EZBL- PIC 24 bootloader ( **PIC24HJ256GP610A)**

The following document will show step by step how to integrate a bootloader for the PIC 24H

1. Install MPLAB-X
2. Download ezbl-v2.04

We need the MPLAB-X IDE version 4.05

<https://www.microchip.com/development-tools/downloads-archive>

why?

The ex\_boot\_uart bootloader is compiled with that version.

I did not manage to get it working with 4.15 – not compiled.

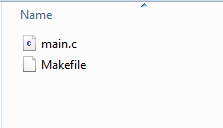
All other applications compiled ok with 4.15

For the github repo I actually removed almost everything, because we can download the ezbl-v2.04 from that link.

The ezbl example is working with PIC24F device , and here I am showing how to work with PIC24H.

The change of the device is important also for the understanding of the process because we need to replace the .S file from the boot loader into the application ezbl\_integration as described in this doc

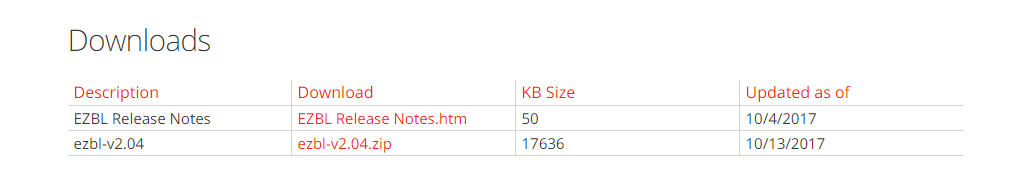
The git rep contains small main.c



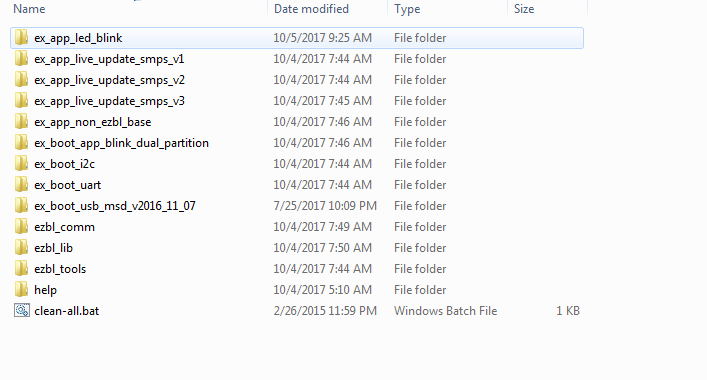
Which have a blink led.

You can replace those files with the main.c of the original ezbl zip file after extract to see a blinking led on the exploer 16/32 reference board.

<http://www.microchip.com/SWLibraryWeb/product.aspx?product=Microchip%20Easy%20Bootloader>



Extract to a folder:

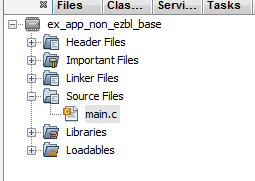


To understands the bootloader process we need to follow that PDF:

