

# TiCODED

&

STRUCTURED EXTENDED BASIC

BY STEFAN 'STEVEB' BAUCH

VERSION 3.0 (C)2025

TICODED <AT> LIZARDWARE.DE

# Table of Content

What is TiCodEd ?.....	3
What is Structured Extended Basic?.....	3
Installing TiCodEd.....	4
TiCodEd Projects.....	4
Writing SXB Code.....	6
Exporting SXB Code.....	6
The Charset Editor.....	7
Structured Extended Basic.....	9
Using REPEAT .. UNTIL.....	9
Using WHILE .. ENDWHILE.....	10
Using Labels and Line-Numbers.....	10
Hexadecimal numbers.....	11
The CASE-Statement.....	11
Implicit and explicit Code-Blocks.....	12
Variables.....	12
Constants.....	12
IN Set Condition.....	13
The BIN\$ Function.....	13
Line Continuation.....	13
Editor Features.....	13
Automation and Integration.....	14
Porting existing XB code to SXB.....	16
Extended Basic Version.....	16
Extended BASIC Compiler Considerations (JEWEL).....	16
Limitations.....	17
Embedded Assembler.....	17
Trouble-Shooting.....	18
Found an Error? Having a suggestion? Future plans?.....	18
Building TiCodEd from Source.....	19
Use LibXBTKN32.dll or LibXBTKN64.dll.....	19
BSD License.....	20
Acknowledgements.....	20
Appendix A – Change-Log.....	21
Appendix B – Standard Subroutine Library.....	24
Appendix C – Extension Packages.....	25
XB256.....	25
T40XB.....	25
T80XB.....	26
TML – The missing link.....	26
RXB 2023.....	26
Extended Basic 2.9 G.E.M.....	26
Extended Basic IIplus.....	26
MultiColor.....	26
Classic99dbg.....	26
Appendix D – Example files.....	27
Appendix E – Keyboard Emulation.....	30
Appendix F – Embedded Utilities.....	31
Hex2BIN\$ - Extract code from text.....	31
TIFILES Editor.....	32
Patch Cart Utility.....	33
Appendix G – Patches.....	34
jewel7.patch.....	34
no-auto.patch.....	34
overdrive.patch.....	34

## What is TiCodEd ?

When I discovered the fabulous [ISABELLA](#) Extended BASIC Compiler I finally found a way to complete my TI-99/4a game programs I had abandoned for quite some years, because the interpreted basic was so slow.

I was disappointed that working with an emulator still meant using the limited edit functions of the TI. I discovered [TidBiT](#), a PHP program to get rid of the line numbers in Extended Basic and replace them with labels. So I started with an PC text editor, pasted the text to TidBiT, converted the program there to standard Extended Basic and pasted it to the [Classic99 emulator](#).

Some say, laziness is the mother of all inventions. I wanted to have a simple way of writing code in a modern environment and test it in an emulator.

TiCodEd (say it like 'decoded', only with a 't' in the beginning: 'TEE-coded') is the TI Code Editor for this. It offers:

- Modern Basic without line numbers
- Translation to Standard Extended Basic
- Saving of files in tokenized format in FIAD
- Usage of assembly libraries like XB256, T40XB, TML and others

Once you wrote your code, just click Export/Build Project, switch to the emulator, OLD DSK1.YourProg and RUN ... or automate this.

Your program is working perfect? TiCodEd can also write the program in MERGE format for direct use in the XB Compiler, manually or automated.

## What is Structured Extended Basic?

I found TidBiT quite useful and ported it to Pascal, to be used in the free [Lazarus](#) IDE. I learned a lot by doing so, but one finding was, that this was not suitable to be extended for structured elements I wanted to use in my programs:

- REPEAT ... UNTIL
- WHILE ... ENDWHILE

Those two forms of loops, head controlled WHILE and tail controlled REPEAT are making GOTO redundant. Labels are supported for all statements where line numbers are used in Standard Extended Basic, i.e. GOTO, GOSUB, RESTORE.

You can now write programs like

```
DATA "THIS IS A TEST ", DOES IT WORK? , "END"
REPEAT
  READ A$
  PRINT A$
UNTIL A$="END"
END
```

TiCodEd translates this to

```
100 DATA "THIS IS A TEST ", DOES IT WORK? , "END"
110 READ A$
120 PRINT A$
130 IF NOT (A$="END") THEN GOTO 110
140 END
```

which can be run with the regular Extended Basic Module.

## Installing TiCodEd

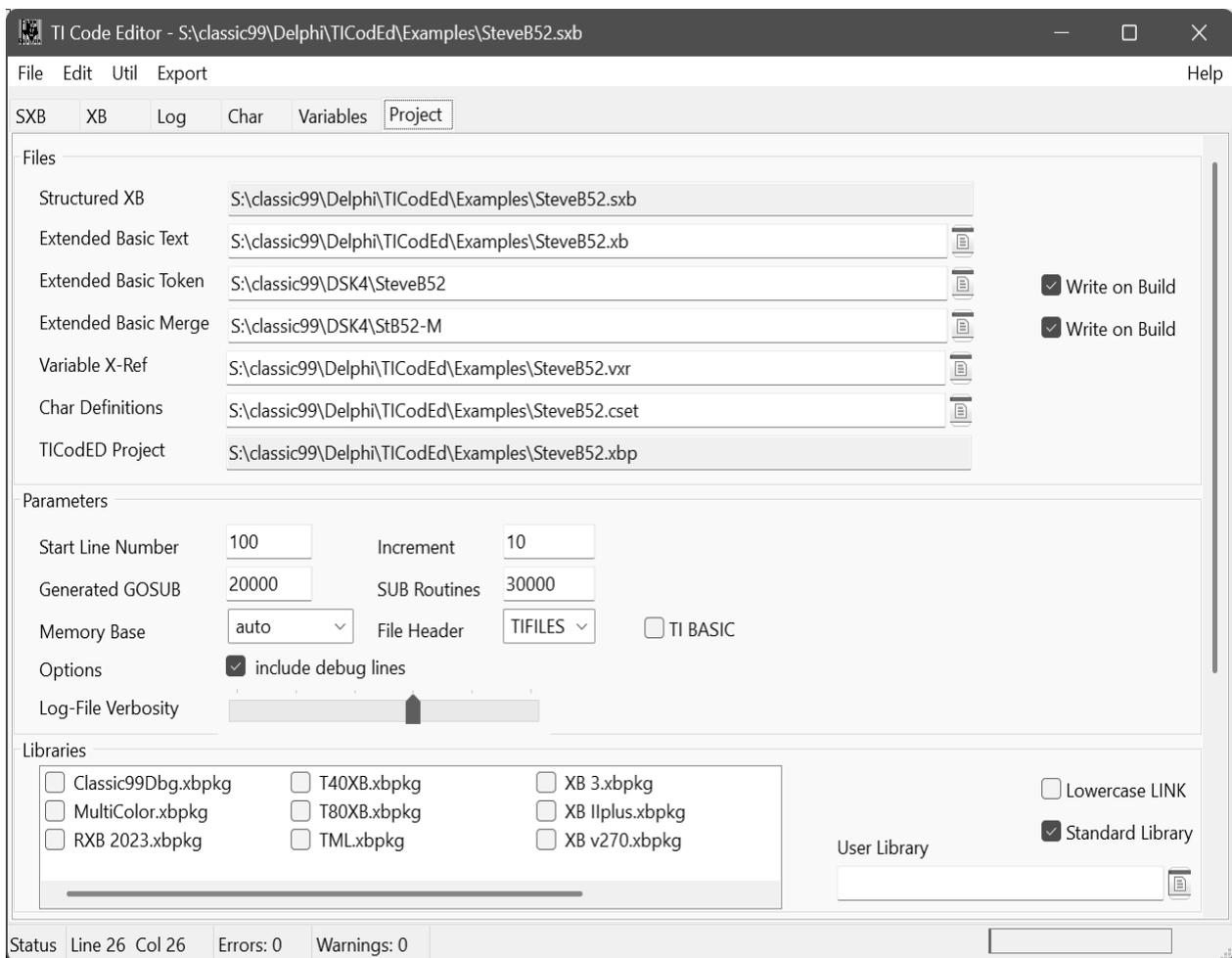
Just download the ZIP file and unpack it to a folder of your choice. As you are reading this documentation you figured this out for yourself, didn't you?

You may associate the SXB extension (Structured Extended Basic) with TiCodEd by double-clicking an SXB-File, search for the TiCodEd.exe file and select always to use this application.

## TiCodEd Projects

When you open an SXB file or create a new one, TiCodEd automatically creates project file in the same directory with the extension XBP (Extended Basic Project).

The project file contains the options for your project and is maintained on the Project page.



Currently the following options are available:

- Which Standard Extended Basic Text file will be written. ('Write on Build' is always checked as this file is mandatory)
- Which Standard Extended Basic Token file will be written (i.e. to a FIAD directory) and this should be written when the 'Build Project' or

'Tokenize' are selected in the Export menu.

