TiCodEd

&

Structured Extended Basic

by Stefan 'SteveB' Bauch

Version 1.10 (c)2021

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...5
Exporting SXB Code...5
Char definition..6
Structured Extended Basic..7
Using REPEAT .. UNTIL..7
Using WHILE .. ENDWHILE..8
Using Labels and Line-Numbers......................................8
Implicit Code-Blocks...9
Editor Features..9
Porting existing XB code to SXB...................................10
Extended Basic Version..10
Limitation..10
Trouble-Shooting..11
Found an Error? Having a suggestion? Future plans?................11
Building TiCodEd from Source......................................12
Use LibXBTKN32.dll or LibXBTKN64.dll..............................12
BSD License 13
Acknowledgements..13
Appendix A – Change-Log...14
  Version 1.10 14
  Version 1.00..14
Appendix B – Standard Subroutine Library..........................15
Appendix C – Extension Packages...................................16
  XB256...16
  T40XB...16
  TML – The missing link..16
  RXB 2020..16
  Extended Basic 2.7..17
Appendix D – Example files 18

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

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

Your program is working perfect? TiCodEd can also write the program in MERGE format for direct use in ISABELLA to be compiled.

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 ISABELLA.
- 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 8, Using Labels and Line-Numbers.
- 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 (see page 17, 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 16 Appendix B – Standard Subroutine Library)

Writing SXB Code

The SXB page is the actual editor. It features syntax highlighting, but currently a generic 'Visual Basic' rule set, which fits most, but not all language features. Especially the # sign is used differently in Extended Basic (Sprites, Files).

A dedicated Extended Basic Highlighter for the SynEdit component of Lazarus is currently not my priority.

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 7 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.

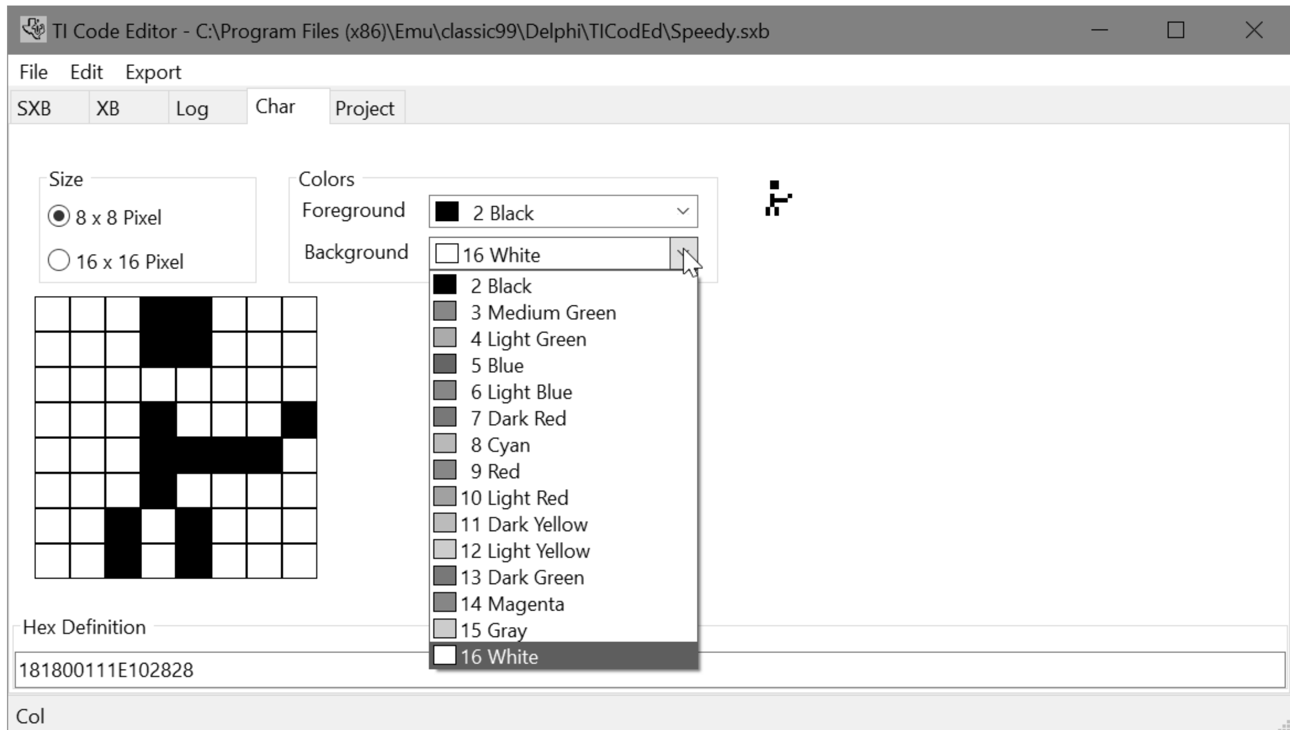
You will find the log of the conversion on the Log page and the Standard Extended Basic file on the XB page, which is read-only by default, but becomes editable double-clicking the text. This might be useful when you want to make a small adjustment and then click Export/Tokenize to create the TI files.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

