Documentation (English)

Documentation written by: Myoda
Traduced by : Paulo Cracel Silva

Version of the document: 20180812

Contents

1 F4ToSerial program 4

1.1 - Introduction 4

1.1 - Methodology 4

1.2 - What type of electronic card should I use with F4ToSerial? 5

1.3 – How does it send data to a cockpit? 6

1.3.1 – Stage 1: Shared Memory 6

1.3.2 – Stage 2: F4ToSerial 6

1.3.3 – Stage 3: Signal recovery and transformation 7

In this step, the Arduino board receives information about the serial port and transforms it into electrical signals that are compatible with active elements (gauges, LEDs, 7-segment displays and displays). 7

1.3.4 - Stage 4: Activation of the cockpit elements 7

2 Download and install the F4ToSerial program 8

2.1 - Download the F4ToSerial program 8

Earlier versions of the program are currently available as well. 8

2.2 - Downloading ++ code for Arduino 8

2.2.1 – Easy method: 8

2.2.2 – Alternative method: 10

2.2.3 - Complete method: 10

3 F4ToSerial User’s Manual 12

3.1 - Configuration backup and restoration 12

3.1.1 - Configuration Backup 12

3.1.2 - Restoring the configuration 12

3.2 – Configuration of active elements 13

3.2.1 - Gauges 14

3.2.2 - Displays 20

3.2.3 - Leds or lightBits 27

3.2.4 - 7 Segments displays 35

3.3 - General settings 42

3.3.1 - Download the latest version 42

3.3.2 - Refresh rate 42

3.3.3 - Serial port speed 43
3.3.4 – Show the debug console 43
4 Bugs and possible issues 43
Nothing happens when I open the serial ports, why? 43

1 F4ToSerial program

1.1 - Introduction

The F4ToSerial program was developed with the aim of including in a single application the control of all "active" elements of a simulation cockpit.

By active elements we mean those who receive information from the simulator and react according to: gauges, lights, 7 segment displays and screens.

F4ToSerial has been developed exclusively for Falcon 4 BMS simulator from version 4.33 or higher.

The information comes from the shared memory of the simulator, and is routed via the serial port to control cards able to drive these so-called "active" elements.

F4ToSerial is an "all in one" program that aims to provide an alternative solution to the multiplicity of programs currently available for Falcon BMS.

The F4ToSerial program nevertheless has two major constraints.

- - It only works in one direction (Falcon BMS → Cockpit)
- - It was developed for the "Block 52" version of the F16.

Sending the data from the cockpit to the simulator (so in the other direction) does not represent any difficulty and will not be discussed here, since it involves simulating keystrokes in 90% of cases.

The F4ToSerial program has been developed to work with Arduino type electronic boards and the code of these boards is provided.

Finally, note that this program was been developed by myself as part of the creation of my home cockpit and was put online for the community. It cannot be used for professional purposes and remains free.

1.1 - Methodology

When I first started designing my F16 simulator cockpit, I quickly realized that money is a determining factor in the quality of the finished cockpit. While time and expertise matters but a high-level and finished cockpit costs tens of thousands of euros ... and I do not lie!

Not having an expandable budget, I had to make sure to consume as few components as possible for the creation of my cockpit.

There are generally two approaches to building a cockpit. In one, the budget is not taken into account, and in the other we pay attention to each euro invested.

This principle applies to "lightBits" and 7 segment displays for example, but also more generally to all modules of the cockpit, buttons, switches, screens etc. but also to ACESII, throttle, etc.

On the other hand, the simpler the composition, the more the connections and this simplification make the assemblies complex and difficult to develop.

Not being a genius handyman or a born electronics engineer, my approach is halfway between the two methods.

That's why in this guide I chose to use this or that method that sometimes simplifies and sometimes consumes the expense.

Of course, everything can be improved and I do not claim to have the quick fix.

1.2 - What type of electronic card should I use with F4ToSerial?

There are several types of cards that are very useful for the simulator cockpit builder.

Photon, Arduino and Pokeys cards etc. The latter will not be discussed in this document because its use with the F4ToSerial program is not yet implemented.

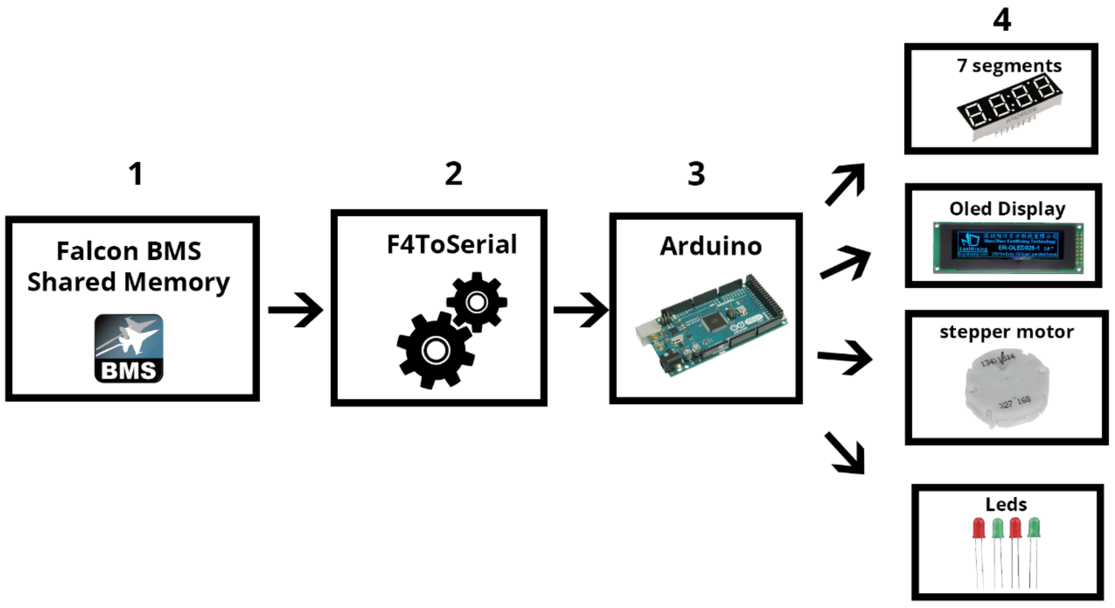
Each has advantages and disadvantages and this document is not intended to make a comparison or guide you when choosing your card.

Note, however, that the F4ToSerial program was developed only for the Arduino Mega board.

The Arduino UNO board is compatible but due to its small amount of memory, it cannot use one of the essential libraries of the C++ program.

Other cards are not supported yet and are not planned for the moment.

1.3 – How does it send data to a cockpit?



Before having the gauges that work in a simulator cockpit or the F16's DED screen that displays information, a number of steps are required. The previous diagram describes these steps which are detailed just after.

1.3.1 – Stage 1: Shared Memory

Shared memory is an area that the developers of Falcon BMS have kindly implemented and documented to allow cockpit builders to read the current information of the aircraft and the flight: (altitude, heading, position of joysticks etc.).

The first step is therefore to access the information register of the shared memory. F4ToSerial uses the Windows F4SharedMem.dll DLL which is installed in the installation directory of the F4ToSerial program.

1.3.2 – Stage 2: F4ToSerial

Stage 2 retrieves data from shared memory, transforming it and sending it to the cockpit. In this step, the F4ToSerial program makes a permanent loop on the memory addresses, then retrieves and prepares the information before sending it to the serial port.

The data is transformed for example from the binary into a string readable by the Arduino card at the next stage.

Examples of data frames in Json format.

Configuring a stepper motor:

```
{"SETUP_STEPPER":{"RPM":{"NAME":"RPM","PINS":[46,44,42,40],"PINSCOUNT":4,"STEPS":600,"MAXSPEED":1000,"ACC
ELERATION":1000}}}
```

Update of an array of LEDs:

```
{"UPDATE_MATRIX":{"A":{"MATRICE":[[1,1,0,0],[1,0,0,1],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]}}}
```

Displaying information from the DED screen:

```
{"SET_DISPLAY":{"DED":[" UHF  292.30   STPT $ 8",""," VHF   1        10:56:20",""," M1 3 C  6400  MAN T 75X"]}}
```

1.3.3 – Stage 3: Signal recovery and transformation

In this step, the Arduino board receives information about the serial port and transforms it into electrical signals that are compatible with active elements (gauges, LEDs, 7-segment displays and displays).

1.3.4 - Stage 4: Activation of the cockpit elements

In this last step the active elements evolve according to the received signals (displacement of the gauges, lighting of LEDs etc.).

Note that all these steps still work in the Falcon BMS direction to the cockpit. The shared memory is accessible by read-only, any interaction with the simulator will be through keystrokes.

2 Download and install the F4ToSerial program

2.1 - Download the F4ToSerial program

You can download the F4ToSerial program from the website:

http://f4toserial.mini-cube.fr/.

The latest version of the application is available here:
http://f4toserial.mini-cube.fr/download-latest-version/

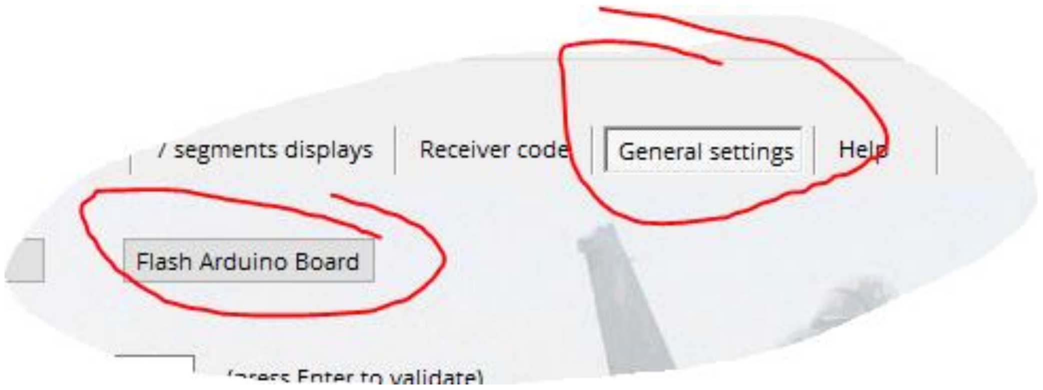
Earlier versions of the program are currently available as well.

2.2 - Downloading ++ code for Arduino

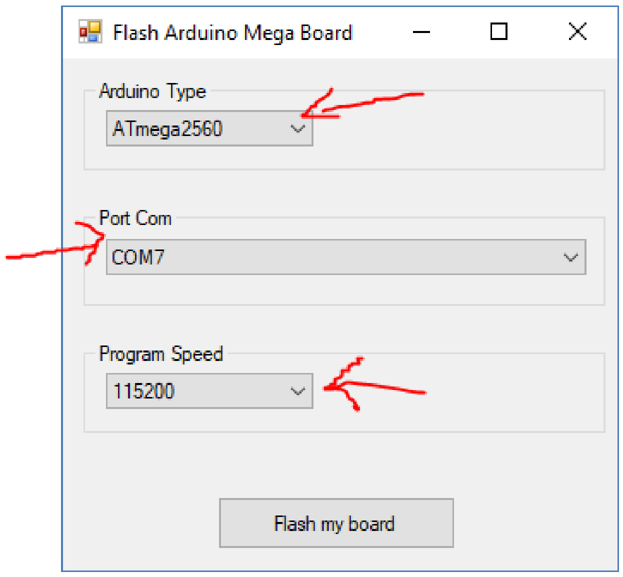
2.2.1 – Easy method:

1) From the F4ToSerial program, if you have an Arduino Mega 2560 board, you will be able to upload the C ++ program directly into your Arduino board.

2) To do this, click on the "General Settings" tab and then on "Flash Arduino Board".



3) 3) Then indicate the type of card (Arduino Type), the serial port (Com Port) and the speed (Program speed) then click on "Flash my board".


Flash Arduino Mega Board

Arduino Type: ATmega2560

Port Com: COM7

Program Speed: 115200

The "Program Speed" parameter is used to set the reading speed of the serial port of the Arduino board.