- Which Standard Extended Basic Token MERGE file will be written (i.e. to a FIAD directory) and this should be written when the 'Build Project' or 'Tokenize' are selected in the Export menu. Very useful when you plan to compile the program using the XB Compiler.
- The Memory Base defaults to "auto", where larger programs will be saved in IV254 format. This can also be forced or the memory base can be set to CALL FILES 0 to 3, where 0 is invalid for real hardware. CALL FILES will never use IV254 format for saving, programs must fit into VDP RAM. When using libraries with additional VDP RAM requirements like XB256, use IV254.
- Check TI BASIC if you need byte-identical files for TI BASIC. If not checked for TI BASIC the created files are valid, but differ slightly.
- Select TI-FILES or V9T9 format for your tokenized files.
- Start Line Number and Increment when creating the Standard Extended Basic program from the Structured Extended Basic program, which usually has no line numbers as there are not needed.  
Note: Scattered line numbers in the SXB file may be used to create logical blocks, see Page 10, Using Labels and Line-Numbers.
- Generated GOSUB- and SUB-Routine Line Numbers (SUBs must be the last statements in any XB program).
- Debug: Lines starting with a hash # will be included for debugging when box is checked, otherwise they will be commented //.
- Log-File Verbosity configures how many messages written to the Log page
  - Error - Only Errors are shown
  - Warning - Errors and Warnings are shown
  - Information - Errors, Warnings and Information are shown
  - Verbose - Also included debug information
  - Very Verbose - More Debug information
  - Unbelievable Verbose - Debug down to the bits

Place the mouse over the scale to get a pop-up of the selected level.

- Libraries
  - Extension Package: Provides simplified CALL routines to popular extension packages and libraries with additional functions. (see page 25, Appendix C – Extension Packages)
  - User Library: Include a text-file with your favorite SUB-Routines to be automatically appended to your program when used.
  - Standard Library: Commonly used functions, still under development (see page 24 Appendix B – Standard Subroutine Library)
  - Lowercase LINK: When compiling with an assembler support library the procedure names need to be in lowercase.
- Post-Processing: A command issued after a successful build-process. The

following variables will be substituted before execution:

- %SXB% - Filename of the SXB file
- %XBT% - Filename of the Extended BASIC Text file
- %XB% - Filename of the Extended BASIC Token file
- %XBM% - Filename of the Extended BASIC Merge file
- %XBP% - Filename of the Project file
- %TIME% - Current time
- %DATE% - Current date

## Writing SXB Code

The SXB page is the actual editor with syntax highlighting. SXB Keywords are bold black, literals and symbols are simple black, identifier red, strings magenta and comments are in turquoise.

Use File/Open to load the demo SPEEDY.SXB, a very simple game to demonstrate the main concepts of SXB.

For an introduction to Structured Extended Basic see page 9 and the provided examples.

## Exporting SXB Code

Once you finished you code, or at least reached a state you want to test, you can click Export/Build Project or start the steps separately, first 'SXB to XB' and then 'Tokenize'. Export/TIDBit starts the port of TIDBit to create the XB file.

Depending on your project settings a tokenized and/or tokenized MERGE file will be written.

If you mount for example DSK4 in Classic99 to your can just type OLD DSK4.<program> and RUN it.

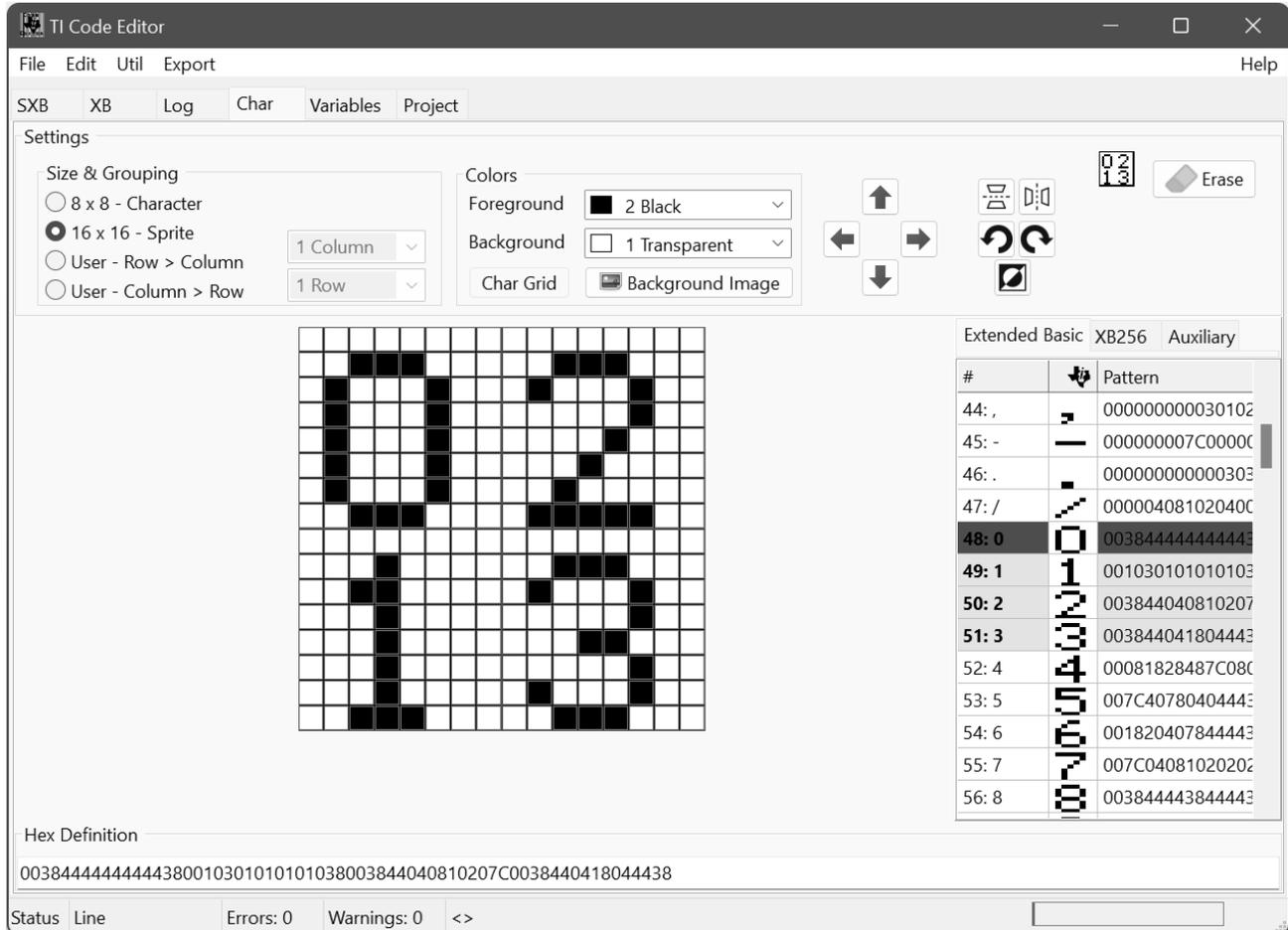
You will find the log of the conversion on the Log page and the Standard Extended Basic file on the XB page, which is read-only by default, but becomes editable double-clicking the text. This might be useful when you want to make a small adjustment and then click Export/Tokenize to create the TI files. The case will be adjusted in the tokenization and remains intact in the XB page.

The Log can be cleared or saved to a file by a left button click on the log text for the context menu.

Please note that TI Files resulting from the export are binary files with the [TIFILES header](#) used by most emulators and may get corrupted when opening with a text editor, but may be viewed or changed with a Hex-Editor like [XVI32](#). I recommend [Ti99Dir](#) by Fred Kaal to manipulate TI files.

# The Charset Editor

One common task in game development is the definition of the graphics. The Char page offers a powerful tool to help with this. You can choose between 8x8 pixel and a 16x16 character definition for single characters or Magnify 3 or 4 groups of four characters, or free defined areas of up to 16x8 characters. In order to support the design you can choose the fore- and background color and see a small preview.



The resulting definition string will be updated with every click you make and can be copied with Ctrl-C to your program. The string is also editable, you can change it any time or paste a hex-string. Illegal, non-hexadecimal characters will be erased when you change focus to another element and leave the edit field, it is even possible to paste a complete CALL CHAR statement.

On the right side of the screen you see a charset table with three tabs:

1. Extended Basic: 32 to 159 (#CHAR1xxx)
2. XB256: 0 to 255 (#CHAR2xxx)
3. Auxiliary: 0 to 255 (#CHAR3xxx)

The first column is the character number and the printable character when available, the second is the preview and the third is the Hex-Definition. You may edit or enter valid hex codes directly in this table.

The "Size & Grouping" setting allows you to group chars as needed. Groups can be selected by clicking on any char within the group. The first line of the group is marked in red, the remaining lines in yellow. Inactive groups are indicated with a bold character number and the first line of a group has a gray background. Regrouping is always possible and does not change any pattern.

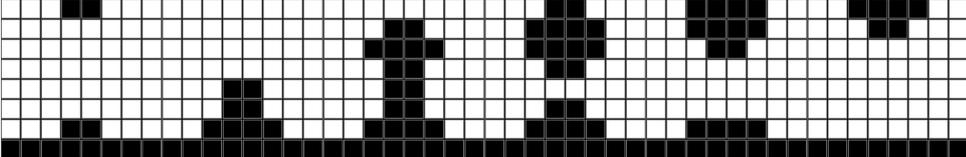
By selecting "Background Image" you may load and unload a graphic in the transparent background as a pattern to copy.

There are buttons to shift the image (green arrows), flip horizontally or vertically, turn 90 degrees left or right or invert the image.

The charset editor is linked to your SXB program in three ways:

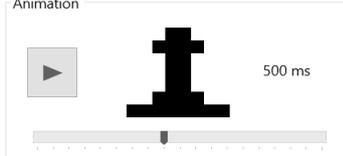
1. CALL AUTOCHAR – Use this subroutine in your program to dynamically create a subroutine defining all chars (re-)defined in the "Extended Basic" and "XB256" tab, the first by using CALL CHAR, the second by using CALL LINK("CHAR2",..) when building your project.
2. You can refer to any char in the three tabs with the literal #CHARnxxx or #CHARnxxx:z , for example as CALL CHAR(132,#CHAR3012:4) which will refer to chars 12 to 15 from the third tab "Auxiliary".
3. You can dump the font (#32-127) or the whole character set as shown in the table to a file using the context menu on table on the right. You may load this file at runtime or attach it to your compiled program using the Cart Patching utility, saving the CALL CHAR space.

