

# **Multicolor**

**A Library for Extended BASIC**

**by Stefan “SteveB” Bauch**

**© 2022–2023 by Stefan Bauch**

**Version 1.0**

<b>Introduction</b>	<b>4</b>
<b>Loading the library</b>	<b>4</b>
<b>General Introduction</b>	<b>4</b>
Color Codes	5
<b>Multicolor Commands</b>	<b>5</b>
<b>Starting and stopping Multicolor Mode</b>	<b>5</b>
CALL MCON	5
CALL MCOLOR(BCOLOR [, MODE])	5
CALL MCDONE	6
<b>Screen and Buffer</b>	<b>6</b>
CALL MCCLR(BCOLOR)	6
CALL MCSYNC	6
CALL VPORT(ROW1,COL1,ROW2,COL2)	6
CALL MCMODE(MODE)	7
Graphic primitives	8
CALL PUTPIX(ROW,COL, COLOR)	8
CALL GETPIX(ROW,COL, COLOR)	8
CALL HLINE(ROW,COL,COLOR,COUNT)	8
CALL VLINE(ROW,COL,COLOR,COUNT)	9
CALL LINE(ROW1,COL1,ROW2,COL2,COLOR)	9
CALL SQUARE(ROW1,COL1,ROW2,COL2,COLOR)	9
Shapes	10
CALL SHAPE(ID,ROWS,COLS,PATTERN\$)	10
CALL SHAPE2(PATTERN\$)	10
CALL BLIT(ID,ROW,COL)	10
Sprites	11
CALL MCMAG(MODE)	11
<b>Using Multicolor with XB</b>	<b>12</b>
<b>Compiling</b>	<b>13</b>

FIXAL to fix all differences	13
The Lowercase Trick	14
Compiling with assembler support	14
Bringing it together	15
<b>Creating Shapes</b>	<b>16</b>
<b>BSD License</b>	<b>19</b>
<b>Appendix A: Examples</b>	<b>20</b>
Lines	20
DirMode	21
Train to Bisbee	22
Quix	22

# Introduction

This library enables the use of the “Multicolor Mode” of the TI-99/4a with Extended BASIC. This mode offers 16 colors in an unbelievable resolution of 64 x 48 pixels.

It has been tested with Extended Basic 1.1, 2.9 G.E.M. and RXB 2022D, but should work with any Extended Basic. Furthermore it requires the 32k Memory Extension. It has been tested with Classic99, JS99er.net and TI994w.

## Loading the library

After CALL INIT you can load the library using CALL LOAD(“DSKx.MCOLOR”), where x is the drive number where the library is stored.

XB 2.9 is recommended as it loads the library very fast and the CALL INIT can be omitted.

## General Introduction

The library provides a set of assembler routines, stored at the Low Memory of your 32k Memory Extension. Beside the code, a screen buffer is also created in this area.

There are two modes available.

1. Writethrough: All write operations are performed directly on the screen.
2. Buffered: All write operations are only done in the buffer and only written to the screen, when a CALL MCSYNC is issued.

Which mode to choose depends on what you want to do. If you have only minor updates, Writethrough may give the best results. If you often change many parts of the screen, the buffered approach will suit you best.

The library is optimized for the buffered mode. Assuming you want to write a game, your game-loop will include

- Clearing the buffer
- Painting the new frame
- writing the frame to the screen
- start all over again

The use of sprites is supported, but differs slightly from the normal use with XB. Please be aware that sprites are independent of the buffer/screen synchronization.

The library comes with a definition file to use the library with the TiCodEd XB development environment and Structured Extended Basic. In SXB you may omit the CALL LINK and use the routines as they were XB standard routines.

Be aware that there are very few checks of parameters for performance reasons. All coordinates are checked against the viewport. Though you might specify coordinates outside the screen, they will not overwrite any other memory-location.

## Color Codes

Please note, that color-codes are one number lower than in Extended BASIC:

0	Transparent
1	Black
2	Medium green
3	Light green
4	Dark blue
5	Light blue
6	Dark red
7	Cyan

8	Medium red
9	Light red
10	Dark yellow
11	Light yellow
12	Dark green
13	Magenta
14	Gray
15	White

## Multicolor Commands

### Starting and stopping Multicolor Mode

#### **CALL MCON**

CALL LINK("MCON")

Starts the "Interrupt Monitor" for XB. The monitor prevents the VDP RAM from getting corrupted by adjusting the stack to the changed VRAM layout. See [Using Multicolor with XB](#).