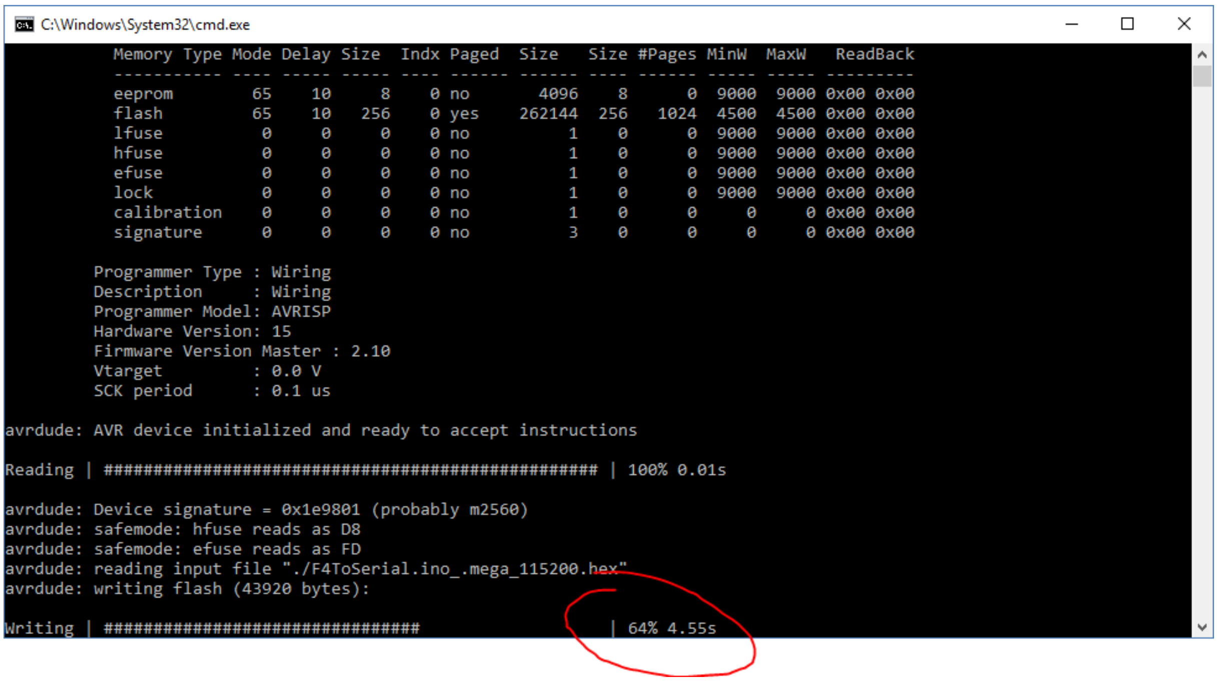
By default, the F4ToSerial program sends data on the serial port at 115,200 bauds.

Some cards can read data at a speed of 1,000,000 bauds.

This parameter corresponds to the Serial.Read () instruction of the C ++ code that is in the Arduino board.

It must therefore be identical to the transfer speed of the Serial Speed Baud Rate (F4ToSerial) program defined in the "General settings" tab..

4) A DOS command window opens, and the program is being transferred to the Arduino board.



C:\Windows\System32\cmd.exe

```
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW   ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- ---------
           eeprom        65    10     8    0 no       4096    8      0  9000  9000 0x00 0x00
           flash         65    10   256    0 yes    262144  256   1024  4500  4500 0x00 0x00
           lfuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           hfuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           efuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           lock           0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           calibration    0     0     0    0 no          1    0      0     0     0 0x00 0x00
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00

         Programmer Type : Wiring
         Description     : Wiring
         Programmer Model: AVRISP
         Hardware Version: 15
         Firmware Version Master : 2.10
         Vtarget         : 0.0 V
         SCK period      : 0.1 us

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9801 (probably m2560)
avrdude: safemode: hfuse reads as D8
avrdude: safemode: efuse reads as FD
avrdude: reading input file "./F4ToSerial.ino_.mega_115200.hex"
avrdude: writing flash (43920 bytes):

Writing | ################################                   | 64% 4.55s
```

2.2.2 – Alternative method:

This method is of interest if the F4Toserial program fails to flash your Arduino board or if the speed needs to be manually set.

1) Download here a program that allows you to transfer a Hex file to an Arduino board http://xloader.russemotto.com/
2) Transfer the Hexa code directly to the Arduino board (Mega (ATMEGA2560)). http://f4toserial.com/resources/

2.2.3 - Complete method:

To run your modules or active elements fully, you will also need to download C ++ code compatible with Arduino boards.

The code download link is here:

https://bitbucket.org/falconbms/

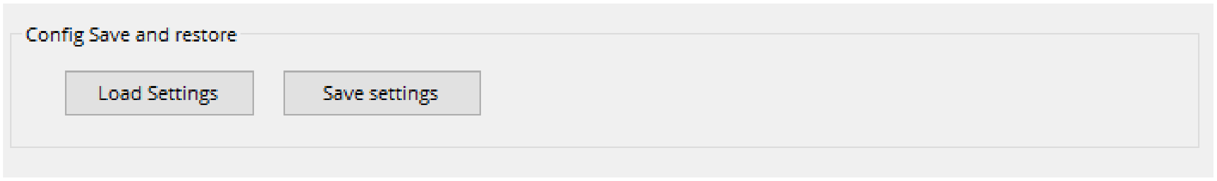
C ++ source files for the Arduino board are available here:
https://bitbucket.org/falconbms/arduino-commons.git

I invite you to clone the files of the repository with a tool of the type SourceTree in order to keep the files updated if new versions of the code C ++ are put on line.

3 F4ToSerial User’s Manual

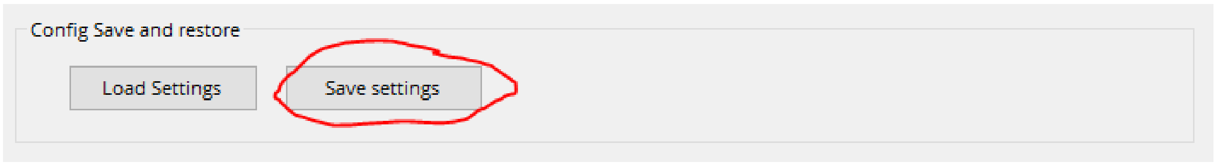
3.1 - Configuration backup and restoration

F4ToSerial allows saving the configuration. The latter is tedious to set up the first time, which is why it is important to register the latter at the end of the setting up of active teams.



3.1.1 - Configuration Backup

The backup of the configuration is done by clicking on the button "Save settings".



You must then specify a location and a name for the configuration file. The backup file extension is ".xml". It is not necessary to specify it.

Configuration files can be easily modified with an HTML / XML editor.

Here is an example of a configuration file:

```
<?xml version="1.0" encoding="utf-8"?>

<serial_settings>{"7_chaff":"","7_flares":"","7_fuel":"","7_uhf_chan":"","7_uhf_freq":"","aft":"","bcd_decoder_0":"COM1","bcd_decoder_1":"COM1","cabinAlt":"","ded":"","epu":"","ffi":"","ftit":"","fwd":"","hyd_press_a":"","hyd_press_b":"","Matrix_A":"","Matrix_B":"","Matrix_C":"","Matrix_D":"","Matrix_E":"","noz":"","oil":"","oxygen":"","pfl":"","pitch":"","roll":"","rpm":"","yaw":""}</serial_settings>
<steppers_speed_accell>{"aft":[1000,1000],"cabinAlt":[1000,1000],"epu":[1000,1000],"ftit":[1000,1000],"fwd":[1000,1000],"hyd_press_a":[1000,1000],"hyd_press_b":[1000,1000],"noz":[1000,1000],"oil":[1000,1000],"oxygen":[1000,1000],"pitch":[1000,1000],"roll":[1000,1000],"rpm":[1000,1000],"yaw":[1000,1000]}</steppers_speed_accell>
```

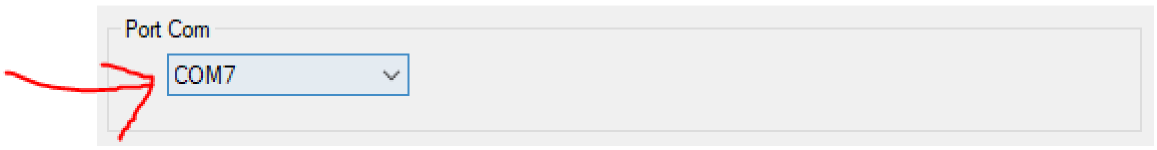
3.1.2 - Restoring the configuration

Loading the configuration is done by clicking on the "Load settings" button.

Any current configuration will be overwritten by the one in the configuration file when loading. So be careful not to overwrite a correct configuration because the operation cannot be cancelled.

3.2 – Configuration of active elements

The principle of setting the active elements or modules (gauges, indicators, 7-segment displays and displays) is first of all by choosing the serial port in the parameters of the element concerned.



Port Com

COM7

Each Arduino board connected to the computer has its own serial port.

Once connected, the list displays the names of available serial ports.

The list of Com ports is not refreshed in real time. Think about plug your equipment before launching the F4ToSerial program.

3.2.1 - Gauges

Overview:

I made several tests before making the decision to use x27.168 stepper motors for gauges because:

- They are not noisy compared to servo motors
- Their angle of rotation is high compared to a servo motor.
- They consume little current (can be plugged without external power supplies on the Arduino board)
- They are very fast.
- The cost of an x27 is very low (less than 2 € a piece).

The cockpit gauges that can be used in F4ToSerial are as follows:

Centre console:

- Rpm
- Noz
- Oil
- Ftit

Right console:

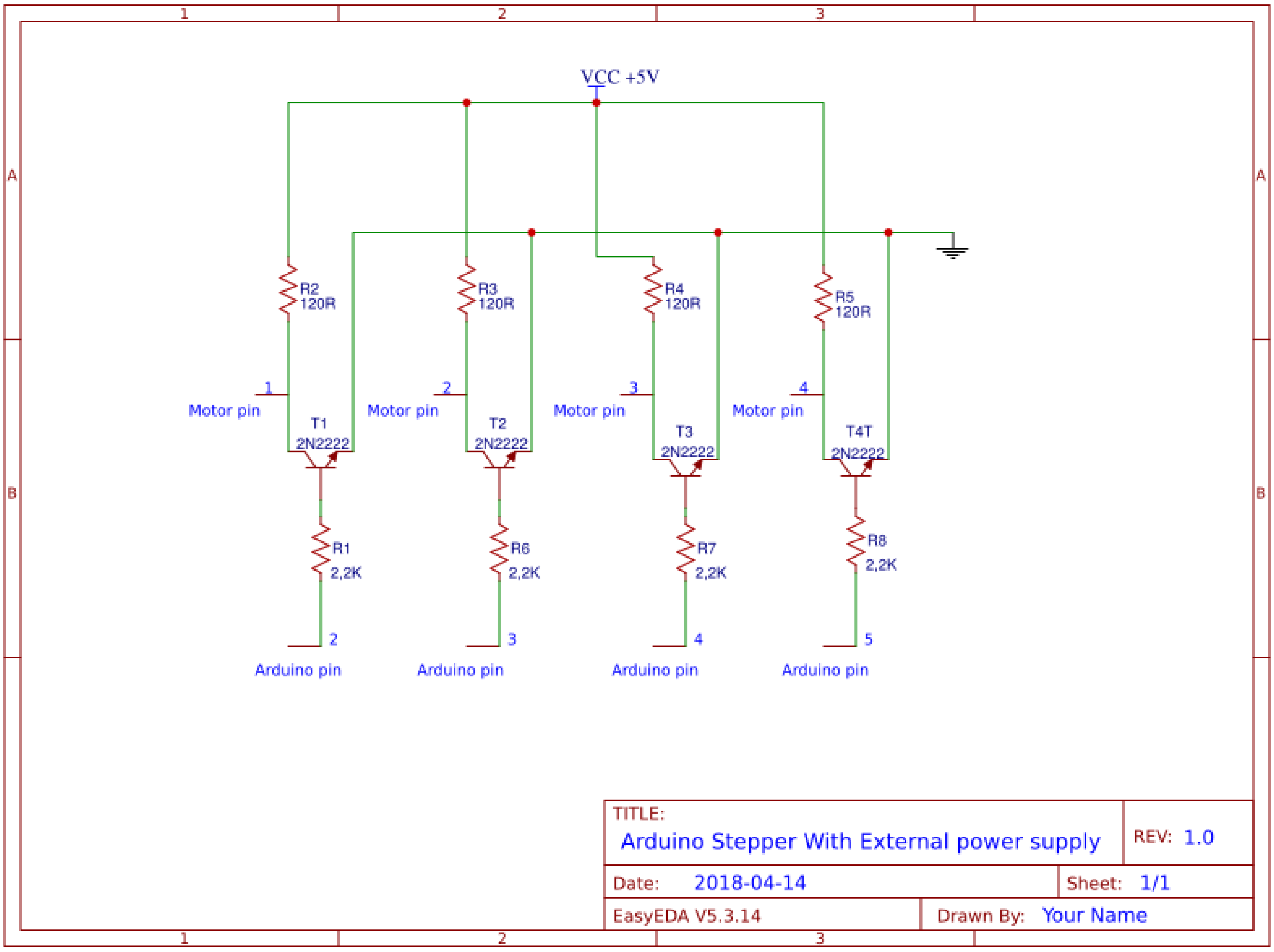
- Epu
- Fwd
- Aft
- Hyd Press A
- Hyd Press B
- Cabin

