NextBuild v7 (Nextlib & Boriel ZX Basic)

Introduction

NextBuild is a collection of tools and libraries which works with Boriel's ZX Basic compiler for working with the ZX Spectrum Next.

Version 7 has a number of significant changes under the hood so it's recommended that you start a new project to get familiar rather than attempt to port a v6 source - in all likelihood v6 sources should run with little or no modification.

Nextbuild V7 now produces NEX files (along with the original .bin files) that will contain all your assets to produce a single file. You can still load and save from SD as and when required, SNA files are no longer produced and the reliance on snasm has been removed.

The preferred IDE is now Visual Studio Code, setup is straightforward with all build scripts / tasks and extensions being automatically configured. Please see : VIDEO for steps. You can still use your own choice of editor (eg NP++, Sublime) and the original BorIDE is still included but will work in a reduced capacity compared with VSC.

Windows, Linux and Mac should now all be supported if you choose to use VSCode as the compilation scripts have been written in python making them cross platform.

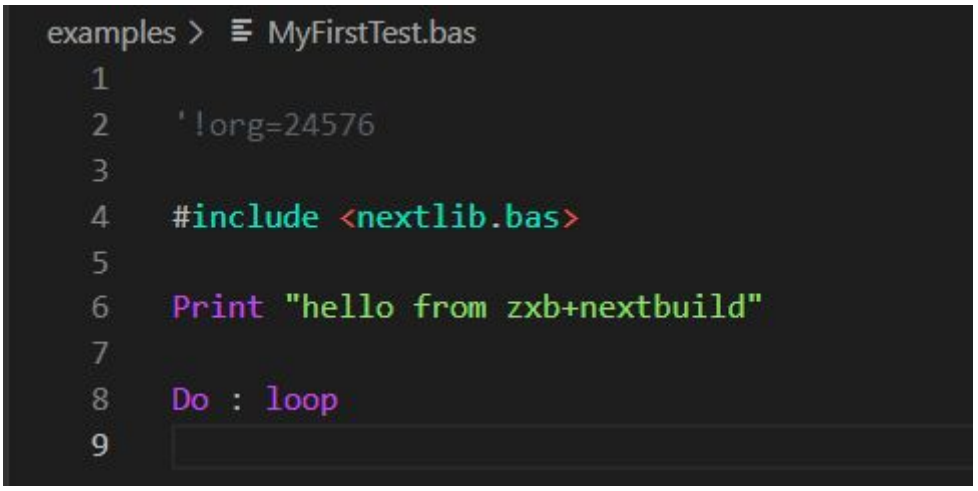
Preprocessor Options

It is important to note in version 7 preprocessor options are set at the top of your .bas file, the following options are available:

```
'!org=nn                nn = program start 24576-65530 if not set def= 32768
'!heap=nn               nn = heap size in bytes, if not set default 2048
'!opt=n                 n = optimization value 0 - 4, if not set 4
'!bmp=filename          filename = bmp loading screen displayed when loading NEX
'!copy=filename         eg. h:\mytest.nex copy your NEX file to h:\mytest.nex
'!noemu                 do not launch emulator after successful compilation
'!asm                   produce ASM output, no emu or NEX
```

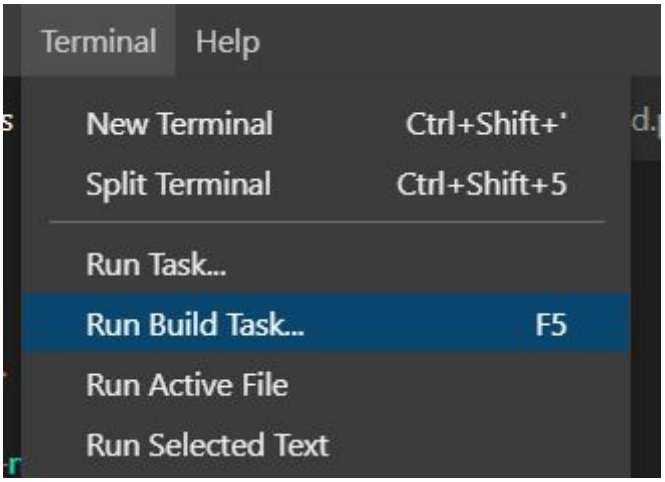
Common template

A normal example .bas file would look as follows:

