Davide Cavaliere
www.monocilindro.com
dadez87@gmail.com
27th October 2016

Fuelino service commands guide V1.1

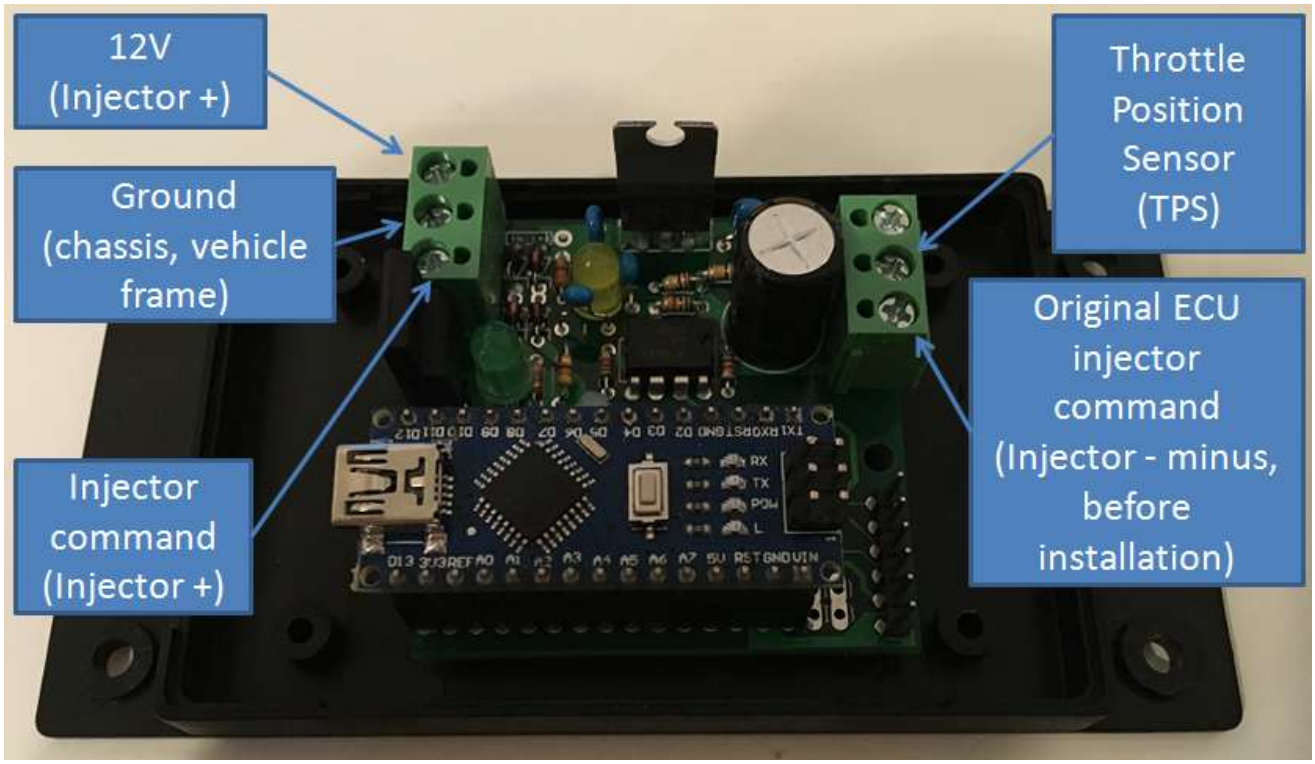
1 Introduction

This guide is aligned with Fuelino software version 1.0 beta1. It has the purpose to explain how to communicate with Fuelino, using Serial communication, to send service commands.
Service commands have the purpose of:

- reading/writing directly the microcontroller (Atmel Atmega328P) EEPROM memory
- reading/writing the calibration maps values, on RAM (temporary) or EEPROM (permanent) memory
- reading real time data, such as: engine rotation speed (time difference between 2 injections), injection time, throttle position sensor signal value, time after power ON, interrupts execution time, and so on.