Left console:

- Roll Trim
- Pitch Trim
- Yaw Trim

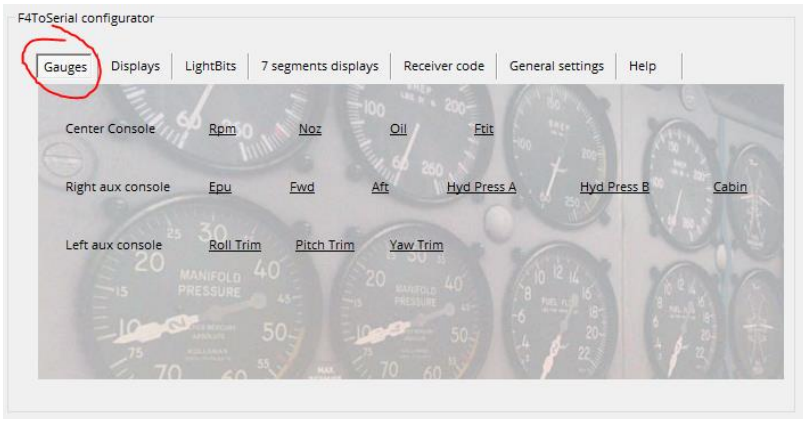
> *A special feature exists for the Hyd Press Gauge. The latter uses 2 needles on the same axis. In this case another type of stepper motor must be used: The x40.168.*

The following diagram illustrates how to wire the x27 motor with an external power supply. This makes it possible to connect all the motors to the same Arduino board.



Use in F4ToSerial.

In F4ToSerial, the gauges are configured in the first tab.

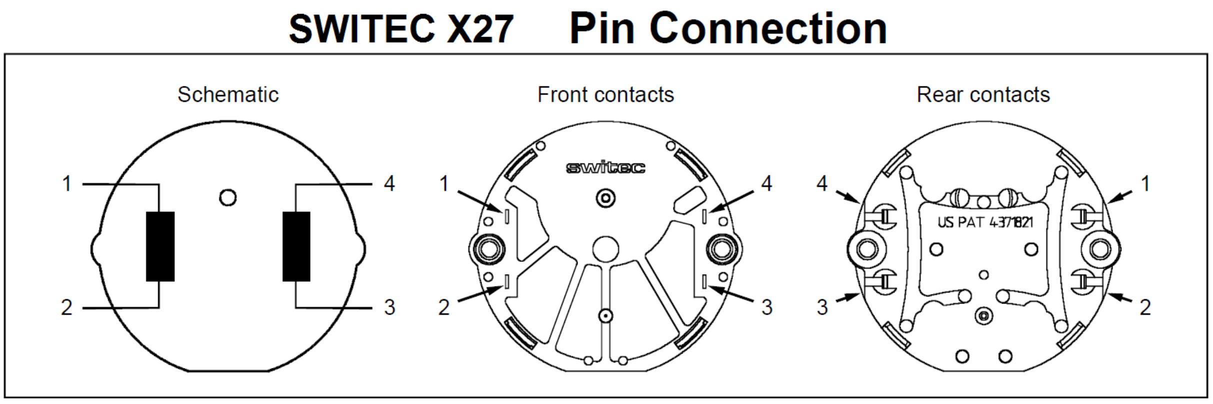

F4ToSerial configurator

Gauges | Displays | LightBits | 7 segments displays | Receiver code | General settings | Help

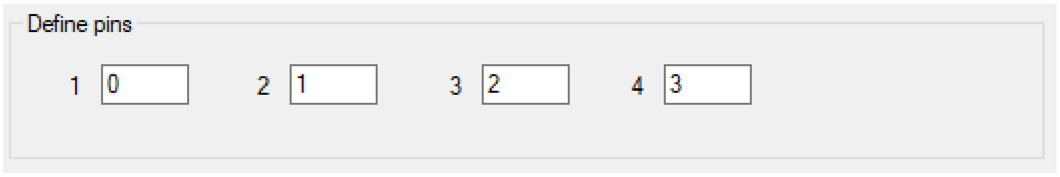
Center Console	Rpm	Noz	Oil	Ftit		
Right aux console	Epu	Fwd	Aft	Hyd Press A	Hyd Press B	Cabin
Left aux console	Roll Trim	Pitch Trim	Yaw Trim			

Click on the link of the module to access its configuration.

X27 stepper motors have 4 pins:



These 4 pins are represented in F4ToSerial in the "Define Pins" area

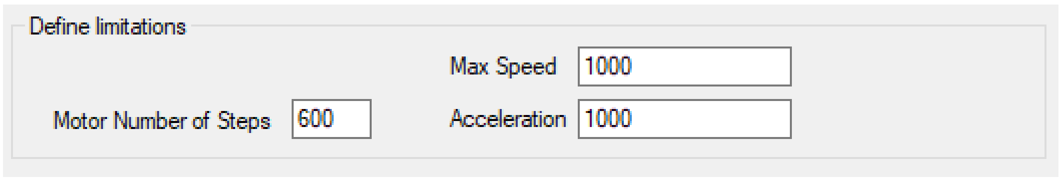

Define pins

1 [0] 2 [1] 3 [2] 4 [3]

Here you must enter the corresponding pin numbers on your Arduino board.

In this example, which is not correct, pin 1 of the motor is connected to pin 0 of the Arduino board and pin 4 of the motor is connected to pin 3 of the Arduino board.

The following area "Define limitations" allow you to configure the limits specific to X27 engines.

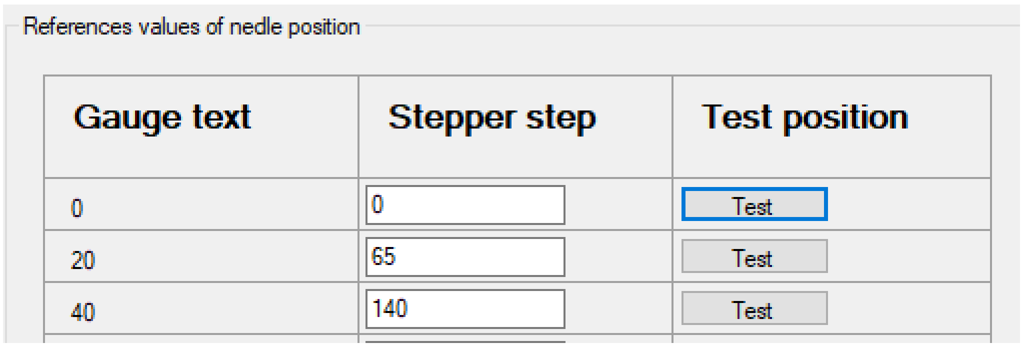

Define limitations

Max Speed	1000
Motor Number of Steps	600
Acceleration	1000

- **Steps**: The X27.168 stepper motors are 600 step motors but you can change the value here. No interest unless your engine is different than an x.27.168.
- **Max Speed**: This parameter corresponds to the maximum speed allowed by the motor. It is calculated on the basis of information from the AccelStepper library: http://www.airspayce.com/mikem/arduino/AccelStepper/
- **Acceleration**: This parameter corresponds to the maximum acceleration allowed by the motor. It is calculated on the basis of information from the AccelStepper library: http://www.airspayce.com/mikem/arduino/AccelStepper/

I recommend being careful if you change the values of the limitations of the motors. You can damage it or display incorrect values. The values 600, 1000 and 1000 come from my many tests.

The "references values of needle position" area allows each gauge of your cockpits to be matched to the values of the Falcon 4 BMS simulator gauges.


References values of nedle position

Gauge text	Stepper step	Test position
0	0	Test
20	65	Test
40	140	Test

Since each cockpit is different, this system allows all types of gauge holders to be used. In my case I used a gauge support that comes from the website: http://hispapanels.com/

But there is other support with other gauge values. It is therefore important to have a system that allows matching all gauges of all cockpits.

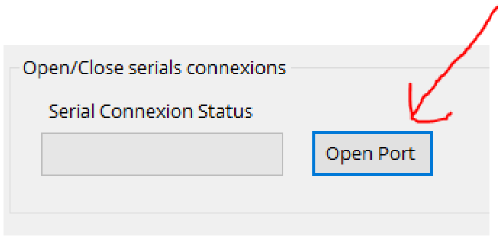
In addition, some F16 gauges with nonlinear values, this option allows to match exactly the values of your cockpit gauge to that of Falcon BMS.

The simulator reference values are shown in the first column.

You must match your gauge values with the simulator values by indicating a number of "steps" in the centre column.

Click on the "test" button to try to match the values between them.

Attention, to try to move the needles you must have correctly connected your motor on the card, closed Falcon BMS and have opened before the serial ports on the main page of the program F4ToSerial.

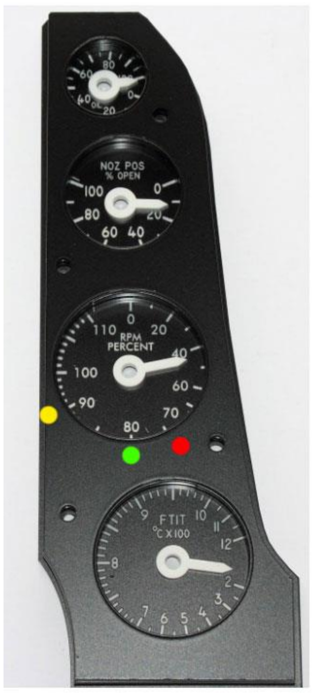


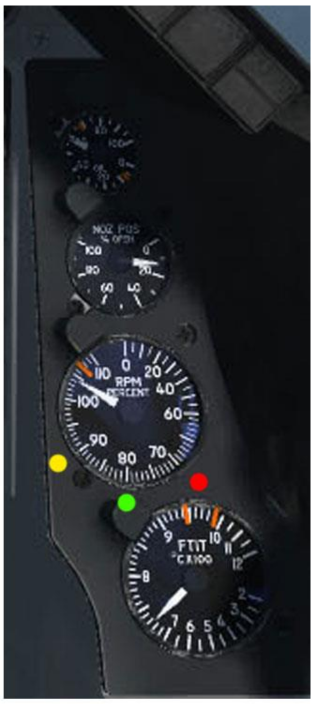
The following photo shows benchmarks between a RPM gauge on Falcon BMS and an RPM gauge in a home cockpit.

You must point your pointer at the values 70 (red dot), 80 (green dot) and 90 (yellow dot) by clicking on the test button and having the same result on your gauge.

Find more help on my video here:

https://youtu.be/dqMFBQkAozs?t=5m55s





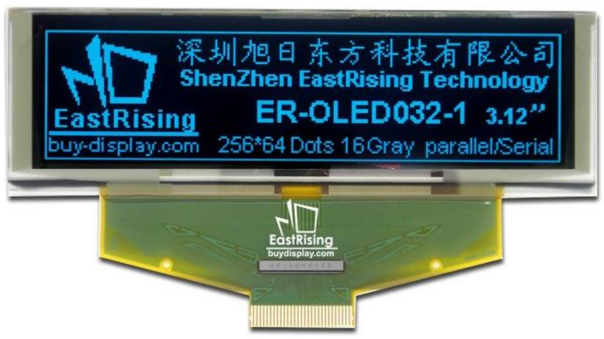
To check if your gauges are calibrated you can launch the Falcon BMS simulator and check the operation of your gauges. To reconfigure your gauges you must close Falcon BMS.