EZBL Hands-on Bootloading Exercises.pdf

To make sure your environment is working correct

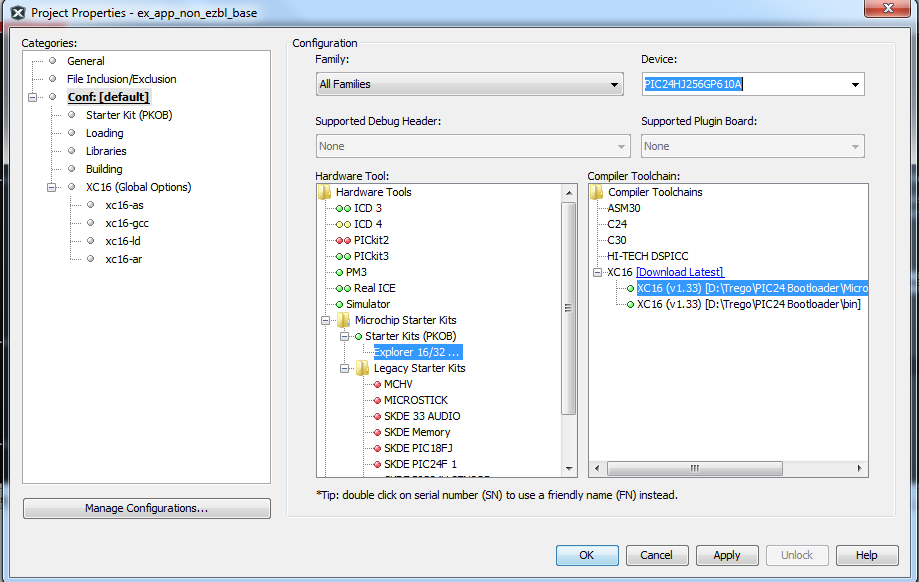
Open the none bootloader application



The device is **PIC24HJ256GP610A**

We assume you already have the compiler installed

Open the properties and select the device:



Select the device and the start kit (Explorer 16/32)

Blinking leds

The example project is a project of a different chip.

It will not compiled.

We want to see some blinking leds in a standalone program before we continue to the bootloader

Lets erase the main.c content and write the below code:

#include <xc.h>

#define FCY 16000000 // Execution speed for \_\_delay\_us()/\_\_delay\_ms() macros

#include <libpic30.h>

int main(void)

{

// Set RA<7:0> LED pins as GPIO outputs

LATA &= 0xFF00;

TRISA &= 0xFF00;

while(1)

{

LATA ^= 0x0055;

\_\_delay\_ms(500);

}

}

When pressing  we need to see:

Connecting to Starter Kit on Board...

Currently loaded firmware on Starter Kit on Board

Firmware Suite Version.....01.51.06

Firmware type..............dsPIC33F/24F/24H

Target voltage detected

Target device PIC24HJ256GP610A found.

Device ID Revision = 3003

Device Erased...

Programming...

The following memory area(s) will be programmed:

program memory: start address = 0x0, end address = 0x3ff

Programming/Verify complete

At this point we need to see 4 leds blink

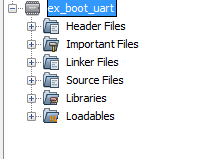
Next step

We need to compile and download the EZBL bootloader into PIC24

This step is explained here:



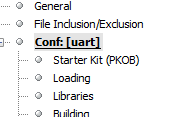
Add ex\_boot\_uart to the workspace

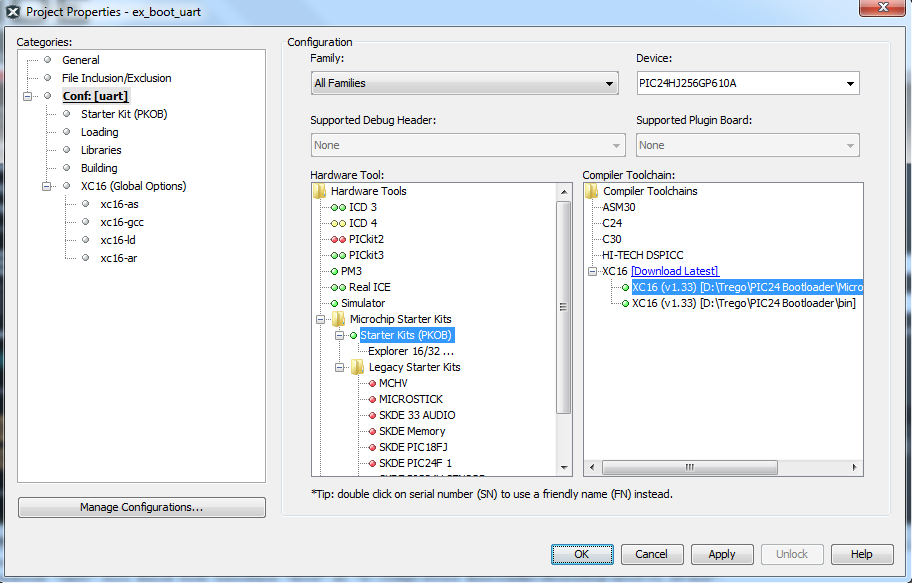


This is the actual bootloader firmware , we shell download this program and not change it.