#### **CALL MCOLOR(BCOLOR [, MODE])**

CALL LINK("MCOLOR",BCOLOR, MODE)

Initializes the library, switches to Multicolor mode and fills the screen and buffer with the given background color.

The MODE is "Buffered" by default and can be set to "Writethrough" with the optional MODE parameter.

BCOLOR	Initial background color	0 - 15
MODE	Optional: Buffered = 0 (default), Writethrough = 1	0, 1

## CALL MCDONE

CALL LINK("MCDONE")

Returns to standard Graphics Mode. At the end of the program the screen will be restored automatically.

## Screen and Buffer

### CALL MCCLR(BCOLOR)

CALL LINK("MCCLR",BCOLOR)

Clears the buffer by setting all pixels to the background color.

BCOLOR	Background color	0 - 15
--------	------------------	--------

Note: CALL MCCLR is the only draw command that does not honor the viewport for performance reasons. If you want to clear only a square on the screen, use CALL SQUARE.

### CALL MCSYNC

CALL LINK("MCSYNC")

Flushes the buffer to the screen (VDP RAM)

### CALL VPORT(ROW1,COL1,ROW2,COL2)

CALL LINK("VPORT",ROW1,COL1,ROW2,COL2)

Sets the "Viewport" of all draw operations. By default, the whole screen can be used and the viewport is set to 0,47,0,63, but can be changed to a part of the screen with this command.

ROW1	Upper row	0 - 47
COL1	Left column	0 - 63
ROW2	Lower row	0 - 47
COL1	right column	0 - 63

Think of (ROW1,COL1) as the upper left point and (ROW2,COL2) as the lower right point of the viewport square. Therefore be sure that ROW1 < ROW2 and COL1 < COL2.

CALL MCCLR will clear the whole buffer and not respect the boundaries set with CALL VPORT. Use CALL SQUARE to initialize a part of the screen.

### **CALL MCMODE(MODE)**

CALL LINK("MCMODE",MODE)

Sets the Mode to write directly to the screen or only to the buffer (default).

MODE	Buffered = 0 (default), Writethrough = 1	0, 1
------	--	------

# Graphic primitives

## **CALL PUTPIX(ROW,COL, COLOR)**

CALL LINK("PUTPIX",ROW, COL, COLOR)

Writes a single pixel to the buffer.

ROW	Pixel row	0 - 47
COL	Pixel column	0 - 63
COLOR	Pixel color	0 - 15

## **CALL GETPIX(ROW,COL, COLOR)**

CALL LINK("GETPIX",ROW, COL, COLOR)

Reads a single pixel from the buffer.

ROW	Pixel row	0 - 47
COL	Pixel column	0 - 63
COLOR	Return Value: Pixel color	0 - 15

## **CALL HLINE(ROW,COL,COLOR,COUNT)**

CALL LINK("HLINE",ROW, COL,COLOR,COUNT)

Draws a horizontal line.

ROW	Pixel row	0 - 47
COL	Left start pixel column	0 - 63
COLOR	Line color	0 - 15
COUNT	Number of pixels	1 - 63

## **CALL VLINE(ROW,COL,COLOR,COUNT)**

CALL LINK("VLINE",ROW, COL,COLOR,COUNT)

Draws a vertical line.

ROW	Upper start pixel row	0 - 47
COL	Pixel column	0 - 63
COLOR	Line color	0 - 15
COUNT	Number of pixels	1 - 47

## **CALL LINE(ROW1,COL1,ROW2,COL2,COLOR)**

CALL LINK("LINE",ROW1,COL1,ROW2,COL2,COLOR)

Draws a line from point (ROW1,COL1) to (ROW2,COL2). This universal routine is more complex and therefore significantly slower than HLINE and VLINE.

ROW1	Start pixel row	0 - 47
COL1	Start pixel column	0 - 63
ROW2	End pixel row	0 - 47
COL2	End pixel column	0 - 63
COLOR	Line color	0 - 15

## **CALL SQUARE(ROW1,COL1,ROW2,COL2,COLOR)**

CALL LINK("SQUARE",ROW1,COL1,ROW2,COL2,COLOR)

Draws a filled square with point (ROW1,COL1) as upper left to (ROW2,COL2) as lower right.