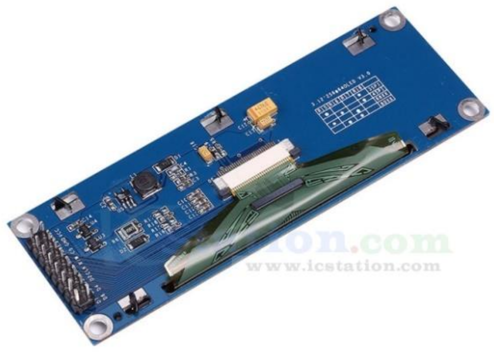
3.2.2 - Displays

Overview:

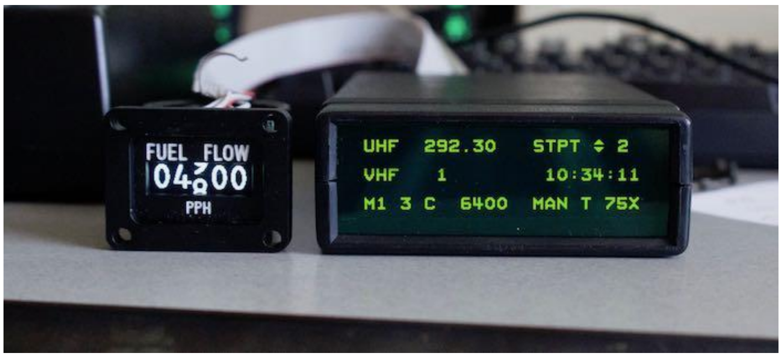
There aren't 36,000 solutions to simulate a DED screen in your cockpit. Among the simplest to implement, the use of OLED displays seems the most interesting.

Exemple d'écran OLED (taille 256x64 px) :





Example of FFI and DED using the excellent DEDuino program:
https://pit.uriba.org/tag/deduino/



F4ToSerial allows obtaining the same level of result, except the scrolling effect of the numbers on the FFI. I would surely develop an update on this.

To work with F4ToSerial, two screen formats are accepted: 256x24 and 128x64.

They are available everywhere on the internet, but F4ToSerial uses only the fastest SPI models. The I2C bus is not implemented in the C ++ program or in F4ToSerial.

I recommend you to pay attention when buying your screen, because in the case of I2C models it is sometimes necessary to unsolder a resistor to switch the BUS in SPI.

In the examples related to my installation I use two models of screens:

The model ER-OLEDM028-1Y whose datasheet is available here:

https://www.buydisplay.com/download/manual/ER-OLEDM028-1_Series_Datasheet.pdf

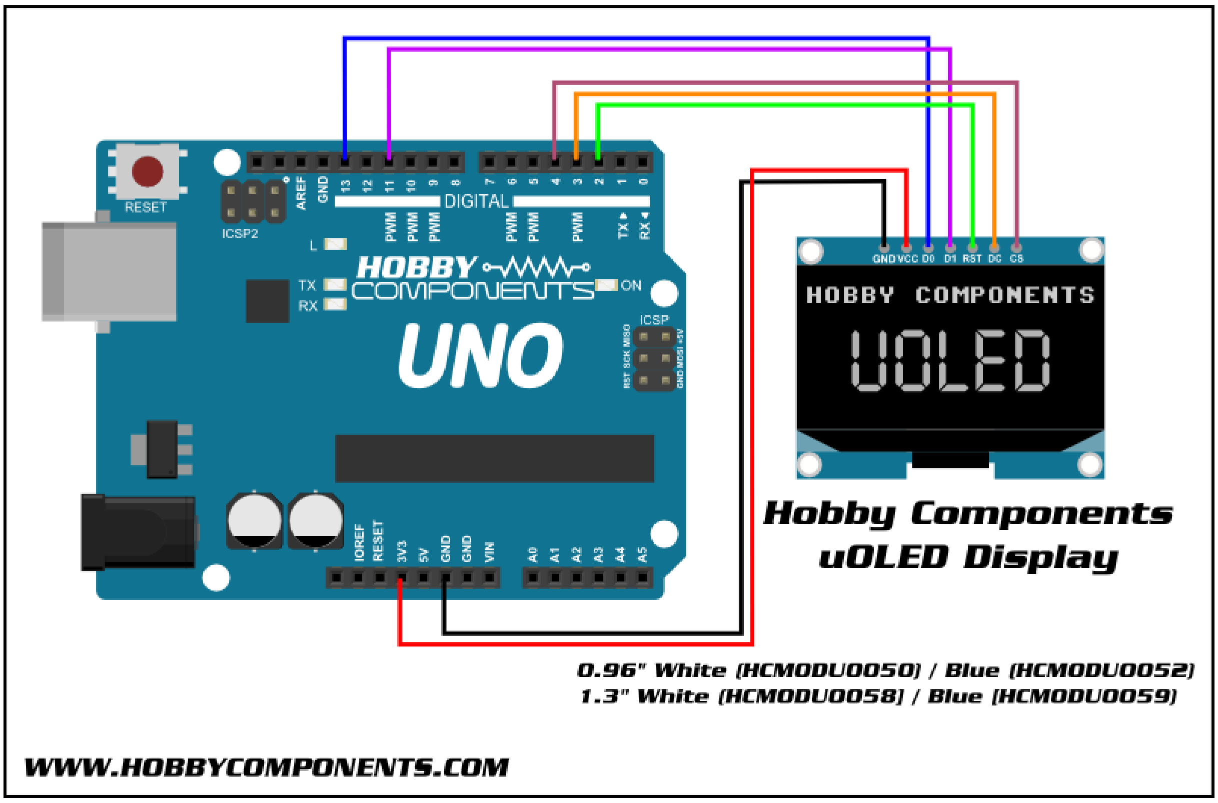
As well as a NONAME model with SSD1322 chipset bought on AliExpress here:

https://fr.aliexpress.com/item/Blue-Color-3-12-3-12inch-OLED-Display-Module-256x64-SPI-Communicate-SSD1322-For-Arduino-STM32/32819511393.html?spm=a2g0s.9042311.0.0.aSjjOi

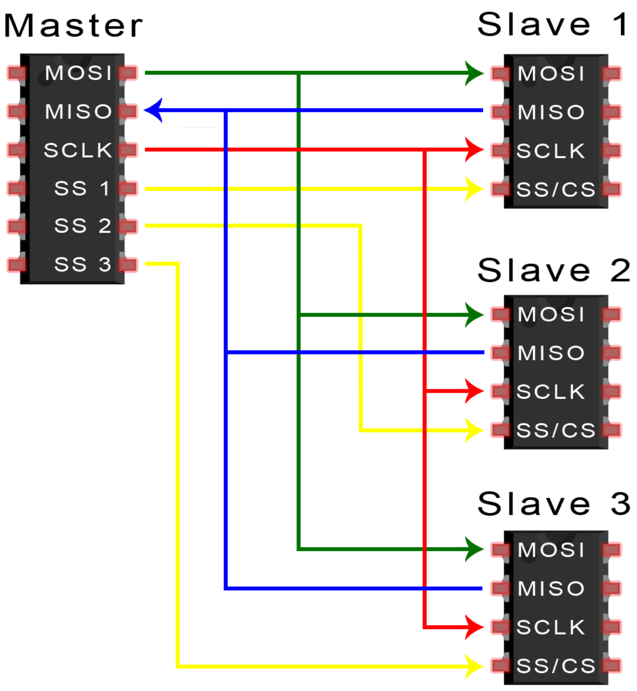
Communication on the SPI bus uses 5 wires:

1. Clock
2. Data
3. CS (Cable Select)
4. DC (Data Contrôle Command)
5. Reset

This example shows the connection of a single 128x64 screen to an Arduino Uno board.

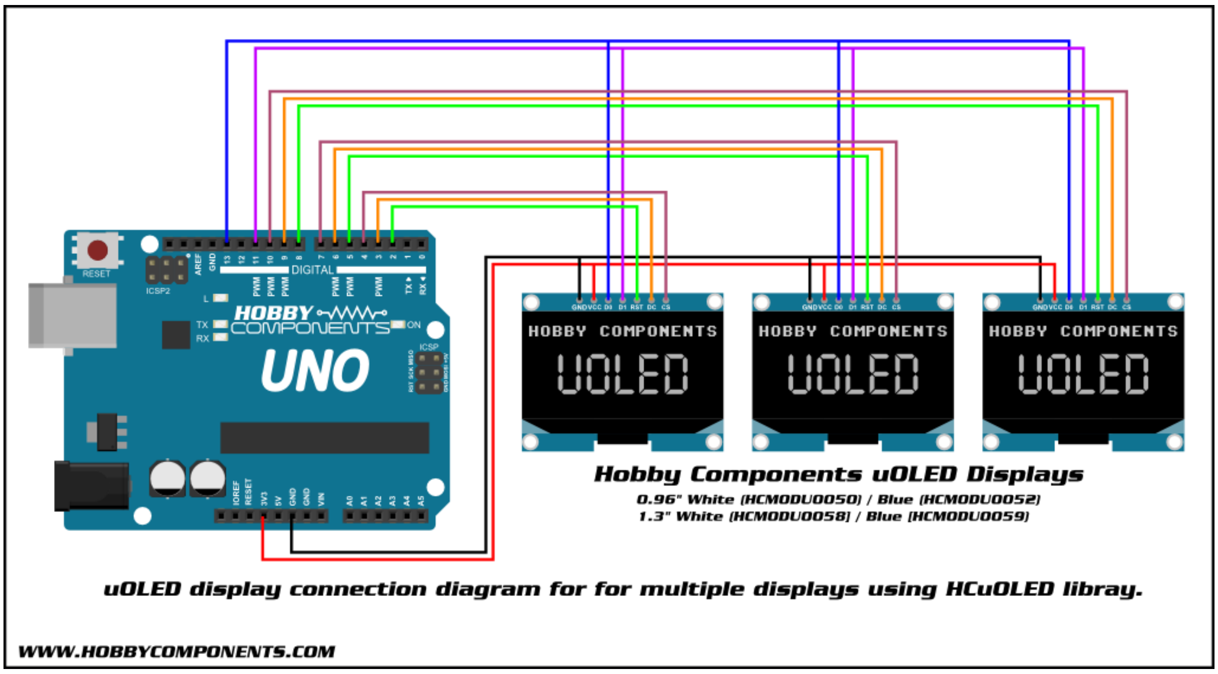


In the cases where several OLED screens are used on the same serial port (the same Arduino board), the clocks (CLOCK / DO) must be connected to each other as well as the data (DATA / M0SI / D1).



RESET, DC and CS are connected separately on the card.

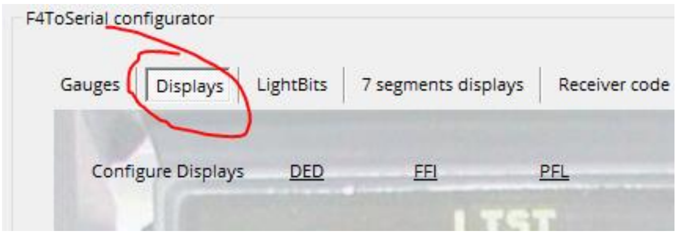
Connection example:



HOBBY COMPONENTS
UNO
HOBBY COMPONENTS UOLED
Hobby Components uOLED Displays
0.96" White (HCMODU0050) / Blue (HCMODU0052)
1.3" White (HCMODU0058) / Blue (HCMODU0059)
uOLED display connection diagram for for multiple displays using HCuOLED libray.
WWW.HOBBYCOMPONENTS.COM

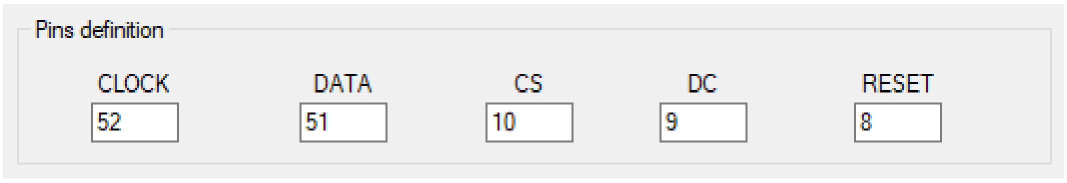
There are specifications for Arduino boards. Only certain pins on the Arduino board are provided to transmit the clock and data. Please refer to the Arduino documentation. https://www.arduino.cc/en/Reference/SPI

Use in F4ToSerial:



Accessible from the "Displays" menu of the configuration block, this option allows you to configure the 3 cockpit screens: DED, PFL and FFI. In Falcon BMS, the last (FFI) is not a screen per se. But its display remains possible.