Again, we need to select the device we have: **PIC24HJ256GP610A**

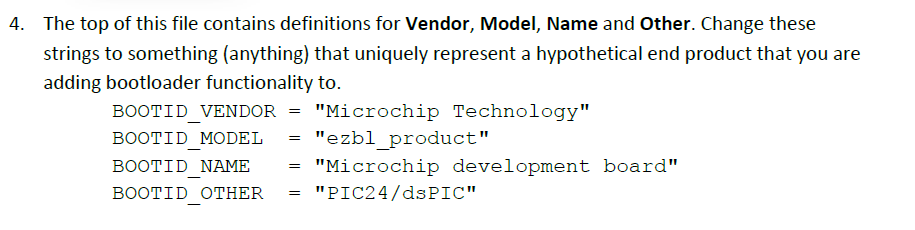
We can see here that the conf is set to Uart



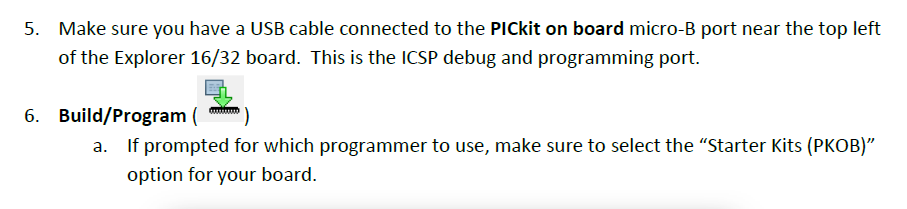


If you don’t see the Ok and apply, close and open MPLAB-X again.

As mentioned, you can change the string:

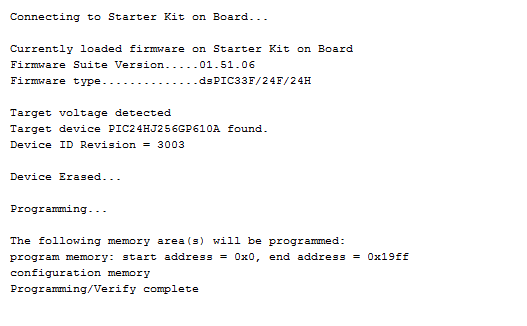


Nothing else to add , lets burned the bootloader into the chip:



Verify **LED D3** is blinking rapidly (8 Hz). If it is, your bootloader is successfully executing and awaiting application upload.

Results:



The bootloader has been downloaded and now the LED D3 is blinking as promise.

Next step

Following the

**Exercise 2 – Loading an example application**

We want to try load a sample application, another blink led application but we want to download it via the UART and not via the Debug port , loader USB Port.

Move it from here:



\*Figure 1

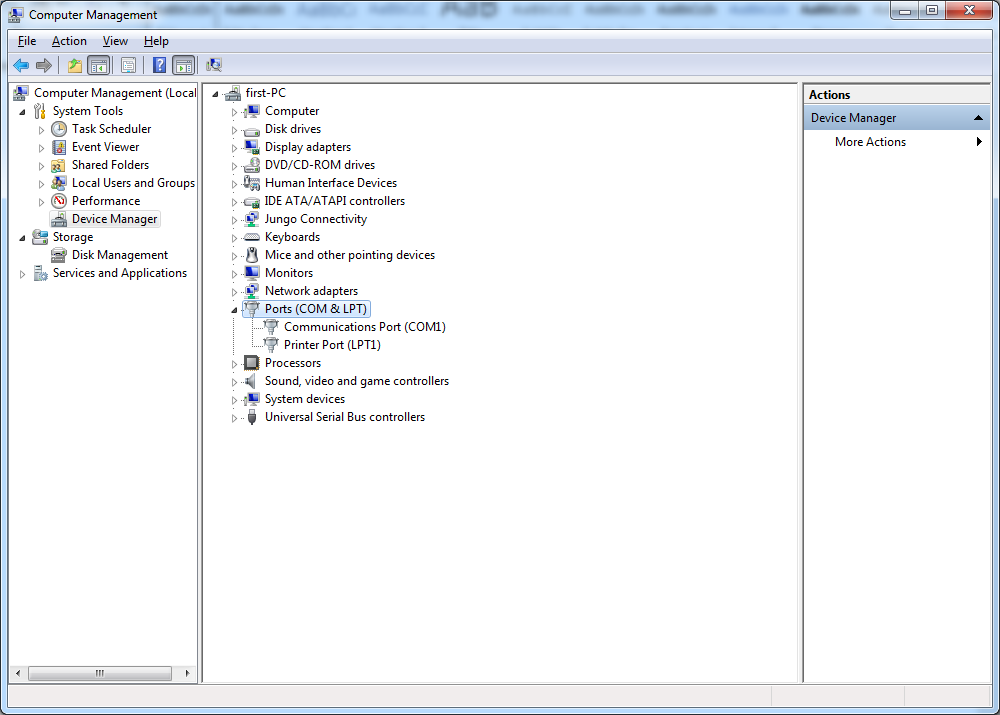
To Here:



\*Figure 2

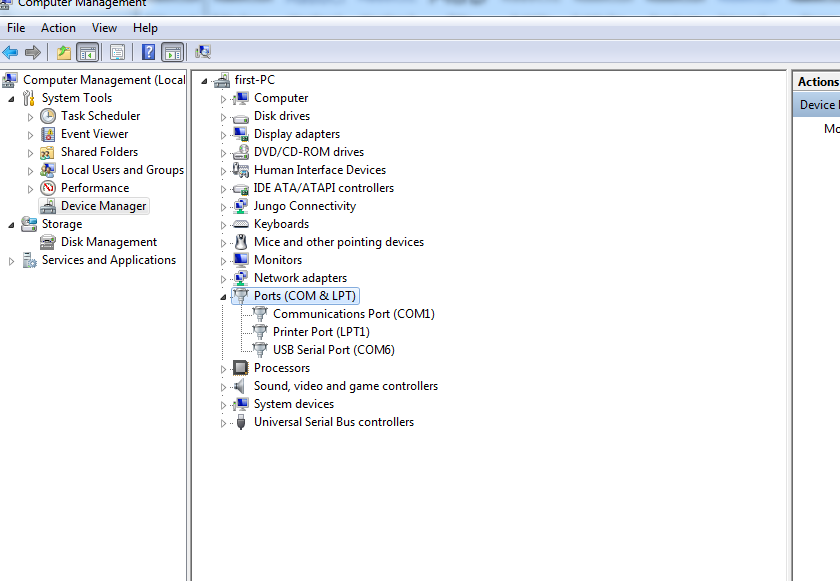
Note, when it connected to the USB Port ( Figure 1)

You will not see it as a serial com port

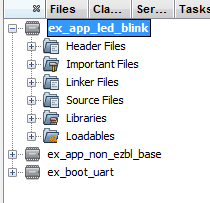


But when connecting it to the Serial port (Figure 2) , you can see in the device manager the serial com port that was assigned to the port ( COM6)