ROW1	Start pixel row	0 - 47
COL1	Start pixel column	0 - 63
ROW2	End pixel row	0 - 47
COL2	End pixel column	0 - 63
COLOR	Line color	0 - 15

# Shapes

Shapes are in contrast to sprites, not hardware supported graphics elements, that are drawn by software to the screen. You can draw shapes as often as you like on the screen.

## **CALL SHAPE(ID,ROWS,COLS,PATTERN\$)**

CALL LINK("SHAPE",ID,ROWS,COLS,PATTERN\$)

Defines a pattern for a shape, with each Hex-Digit defining the color of a pixel, color 0 is transparent, not overwriting the background. The string defines the colors column by column and with an even number of rows. See chapter [Creating Shapes](#) for hints.

Use CALL SHAPE2 to continue a shape's pattern if your shape is bigger and you need to split the definition. Please note a maximum pattern length of 128 characters per call.

ID	The address of the pattern will be returned	mem-addr
ROWS	Number of rows	2 - 48
COLS	Number of columns	1 - 64
PATTERN\$	Hex-String with Pixel Color information	Hex-String

Technical Note: You may PEEK the ID and get the rows first, then the columns, followed by two pixel-values per byte.

## **CALL SHAPE2(PATTERN\$)**

CALL LINK("SHAPE",PATTERN\$)

Continues the previous shape with an additional pattern string.

PATTERN\$	Hex-String with Pixel Color information	Hex-String
-----------	---	------------

## **CALL BLIT(ID,ROW,COL)**

CALL LINK("BLIT",ID,ROW,COL)

Draws a shape to the buffer at position (ROW,COL)

ID	A number to identify the shape	0 - 63
ROW	Row of the left upper pixel	0 - 47
COL	Column of the left upper pixel	0 - 63

# Sprites

All sprite functions of Extended BASIC can be used with the Multicolor Library, except for CALL MAGNIFY, which resets the screen to the default graphics mode.

Default is CALL MAGNIFY(4), but this may be changed with CALL MCMAG.

## CALL MCMAG(MODE)

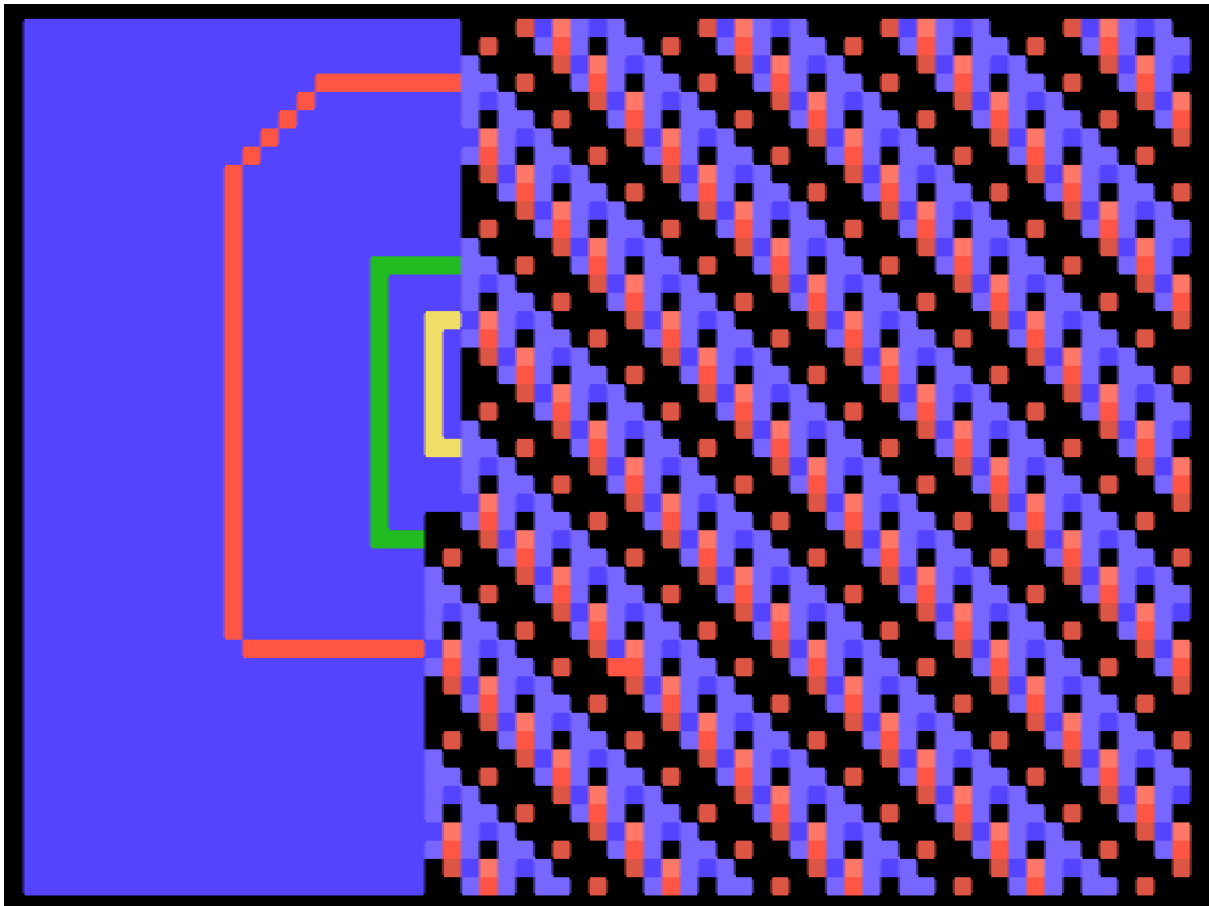
CALL LINK("BLIT",ID,ROW,COL)

