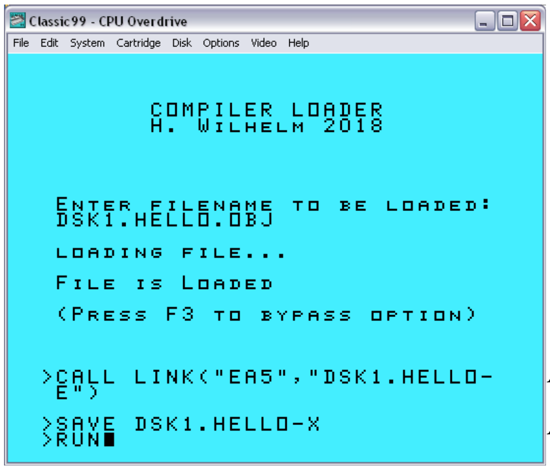

```
 MEMORY OPTIONS
>USE 24K RAM (MOST PROGRAMS)
 USE 32K RAM (LARGE PROGRAMS)
```

USE 24K RAM – This runs the Loader. CLOADER uses the assembly loader from the EA cartridge instead of the GPL loader in the XB cartridge, so it is *much* faster. The loader prompts you to:
Enter filename to be loaded: The filename in the mailbox will be suggested with the .OBJ extension To load the file, press Enter, or change the filename or type in a new one and press Enter.

After the file has been loaded three prompts are automatically suggested in the XB command line. You can press F3 or F4 to skip any of the three options.
CALL LINK("EA5","DSKn.FILENAME-E") Press Enter to save the compiled program in EA5 format.

SAVE DSKn.FILENAME-X Press Enter to save the compiled program embedded in an XB loader.

RUN P.ress Enter to run the compiled program. (If Classic99 is in CPU overdrive you should return to normal speed)
Autocomplete has no "undo." Once you press Enter, F3 or F4 you cannot return to that option.


Classic99 - CPU Overdrive
File Edit System Cartridge Disk Options Video Help

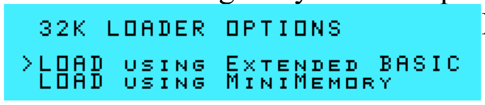
```
      COMPILER LOADER
      H. WILHELM 2018

  ENTER FILENAME TO BE LOADED:
  DSK1.HELLO.OBJ
  LOADING FILE...
  FILE IS LOADED
  (PRESS F3 TO BYPASS OPTION)

 >CALL LINK("EA5","DSK1.HELLO-
  E")
 >SAVE DSK1.HELLO-X
 >RUN
```

Autoprompt to save in EA5 format

Autoprompt to save in an XB loader

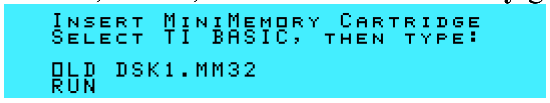
Depending on the length, the EA5 program will be saved in from one to four sequential files (-E,-F,-G,-H).. The XB program will always be saved in one file (-X) unless you selected low memory runtime routines, in which case there will be two files (-X,-Y). If you copy them, be sure to copy all the files.

USE 32K RAM gives you these options: (When compiling you must choose "put runtime in low memory")


```
 32K LOADER OPTIONS
>LOAD USING EXTENDED BASIC
 LOAD USING MINIMEMORY
```
I

LOAD using Extended BASIC prompts you for a filename to load. This uses the slow assembly file loader built into XB. As with the standard loader, you can save the program in EA5 format and/or in XB format.

LOAD, SAVE, TEXT in MiniMemory gives you this screen:


```
 INSERT MINIMEMORY CARTRIDGE
 SELECT TI BASIC, THEN TYPE:
 OLD DSK1.MM32
 RUN
```

MM32 is a loader program that works the same as the loaders described above. This assembly language loader is much faster than the XB loader, but you have to swap cartridges to use it.

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 have been working on is still in the mailbox and will be suggested as the prompt when you select XB256 or XB.

IS THERE ENOUGH MEMORY?

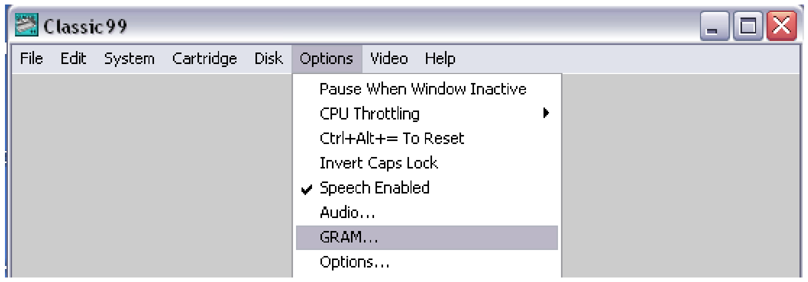
The loader will report how many bytes of memory are left when the compiled code is embedded in an XB loader. There are 56 additional bytes if you run the program as EA5. This remaining memory is used for string variables and the subroutine stack, and you must be sure there is enough room. Unfortunately, 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 gives you the option to put the runtime routines into low memory, which frees up from 6K to 8K of memory. When it is time to load the program, if you stay with Extended BASIC, the loader uses the slow GPL assembly loader built into XB. If you switch cartridges you can use the fast loader that is included with the MiniMemory cartridge. Two files are created when saving as an XB program (-X and -Y). Since this process is not quite as simple as the normal procedure, it should only be used if the program is so large the normal method does not work.