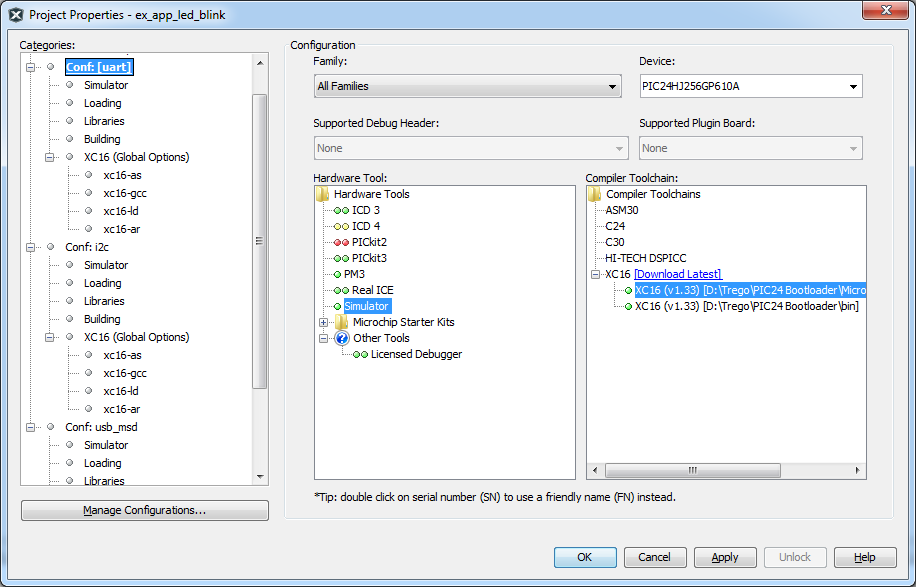
COM6



Insert the ex\_app\_led\_blink into the workspace, and make it active project



Again , we need to select the right device: **PIC24HJ256GP610A**



We can see here several things

1. There is not USB exploer any more because we are not connected to the USB debug port
2. The categories become extended , Uart, I2c usb\_msd.

The serial bootloader is done with a program written in Java.

The java utililty is integrated in the MPLAB-X , means, we can download the application via the bootloader directly from mplab.

Later on we also see how to do it out side the MPLAB-X environment.

We will also , Later on, we will need to learn how to integrate the bootloader code into our own code.

Let’s review the project code and sources before we download the led blink app via the bootloader

As we can see in figure 3, the led blink is not a simple main.c file by its own.

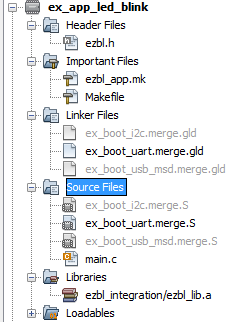
It contains , the ezbl bootloader files.

Ezbl.h

Ez\_boot\_uart.merge.S

Ez\_boot\_uart.mergde.S

Ezbk\_integration/ezbl\_lib.a

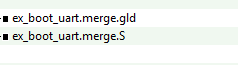


\*Figure 3

The content of the application that works with the ezbl bootloader should have the ezbl\_integration directory:



The two files



Need to be taken from the ezbl\_integration of the bootloader , means from the

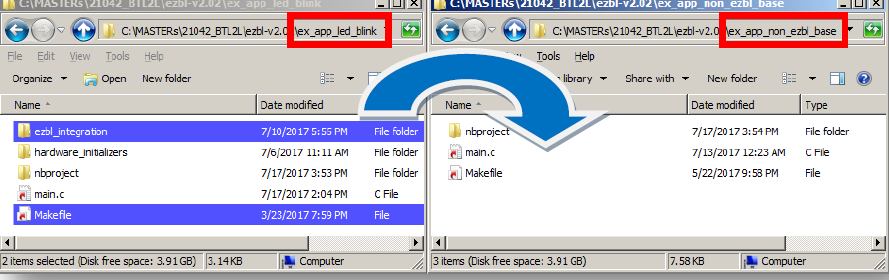
ex\_boot\_uart

the ex\_boot\_uart.merge.S will be different for different PIC 24 devices , so it is important to take it from the bootloader.