For sizes 1xN or 2x2N an additional section for animations appears:



Extended Basic		XB256	Auxiliary
#	Pattern		
17	0F01010100000000		
18	000000000101010F		
19	18000000000018FF		
20	0000000018183CFF		
21	00183C1818183CFF		
22	183C3C1800183CFF		
23	3C3C18000003CFF		
24	3C180000000000FF		
25	FF818181818181FF		
26	FF00000000000000		
27	0000000000000000		
28	0000000000000000		
..	.....		

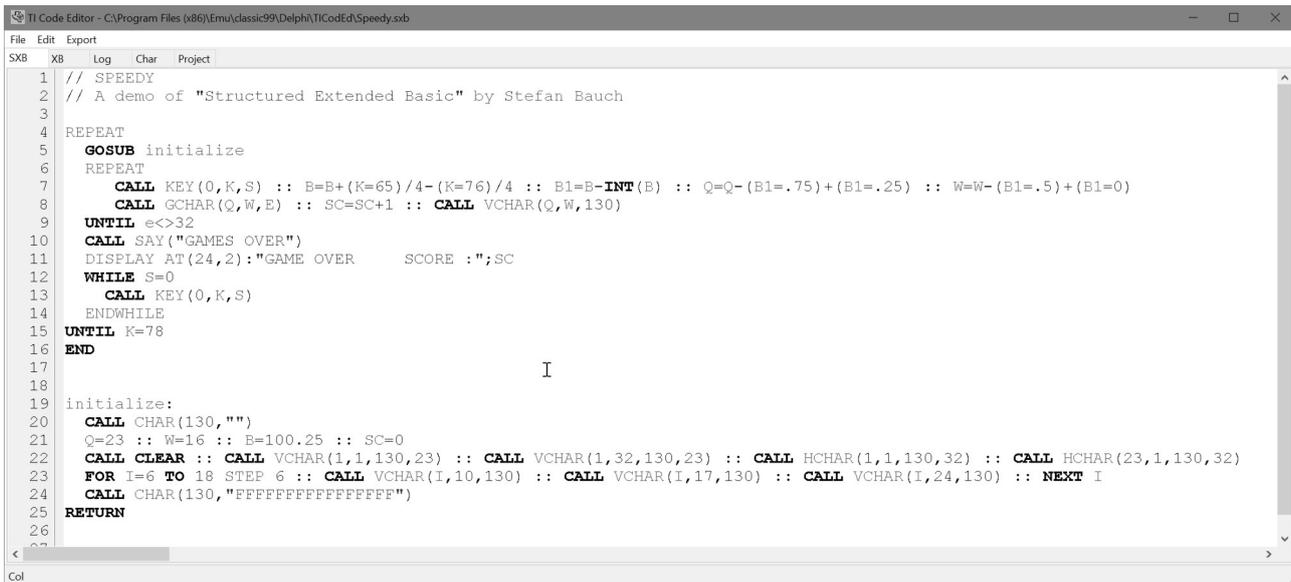
Animation



You can play the animation while editing to see the result immediately.

# Structured Extended Basic

Let's start with a small included demo game called „Speedy“.

The screenshot shows a TI Code Editor window titled "TI Code Editor - C:\Program Files (x86)\Emu\classic99\Delphi\TICodEd\Speedy.sxb". The editor displays the source code for a game called "Speedy". The code is structured with comments, subroutines, and loops. Line 19 is labeled "initialize:". The code includes a REPEAT loop for initialization, a WHILE loop for game logic, and a FOR loop for character placement. The code is as follows:

```
1 // SPEEDY
2 // A demo of "Structured Extended Basic" by Stefan Bauch
3
4 REPEAT
5   GOSUB initialize
6 REPEAT
7   CALL KEY(0,K,S) :: B=B+(K=65)/4-(K=76)/4 :: B1=B-INT(B) :: Q=Q-(B1=.75)+(B1=.25) :: W=W-(B1=.5)+(B1=0)
8   CALL GCHAR(Q,W,E) :: SC=SC+1 :: CALL VCHAR(Q,W,130)
9 UNTIL e<>32
10 CALL SAY("GAMES OVER")
11 DISPLAY AT(24,2):"GAME OVER   SCORE :";SC
12 WHILE S=0
13   CALL KEY(0,K,S)
14 ENDWHILE
15 UNTIL K=78
16 END
17
18
19 initialize:
20 CALL CHAR(130,"")
21 Q=23 :: W=16 :: B=100.25 :: SC=0
22 CALL CLEAR :: CALL VCHAR(1,1,130,23) :: CALL VCHAR(1,32,130,23) :: CALL HCHAR(1,1,130,32) :: CALL HCHAR(23,1,130,32)
23 FOR I=6 TO 18 STEP 6 :: CALL VCHAR(I,10,130) :: CALL VCHAR(I,17,130) :: CALL VCHAR(I,24,130) :: NEXT I
24 CALL CHAR(130,"FFFFFFFFFFFFFFFF")
25 RETURN
26
27
```

You can see the program structure just by looking at the optional indentation. In the first lines you see comments. Everything after the // is ignored to the end of the line and not included in the exported XB file. REM or ! may be used to have persistent comments in XB.

A leading # indicates a Debug-Line. Depending on the checkbox *Debug* on the project page the line gets included (checked) or dismissed (unchecked). This way you can include additional functions during development and just switch them off when you build a release version.

Blank lines will be ignored and may be used for grouping of lines.

The program contains labels (in line 19, used in line 5) and structured loops, REPEAT .. UNTIL and WHILE .. ENDWHILE, discussed in detail in the next two paragraphs. These loops may be nested like FOR..TO..NEXT.

For Structured Extended Basic it is mandatory to have these statements not combined with other statements on the same line, as it is common practice for a better readability of the source code.

[for playing Speedy use A key to turn left, L key to turn right, N key for end.]

## Using REPEAT .. UNTIL

REPEAT .. UNTIL is a foot-controlled loop. This means, the code is executed at least once. The condition at the end of the block determines whether to repeat the section starting with REPEAT or leave it.

The loop will be translated in two steps as follows:

```
REPEAT
  <CODE>
UNTIL <CONDITION>
```

### Step #1: Create Labels

```
REPEAT001:  
<CODE>  
IF NOT <CONDITION> THEN REPEAT001
```

### Step #2: Assign Line-Numbers

```
100 <CODE>  
110 IF NOT <CONDITION> THEN 100
```

You may use any valid XB code and any valid XB condition. Check the SPEEDY.SXB example and compare the SXB with the XB page.

## Using WHILE .. ENDWHILE

WHILE .. ENDWHILE is a head-controlled loop. This means, the code may be omitted completely if the condition is not met.

The loop will be translated in two steps as follows:

```
WHILE <CONDITION>  
  <CODE>  
ENDWHILE
```

### Step #1: Create Labels

```
GOTO ENDWHILE001  
WHILE001:  
<CODE>  
ENDWHILE001:  
IF CONDITION THEN WHILE001
```

### Step #2: Assign Line-Numbers

```
100 GOTO 120  
110 <CODE>  
120 IF <CONDITION> THEN 110  
130 ...
```

This has been changed since version 1.0. The line 100 is not an 'IF' in order to save memory and results in one additional jump when the condition is true.

As many BASIC variants use WHILE .. WEND it is also supported to use WEND instead of ENDWHILE.

## Using Labels and Line-Numbers

You may use Labels in your program anywhere in your Basic program where line-numbers are used in Standard Extended Basic, i.e. GOTO, GOSUB, RESTORE etc.

Labels are defined by a name, followed by a colon in the first column of your program. Please make sure not to use any reserved words or use a label more than once.

The label is referenced without the colon.

### Example:

```
GOSUB INITIALIZE
END
INITIALIZE:
  <CODE>
RETURN
```

Labels may be used in ON GOTO and ON GOSUB, i.e. 'ON A GOSUB LB01,LB02,LB03'.

You can always use line numbers as in Extended Basic. If the internal line-number is lower than the explicit line-number, the internal number will be adjusted, otherwise the internal number will be used. This makes sure that line-numbers are in ascending order, but may break existing GOTO/GOSUB/etc. statements using the explicit number.

It is generally depreciated to use line-numbers. For a better readability of the generated Extended Basic source it is suggested to use scattered line-numbers for blocks of code to be found again easily on the TI. Do so by only entering a line-number on a line of its own or in front of a statement. This will offset the line-number, but keep the increment, i.e. GOSUB routines start at line 10000, DATA statements at 20000 and SUB routines at 30000.

## Hexadecimal numbers

When using CALL LOAD, CALL MOVE, CALL POKEV etc. you need to address memory in the decimal notation, while many technical references are made for assembly language using the hexadecimal notation. As the TI typic > symbol is ambiguous, you may use the \$ sign and a two or four digit hexadecimal number wherever a number is required, i.e. CALL LOAD(\$3F7E,\$FF).

## The CASE-Statement

Extended Basic has ON x GOTO and ON x GOSUB statements to jump to different lines, but the value of x needs to be 1..n for n destination-lines.