Changes the sprite magnification in Multicolor Mode.

MODE	MAGNIFY value as in XB	1 - 4
------	------------------------	-------

# Using Multicolor with XB

In order to use the Multicolor Library the memory layout of the VDP RAM needs to be changed. Neither the console ROM nor XB is aware of this change, which might cause them to overwrite parts of your Multicolor screen, like in the following screenshot:



When using interpreted XB make sure to activate the included Interrupt-Monitor, written by Harry Wilhelm. It makes sure that your screen does not get overwritten by the Garbage Collection or other routines. You need to CALL MCON first and then restart the program with RUN <next-line-number>.

TiCodEd SXB Code	Regular XB Code
CALL MCON	100 CALL LINK("MCON")
RUN PRGRESTART	110 RUN 120
PRGRESTART:	120 ...

**This is not needed when compiling and the “RUN” statement will fail. Remove or comment those lines before compiling.**

# Compiling

Once your program is tested you may want to compile it. Starting with JEWEL Harry Wilhelms BASIC Compiler allows the use of assembler subroutines to be used in compiled programs.

There are some differences between compiled and interpreted code, the most significant is that the interpreted code uses floating point variables, while the compiled code works with integers only. Most assembler routines today also work with 16 bit integers, like The Missing Link or T40XB. The same is true for this Multicolor library, which takes the BASIC floating point variables and converts them to 16 bit signed integers. This is not needed and must be changed when compiling. Jewel already offers tools to do so.

The second difference is the way the new routines are integrated in the compiled code. The dynamic CALL LINK from XB needs to be converted to a static subroutine call and mixed with the runtime and the provided library to be assembled together.

Let's have a look at the steps to take. You may use the included demo BISBEE for your first attempts.

## FIXAL to fix all differences

First we prepare our library to be used with the compiler. This has already been done for you with the Multicolor library. You will find the file MCOLORC in the package.

This chapter is only needed if you want to change the code and prepare a modified version and for your better understanding of the mechanism.

First you need to remove or comment out the line "AORG >2600" and assemble the code to MCOLOR.OBJ. When interpreted, AORG makes sure that the code is always loaded to the same memory location. This is helpful for repeated loading of the library. Without AORG the code will be appended and not overwritten. The following loader will not work with AORG though.

When you assembled the changed MColor.txt to MCOLOR.OBJ you start fresh with Jewel in Disk 1:

```
NEW
CALL INIT
CALL LOAD("DSKn.MCOLOR.OBJ")
CALL LOAD("DSK1.FIXAL")
CALL LINK("X")
```

This changes the integrated assembler support routines to be compatible with the compiler, so you don't have to change your parameter handling manually. Now save the new, fixed library. The convention is to add a "C" for the compiler-enabled version:

```
SAVE DSKn.MCOLORC
```