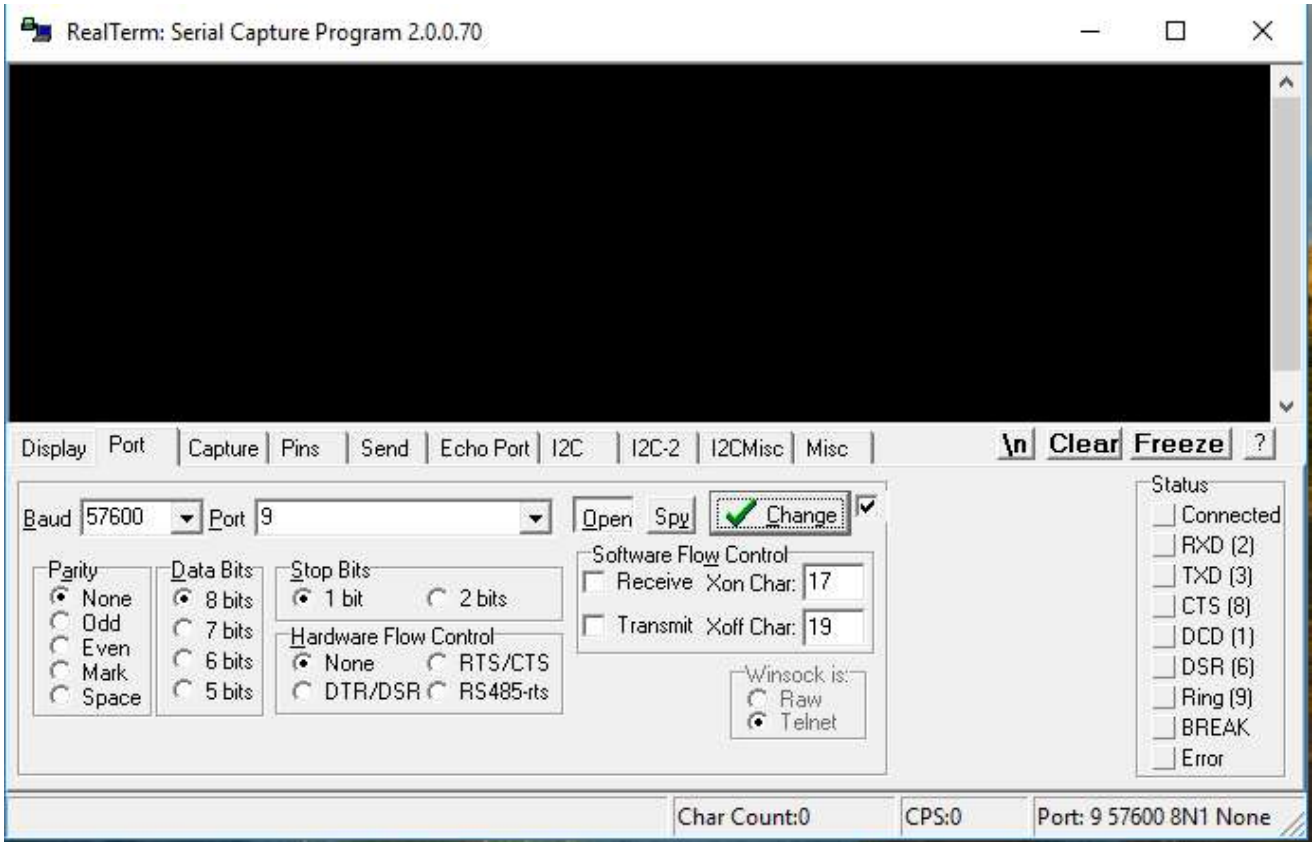
2 System Layout

The following image shows the pinout of Fuelino.



3 Service commands:general explanations and syntax

In order to send service commands to Fuelino, a serial communication must be previously established, at 57600 baud, as shown in the picture below. Communication can be extablished using RealTerm, or Arduino IDE “Serial Monitor”, or any other program which allows to send messages through serial communication.



Each service command sent from service tool (PC) to the Fuelino, via USB cable (Serial communication), is composed by 8 ASCII characters (1 ASCII character = 1 byte) plus a final “line feed” or “line return” ASCII character.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
‘e’ or ‘c’ or ‘d’								‘/r’ or ‘/n’

As a consequence, a single service command has a size of 9 bytes.
The first byte corresponds to the command type:

- “e” = EEPROM read/write direct access command
- “c” = Calibration maps values read/write command
- “d” = Data reading command (real time measurements)

Fuelino will reply with a message:

- In case of a read request, Fuelino will reply with the read value
- In case of a write request, Fuelino will reply with an echo response (same format as the command)

In case "*#define COMM_ENABLE_CHECKSUM*" is set to "1" (it is "0" as standard), all messages sent from Fuelino have, at the end, 2 binary bytes which correspond to the checksum calculated on all message bytes (except for the checksum itself). This checksum can be used to verify that no error happened during the bytes transmission.

3.1 EEPROM direct access command

The following command can be used to read/write directly the EEPROM memory addresses, with a limitation to the first 256 bytes.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
'e'	'0' = read '1' = write	EEPROM memory address [000 : 255]			EEPROM data Read case: 000 Write case: 000 : 255			'/r' or '/n'

3.2 Calibration maps access command

There are 3 maps in total, which can be accessed using a"map number" from 0 to 2. Each map (1-D array) has a specific number of values, which can be read (byte 1 = '0') or written.
In case any index of a map is written on RAM memory (byte 2 ='1'), Fuelino will suddenly use this new value; however, if a power OFF or a microcontroller reset happens, this value is lost, and at next power ON, the microcontroller will load the values stored in EEPROM memory.
For this reason, writing values on RAM memory is useful if you want to test some new calibration settings without having impact on EEPROM permanent memory, such as if you want to do some temporary tests and then come back to the initial settings at next power ON.
However, if you notice that your temporary calibration settings are working pretty well, and you want to save them permanently on the EEPROM memory, you need to do it with EEPROM writing command (byte 2 = '2'). This command will store your temporary map from RAM to EEPROM (permanent memory).

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
'c'	'0' = read '1' = write RAM '2' = write EEPROM	Map number [0 : 3]	Index [00 : (map_size - 1)]		Value [000 : 255]			'/r' or '/n'

In Fuelino software, there are 3 maps which define the injection time increase (in percentage), depending on different conditions:

- **incrementi_rpm**[]. Map number **0**. This array has *INJ_INCR_RPM_MAPS_SIZE=8* values. Each one defines the injection time increment (0=0%, 255=49.8%) depending on engine rotation speed (rpm).
- **incrementi_thr**[]. Map number **1**. This array has *INJ_INCR_THR_MAPS_SIZE=16* values. Each one defines the injection time increment (0=0%, 255=49.8%) depending on throttle position sensor signal (%).
- **incrementi_tim**[]. Map number **2**. This array has *INJ_INCR_TIM_MAPS_SIZE=8* values. Each one defines the injection time increment (0=0%, 255=49.8%) depending on original injection time (micro seconds, us). Currently, this map is not used.