With SXB you may now use a CASE statement for numeric and string variables with arbitrary values. Multiple values may be separated by comma.

```
CASE x OF
  10 : <STMT1>
  20 : <STMT2>
  30,40,50 : <STMT3>
  ELSE <STMT4>
ENDCASE
```

The statements may be a group of statements, separated by ::, but must fit on a line. For larger branches you may use BEGIN-END block, see next chapter. There is a CASE.SXB file in the examples. The ELSE branch is optional. Technically, the CASE will be translated to an ON x GOSUB with a complex boolean calculation of x. For this reason CASE statements may not be nested. Both calculation and line-number list must fit one XB line, limiting the number of branches. Be sure to use the ELSE option when you can not guarantee that the value is in your list, otherwise a "Bad Value" error will occur.

## Implicit and explicit Code-Blocks

Extended Basic has neither BEGIN/END like Pascal nor { } like C to build blocks of statements. But you may group some few statements in an IF-THEN-ELSE clause by using :: for multiple statements in one line, i.e.:

```
IF A<0 THEN <STMT1> :: <STMT2> ELSE <STMT3> :: <STMT4>
```

This technique is limited by the allowed length of the line in Extended Basic. It can be used in Structured Extended Basic as well and is the reason that in contrast to other programming languages the line has a meaning and may not be split without changing semantics. Each line with executable code in Structured Extended Basic will be one line in the resulting Standard Extended Basic code.

Additionally SXB now offers BEGIN-END blocks. It does so by fitting the block in a generated subroutine and use a GOSUB to call it. BEGIN-END blocks may be used in IF-THEN-ELSE and in CASE Statements. See the examples "Begin.sxb" and "Case.SXB". BEGIN-END blocks may be nested.

```
IF A<0 THEN BEGIN
  <STMT1>
  <STMT2>
END ELSE BEGIN
  <STMT3>
  <STMT4>
END
```

Starting TiCodEd 3.0 BEGIN/END can also be used in SUB-routines. You may exit a BEGIN/END block with RETURN. Caution: Leave BEGIN/END without RETURN does not clear the stack and may cause memory problems.

## Variables

Variables are stored as unquoted text, therefore the longer the names are, the more memory they need. As memory is valuable on the TI, often short variable names are used, which are hard to read and to remember. The Variable tab shows you all variables from the last build-process and the XB lines where they are used as a cross-reference. You are able to assign short-names to variables. They will replace the names in the SXB and XB page when exported in tokenized format, saving the memory in the TI. Checks for duplicates and string/number-mismatches are performed when entering the optional short-names.

## Constants

You may use named constants in SXB, which gets replaced when creating the XB code with the given value.

```
CONST GRAVITY=9.81
PRINT GRAVITY
Will lead to an XB line like
10 PRINT 9.81
```

Any key=value pair might be given after the CONST keyword, the value might also be a formula or a statement. The CONST keyword needs to stand alone in one line of code.

## IN Set Condition

When you want to test if a variable is in a group of values you can now use the IN function:

```
IF K IN[69,83,68,88] THEN <STMT1>
```

This will be translated to IF K=69 OR K=83 OR K=68 OR K=88 THEN <STMT1>. Strings can be used as well and "K NOT IN[...]" is supported. Please note that there must be exactly one space between NOT and IN.

## The BIN\$ Function

You can use the pseudo-function BIN\$("HEX-STRING") to include binary data in a string, for example to use it with CALL VWRITE. The function needs a static string in quotes as it is executed when tokenizing the code and not at runtime.

Under Util / Hex to BIN\$ you will find a utility to extract hex-code from a pasted text and create BIN\$-statements, i.e. from scans or listings.

## Line Continuation

One SXB line is usually one XB line. The following exceptions are implemented:

- Lines ending with THEN
- Lines ending with ELSE
- Lines ending with ..

will get the following line merged.

## Editor Features

There is a code-completion implemented for all keywords and standard routines. Press Ctrl-Space to get a list and select the word. Code completion works for statements, subroutines (standard, local and libraries), labels and variables with at least three characters. Dynamic attributes are based on the last build.

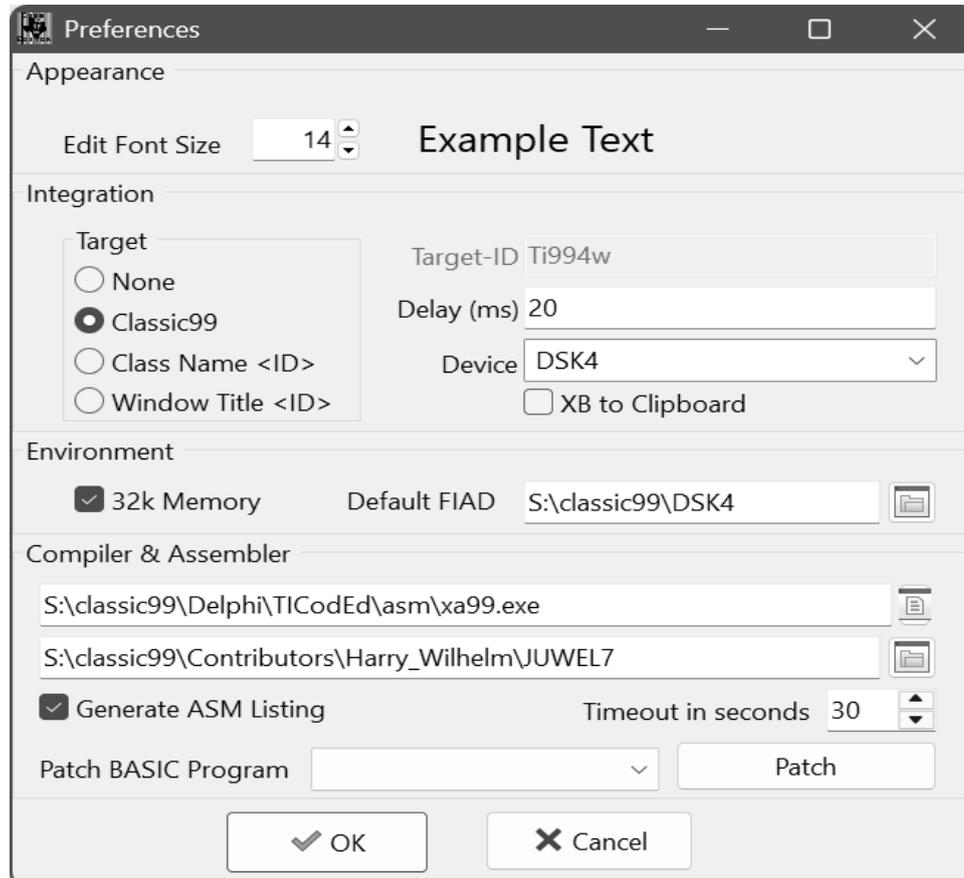
Lazarus offers the SynEdit component, which is used on the first page of TiCodEd. Beside the syntax-highlighting it offers some useful shortcuts beside the usual navigation:

Undo	Alt Backspace
Redo	Shift Alt Backspace
Block Indent	Shift Control I
Block Unindent	Shift Control U
Column Select	Shift Control C
Line Select	Shift Control L
Normal Select	Shift Control N

Double-Clicking on a label or SUB-routine name will scroll to the respective label/routine using the list generated at the last build of your project.

## Automation and Integration

TiCodED can be configured to send key-stroke messages to an emulator, especially to Classic99. This can be used to load, run or compile the program automatically in the emulator. In the Preferences in the menu Edit you find:



Select Classic99 and the Device you use in the emulator for loading the FIAD programs you create with TiCodEd. The Delay in milliseconds is the delay between keys sent to the emulator, the typing speed. 20 should work in most environments. Other programs might be identified by Windows Class Name or by exact window title, but keyboard input may not work in other programs.

For manual copying you may select "XB to Clipboard" to get the XB code copied to the clipboard on a successful build.

You may choose a default FIAD directory where all tokenized files are stored for the use of Classic99.

In Windows you may use Fred Kaal's Xa99 assembler for automated compiling. It is included in the *asm* directory of the installation. It needs to find the compiler runtime files to be included, so specify the directory of the compiler in the next field. Finally you can enable the writing of the assembler listfile and set a timeout for the compiler to finish.

See Appendix G – Patches for patching the compiler.

If anything but "None" is selected as a target, an additional section appears on the project tab:

Integration		
<input type="checkbox"/> Auto-Load	<input type="checkbox"/> Auto-Run	<input type="checkbox"/> Auto-Compile
OLD DSK4.test :RET.	RUN :S+:OEM7.:S-DSK4.test:S+:OEM7.:S- :F	RUN :S+:OEM7.:S-DSK1.COMPILER:S+:OEM :WAIT4.:F3. DSK4.test-M :RET. :RET. Y:RET. .....

If the edit field are empty they can be populated with a default by double-clicking in the edit box. There is a fine, but important difference between sending characters or emulating keyboard entries. There is for example no key for the quote-character. For typing a quote you have to press the Shift-Key, press the key in the middle between Return and L (Labeled differently in each country setting), and release the Shift-Key. This can be constructed as:

:S+:OEM7.:S-

:S+ is pressing Shift, OEM7 is the internal Microsoft name of the mentioned key and :S- releases Shift. The defaults should work for you and in Appendix E – Keyboard Emulation – you will find a list of keys to send.

When a build is successful, which means without errors, three buttons appear below the log. You can press Load, Run or Compile or hit the keys L,R or C to send the specified key-sequence to the emulator.

```
Writing MERGE-File S:\classic99\Delphi\TICodEd\test-M
Writing PROGRAM-File S:\classic99\Delphi\TICodEd\test
Writing variable X-Ref file S:\classic99\Delphi\TICodEd\test.vxr
```

```
Errors: 0
Warnings: 0
```

Load

Run

Compile

By selecting one of the three Auto check-boxes on the preference tab you can even automate this for each successful build process.