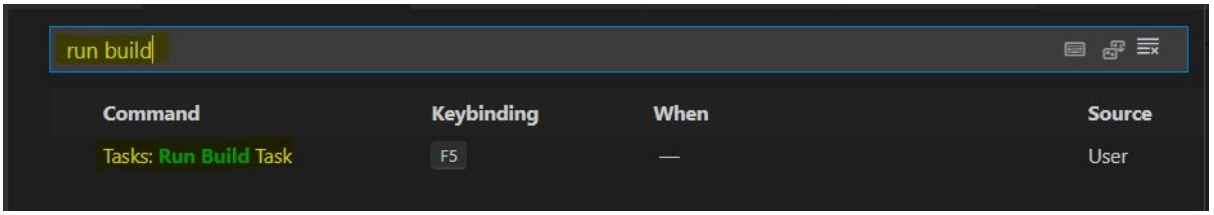


The Do ： Loop at the end prevents the NEX from finishing which would cause the Next to soft reset

Save this file in **Sources/examples** as **MyFirstTest.bas**, then **COMPILE** by choosing the **TERMINAL** menu and selecting **“Run Build Task”**



You should see the compilation info being printed at the bottom of VSCode. Note you can customise the ***keyboard shortcut*** for **Run Build Task** in VSCode in the **File menu/Preferences/Keyboard Shortcuts**, in the search type “run build” and you should see the current binding, right click and chose “**Change KeyBinding**” to pick a new shortcut.



```
================================================================================================
NextBuild v7      : David Saphier / em00k - 14-Jan-2021     https://github.com/em00k/NextBuild
ZX Basic Compiler : Jose Rodriguez aka Boriel               https://zxbasic.readthedocs.io/
Cspect Emulator   : Mike Dailly                             https://cspect.org
================================================================================================
Input File : C:\NextBuildv7\Sources\examples\MyFirstTest.bas

Looking for ORG...
Found ORG    :  24576
Working Dir  :  C:\NextBuildv7\Sources\examples
Compiling    :  C:\NextBuildv7\Sources\examples\MyFirstTest.bas
ZXbasic ver  :  1.15.0-beta4

YAY! Compiled OK!
================================================
```

Note in the above you can see useful information such as the **ORG** address detected, name of the source file and location along with the version of ZXBasic being used.

If all is successful with no errors you will see **YAY! Compiled OK!** If you have an error, your machine's text viewer will open with the **COMPILE.TXT** file which will explain the error.