The first step is the definition of the serial port and then the pins of the OLED screen.

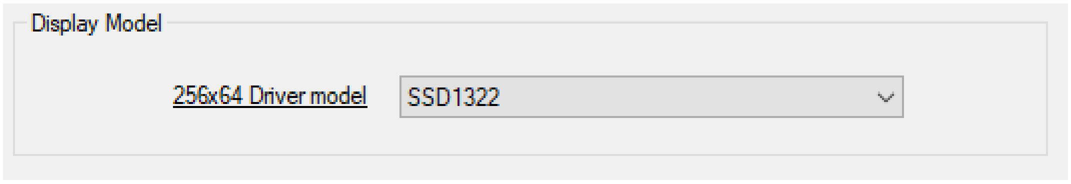

Pins definition

CLOCK	DATA	CS	DC	RESET
52	51	10	9	8

Indicate here the pins of your Arduino board corresponding to the SPI pins of your OLED screens.

By default, the pin numbers CLOCK (52) and DATA (51) correspond to those used on a Mega Arduino board.

Then indicate the model of the microcontroller of your screen:

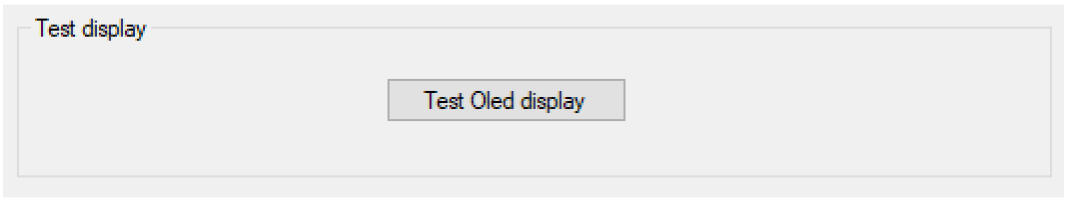

Display Model

256x64 Driver model: SSD1322

F4ToSerial is compatible with the following OLED chipsets:

- **Format : 256x64**
  - SSD1322 (default)
- **Format : 128x64**
  - SSD11306 (default)
  - SH1106
  - SSD1309
  - SSD1325
  - ST7565

Finally, after saving the configuration, and opening the Serial ports, it is possible to perform tests to make sure your OLEDS displays work properly. Click on the "TEST" button.



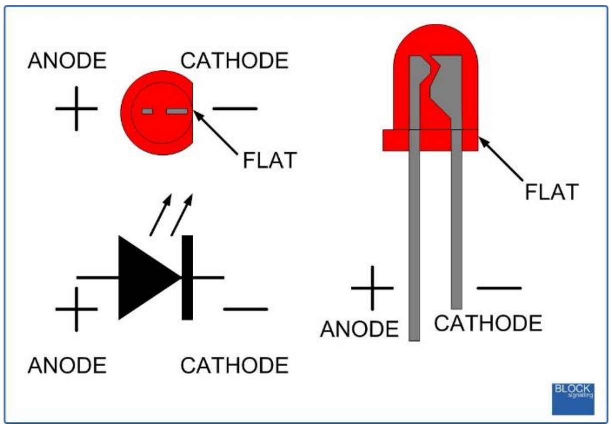
3.2.3 - Leds or lightBits

Overview:

The use of LEDs or LEDs in a cockpit is essential to simulate the many lights on or off in the Falcon BMS simulator.

This is also the same type of components that are used in any real device.

In the Falcon BMS simulator, the LEDs are called "LightBits" and their status (on or off) is changed to "binary" (O or 1) in shared memory.



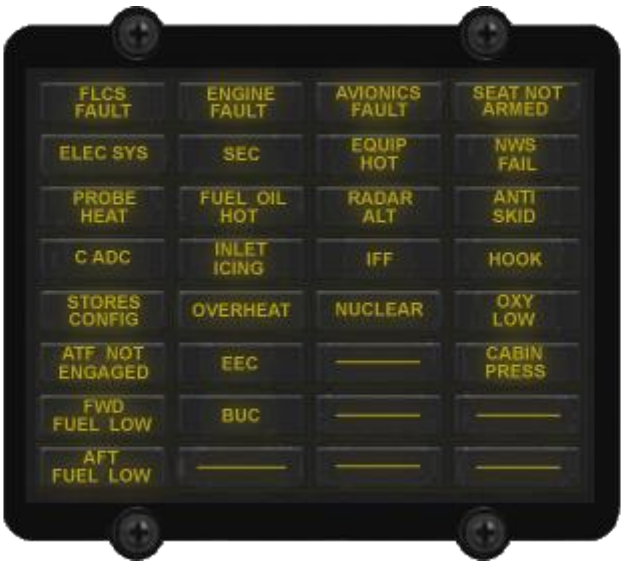
There are two approaches to using Lightbits. One "economic" and the other "we do not care".

In the "we do not care" approach, you need to count 1 pin of the Arduino board for each lightbits. It's easier to wire, but it quickly occupies all the pins on your Arduino board.

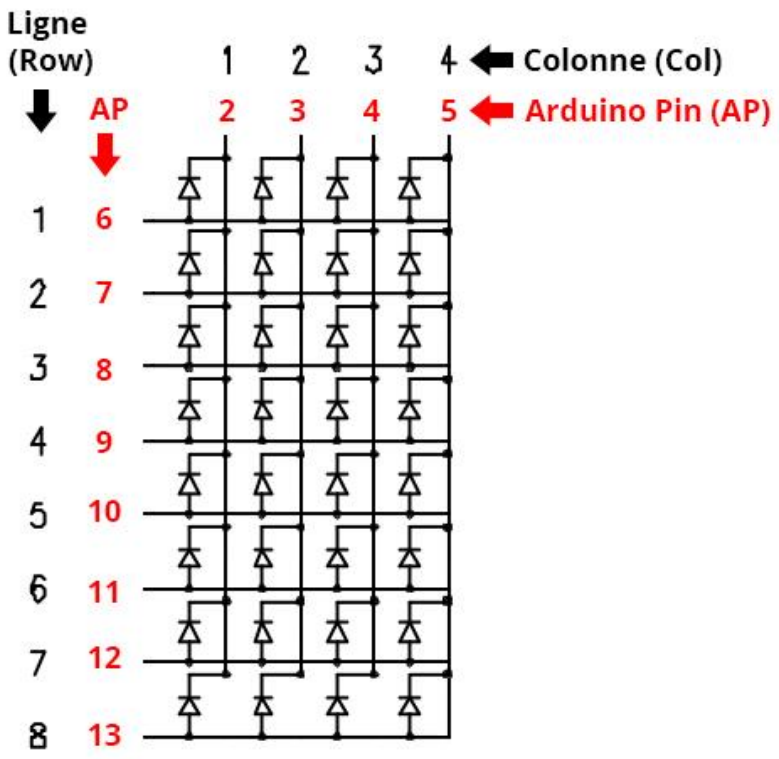
In the "economical" approach, 4 pins can power up to 16 lightbits. This is called "Matrix" or "Matrix" in English.

Personally I use both methods with the application of a matrix for some components like the PFL for example.

Hereinafter, the PFL of F16 Block 52.



The PFL can be represented with a matrix of this type:



Of course, we must not forget to add the resistances in entries of each column.

I will not dwell on the use and operation of a matrix here because there are very good tutorials on Google by searching for "LED Matrix" for example.

(French) https://openclassrooms.com/courses/perfectionnez-vous-dans-la-programmation-arduino/concevez-des-matrices-de-led

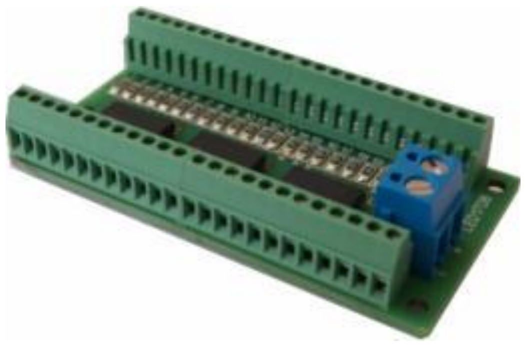
(English) https://www.arduino.cc/en/Tutorial/RowColumnScanning

The only thing to know is that the use of the matrix does not allow having several LightBits lit at the same time. It is therefore cunning to light several LEDs at the same time and for that use the effect of retinal persistence.

The good news is that you don't know how to do it, F4ToSerial does it for you!

In my case for the use of LEDs connected live on the card without going through a matrix, I use a very practical card.

http://www.flightsimparts.eu/shop_ledextension.htm



This card has 2 advantages

1. It already has all the resistances
2. It uses transistors to power the LEDS with an external power supply.

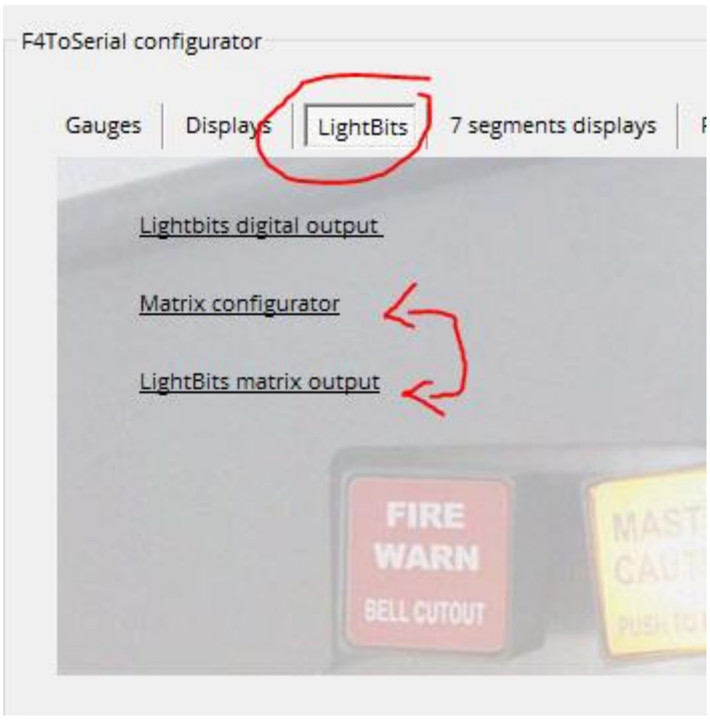
On this last point, remember that the more components you plug in (OLEDS, LEDs, Motors, etc.) on your Arduino board, the more power you draw! An Arduino board alone cannot light more than 5 or 6 LEDs!

I strongly recommend the use of an external power supply!

In the case of a matrix, only one LED is lit, which consumes little power!