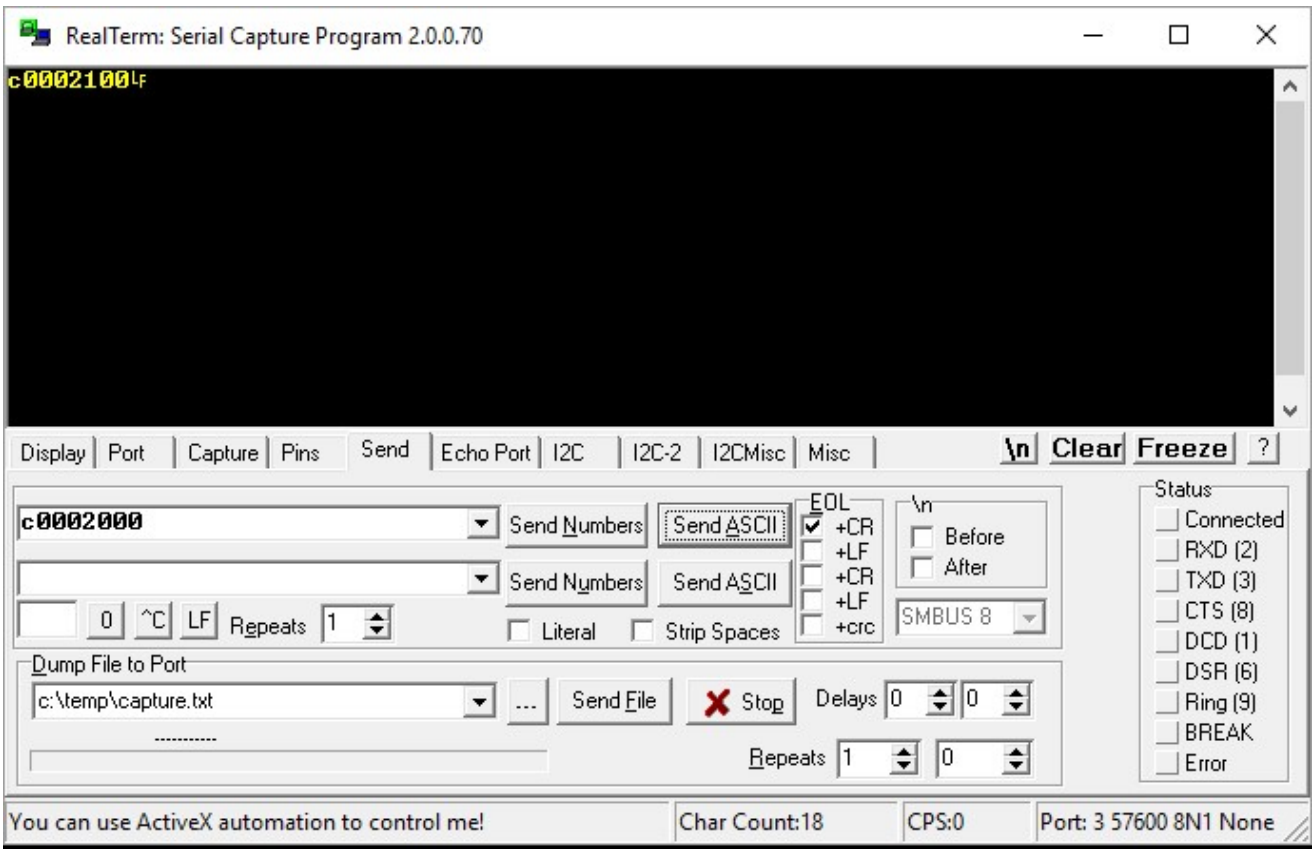
The total injection time increase percentage is calculated as the sum of the 3 maps described above. In order to calculate the injection time increment, in microseconds, the microcontroller multiplies the original injection time coming from original ECU, by the injection time increase percentage, which is the sum of the values read in each map.

3.2.1 Example 1: reading the calibration value of a specific map

The following example shows how to read the calibrated values, inside RAM memory, of a specific map, at a specific index.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
Calibration 'c'	Read/Write '0' = read	Map number '0'	Index "02"		Value 000			'/r' or '/n'

As shown below, the microcontroller replies (3rd line) that map 0 index 2 is equal to 100 (*incrementi_rpm[2] = 100*). A value of 100 corresponds to about 19.53% (256 = 50%).