Please note that the compile process will start the Extended BASIC compiler (assumed to be on DSK1) for the external Assembler and not using the low memory for the runtime. This can be changed when needed. Using ASM994a the Assembler and the Loader need to be executed manually.

When specifying a path to Fred Kaal's [XA99 Assembler](#) in the preferences, the compile integration will generate code to use this instead of ASM994a. You may need to create a new automation-script when changing settings or filenames by deleting all content of edit-box and do a double click in the empty box.

You may need to adjust or add an additional :WAITx, when the timing is not correct.

## Porting existing XB code to SXB

There is no general approach, but the following worked quite good for me:

An Extended Basic program with line numbers should be usable as SXB program to start with, as line numbers are depreciated but allowed.

Go through the code and look for any reference to a line-number. Replace the line-number with a label and add this label right in front of the target line. Once finished you may remove all your line-numbers and start formatting the code. This process is now automated with the File/Import function. Each referenced Destination-Line gets a Label LABELnnnnn with nnnnn as the line-number and a renaming-function to replace them with meaningful names.

Add blank lines to separate logical blocks of code. Use line-numbers to group your code. Add // comments to make the code easier to understand. Refactor blocks by using REPEAT and WHILE loops instead of IF/GOTO.

Pretty soon your code looks completely different. Export the code to XB and compare to the original code.

## Extended Basic Version

TiCodEd is tested against the most popular XB 'Solid State Software' module Version 110 and G.E.M 2.9 included in Classic99. MyArc Extended Basic contains additional tokens not supported. If an XB Version uses the same tokens as XB 110 it should run. Limited testing has been done on RXB 2015E up to RXB 2024, XB II+ and Extended Basic v2.7.

If you are unsure which cartridge you own you may try

```
CALL VERSION(A)  
PRINT A
```

This will print 110 for the most common module. If you use CALL Subroutines unknown to this version you may get warnings as the sub-routines are neither known to TiCodEd nor included in a Library file. For XB 2.9 and RXB 2024 are packages included to declare these additional subroutines as internal.

## Extended BASIC Compiler Considerations (JEWEL)

TiCodEd is designed to ease the development of compiled programs. Most obvious, it supports the export of the needed MERGE-File. But it also checks the names of your SUB-ROUTINES. There are strict rules listed in the XB Compiler manual you may forget while coding. TiCodEd will give warnings, when you use

- Too similar user-routine names (only 6 significant chars)
- compiler reserved words (long list in the compiler manual)
- Letters NC, NV, NA, SC, SV, SA, L followed by a numbers

## Limitations

There are some design limitations you should be aware of, but usually should not limit your options:

- Nesting level of REPEAT/WHILE has combined a maximum of 32
- Not more than 999 REPEAT/WHILE loops may be used in one program
- Labels may have up to 32 characters and must not be named BLOCKnnn, REPEATnnn, WHILEnnn and ENDWHILEnnn (nnn a number) as they are used internally.
- You may use up to 200 labels (including generated labels)
- SXB Programs may have up 2048 lines, including imported SUB-ROUTINES
- A line may have up to 128 token (remember, special characters like plus or parenthesis are [also tokens](#)). A "Line too long" error will be shown when the limit is exceeded
- REPEAT, UNTIL, CONST, WHILE, ENDWHILE and WEND need to be the only statements on a code-line
- Memory Base "Files(X)" will not check for memory overruns
- The status-bar contains only information on the SXB tab and the last build-process (Errors, Warnings and memory usage).
- TIFILES and V9T9 header will be only partially filled
- SXB files will be automatically saved before conversion to XB
- A VXR file with the variable cross-reference is needed for variable short names and will be created with every "Build Project"
- Syntax errors may not be detected as both conversions (to XB and to Tokens) are based on substitution of strings, not a syntax-graph
- DATA statements may have strings with quotation. Please be aware of the removal of spaces at the beginning and the end
- The included assembler Xa99 can handle 3.000 labels.

This section is likely to be extended when people send me errors I can not easily fix.

## Embedded Assembler

Starting with Jewel 7 the BASIC Compiler is able to process custom assembly routines. SXB now has ASM/ASMEND blocks similar to SUB/SUBEND to facilitate this. Use the "Patch" function with "jewel7.patch" in the Preferences to prepare the compiler by switching from an interactive handling to a READ/DATA handling of the custom routines managed by TiCodEd.

For details on using ASM/ASMEND see the chapter "[Embedded Assembly ASM ... ASMEND](#)" in the included "TiCodEd Beginners Manual" and Appendix G – Patches.

## Trouble-Shooting

Something not working? Take a look at the log page. If you set the log-level to 5 = 'unbelievable verbose' you get log information about seven times the size of your source when running 'Export / Build Project'.

First, check if the code is correctly translated from SXB to XB. The log gives you indication of all substitutions.

The second step is more tricky, as it involves the binary token format. The log file also lists each line in token format. First the XB line is shown, then after the -> arrow how it is split up separated by pipe symbols | and finally the hex presentation. For bytes larger 0x80 the corresponding token is listed in square brackets for easier interpretation, disregarding whether or not it is actually a token (i.e. part of a line number).

```
120 FOR I=0 TO 14 :: CALL COLOR(I,16,2) :: NEXT I :: CALL
COLOR(9,10,16) -> FOR:I|=!0!TO!14!::!CALL!COLOR!(!I!,!16!,!2!)!::!
NEXT:I!::!CALL!COLOR!(!9!,!10!,!16!)! -> 8C[FOR] 49 BE[=]
C8[UNQUOTEDSTRING] 01 30 B1[TO] C8[UNQUOTEDSTRING] 02 31 34 82[::]
9D[CALL] C8[UNQUOTEDSTRING] 05 43 4F 4C 4F 52 B7[(! 49 B3[, ]
C8[UNQUOTEDSTRING] 02 31 36 B3[, ] C8[UNQUOTEDSTRING] 01 32 B6[)]
82[::] 96[NEXT] 49 82[::] 9D[CALL] C8[UNQUOTEDSTRING] 05 43 4F 4C 4F
52 B7[(! C8[UNQUOTEDSTRING] 01 39 B3[, ] C8[UNQUOTEDSTRING] 02 31 30
B3[, ] C8[UNQUOTEDSTRING] 02 31 36 B6[)] (?I????0????14?????COLOR?I???
16????2????I?????COLOR????9????10????16?)
```

If the tokenized file does not load try pasting the XB source code to the emulator, save it and compare the files. Or try loading the MERGE file instead.

## Found an Error? Having a suggestion? Future plans?

I am happy to hear from you, whether TiCodEd works for you or you found a problem. I won't debug your code though. When you have an example of something going wrong, chances increased when your code has less than 15 lines or you can point to an exact line where an error occurs and what is actually right.

I set up an email address I check when I have time for my TI hobby:

TiCodEd <at> lizardware <dot> de

Please accept my apologies that it may take some time for me to respond. Rainy autumn weekends may be better than sunny spring workdays.

There is also a support-thread on AtariAge:

<https://forums.atariage.com/topic/314536-introducing-structured-extended-basic-and-ticoded/>

## Building TiCodEd from Source

TiCodEd is written in PASCAL using the free Lazarus IDE and libraries.

<https://www.lazarus-ide.org/>

This IDE offers cross-platform support for Linux and Mac.

## Use LibXBTKN32.dll or LibXBTKN64.dll

I wrote a wrapper for the uTokenize.pas unit to be used in other programs. It is used to convert Extended Basic Text-Files to tokenized XB and publishes the following function:

```
Function XBTokenize(xbin,tknout,mrgout,logfn,opt:PChar):Integer; cdecl;
```

Parameter	Description
xbin	Input Text filename with Standard Extended Basic Program
tknout	Output filename with TIFILES Header, empty to omit
mrgout	Output filename with TIFILES Header in MERGE format, empty to omit
logfn	Logfile name, empty to omit
opt	Options: v=[0..5] Verbosity, default 3 -IV254 forced IV254 format also for small programs
Return-Code	00 : No Errors 10 : Could not open XB input file 11 : Error reading XB input file 20 : Error converting file 30 : Error writing MERGE file 40 : Error writing PROGRAM file 50 : Could not open log file

## BSD License

Copyright (c) 2025, Stefan Bauch

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Acknowledgements

Some people helped me, for pandemic and geographical reasons mostly by mail.

- Matthew Hagerty with hints on porting his TIDBiT to Pascal
- Ralph Benzinger for hints on tokenizing and the 'long format'
- Harry Wilhelm for writing the XB Compiler, which encouraged me to resume coding for the TI-99/4a
- Fred Kaal for Xa99 and Ti99Dir, testing of TiCodEd and the Tokenizer LibXBTKN32.DLL and including it in Ti99Dir
- Helge Spahrbier for testing of TiCodEd and the Mac version
- AirShack, Oddemann, dhe and Vorticon for their input on AtariAge

In memory of Frank Euler, my late TI programming companion. I miss you.