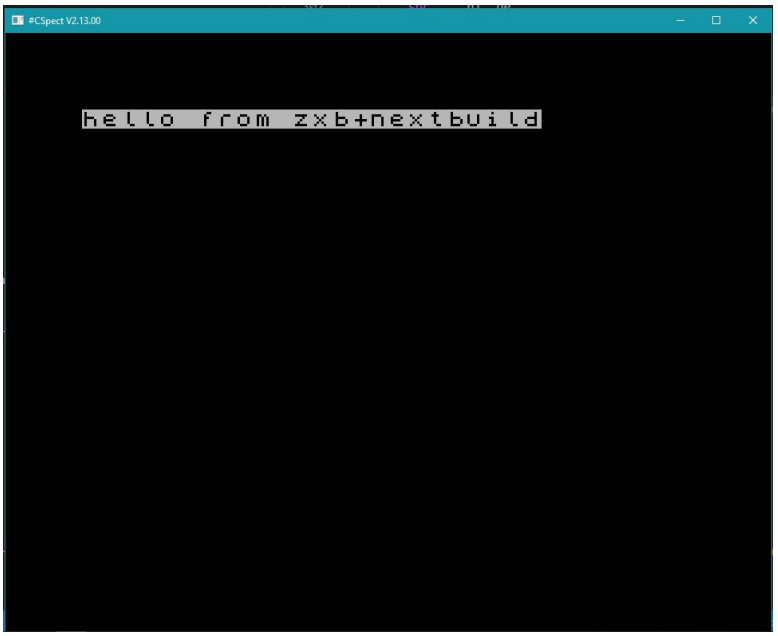
Let's assume everything has compiled OK and no mistakes have been made you will see the NEX creation part:

```
================================================
Generating Nexcreator Config ...

Saved config file : MyFirstTest.cfg
================================================
Generating NEX : C:\NextBuildv7\Sources\examples\MyFirstTest.nex

Requires Core 3.0.0 or greater
File '../../Tools/sysvars.bin' 8K bank 10, 1c00 (16K bank 5, 1c00)
bank=5,addr=1c00,realbank=0,256
PC=$6000
SP=$5ffe
bank=5,addr=2000,realbank=0,3609
Generating NEX file in V1.1 format
Generating NEX file for 1MB machine

Compile Log  :  C:\NextBuildv7\Sources\examples\Compile.txt
Memory Log   :  C:\NextBuildv7\Sources\examples\Memory.txt

NEX created OK! All done.
Overall build time : 0:00:01.0s
```

You should see any data files that have been included (there are none in this example) and the NEX being generate OK.

The Compile & Memory Log can easily be opened in VSCode with CTRL+click on the filename

If successful CSpect emulator should launch with our test program:



You can press ESC to quit CSpect (nb F1 will enter the debugger but more on the later)

Lets example our example to include some data in our NEX file and use Layer2

```
examples > MyFirstTest.bas
1
2   '!org=24576
3
4   #include <nextlib.bas>
5
6   LoadSDBank("font13.spr",0,0,0,35)
7
8   L2Text(0,0,"HELLO FROM NEXT BUILD",35,0)
9
10  Do : loop
11
```

A few notes on the code above

We can load data from SD into a memory bank on the Next with the following command:

```
LoadSDBank(fname,address,size,offset,bank)
```

This command is not designed to enabled at runtime, it's more of a placeholder so when we generate a NEX cfg file we can see what data we need to put into what banks. Here we load the file “font13.spr” that is in the data folder into 8kb bank 35, 0 bank address offset, length 0 means it will be automatically detected, 0 offset inside the file. Important : when producing

your “production” NEX file its important to use a define before your include the nextlib like so :

```
4   #define NEX
5   #include <nextlib.bas>
```

#define NEX will disable the LoadSDBank commands as all data will be included when we generate the NEX and we do not need to load the files from SD when the program runs.

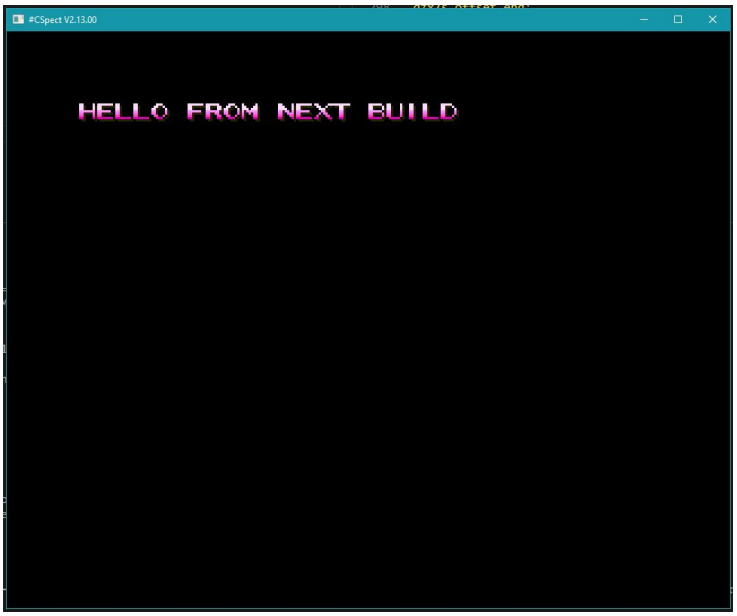
```
L2Text(x, y, string$, bank, colour mask)
```

This command uses the font in “bank” to draw text on Layer2. The colour mask is which colour do we want to mix with, so we can overly text on an image without have black portions around the text.