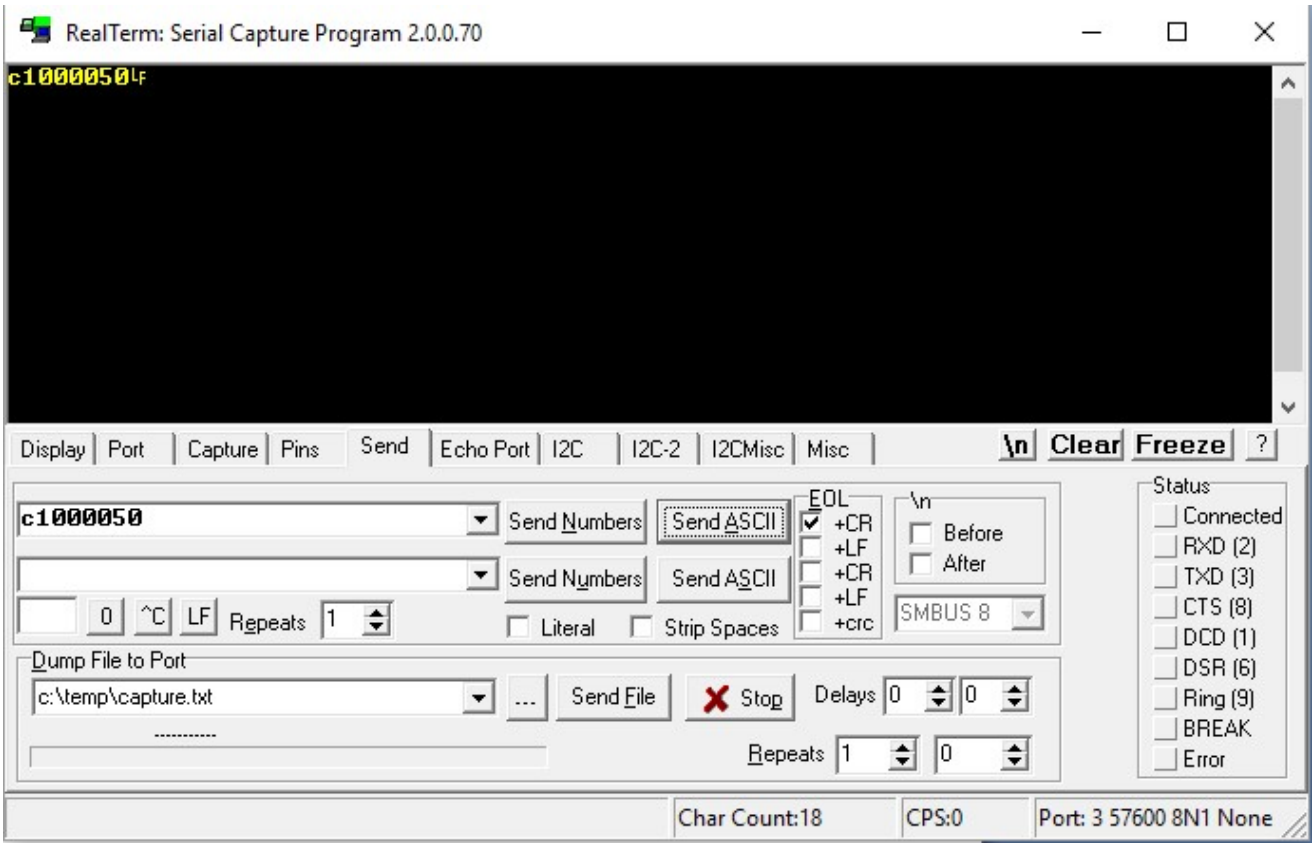


3.2.2 Example 2: writing the calibration value of a specific map

This example is dual to the previous one. Map 0, index 0, is set to 50 (*incrementi_rpm[0] = 50*).
50 correspons to a percentage of increase of about 9.77% (256 would be 50%).

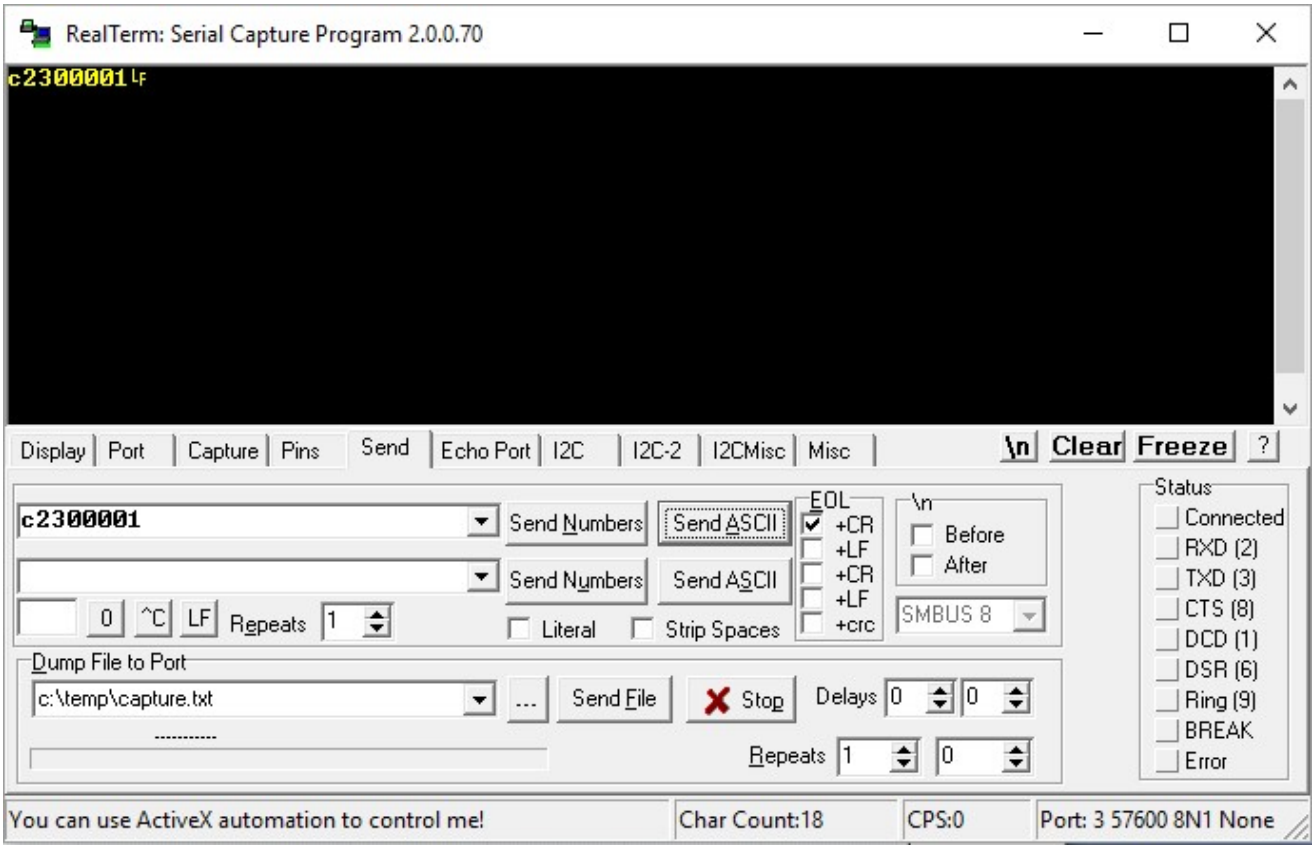
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
Calibration 'c'	Read/Write '1' = write RAM	Map number '0'	Index "00"		Value 050			'/r' or '/n'