## Appendix A – Change-Log

### Version 3.00

- Use \$xxxx or \$xx hexadecimal numbers where numerical literals are used
- BEGIN/END and CASE may now be used within SUB/SUBEND
- ASM/ASMEND may include Assembly code in compiled XB (Windows only)
- New Patch Cart utility to add data to your compiled program (Appendix F)
- Load or dump fonts and charsets from the Char-Page using the right mouse-button context menu on the character table.
- “:: REM” issues a warning now (can’t be compiled)
- Warning dialog for assembler errors
- Now multiple Packages may be selected, i.e. XB256 and XB 2.9 GEM
- Classic99 Debug-Opcodes library using the ASM/ASMEND technology
- Various clean-ups and fixes; new examples and reworked manual

### Version 2.51

- Switch from Log to SXB page after successful integration run
- New library files for T80XB and XB II+ with short examples
- Bugfix: IMAGE was tokenized wrongly
- Bugfix: The first characters on Char-Page got corrupted

### Version 2.50

- New CONST statement for universal constants
- New “Replace Identifier” in Edit-Menu (the general Replace “i -> count” would also change the i in PRINT or DIM)
- Option to use XA99 Assembler with Integration to automate the compilation even more. Edit-boxes now can have up to 20 lines.
- V9T9 files are now supported on the Project Page for generated files
- Default FIAD directory for tokenized files
- Util Menu: File Editor for Variable/Fixed Internal/Display data files in FIAD in TI-FILES and V9T9 format up to 512 records/16 fields per record.
- Char page: Unload Background-Image
- Integration: Copy XB to Clipboard option
- New library files for RXB 2023 (2022), XB 2.9 (G.E.M.) and MColor
- Standard-Library: CALL CharInv(org\$,inv\$) and CALL Scramble(a\$)
- Lowercase LINK flag for compiling with Assembler Support-Routines
- Hex2BIN\$ extended with a quick “Replace all” function
- File- and directory fields have a file-dialog icon now
- Check for unique labels added

- Bugfix Char page: Copied pattern accidentally when Hex-Editor was active

### **Version 2.40**

- Util Menu: Hex2BIN\$ is a dialog to extract HEX data from a text
- Var page: PreScan List of variables
- Char page: Flip, rotate and invert added
- Bugfix: BIN\$ had issues with >22 "Quote" character

### **Version 2.30**

- Memory Base replaces the IV254 flag with more options, i.e. CALL FILES()
- TI BASIC flag allows byte-identical files in TI BASIC
- Bugfix: Multiple IN[] in one line now work
- Bugfix: DATA statements ending with "," now work properly

### **Version 2.2x**

- New function BIN\$ to include binary data provided as Hex-String
- Code-Completion with Ctrl-Space for keywords and variables
- Linux version
- New RXB 2022 and XB 2.8 Package file
- Bugfix: Generated GOSUB Line-Numbers correct when no SUB is used
- Bugfix: Single-Word Statement followed by :: not misread as labels

### **Version 2.10**

- New option for forced IV254 on project page
- Edit Menu: New Toggle Comment feature
- New RXB 2021 and TML 2.0 Library files; bug fix XB256 "WINDOW"
- 200 instead of max. 100 labels due to many generated labels
- Semicolon can now be used in filenames in the integration
- ON GOTO / GOSUB does not require a space before each label anymore

### **Version 2.00**

- New BEGIN-END Blocks in IF-THEN-ELSE and CASE
- New CASE-Statement
- IN[...] Set testing
- Import for XB programs supporting assignment of labels
- Syntax Highlighter adapted for XB / SXB (from Visual Basic)
- Changed Char Editor to complete Charset Editor incl. animations
- Find/Replace reworked to avoid accidental deletions
- Scroll-Bar on Parameter and Char Tab for smaller screens
- Parameters for generated GOSUB and SUB/SUBEND line numbers
- Error-Messages for Non-ASCII Characters and undefined Labels

- Finally a meaningful status-bar for the SXB editor and last build
- Code-Completion with Ctrl-Space and new Edit/Highlight function
- Remote Control of Classic99 and other emulators by Keyboard Emulation

### **Version 1.20**

- New Variable Page
  - Cross-Reference with option for XB short names to save memory
- Char Page
  - Switching 8x8/16x16 now changes the hex-string
  - Erase Button added
- Project Page has now a Post-Processing Command-String
- Window size and position will be restored on next execution
- Auto-Scaling disabled to fix display errors
- Several minor bug fixes and usability enhancements

### **Version 1.10**

- Libraries in Project page
  - Extension Packages (XB256, T40XB,...)
  - User Library
  - Standard Library
- Help-Menu added
- Preferences dialog added
  - Change font size for editor
- WHILE-ENDWHILE logic optimized
- Program statistics in log file for log level 'Information' and higher
- More SXB Examples provided

### **Version 1.00**

- Initial release

## Appendix B – Standard Subroutine Library

Still under development. Suggestions and code donations welcome!

All routines are XB Compiler compatible (except where noted).

**CALL ScrInit(fg,bg)** Clears the screen, deletes all sprites, sets a background color bg and all chars in foreground color fg with a transparent background.

**CALL RAND(SEED,UL,RES)** Returns an integer  $0 \leq RES < UL$  and advances the seed. Useful when you want a repeatable sequence, but be aware that neighbor seeds will compute similar results, while the seed is updated with distinct values.

**CALL CreateQ(a\$,L)** Initializes a Queue with the length of L (max 84) and stores it in the string a\$. Each queue entry consists of three byte, the last byte of the string is the current entry.

**CALL enQ(a\$,c,p1,p2,d)** Adds a record to the first free entry of the queue d steps ahead of the current position (use 1 for the next available) with the command c and the parameters p1 and p2 (values 1-255 for command, 0-255 for parameters). If the queue is full a\$ is set to 'full' and appropriate actions should be taken.

**CALL deQ(a\$,c,p1,p2)** Gets next entry from the queue, command 0 means empty slot.

**CALL trim(a\$)** Removes leading and trailing spaces and unprintable characters.

**CALL upStr(a\$)** Converts a\$ to uppercase characters.

**CALL loStr(a\$)** Converts a\$ to lowercase characters.

**CALL CharInv(org\$,inv\$)** inverts a given hex char-pattern

**CALL Scramble(a\$)** Scrambles a string

**CALL Mod(n,d,m)** Calculates the modulo  $n \text{ MOD } d$  (Remainder  $n/d$ ). Results may differ when compiled if n or d are no integer values.

## Appendix C – Extension Packages

Extension Packages are translation tables for popular extensions in a very easy and simple format. Whenever a subroutine has parameters they are universally referred to as 'P'. All parameters are passed as-is to the LINK call. When a subroutine has optional parameters it is required to have two translation lines in the XBP Package file, first the line with parameters, followed by the line without parameters, i.e. SCRLUP in XB256.

```
CALL SCRLUP(P) -> CALL LINK("SCRLUP",P)
CALL SCRLUP -> CALL LINK("SCRLUP")
```

Each line starts with the SXB simplified code, followed by ' -> ' and the resulting LINK call. Please take care of possible name conflicts, i.e. CALL LINK("SCREEN",P) must not be referenced as CALL SCREEN, as this is already taken by XB.

Immediate calls are not defined as they must not be used in programs.

For additional routines in newer basic variants the INTERNAL command can declare internal subroutines which are neither searched in libraries nor raising errors. The extension package may also contain specific subroutines to the package. These are read after the user- and before the standard-library file.

You may combine files to create a combined package, i.e. RXB with XB256.

### XB256

All routines of XB256 have been translated to identical CALL routines except for:

- CALL LINK("SCREEN",P) is to be used as CALL SCREEN2(P) in SXB
- CALL LOAD(-1,N) may be used as CALL SYNC(N) to set the interval

See file LIB/XB256 for syntax and the very good XB256 documentation for semantics and usage of XB256.

Additional Routine:

- SUB SPEED(S,X,Y) – Returns the speed of Sprite S.

### T40XB

All routines of T40XB have been translated to identical CALL routines except for:

- CALL LINK("COLOR",P) is to be used as CALL COLOR2(P) in SXB
- CALL LINK("CHAR",P) is to be used as CALL CHAR2(P) in SXB
- CALL LINK("HCHAR",P) is to be used as CALL HCHAR2(P) in SXB
- CALL LINK("VCHAR",P) is to be used as CALL VCHAR2(P) in SXB

See file LIB/T40XB for syntax and the very good T40XB documentation for semantics and usage of T40XB.

## T80XB

All routines of T80XB have been translated to identical CALL routines except for:

- CALL LINK("COLOR",P) is to be used as CALL COLOR2(P) in SXB
- CALL LINK("CHAR",P) is to be used as CALL CHAR2(P) in SXB
- CALL LINK("HCHAR",P) is to be used as CALL HCHAR2(P) in SXB
- CALL LINK("VCHAR",P) is to be used as CALL VCHAR2(P) in SXB

See file LIB/T80XB for syntax and the very good T80XB documentation for semantics and usage of T80XB.

## TML – The missing link

All routines of TML 2.0 have been translated to identical CALL routines except for:

- CALL LINK("COLOR",P) is to be used as CALL COLOR2(P) in SXB
- CALL LINK("CHAR",P) is to be used as CALL CHAR2(P) in SXB
- CALL LINK("CLEAR",P) is to be used as CALL CLEAR2 in SXB

## RXB 2024

Rich Extended Basic (RXB) contains numerous new CALL subroutines. This packages declares them as internal routines on top of the standard routines.

## Extended Basic 2.9 G.E.M

Extended Basic 2.9 contains numerous new CALL subroutines. This packages declares them as internal routines on top of the standard routines.

## Extended Basic IIplus

Declares additional Routines of XB IIplus as internal and translates all Apesoft High Resolution Graphic commands to the simplified CALL syntax.

## MultiColor

Use my [MColor library](#) with TiCodEd for the MultiColor mode with 64x48x16 pixel.

## Classic99dbg

Provides Classic99 Debug opcodes for compiled Extended Basic, see chapter 7.9 of the Classic99 manual for more information on debug opcodes in Classic99.

## Appendix D – Example files

In the directory `.\Examples` you will find some demonstrations of SxB you may use for exploring the possibilities of the TiCodEd environment.

### COINC

A very basic program for testing the CALL COINC subroutine for a game. No line-numbers and just a REPEAT loop for starting and make yourself comfortable with the environment and the different pages. Paste the character-definition to the Char page, modify it and paste it back.