Char definition

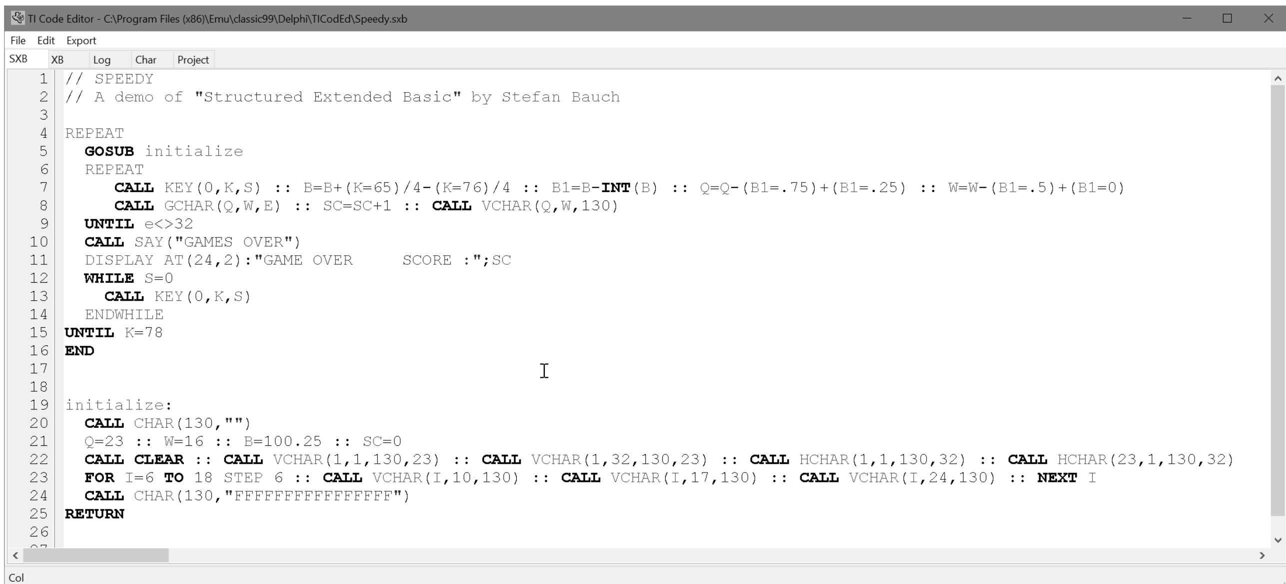
One common task in game development is the definition of the graphics. The Char page offers a little tool to help with this. You can choose between 8x8 pixel and a 16x16 character definition for single characters or Magnify 3 or 4 groups of four characters. In order to support the design you can choose 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.

Structured Extended Basic

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



You can see the program structure just by looking at the optional indention. 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 safe 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, but require a blank between the comma and the label name, 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. 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.

Implicit 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.

Editor Features

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

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. 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. 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, RXB 2020 and Extended Basic v2.7 included in Classic99.

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 2.7 and RXB 2020 are packages included to declare these additional subroutines as internal.

ISABELLA Considerations

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 ISABELLA 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 manual)
- Letters NC, NV, NA, SC, SV, SA, L followed by a numbers

Limitation

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
- You may use up to 100 labels

- SXB Programs may have up 1024 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, WHILE, ENDWHILE and WEND need to be the only statements on a code-line
- Tokenized files over 11775 bytes will be written in 'long format'
- TIFILES header will be only partly filled
- SXB files will be automatically saved before conversion to XB
- Syntax errors may not be detected as both conversions (to XB and to Tokens) are based on substitution of strings, not a syntax-graph
- The editor uses the Visual Basic highlighter and show some weird interpretations in SXB, especially the # sign in sprite- and file-numbers is misinterpreted. Lazarus volunteers are welcome to do a SXB highlighter
- DATA statements may have strings with quotation. Please be aware of the removal of spaces at the beginning and end
- Labels in statements need to have a preceding space in front, which is natural in most places, but special in 'ON GOTO' and 'ON GOSUB', it reads 'ON A GOSUB LB01, LB02, LB03'
- GUI status bar is still without function

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

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 line 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.

I already have some features in my mind for future releases.

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 (sadly not yet compatible with the latest MacOS). Ports to those platforms may not be too complicated. Currently only Win32 and Win64 are supported and available for download. I work with and test the Win64 bit version.

A new Syntax Highlighter for SXB would be nice if someone feels the need to dig into the depths of the SynEdit component. I currently prefer not to.

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
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) 2021, 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 ISABELLA, which encouraged me to resume coding for the TI-99/4a
- Fred Kaal for Ti99Dir and testing of TiCodEd and the Tokenizer LibXBTKN32.DLL and including it in Ti99Dir
- Helge Spahrbier for testing of TiCodEd

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

Appendix A – Change-Log

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 ISABELLA 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 <= 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. For performance reasons the boundaries are determined first so the removal of unprintable characters may lead to trailing or leading spaces again.

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

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

CALL Mod(n,d,m) Calculates the modulo n 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 an 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.

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.

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.

TML – The missing link

TML has not been converted yet.

RXB 2020

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

Extended Basic 2.7

Extended Basic 2.7 contains numerous new CALL subroutines. This packages declares them as internal routines on top of the standard routines.

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 disply. 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 1 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.

SUBTEST

The ISABELLA 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.