You should be able to see in this example that when the NEX file is being created a data file is now included

```
Generating Nexcreator Config ...

Saved config file : MyFirstTest.cfg
==================================================
Generating NEX : C:\NextBuildv7\Sources\examples\MyFirstTest.nex

Requires Core 3.0.0 or greater
File '../../Tools/sysvars.bin' 8K bank 10, 1c00 (16K bank 5, 1c00)
bank=5,addr=1c00,realbank=0,256
File './data/font13.spr' 8K bank 35, 0000 (16K bank 17, 2000)
bank=17,addr=2000,realbank=17,4096
PC=$6000
SP=$5ffe
bank=5,addr=2000,realbank=0,3257
Generating NEX file in V1.1 format
Generating NEX file for 1MB machine
```

Once again our program should launch in CSpect



Well done! Your first program. Please see the other examples for more informations

Command Summary

SD Access

When running your code in CSpect the SD card will be mounted inside the /data/ folder.

LoadSDBank(filename$,address in bank, size, offset in file, 8kb start bank)

This is a very powerful command designed to simplify loading large data files into the Next's memory banks. There is no limit on the size of file apart from the number of banks available on a 2Mb Next.

filename$	= string of file to load
Address	= MOD 8192 into the bank (so you could start halfway in)
Size	= size to load, use 0 to autoload
Offset	= offset into file, so you can skip bytes
8kb bank	= bank to start loading, the bank will increment for every 8kb

Note using Address offset can lead to issues if you have not set #DEFINE NEX. Remember you can use #DEFINE NEX to disabled all the LoadSDBank code when you are finalising your NEX as this will save lots of bytes, the files will be added at the NEX creation stage rather than loaded in at the start of the program

LoadSD(filename$,address,size,offset) -

Loads a file from SD to address. Note you must specify the size it is not automatically detected unlike LoadSDBank

- `LoadSD("myscreen.scr",$4000,6912,0)`
    - Will load a file called myscreen.scr to memory address $4000

SaveSD(filename$,address,size)

Creates/Overwrites a file on SD from address by size

- SaveSD("mydata.dat",$8000,256)
    - Saves 256 bytes from $8000 to SD called mydata.dat

LoadBMP(filename$)

Loads a 256*192 256 colour BMP from SD card and displays on layer2. The palette must be in Next Index order or you will need to upload your own palette with PalUpload()

Layer2 Access

ShowLayer2(N)

0 = disables Layer 2
1 = displays Layer 2

CLS256(N)

N = clears layer 2 256*192 with palette index 0 - 255

ScrollLayer2(X,Y)

X = Scroll Layer 2 X pixels 0-319
Y = Scroll Layer 2 Y pixels 0-255

ClipLayer2(X1,X2,Y1,Y2)

Clips the visible area of Layer2
X1 = Left edge 0-255 , X2 = Right edge 0-255
Y1 = Top edge 0-255 , Y2 = bottom edge 0-255

Layer2 Tiles

DoTile(X,Y,Tile) (deprecated)

Draws a 16x16px tile at X,Y to Layer2 256*192. X & Y are in steps of 16 pixels.
You can draw a total of 16 tiles horizontally and 12 vertically.

Tile = tile number, the tile data should be stored at $c000 and would be 16kb in size. The maximum tile would be 63 - it is recommended to use DoTileBank16 as this limitation is removed.

DoTile8(X,Y,Tile) (deprecated)

Draws an 8x8 256 colour tile to Layer 2 256*192. The tile data should be stored at $c000 and the maximum tile would be 255. It is recommended to use DoTileBank8.

TBC

NB for now VScode has code snippets on all commands press CTRL+Space to show the helper.