Fuelino replies with an echo message, including 2 bytes of "checksum".
Please notice that this writing operation affects RAM memory, and not EEPROM memory, therefore a power OFF will delete the change. In order to store map 0 on the EEPROM memory, command "c2000001" must be sent. Alternatively, command "c2300001" will save all maps on the EEPROM.



3.2.3 Example 3: save calibration maps into EEPROM permanent memory

- In order to save permanently map number “n” into EEPROM permanent memory, use the command: “c2n00001” (example, for map 2, send “c2200001”).
- In order to save permanently all maps into EEPROM permanent memory, use the command: “c2300001”.

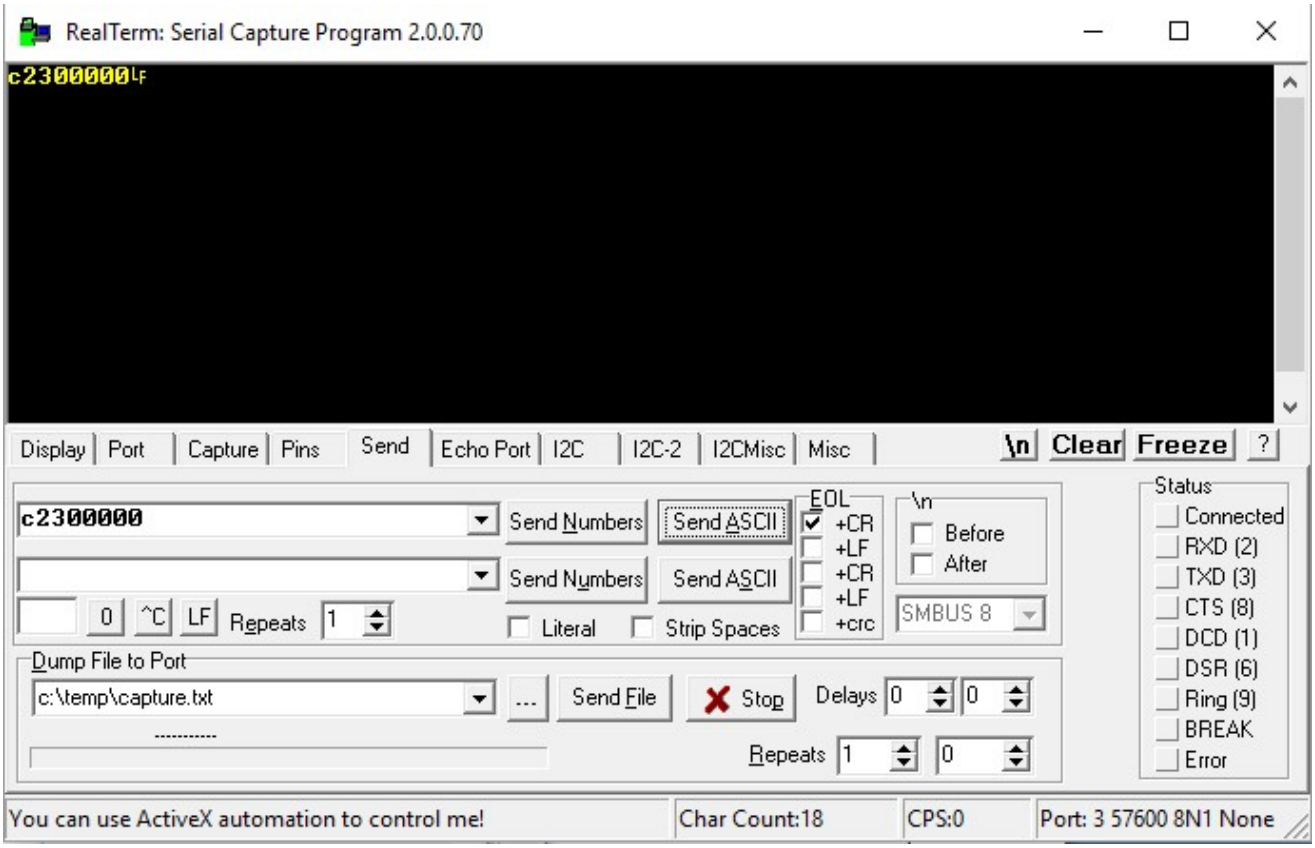


3.2.4 Example 4: restore standard values for a given map

For a given map number “n”, standard values can be restored using the command “c2n00000”. For example, in case you would like to restore standard values for map number 0, you should send “c2000000”.
If you would like to store standard values for all maps, you should send “c2300000” (‘3’ means “all maps”).