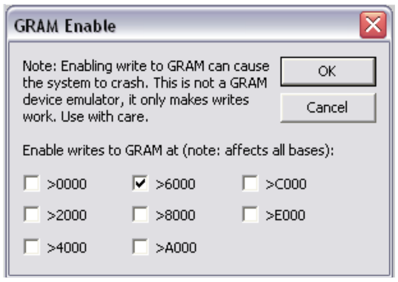
Enable GRAM at >6000

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"


GRAM Enable

Note: Enabling write to GRAM can cause the system to crash. This is not a GRAM device emulator, it only makes writes work. Use with care.

Enable writes to GRAM at (note: affects all bases):

☐ >0000 ☑ >6000 ☐ >C000
☐ >2000 ☐ >8000 ☐ >E000
☐ >4000 ☐ >A000

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. This means you have to re-enable the gram when you want to use this 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 CLOADER MB MM32 MM32ASSM.OBJ RUNTIME1.TXT RUNTIME2.TXT RUNTIME3.TXT RUNTIME4.TXT RUNTIME5.TXT RUNTIME6.TXT RUNTIME7.TXT RUNTIME8.TXT YRUNTIME1.TXT SINE255 SOUNDLIB.TXT XB32 XB32A.OBJ	(all below are used by the compiler) assembler from Editor/Assembler loads compiled and assembled program mailbox used to pass info to MiniMemory low memory loader for MiniMemory assembly support for MM32 XB routines (runtime) XB routines (runtime) XB routines (runtime) XB256 routines (runtime) StarWars scroll (runtime) CHSETD routines (runtime disk access routines (runtime) speech support (runtime) runtime routines for old BASIC compiler table of sine values crash and chime sounds from EA manual low memory loader for Extended BASIC object code for XB32
APERTURE 256DEMO 256DEMO2 HELLO 8QUEENS	(demo programs)
XB32K.OBJ	Lets you use all 32K for XB program
Asm994a.exe	Asm994a assembler (windows program)

Appendix B - 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 C – 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 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("XB256**A**"):: 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

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 if you want to try it out uncompiled.

Appendix D

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

There may be those who prefer to use the entire XBGDP on a real TI99 computer. ISABELLA99 is a folder containing files modified to be be used on “real iron.” The procedure for use is essentially the same as described above. Copy the files onto a floppy disk and put it in disk drive #1. The main difference is that assembly source code must have a -S extension instead of .TXT, and assembly object code must have a -O extension instead of .OBJ. The autoprompts have been modified accordingly.

Something to keep in mind is that floppy discs have less capacity than the (almost) unlimited disc capacity of Classic99. If necessary, you can remove some of the files to open up more space. The table below ranks the files in order of importance, with A being essential and E being totally optional.

Files in XB Game Developer's Package for real TI-99/4A (ISABELLA99)

LOAD	A	menu program
XB256 COMPRESS SLCOMPILER SLCONVERT	A B B B	(all are used by XB256)
COMPILER ASSEMBLER CLOADER MB MM32 MM32ASSM-O RUNTIM1-S RUNTIM2-S RUNTIM3-S RUNTIM4-S RUNTIM5-S RUNTIM6.-S RUNTIM7-S RUNTIM8-S YRUNTIM1-S SINE255 SOUNDLIB-S XB32 XB32A-O	A A A C C C A A A A A A A A D B B C C	(all below are used by the compiler) assembler from Editor/Assembler loads compiled and assembled program mailbox used to pass info to MiniMemory low memory loader for MiniMemory assembly support for MM32 XB routines (runtime) XB routines (runtime) XB routines (runtime) XB256 routines (runtime) StarWars scroll (runtime) CHSETD routines (runtime disk access routines (runtime) speech support (runtime) runtime routines for old BASIC compiler table of sine values crash and chime sounds from EA manual low memory loader for Extended BASIC object code for XB32
APERTURE 256DEMO 256DEMO2 HELLO 8QUEENS	E E E E E	(demo programs)
XB32K-O	E	Lets you use all 32K for XB program
ISABEL-ARC	F	All 30 files above archived into one file

A – essential files
B – Needed for sound list and compressed files.
C – Only used when loading runtime into low memory.
D – Only used if you want to use original TI BASIC compiler.
E – Demo programs – totally optional.
F –An alternative is to use Archiver 3.03 to extract all 30 files from ISABEL-ARC.