In the exercise , because they already prepare a bootloader ex\_app\_led\_blink

They tell us to take the files, including the makefile directly from this directory into our new project.

For example , the muze.



We can do that but make sure the .S and.gld files are the same as the bootloader files!

In the ex\_boot\_uart.merge.S we can see at the beginning the following:

/\*\*

\* EZBL Bootloader Code and RAM Allocation

\*

\* Automatically generated file - not intended for manual editing. If changes

\* are made here, they will normally be overwritten when you rebuild your

\* Bootloader. If necessary, maintain a backup copy and manually merge your

\* customizations back in.

\*

\* Built for:

\* PIC24HJ256GP610A

\* From:

\* dist\uart\production\ex\_boot\_uart.production.elf

\* Using build configuration:

\* uart

\*/

That help to understand for which device the bootloaded was prepared for.

Now , lets update the ex\_app\_led\_blink via the uart with the help of the bootloader

Open the file:

**ezbl\_app.mk**

ifneq (,${filter default uart,${CONF}}) # Check if "default" or "uart" MPLAB project build profile is used

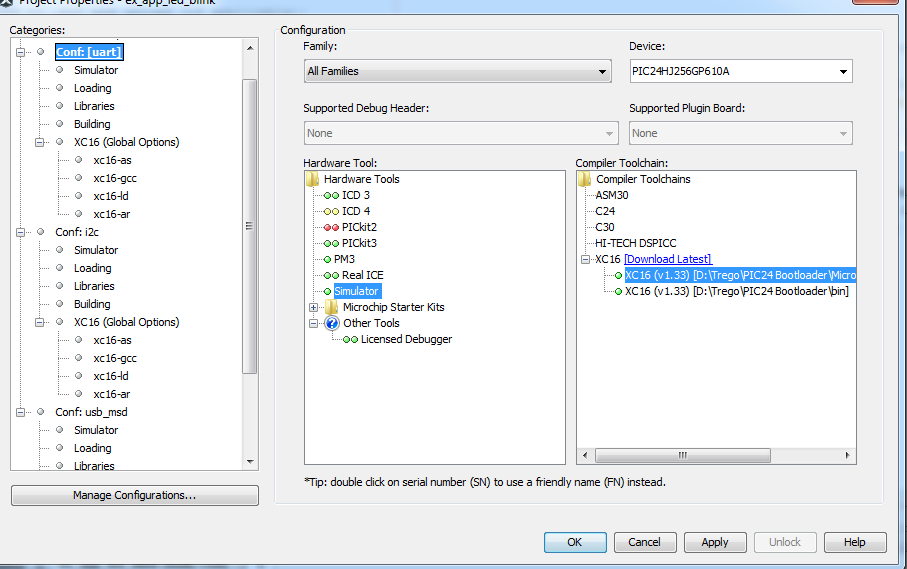
@echo EZBL: Attempting to send to bootloader via UART

${MP\_JAVA\_PATH}java -jar "${thisMakefileDir}ezbl\_tools.jar**" --communicator -com=COM6** -baud=230400 -timeout=1100 -artifact="${DISTDIR}/${PROJECTNAME}.${IMAGE\_TYPE}"

And change the right COM port that was selected to your serial port.

Instead if com21, I change it to COM6.

In the properties, select simulator, make sure your device is correct and select conf:uart



The download is done immediately after the build process is done.

Right click on the project and select build.

make[2]: 'dist/uart/production/ex\_app\_led\_blink.production.hex' is up to date.