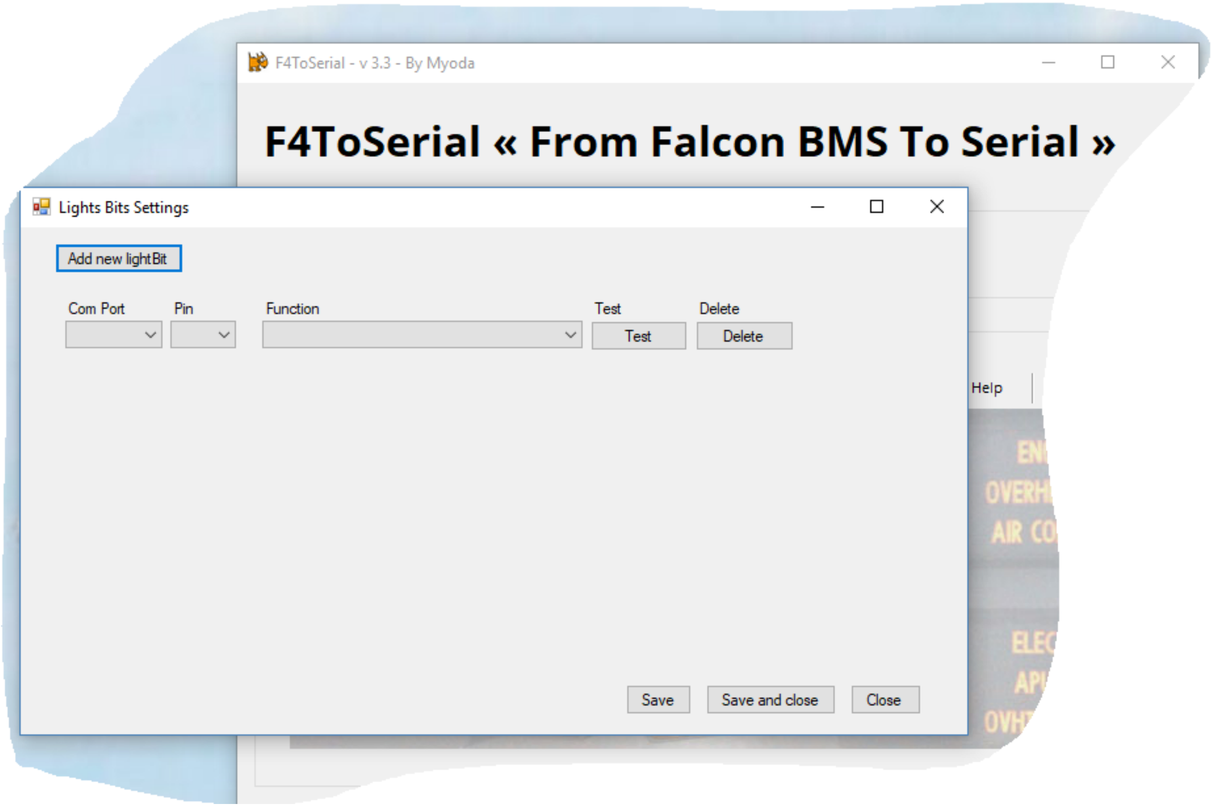
Use in F4ToSerial:

To set up LightBits in F4toSerial start by clicking on the "LightBits" tab



There are 3 links in this topic. The first link "LightBits Digital output" is used to connect the LEDS directly on the card. The other two are used to create and configure a matrix.

Creating a LightBit directly on the "LightBits Digital output" card

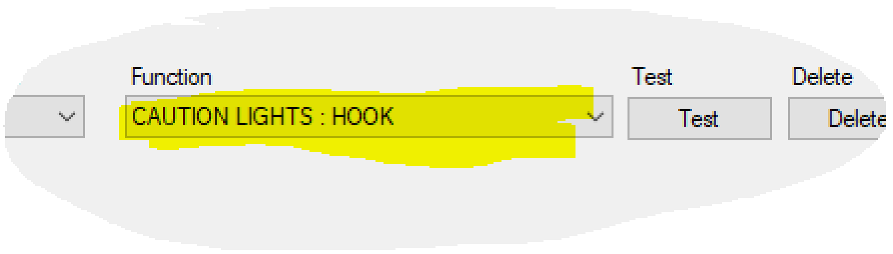


Clicking on "Add New LightBit" adds a new "entry" to our LightBits board.

As always, the serial port (Com Port) must first be defined.

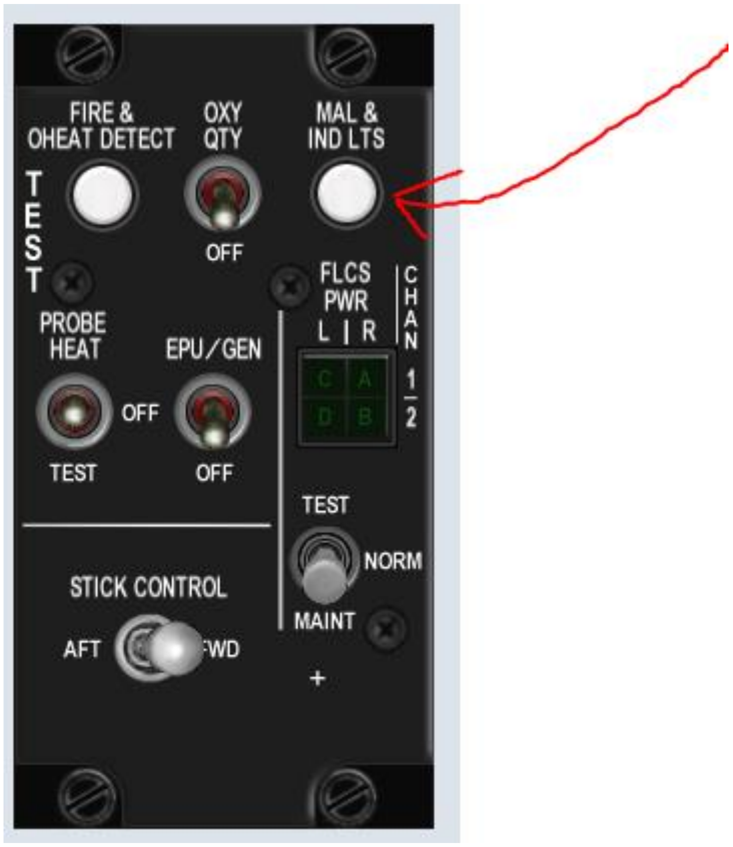
Then indicate the "pin" output from the Arduino board corresponding to your LightBit.

Finally, define its function:



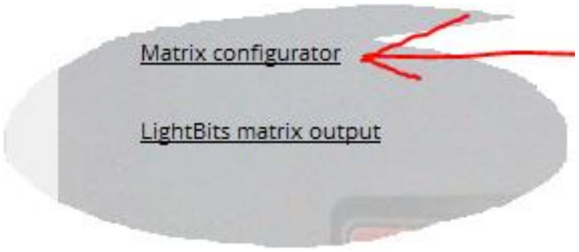
The "Function" column corresponds to the function of the lightBit in the Falcon BMS simulator. I invite you for testing to use a simple function like the "Hook" or the seat lock that immediately lights the PFL.

You can still do a test by pressing (MAL & IND LTS) to turn everything on.

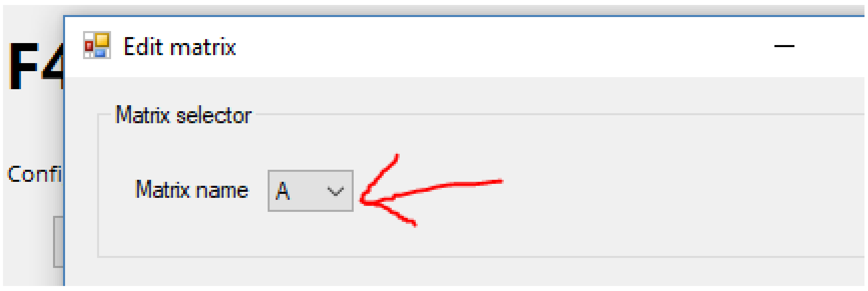


Creating a LightBits matrix Matrix configurator and LightBits matrix output

Click on "Matrix configurator"



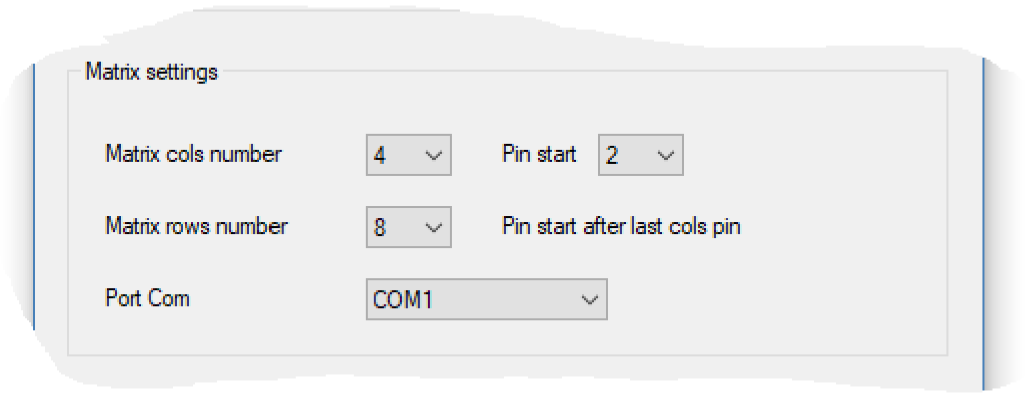
Then start by choosing the name of your Matrix


F4
Edit matrix
Matrix selector
Matrix name A
Confi

F4 Serial allows you to create up to 5 8x8 Matrices (8 rows and 8 columns).

They are named A, B, C, D or E.

Then indicate the 4 key parameters of the matrix:


Matrix settings
Matrix cols number 4
Pin start 2
Matrix rows number 8
Pin start after last cols pin
Port Com COM1

1. The number of columns (Matrix cols number)
2. The number of rows (Matrix rows number)
3. The serial port (Com Port)
4. The starting pin (Pin Start).

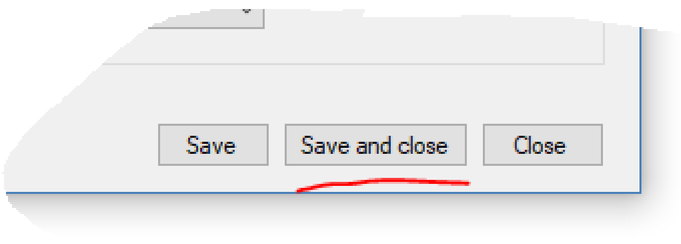
The last parameter (PinStart) is very simple to understand. It tells your Arduino board which is the first pin of your matrix knowing that they are all in order.

You still need to wire your matrix on the Arduino board starting with the columns with the F4ToSerial program.

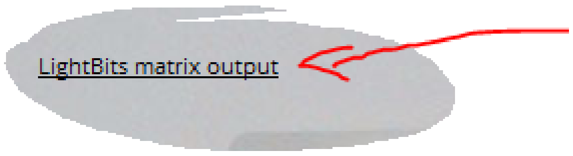
In the example above, the 4x8 PFL array (4 columns and 8 rows) is sent to Com port 1, and the pins are wired from 2 through 13.

2,3,4 and 5 (4 columns) + 6,7,8,9,10,11,12 and 13 (8 lines).

You can then save the configuration.



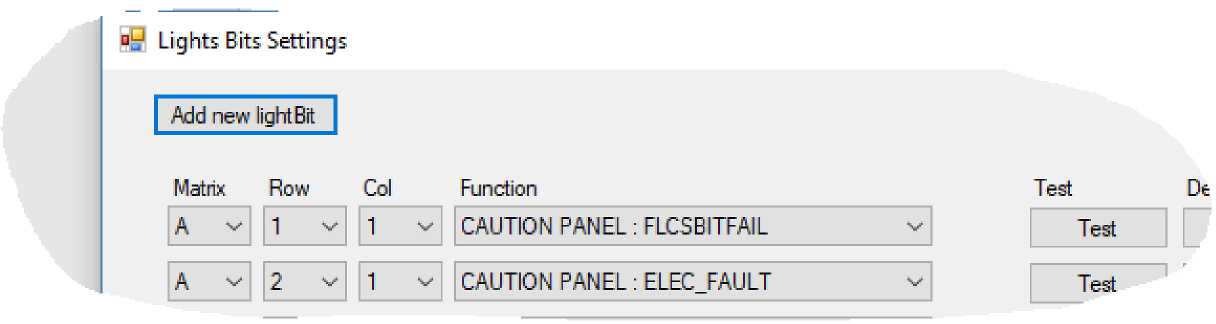
Now you have to define the LightBits and their functions in your matrix. To do this, click on "LightBits matrix output" in the main window.



The principle is the same as for a lightBit connected to the Arduino board directly except that now you have to indicate

- The matrix
- The column
- The line
- The function

Your lightBit is at the intersection of the line and the column.



Lights Bits Settings

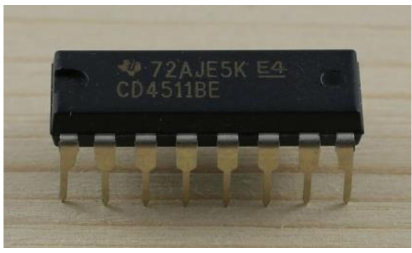
Add new lightBit

Matrix	Row	Col	Function	Test	De
A	1	1	CAUTION PANEL : FLCSBITFAIL	Test	
A	2	1	CAUTION PANEL : ELEC_FAULT	Test	

3.2.4 - 7 Segments displays

7 segment displays have been around for a very long time and are used in many devices including the F16 Falcon Block 52.