### WHILE

Demonstrates the use of nested WHILE loops. Take a look at the generated code on the Xb page. The WHILE becomes a GOTO, the condition is moved to the end of the loop. This example uses ENDWHILE and WEND to end the loop as they are synonyms.

### XB256

This program demonstrates to use of the package 'XB256'. Make sure that on the project page LIB\XB256.xbpkg is selected and you load it first when you execute the program. This package defines the translation of the simple CALLs to the CALL LINK statements. Have a look at the source and open LIB\XB256.xbpkg in a text-editor if you are curious.

The screen is prepared with CALL scrn2, screen2, color2 and displ. Check the Xb page how they are translated to CALL LINK. Please note that the text 'XB256' is not moved, but the whole screen is scrolled in all four directions just to show how mighty the XB256 package is and how easy it can be used.

### QUEUE

This program demonstrates the use of the TiCodEd Standard Library. Make sure that the box 'Standard Library' is checked on the Project page. First it uses CALL ScrInit out of the library to clear the screen, set a black background and white characters. Note the scattered line-numbers to build logical blocks in the program, the main program starts at 1000, the GOSUB routines at 20000 and the trailing 30000 sets the base line-number for the subroutines out of the Standard Library.

The program asks first for the length (capacity) of the queue, which may be up to 84 entries, and the delay, meaning how far from the current position a new entry will be inserted.

When you press letters A to Z, they will be shown in one of three rows and the position will be added in the queue (p1,p2). After d iterations the letter will be deleted. The command `l` stands for 'delete char', others may be added and `ON c GOTO / ON c GOSUB` could be called.

On the Xb page you can study how the code from the library is appended. A compiled version of this program is included.

### SPRITE256

Demonstrates the usage of package-specific SUB-Routines with the XB256 package. XB256 provides VREAD to read the sprite control table, used in a new SUB-routine to read the speed of a sprite.

## **MODULO**

The usage of `CALL MOD(n,d,m)` to calculate the `n MOD d`.

## **STEVEB52**

A little game demo. Press Space to drop a bomb. Clear the runway before you land.

## **STRINGS**

String utility functions `CALL LoStr`, `CALL HiStr` and `CALL Trim` from the standard library.

## **BINSTR**

Demonstrate the `BIN$` function to create a binary string and use it in `XB256 CALL VWRITE` to create a fast `CALL CHAR`.

## **CASE**

Demonstrates the `CASE` statement including the option to use `BEGIN-END` blocks with `CASE` and the `ELSE` branch.

## **BEGIN**

Nested `BEGIN-END` blocks and a simple `CASE`.

## **IMPORT.XB**

A test- and training `XB` program for the `import` module (`File/import`) using all available `XB` statements with line-numbers.

## **TMLSXB**

A very basic demo for `TML` with `SXB`. Be sure to select the `TML` package on the project page.

## **T80Demo**

A very basic demo for `T80XB` with `SXB`. Be sure to select the `T80XB` package.

## **XB2+**

A basic demo for `XB IIplus` hires graphics with `SXB`. Be sure to select the `XB IIplus` package on the Project page.

## **BISBEE**

MColor library demo using shapes and the double-buffer

## **QUIX**

MColor library demo with unbuffered write-through and line-drawing

## **SPEEDYMC**

[Multicolor Speedy](#)

## **FLIP & MPY**

Adoptions of the two examples from Harry Wilhems compiler manual for custom assembler routines.

## **Classic99dbg**

Use the Classic99Dbg.xbpkg library to demonstrate the usage of ASM/ASMEND in a library and how the debug opcodes work in Classic99.

## **CartPatch**

Examples to use the patch functionality to store and retrieve screen data in the Extended BASIC module image (for development) and append it to the finished module file.

## **SUBTEST**

The BASIC compiler is very picky about names of sub-routines. Only the first 6 characters are significant and there are plenty of reserved words (check Isabella documentation for details).

TiCodEd has a check implemented and gives warnings:

1. Too similar user-routine names
2. Use of reserved words
3. Letters NC, NV, NA, SC, SV, SA, L followed by a numbers

While the program itself is without any use it provides examples of the implemented checks.

Warning: VREAD is a reserved word in ISABELLA.

Warning: !NV1000 and !NV1000X too similar for ISABELLA (only 6 significant chars)

Warning: !CHARSET2 is too similar to standard routine for ISABELLA (only 6 significant chars)

Warning: L1000 is invalid in ISABELLA.

Warning: NV1000 is invalid in ISABELLA.

Warning: Subroutine NONEXISTANT not declared.

The preceding '!' indicates the matching presence of a SUB command, otherwise a warning will be issued.

## Appendix E – Keyboard Emulation

Please visit the Microsoft page for [virtual key codes](#). Many of these keys can be used in the TiCodEd Keyboard Emulation.

- Standard Characters A-Z and 0-9
- :RET. Return Key
- :TAB. Tabulator
- :ESC. Escape
- :F1. to :F24. Function Keys (i.e. :F3. for "erase")
- + - , .
- :S+ :S- Shift on / Shift off
- :C+ :C- Control on / Control off
- :A+ :A- Alt on / Alt off
- :OEM1. to :OEM8., :OEM102. See Microsoft [virtual key codes](#) for details

Special commands:

:WAITx. waits for x seconds before continuing (1-9 sec.).

:WAITFILE. <Filename> waits to the file to have more than zero bytes

:DIALOG. <Message> Displays a message-dialog with "Yes" and "Cancel"

:ASSEMBLER. <Source> Assembles the given source

You may use up to twenty lines. Note that for each line in the script:

- The focus is set to the specified program (Classic99)
- Before the first character 20 times the delay-rate will be waited

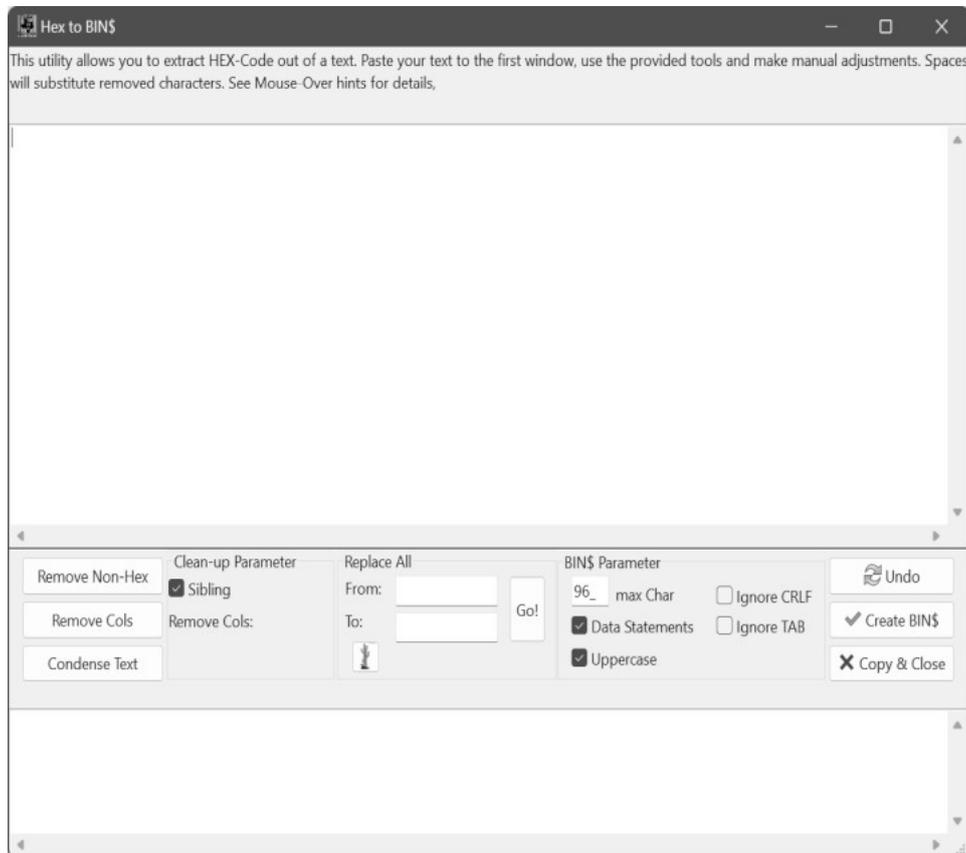
Increase the Delay in the Preferences if characters are missing.