EZBL: Converting .elf to .bl2

test "dist/uart/production/ex\_app\_led\_blink.production.bl2" -nt "dist/uart/production/ex\_app\_led\_blink.production.elf" || "C:\Program Files (x86)\Microchip\MPLABX\v4.05\sys\java\jre1.8.0\_144/bin/"java -jar "ezbl\_integration/ezbl\_tools.jar" --blobber -artifact="dist/uart/production/ex\_app\_led\_blink.production.elf"

EZBL: Attempting to send to bootloader via UART

"C:\Program Files (x86)\Microchip\MPLABX\v4.05\sys\java\jre1.8.0\_144/bin/"java -jar "ezbl\_integration/ezbl\_tools.jar" --communicator -com=COM6 -baud=230400 -timeout=1100 -artifact="dist/uart/production/ex\_app\_led\_blink.production"

Upload progress: |0% 25% 50% 75% 100%|

|..................................................|

11606 bytes sent in 0.763s (15211 bytes/second)

BUILD SUCCESSFUL (total time: 1s)

Loading code from D:/Trego/Pic24\_Bootload\_4\_1\_2018/ezbl-v2.04/ex\_app\_led\_blink/dist/uart/production/ex\_app\_led\_blink.production.hex...

Loading completed

You can change the speed:

ledTimer = NOW\_32();

while(1)

{

// Every 500ms toggle an LED

if(NOW\_32() - ledTimer > NOW\_sec/2u)

{

ledTimer += NOW\_sec/6u;

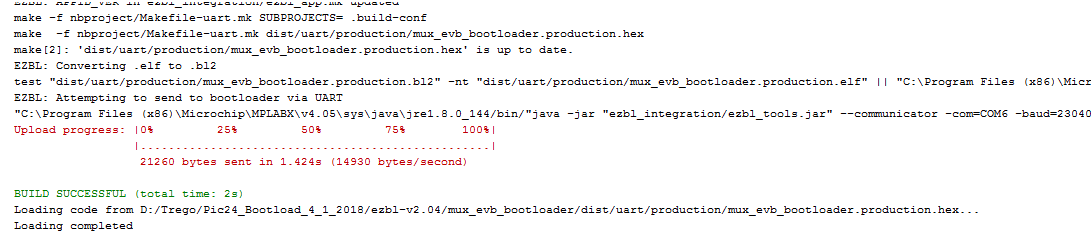
LEDToggle(0x01);

}

Idle();

}

Build , and see that it is being updated each time.



I needed to change the timers because the bootloader is sharing timers.

See document muze\_evb\_bootloader.docx

Next step

Using a command line script to upload the firmware via the bootloader

See document:

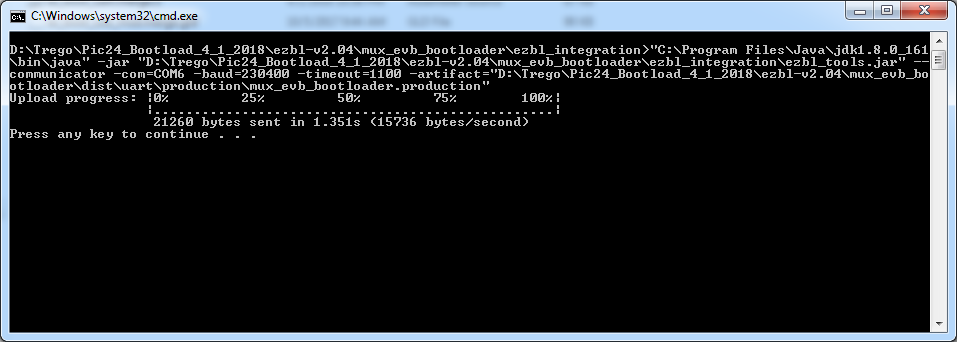
ezbl\_java\_loader.docx

In case your java.exe is not in the path, you can add it to path or add the entire path:

"C:\Program Files\Java\jdk1.8.0\_161\bin\java" -jar "%~dp0ezbl\_tools.jar" --communicator -com=COM6 -baud=230400 -timeout=1100 **-artifact="D:\Trego\Pic24\_Bootload\_4\_1\_2018\ezbl-v2.04\mux\_evb\_bootloader\dist\uart\production\mux\_evb\_bootloader.production"**

@pause

The end 😊



Our application is being updated using external script based on java via UART.