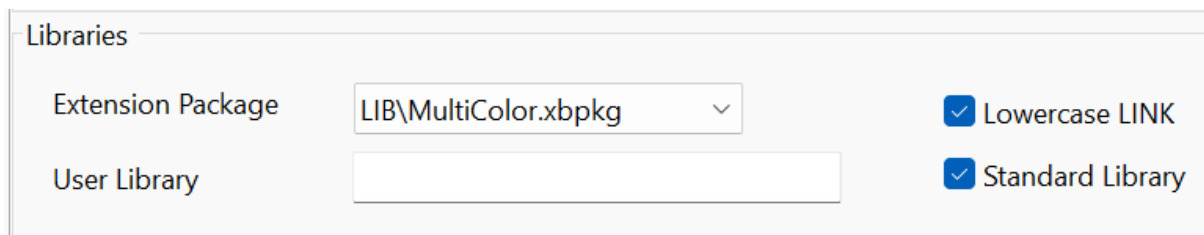
This file now contains the runtime in low memory to be used by your compiled program. We need to copy and modify it for each compiled program that uses the library.

## The Lowercase Trick

In order to distinguish and identify the library assembly routines they are coded in lowercase.

CALL LINK("MCOLOR",13) now becomes CALL LINK("mcolor",13) in your program.

When you use TiCodEd 2.5 or higher, you can simply activate "Lowercase LINK" in the Project settings in the Libraries section:



The regular tokenized and the merge-file will be created with lowercase LINK calls. Be sure to remove this setting when finished compiling and returning to interpreted mode.

If you are not using TiCodEd you can use the UC2LC utility included with Jewel:

```
OLD DSKn.YOURPRG
CALL LOAD("DSK1.UC2LC")
CALL LINK("X")
```

You may want to use another name for the converted program, i.e. add an L:

```
SAVE DSKn.YOURPRGL
```

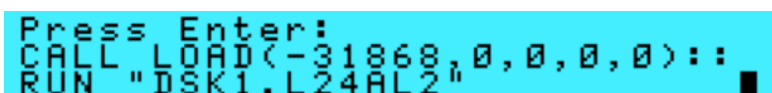
## Compiling with assembler support

You compile your program to object-file as always. When you come to the LOADER, things are getting a little bit different. Answer Y when asked "Using Assembly Support?".

Enter the object-file of your program when asked for the compiled file to load.

When asked "Assembly routines to load" give the library file DSKn.MCOLORC.

After a short while a cryptic:



appears. Press ENTER when the cursor starts blinking. The rest of the loader works as known, first creating the EA5 file, then the XB loader.

Please note, that the library is not merged with your program, it is only dynamically linked. You need both to make it work.

## Bringing it together

Of course you can always load the MCOLORC program to load the library before running your program, but it would be easier to link those two programs. The disadvantage is that you need to decide which disk drive you will use for the files.

Load the library

OLD DSKx.MCOLORC

Edit line 10 to include a CALL INIT and to start the actual program with RUN:

```
10 CALL INIT::CALL LOAD(8192,255,172):: CALL LINK("X"):: RUN "DSKn.YOURPRG-X"
```

Then save it as DSKn.YOURPRG-C.