When maps values are restored, the following standard values are written for all indexes of each map:

- #define INJ_INCREMENT_RPM_STD (uint8_t)100 // Injection increment standard (%)
- #define INJ_INCREMENT_THR_STD (uint8_t)0 // Injection increment standard (%)
- #define INJ_INCREMENT_TIM_STD (uint8_t)0 // Injection increment standard (%)



3.3 Measurements data reading command

Real time measurements data, in binary format, can be read using the command “d1000000”. Practically, all command sizes between 4 bytes (“d100”) and 8 bytes (“d1000000”) are accepted.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
‘d’	‘1’	‘0’	‘0’	‘0’	‘0’	‘0’	‘0’	‘/r’ or ‘/n’

The microcontroller replies with 20 bytes data, as following. Each field has little endian format.

- Message type, 1 byte, ‘d’ (ASCII 0x64)
- Message type, 1 byte, ‘1’ (ASCII 0x31)
- Time, in milliseconds, since Power ON or reset, 4 bytes
- Injections counter, 2 bytes
- Time between consecutive injections (=2 engine rotations), 2 bytes, 1 bit = 4 us
- Injection time from original ECU, 2 bytes, 1 bit = 4us
- Throttle position sensor signal, 2 bytes, 0=0V and 1023 = 5V
- Lambda sensor signal, 2 bytes, 0=0V and 1023 = 5V [valid only in Fuelino V2]
- Extension time ticks, 2 bytes, 1 bit = 0.5 us
- Injector ON interrupt time, 2 bytes, 1 bit = 4us
- Injector OFF interrupt time, 2 bytes, 1 bit = 4us
- Checksum, 2 bytes

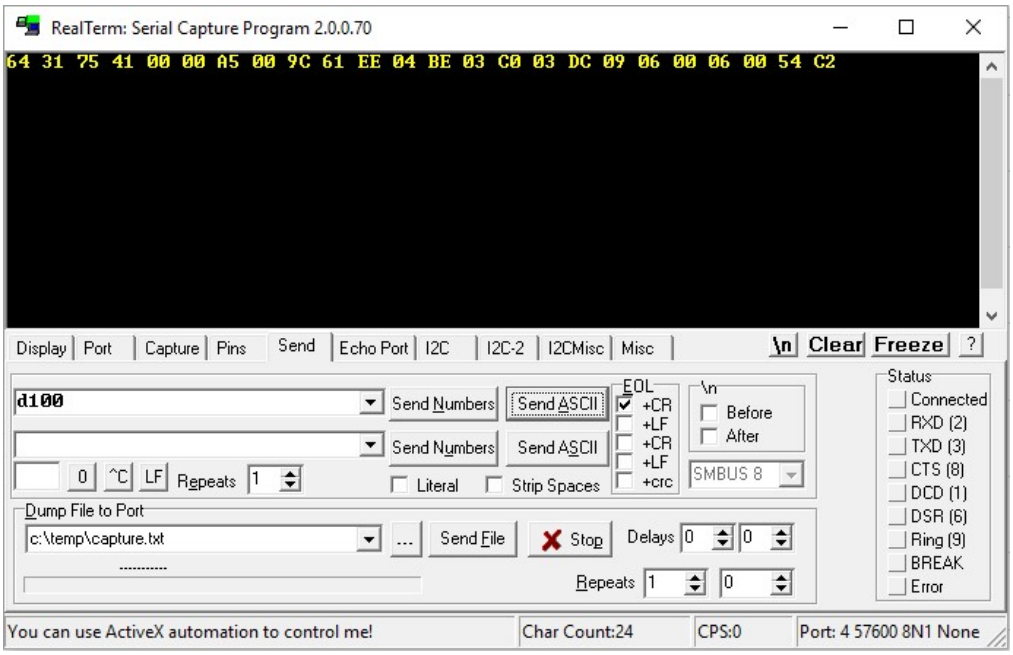
It is also possible to read single information data in ASCII format, using the following command. Also in this case, the size accepted is between 4 (“d0xy”) and 8 (“d0xy0000”).