This component, relatively simple to nevertheless a defect: it is greedy in wiring! Indeed, each "LED" of the display is a full LED and needs to be powered to operate, just like a lightBit.

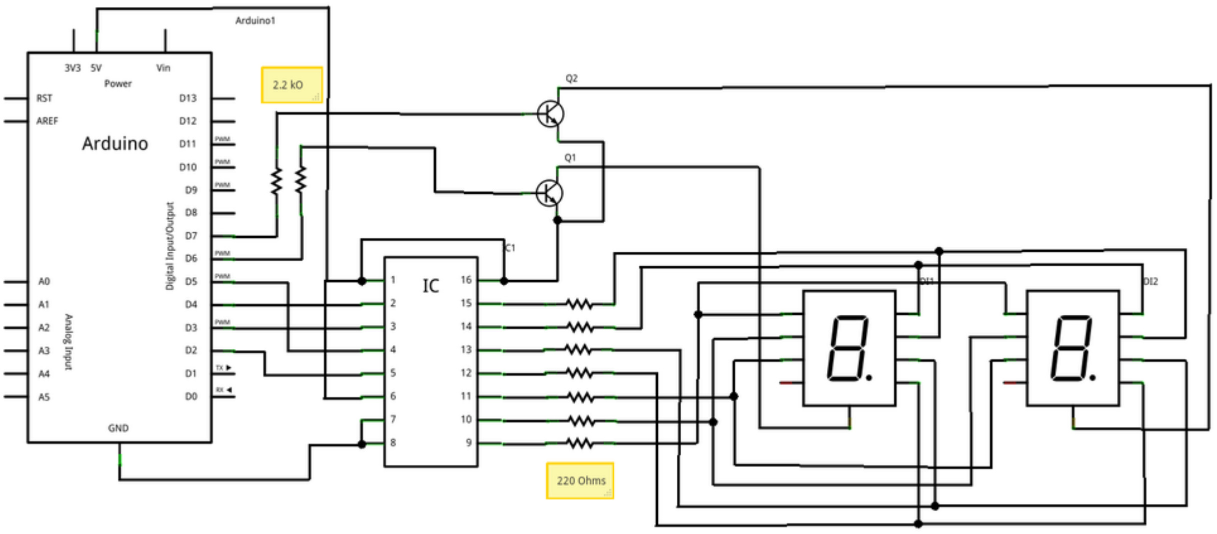


To avoid using too many pins on the Arduino board, it is strongly recommended to go through a BCD decoder.

Again, I will not go into the details of the operation of the BCD decoder, because very good documentation exist on Google with some research.

https://www.carnetdumaker.net/articles/utiliser-un-afficheur-7-segments-avec-une-carte-arduino-genuino/#question-piege-cathode-ou-anode-commune

The following is an example of connecting an Arduino board to a BCD decoder and two 7-segment displays..



Note that in the example above, we use two bipolar NPN transistors. They are used to switch display power to play on the effect of retinal persistence. Because just like for a lightBits, when one boards several displays on the same decoder BCD, only one can be used at the same time.

BCD decoders can only display values from 0 to 9. I invite you to read this tutorial to understand the principle of operation:

http://eskimon.fr/tuto-arduino-205-afficheurs-7-segments

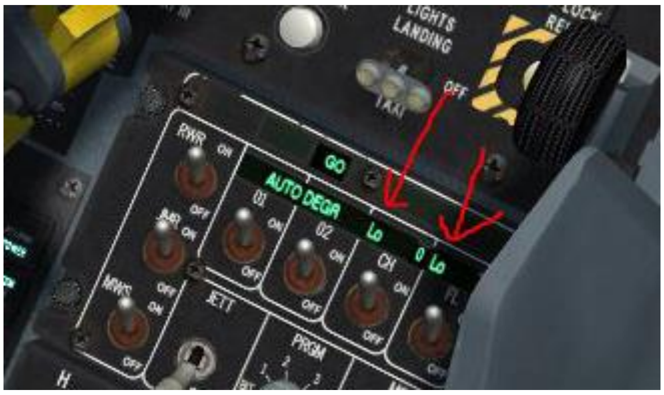
In F4ToSerial, to use a 7-segment display, you have to go through a BCD decoder.

The 7-segment displays usable with F4ToSerial are:

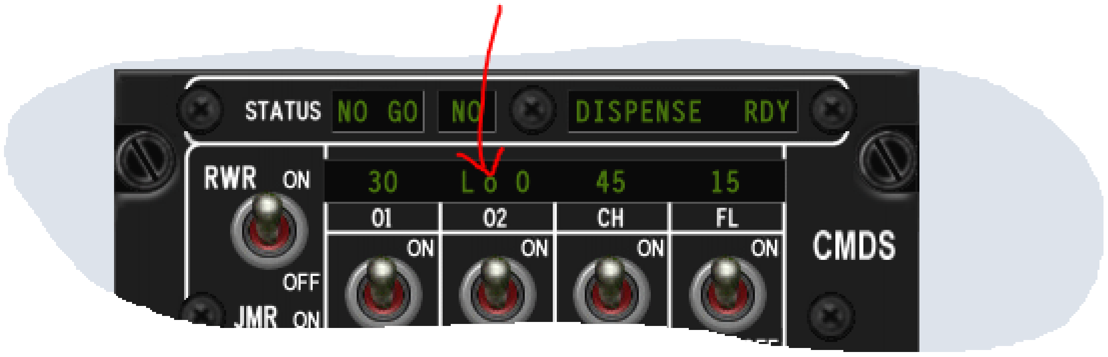
- UHF FREQ
- UHF CHAN
- FUEL (Gauge)
- CHAFF/FLARES (CMDS pannel)

Attention:

For CHAFF and FLARES, when the amount of capsule of CHAFF and FLARES reaches level 0 the indication "Lo" for "Low" (minimum) appears in front of the number of Chaff / Flares remaining.



Visible also here (saveed that 01 and 02 are not used on the F16)



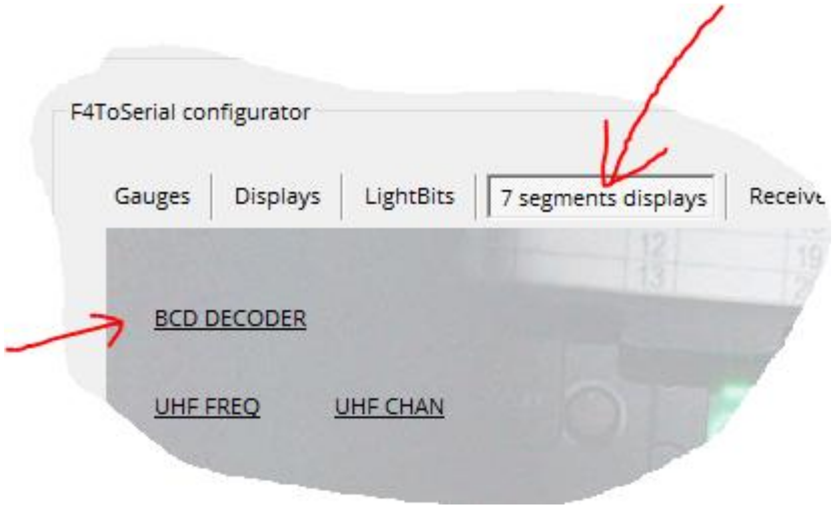
Unfortunately BCD decoders can only display values from 0 to 9.

It is not planned for the moment to update on this because it does not interfere with the understanding of the CMDS panel and it is not a constraint called "strong".

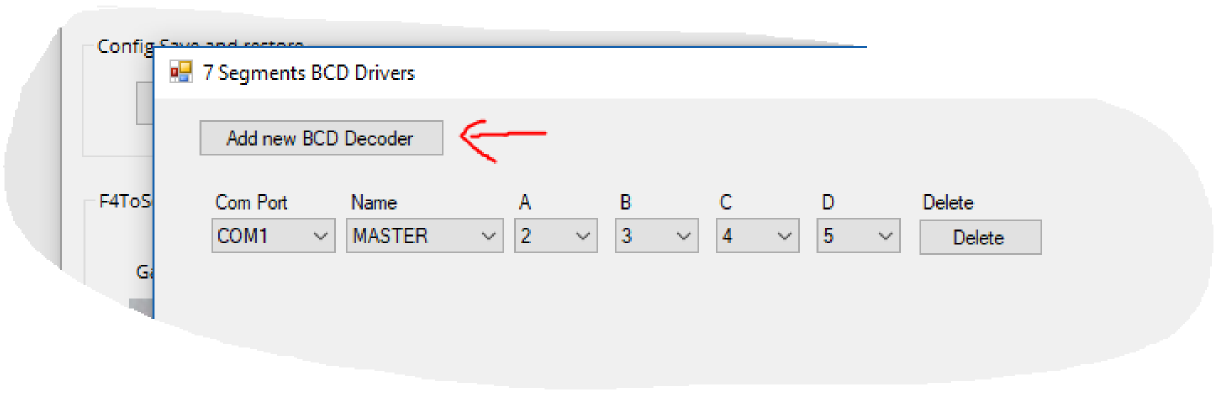
Use in F4ToSerial:

Start by clicking on the "7 segments displays" tab.

Then click on "BCD DECODER" to define and add a new decode BCD.



Click on the button "Add new BCD decode" to add a new line.



Com Port	Name	A	B	C	D	Delete
COM1	MASTER	2	3	4	5	Delete

For each line you have to define the following parameters:

• The serial port (Com Port)

• The name of the decoder (Name)

• The input pin A of the BCD decoder

• Input pin B of the BCD decoder

• The input pin C of the BCD decoder

• The input pin D of the BCD decoder

Since BCD decoders can be distributed among several Arduino boards, you can add multiple lines and you will need to choose a name for your decoder.

The name simply defines the function of the decoder. By choosing for example "Master", you define a decoder that will have several functions like UHF, CAN, FLARES etc.

This name will be used by the Arduino board to identify which decoder will receive information from the Falcon BMS simulator.

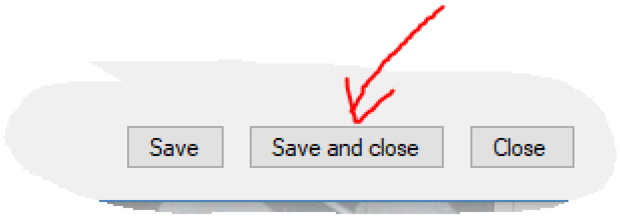
Example:

On an Arduino board connected to the COM2 port you use a decoder for the CHAFF, FLARES and FUEL displays. You can then define it as "Master" name.

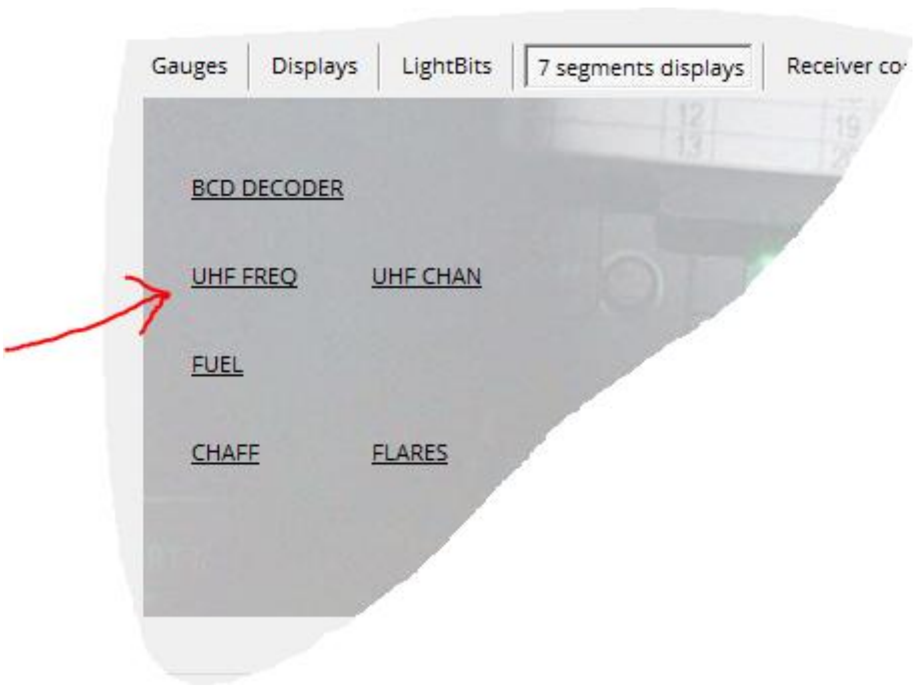
On another Arduino board on the COM3 port you use a decoder for the UHF CHAN and UHF FREQ displays, you can then set "UHF" or "UHFCHAN" or "UHFFREQ" as the name.