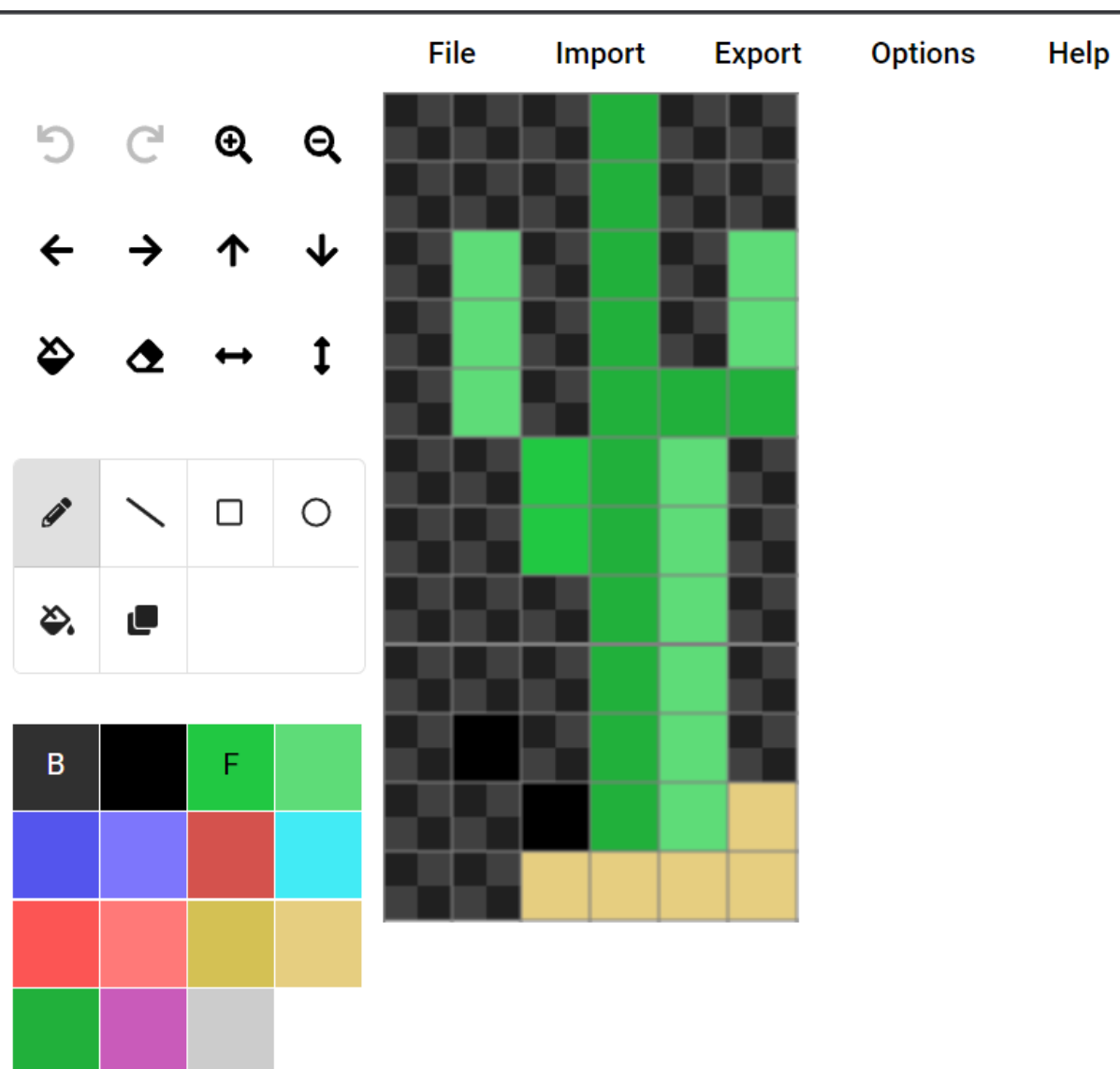
This is your chained program, combining the loading of the library in lower memory and then your compiled program.

# Creating Shapes

While it is possible to define shapes manually by listing the colors as hex values, column by column, the easiest way to define neat shapes is using **Raphael**.

<https://raphael.js99er.net/>

This TI oriented graphics editor offers all you need to define a shape to be used in a Multicolor game or application. Please be aware that you need an even number of rows, the number of columns may be even or odd.



Once finished, export your shape to "Assembly data / By column 8 bpp". The export looks like this:

```

byte >00,>00,>00,>00,>00,>00,>00,>00
byte >00,>00,>00,>00,>00,>00,>03,>03
byte >03,>00,>00,>00,>00,>01,>00,>00
byte >00,>00,>00,>00,>00,>02,>02,>00
byte >00,>00,>01,>0b,>0c,>0c,>0c,>0c
byte >0c,>0c,>0c,>0c,>0c,>0c,>0c,>0b
byte >00,>00,>00,>00,>0c,>03,>03,>03
byte >03,>03,>03,>0b,>00,>00,>03,>03
byte >0c,>00,>00,>00,>00,>00,>0b,>0b

```

There are two ways to get this into the desired format.

1. Use a Text-Editor and use the Replace function

From	to
" byte"	"
">0"	"
" "	"

This leaves:

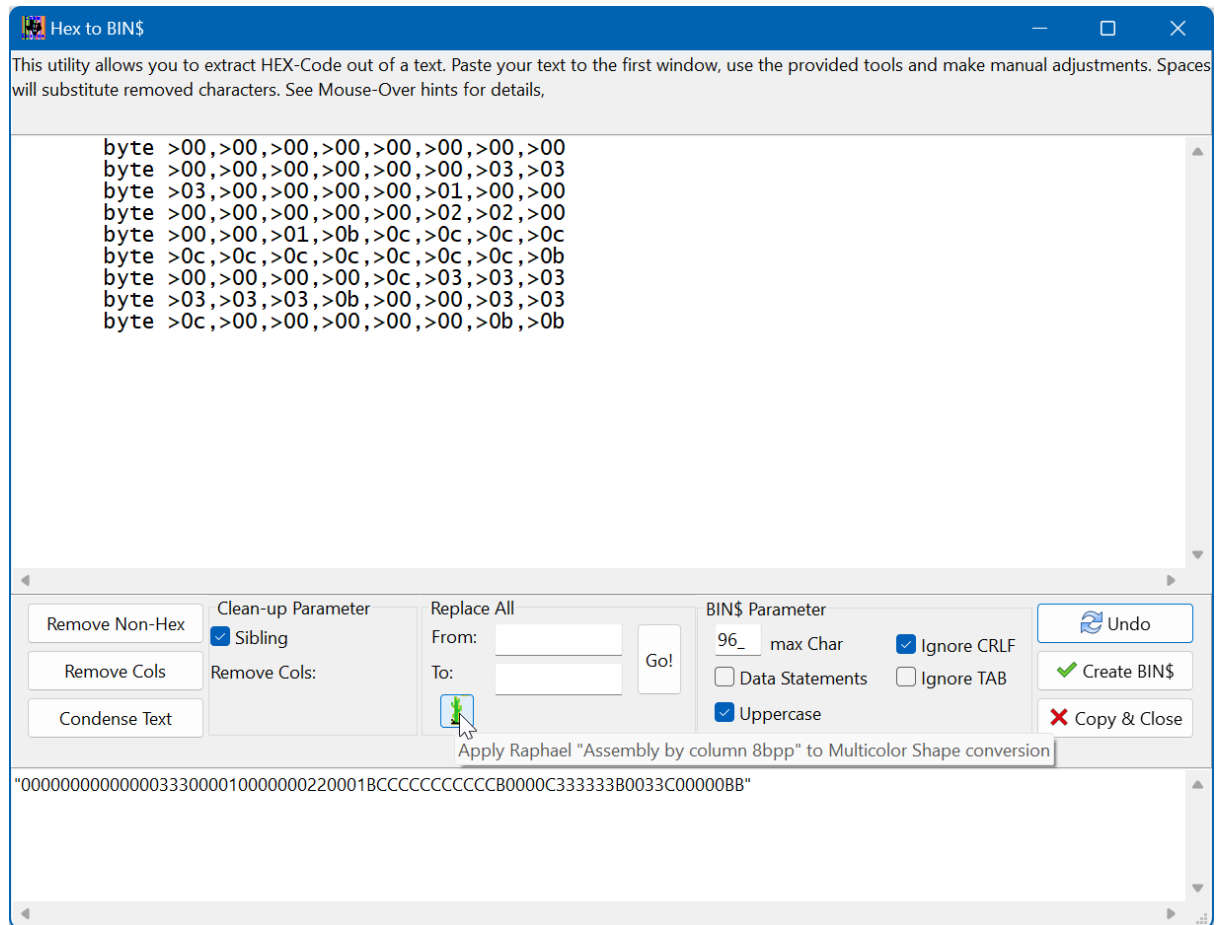
```

00000000
00000033
30000100
00000220
001bcccc
cccccccb
0000c333
333b0033
c00000bb

```

Change lowercase letters to uppercase and combine the lines as needed, or

2. Use the TiCodEd integrated function “Hex to BIN\$” in the Util menu:



Paste your Raphael-Export and the Cactus-Speedbutton will make all needed replacements.

Select “Uppercase” and “Ignore CRLF” to get a shape-string with “Create BIN\$” and copied to the Clipboard when exiting with “Copy & Close”.

# BSD License

Copyright (c) 2023, 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.

# Appendix A: Examples

## Lines

This is a very simple demo:

CALL LOAD("DSK4.MCOLOR.OBJ") CALL MCON RUN PRGRESTART  PRGRESTART: CALL SCREEN(2) CALL MCOLOR(4,0)  FOR I=1 TO 20 CALL MCCLR(4) CALL HLINE(24-i,32-i,11,i*2) CALL VLINE(25-i,32-i,11,i*2) CALL HLINE(24+i,33-i,11,i*2) CALL VLINE(24-i,32+i,11,i*2) CALL LINE(8,30-i,32,30+i,9) CALL LINE(8,1,i,64-i,15) CALL MCSYNC NEXT I END  CALL WAITKEY	100 CALL LOAD("DSK4.MCOLOR.OBJ") 110 CALL LINK("MCON") 120 RUN 130 130 CALL SCREEN(2) 140 CALL LINK("MCOLOR",4,0) 150 FOR I=1 TO 20 160 CALL LINK("MCCLR",4) 170 CALL LINK("HLINE",24-i,32-i,11,i*2) 180 CALL LINK("VLINE",25-i,32-i,11,i*2) 190 CALL LINK("HLINE",24+i,33-i,11,i*2) 200 CALL LINK("VLINE",24-i,32+i,11,i*2) 210 CALL LINK("LINE",8,30-i,32,30+i,9) 220 CALL LINK("LINE",8,1,i,64-i,15) 230 CALL LINK("MCSYNC") 240 NEXT I 250 END 260 CALL WAITKEY 270 SUB WAITKEY 280 CALL KEY(3,K,S) 290 IF NOT (S=1) THEN 280 300 SUBEND
---	--