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	End byte
‘d’	‘0’	x	y	‘0’	‘0’	‘0’	‘0’	‘/r’ or ‘/n’

The information type is determined by the information number (2 characters, xy):

Number	Type	Resolution
0	Distance between 2 injections = 2 engines rotations	4 us
1	Injection time	4us
2	Extension time	0.5us
3	Throttle sensor	0=0V, 1023=5V
4	Lambda sensor [not valid on Fuelino V1]	0=0V, 1023=5V
5	Combustion counter	1 combustion

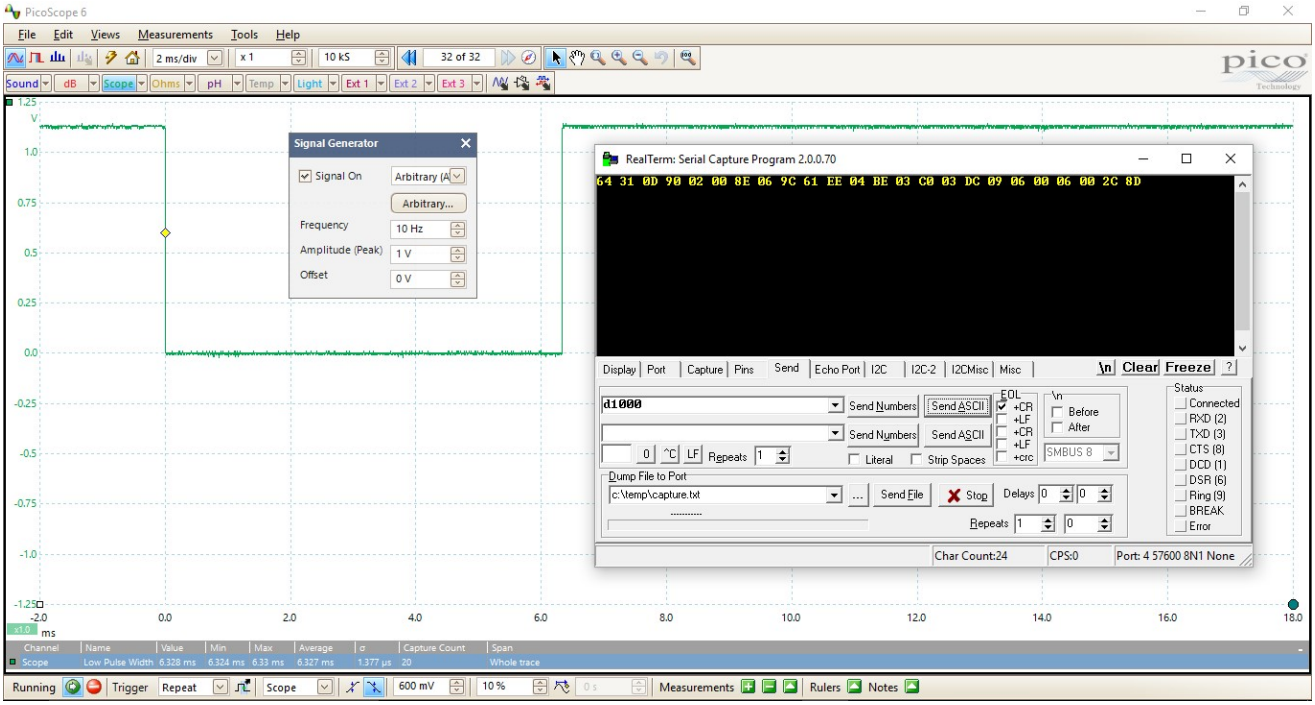
3.3.1 Example: read real time data in binary format



The following picture shows the message received from Fuelino, when it is fed with 10Hz, 5% duty cycle injection input pulses, which means that the distance between injections is 100 milliseconds, and the injection time is 5 milliseconds.

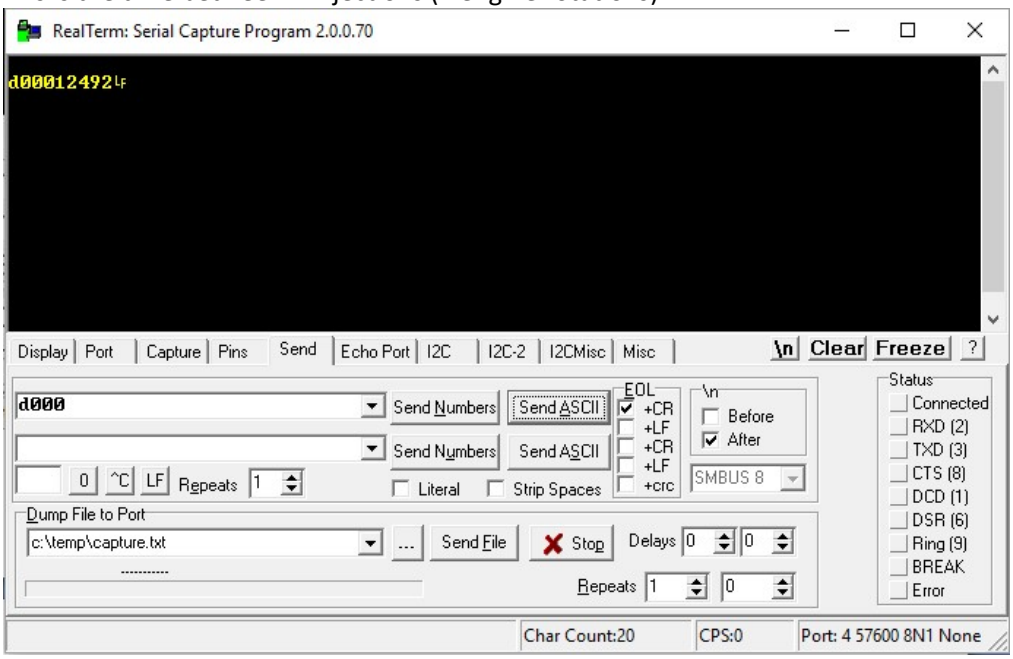
- Distance between injection is 0x619C = 24988 ticks = 99952us (1 tick = 4us)
- Original ECU injection time is 0x04EE = 1262 ticks = 5048us (1 tick = 4us)

Since the injection percentage increment is set to 128 (25%), the injector signal output by Fuelino is increased. The oscilloscope (Pico DrDaq) measures pulses of 6.330ms.