Pins A, B, C and D of the BCD decoder are to be defined also and I invite you to look for the documentation of your decoder for more information if you do not understand this part.

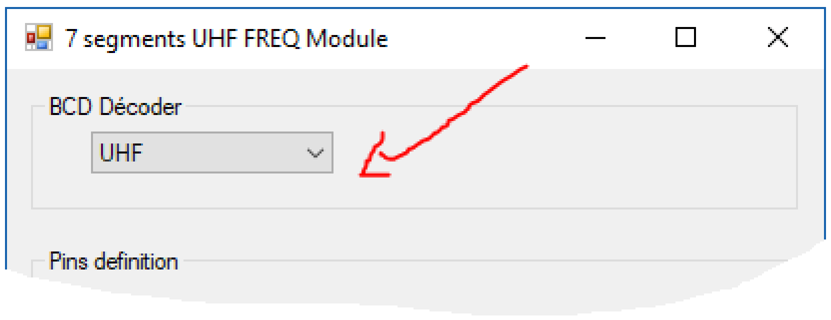
Once you have finished the settings and saved the configuration:



Click on the display in the main window.



Then select the BCD decoder for this display.


7 segments UHF FREQ Module

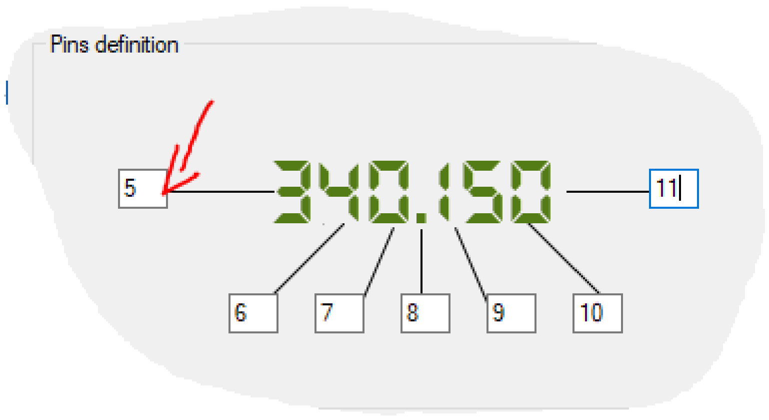
BCD Décoder

UHF

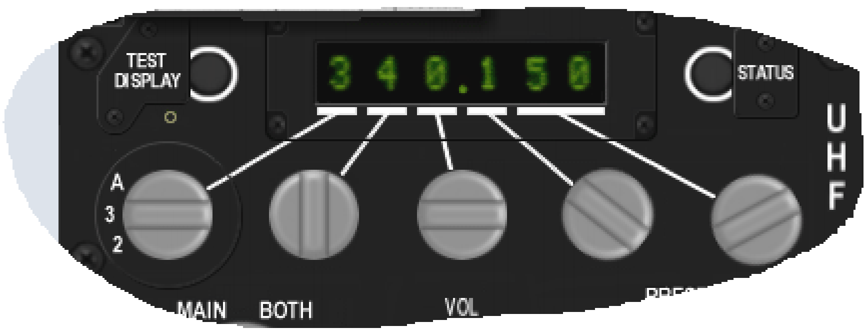
Pins definition

Once the decoder has been selected, enter the corresponding pin number for each display (represented here by a number in green).

In this example it is the 6 digits that make up the UHF FREQ (7 with the dot).

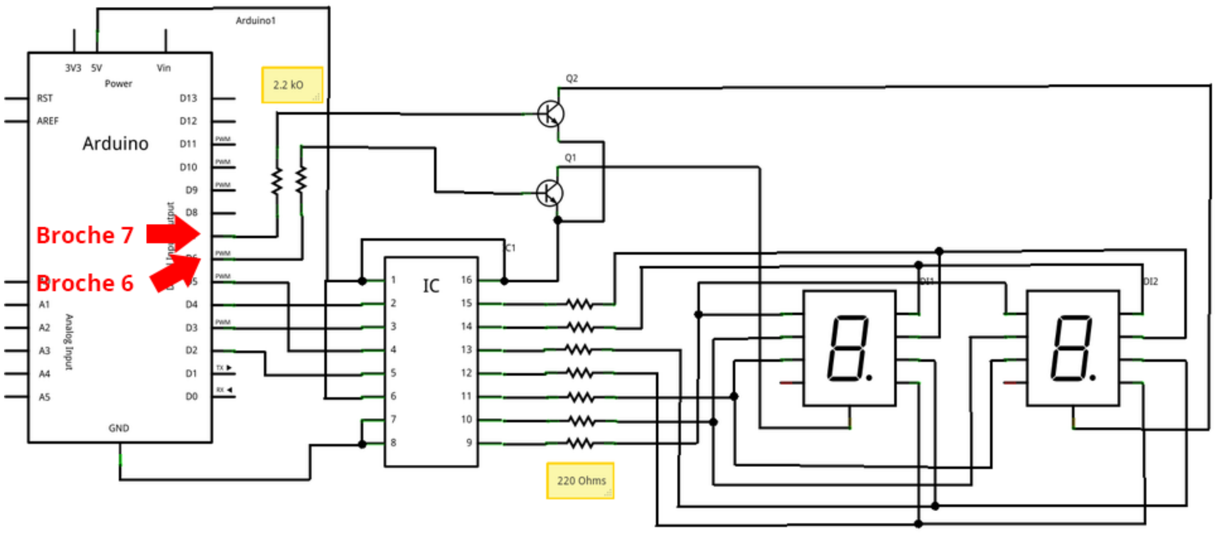


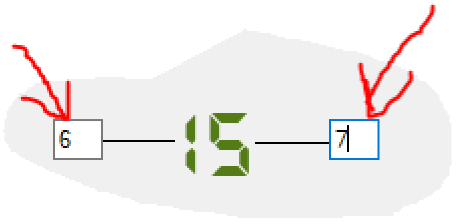
This window represents the panel.



To better understand how to define the pins that allow choosing the display I invite you to look at this diagram.

Example:



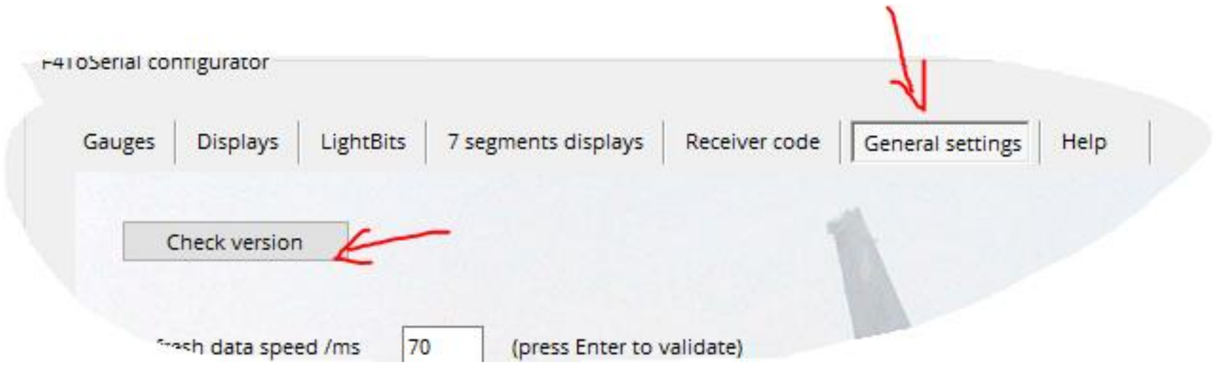


3.3 - General settings

3.3.1 - Download the latest version

Downloading the latest version is important to keep the software up-to-date.

For this, click on the "Check version" button.

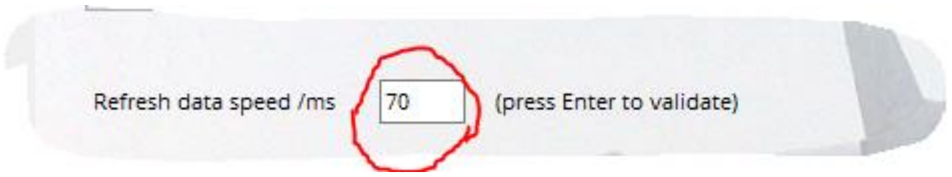


Refresh data speed /ms 70 (press Enter to validate)

3.3.2 - Refresh rate

The refresh rate is the time elapsed between each new sending of information to your Arduino board from the F4ToSerial program.

Attention, this speed is calculated and depends on the speed of transfer (Baud Rate).



Refresh data speed /ms 70 (press Enter to validate)

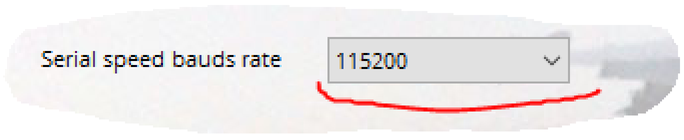
- The more you reduce this number, the faster you send information to your Arduino board. **You risk the saturation of the "buffer" of your Arduino board**.
- The more you increase this number, the more time passes between the information. **The risk is to note a delay on the elements of your cockpit compared to Falcon BMS (example gauge that do not follow).**

3.3.3 - Serial port speed

The serial port speed is the rate at which each data "bit" is transferred from the F4ToSerial program to the Arduino board.

Arduino natively supports the following frequencies.

300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200

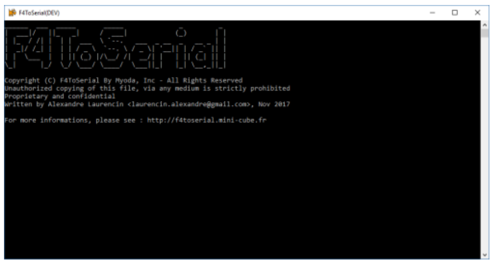


Serial speed bauds rate: 115200

My experience has shown me that the Arduino board can support well over 115,200 bauds. In practice, I invite you to use a higher speed to avoid any dropouts of active elements including OLEDS displays.

I recommend the use of 1,000,000 as the minimum transfer speed.

3.3.4 – Show the debug console



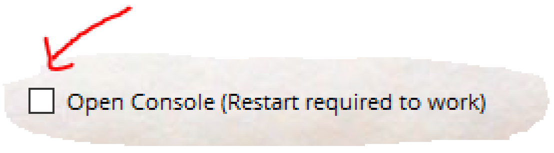
```
F4ToSerial(DEV)

Copyright (C) F4ToSerial By Myoda, Inc - All Rights Reserved
Unauthorized copying of this file, via any medium is strictly prohibited
Proprietary and confidential
Written by Alexandre Laurencin <laurencin.alexandre@gmail.com>, Nov 2017

For more informations, please see : http://f4toserial.mini-cube.fr
```

The console is a very useful mode for debugging your application. When using F4ToSerial for the first time, or when you are configuring a module.

It displays the data that passes on the card and alerts you if there is a problem in the program.



☐ Open Console (Restart required to work)

When you click on "Open Console" you have to restart F4ToSerial.

4 Bugs and possible issues

Nothing happens when I open the serial ports, why?

If nothing happens when you open the serial ports and click the test buttons items (lightbits, motors etc.)

Start by looking in the console if you have information that is sent.

If nothing happens, you are probably sending the data to a card at the wrong speed (baud). To do this, check that the program of your Arduino board communicates at the same speed as the program F4ToSerial.

When I add more than 6 LEDs or more than 2 gauges it stops working, why?

If you draw too much power on your Arduino board, it will not work properly. You must power your elements with an external source.

In general you need to connect all your equipment to an external source other than the Arduino board, itself powered by the USB port of your computer.