- Load the library
- Start the Monitor to avoid overwriting of the screen by XB
- Restart with the following line-number
- Initialize the screen
- Loop of line drawings
  - clear the screen buffer
  - draw lines
  - synch buffer to the screen
- wait for a key to be pressed

The files provided:

Lines.sxb	SXB source (left column)
Lines.xb	XB source as text (right column)
Lines	FIAD format XB File

# DirMode

This example uses the “direct draw mode”, instead of writing to the buffer first and then do a MCSYNC. This mode is enabled with the second and optional parameter CALL MCOLOR(4,1). It may be also be activated/deactivated with CALL MCMODE(x).

This demo lets you paint on the screen with the joystick or ESDX, switching color with the fire-button or Q and exit with B.

CALL LOAD("DSK4.MCOLOR.OBJ") CALL MCON RUN PRGRESTART  PRGRESTART: CALL SCREEN(2) CALL MCOLOR(4,1)  x=31 :: Y=23 :: C=11  repeat call joyst(1,jx,jy) call key(1,k,s) x=x+(jx/4)+(k=2)-(k=3) y=y-(jy/4)+(k=5)-(k=0) c=c-(k=18) if k=16 then k=5 call putpix(y,x,c) until k=16 end	100 CALL LOAD("DSK4.MCOLOR.OBJ") 110 CALL LINK("MCON") 120 RUN 130 130 CALL SCREEN(2) 140 CALL LINK("MCOLOR",4,1) 150 x=31 :: Y=23 :: C=11 160 call joyst(1,jx,jy) 170 call key(1,k,s) 180 x=x+(jx/4)+(k=2)-(k=3) 190 y=y-(jy/4)+(k=5)-(k=0) 200 c=c-(k=18) 210 if k=16 then k=5 220 CALL LINK("PUTPIX",y,x,c) 230 IF NOT (k=16) THEN 160 240 end
---	---

The files provided:

DirMode.sxb		SXB source (left column)
DirMode.xb		XB source as text (right column)
DirMode		FIAD format XB File
Compiled		Directory with the compiled files and adjusted sources
	DirMode-M	Generated MERGE file with lowercase CALL LINK
	DIRMODE.TXT	Assembler Source generated from the compiler
	DirMode.obj	Assembled object-code
	DirMode-X	Compiled and linked program
	DirMode-C	Copied and adjusted MCOLORC to load the Multicolor Library and start the DirMode-X program

# Train to Bisbee

A more complex demo, using CALL SHAPE and CALL BLIT to draw saguaros of different sizes, the sun, two clouds and the valley/hill element on the horizon. The screen is initialized in yellow, a square is used to paint the sky and then all shapes are drawn and their position updated. The X-position is tenfold and therefore divided by 10 when drawing the shape in order to allow one decimal in the position and increment and still use integers with the compiler.

The files provided:

Bisbee.sxb		SXB source
Bisbee.xb		XB source as text
Bisbee		FIAD format XB File
Compiled		Directory with the compiled files and adjusted sources
	Bisbee-M	Generated MERGE file with lowercase CALL LINK
	BISBEE.TXT	Assembler Source generated from the compiler
	Bisbee.obj	Assembled object-code
	Bisbee-X	Compiled and linked program
	Bisbee-C	Copied and adjusted MCOLORC to load the Multicolor Library and start the DirMode-X program
	Bisbee.xbp	TiCodEd Project-Definition
	Bisbee.vxr	TiCodEd variable cross-reference
Graphics		Graphics used in the demo
	*.rap	Raphael graphics files
	*.a99	exported assembler code of the graphics

## Quix

Simple line-draw demo in writethrough-mode. Files similar to "Train to Bisbee".