Class Names for emulators are

- Classic99: TIWndClass
- Ti994w : Ti994w
- Win994a: n/a (use Window Title 'Win994a Simulator - v3.010 (x64)')

## Appendix F – Embedded Utilities

### Hex2BIN\$ - Extract code from text



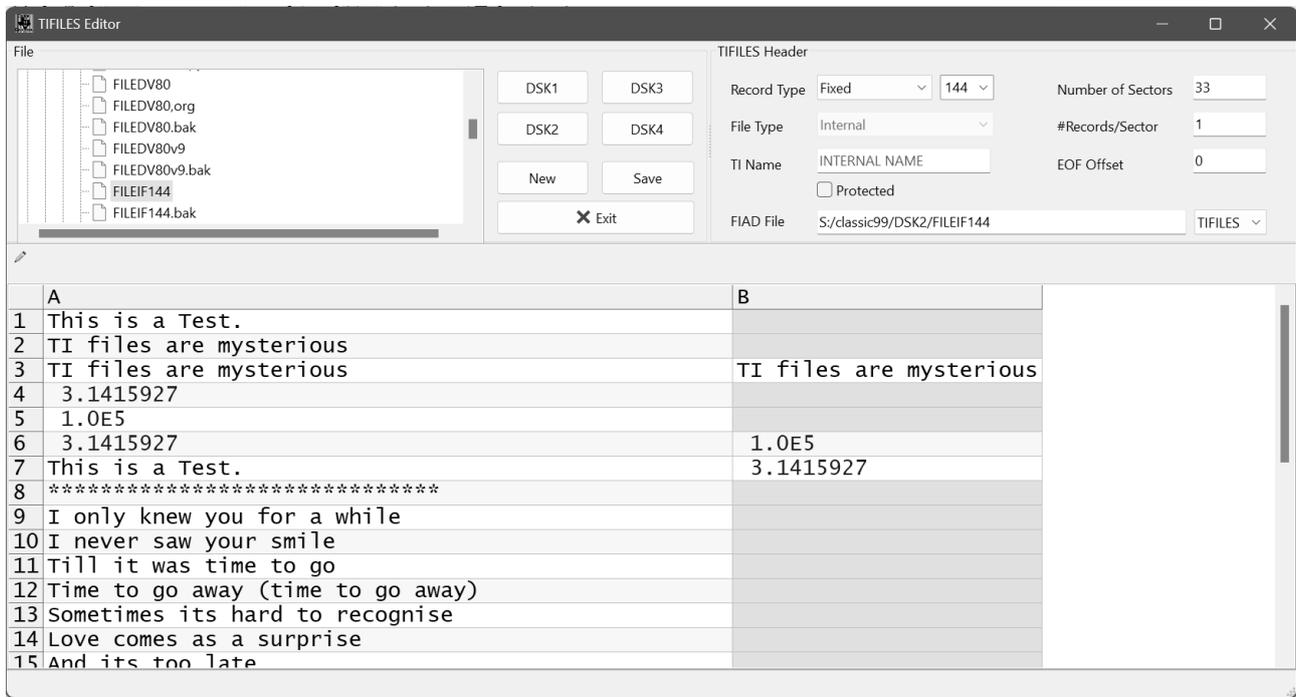
Sometimes you have a listing file from a tool or scan ... and need the embedded hex-code .. just paste your code in the upper editor and start playing around. Use the Undo if something does not work out (sorry, only one level of undo).

- Select chars in the first row and press "Remove Cols" to delete the marked columns in all rows.
- Use "Replace all" to get rid of repeating patterns .. "To" may be empty.
- "Remove Non-Hex" removes all characters other than 0..9, A..F, a..f; Check "Sibling" to require at least two of them together.
- Condense Text removes spaces
- Export remaining characters with "Create BIN\$"
- When done, leave with "Copy & Close"

The Quick-Action "Saguaro" performs a combination of those steps to extract the shape definition from <https://raphael.js99er.net/> exported by "Assembly data / By column 8 bpp" to be used with the Multicolor library.

## TIFILES Editor

With this editor you are able to open FIAD (File in a Directory) data files created with (Extended) BASIC in Variable/Fixed and Display/Internal format. Internal files are opened in a grid where each line is a record, each cell a variable, whereas Display files are less structured and opened in a generic editor control.



You may adjust all settings like Fixed/Variable, record length, internal name, file name and header (TIFILES/V9T9) before saving, only the File Type Internal/Display remains fixed as the formats differ too much.

There is no meta-information except the length of a field. Therefore some knowledge of the data is required, especially for 8 character wide fields in an Internal format file. They may represent a string with 8 characters or a RADIX-100 coded floating point number, i.e. "ABCDEFGH" may also be read as 6667.6869707172 resp. 6.6676869707172E3.

The following heuristic is applied: If an 8 byte field is a valid RADIX-100 encoding, it is interpreted as RADIX-100. If it contains bytes below 32, it is considered safe to be a RADIX-100 and shown in blue, if it consists only of displayable characters 32 to 99, it is marked in purple and should be checked, like the above example.

Checks and conversions can be done by a right-mousebutton context-menu. You may always convert from Text to RADIX-100 and back. If a field is set to RADIX-100 and does not contain a valid number, the value is shown in red.

You can not save the file with red entries.

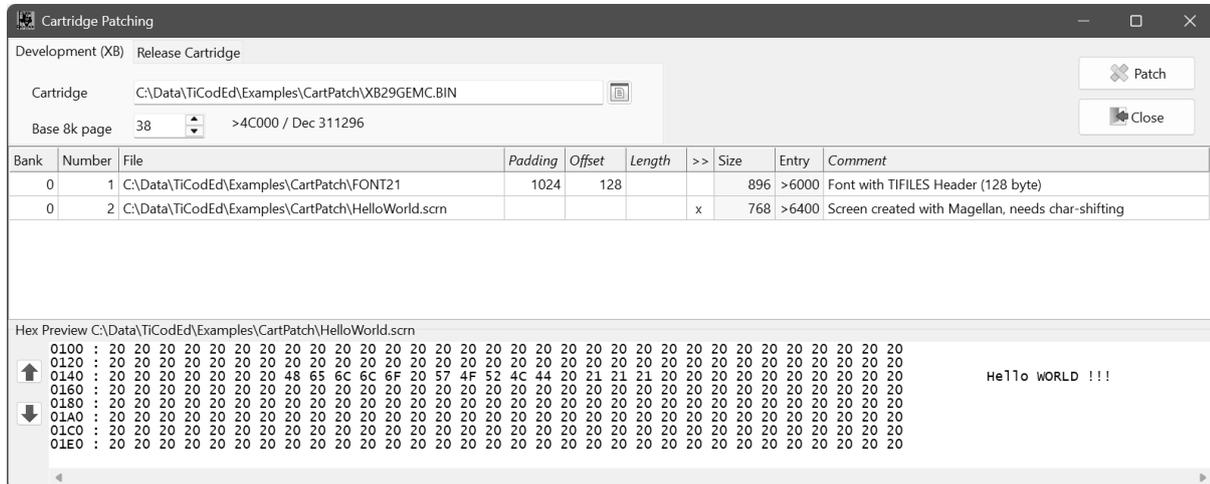
Unused fields are gray and might be populated simply by typing, and deleted (inactivated) with the context menu, where also whole lines (records) may be added or deleted.

Be sure to have a backup before editing any file!

## Patch Cart Utility

Access banked ROM from your compiled program! This does not work when not compiled, as bank-switching also disables the Extended BASIC module.

The utility has two tabs, one for patching the Extended BASIC module to use some free space within the module while developing, and a "Release Cartridge" for your final module.



XB 2.9 G.E.M. only uses about 300 of the 512kB in pages of 8kB. You may use page 38 and above for your own data. The hex preview pane should be initially full of zeros. If not, use the arrows to find an empty area.

Bank	Offset to the Base 8k Page, starting 0
Number	Any number to indicate the ascending sequence of files
File	Name of the file to be included, a double-click for file-dialog
Padding	Fill up with zeros if the file is shorter
Offset	Skip given numbers of bytes in the beginning (i.e. header)
Length	How many bytes to include (blank = all)
>> (Shift)	VRAM values are shifted by 96 from their ASCII values. Set "X" to shift the ASCII input by 96 or give another numeric value
Size	Size of the input file
Entry	Resulting entry address (calculated only when patching)
Comment	Some additional information on the file

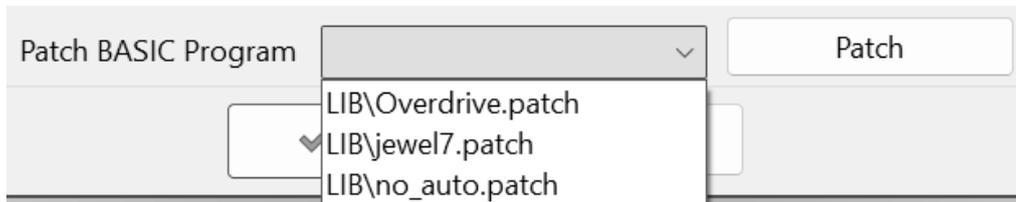
Check the "TiCodEd Beginners Manual" for more details and the CartPatch folder in the Examples directory.

Press the "Patch" button to apply the patch. Your settings will be stored in a comma-separated file <project>.cart, including the entry-points, and can be opened with any spreadsheet program.

## Appendix G – Patches

Starting version 3.0, TiCodEd is able to patch the compiler to your specific needs. This is done with simple macros by loading, modifying and saving the compiler in Classic99 or any other configured emulator on Windows.

Start the Preferences dialog in the Edit menu and look for the "Patch BASIC Program" at the bottom:



### jewel7.patch

This is the main patch to change the compiler to be able to use the TiCodEd ASM/ASMEND embedded assembler feature. It replaces the manual interactive maintenance of the custom routines with those based on READ/DATA, where the DATA line 10000 is maintained by TiCodEd. The custom runtime-file DSK1.CUSTOM.TXT is updated by TiCodEd as well

### no-auto.patch

This patch changes line 180 as described in the Appendix B of the XBGDP manual to not use the autocomplete feature, which sometimes conflicts with the automation coming from TiCodEd.

### overdrive.patch

This patch adds a line 1 to the compiler switching on the Classic99 CPU Overdrive mode via the Classic99 Debug OpCodes described in chapter 7.9 in the Classic99 manual.

**Hint:** If you want to have your compiled programs always switch to "normal speed mode", add DATA >0110 in RUNTIME1.TXT as shown:

```
*****
RUNV  LI R0,1      RUNV WILL RUN PROGRAM BUT NOT CLEAR OUT SCREEN2
      JMP RUN1
*****
RUN   SETD R0
      DATA >0110   SWITCH CLASSIC99 TO NORMAL SPEED
RUN1  CLR @XBEA5
      LI R1,CLRSCN
```

You need to enable the Debug OpCodes by setting enableDebugOpCodes=1 in the Classic99.ini file.