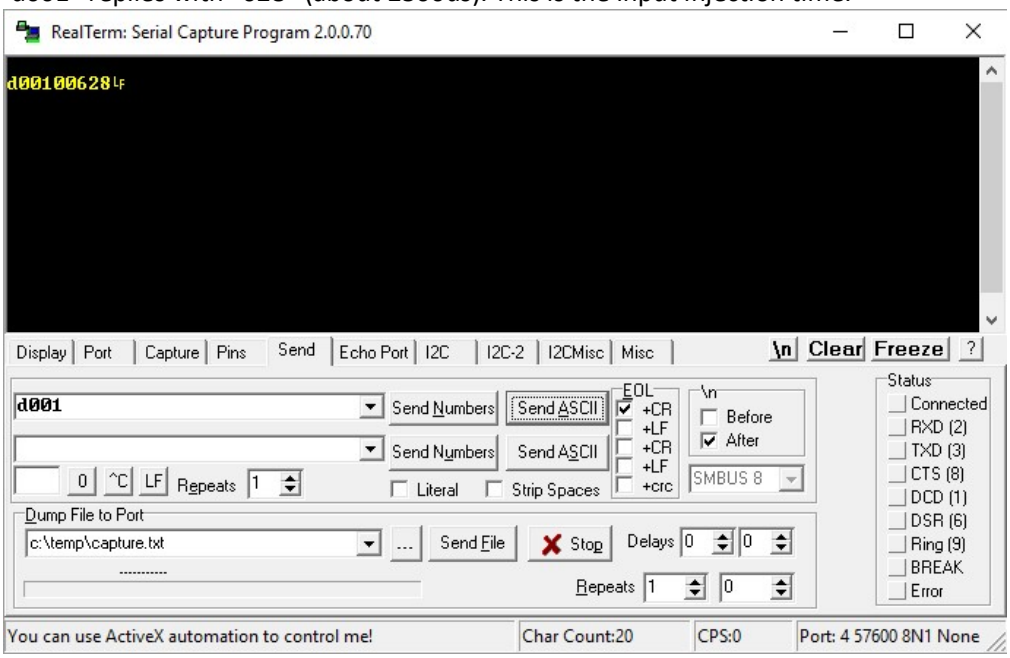


3.3.2 Example: read real time data in ASCII format

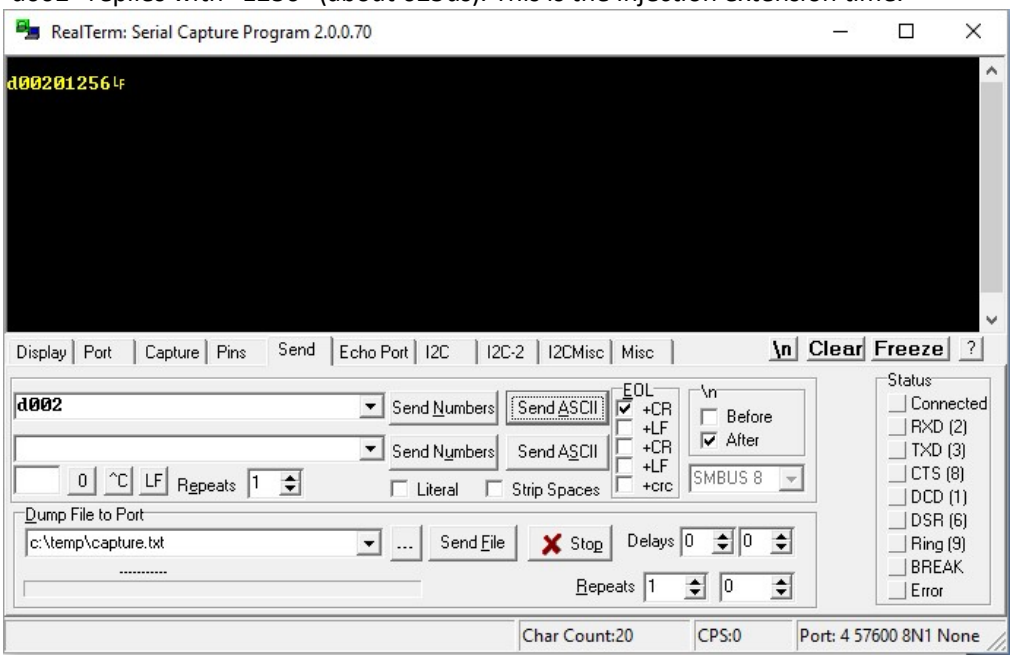
Fuelino is fed with 20Hz, 5% duty cycle input. “d000” replies with “12492” ticks (about 50000us). This is the time between 2 injections (2 engine rotations).



“d001” replies with “628” (about 2500us). This is the input injection time.



“d002” replies with “1256” (about 625us). This is the injection extension time.



4 Appendix

4.1 Appendix 1: "incrementi_rpm[]" map breakpoints

The relation between map 0 indexes (from 0 to 7) and rpm breakpoints is visible below.
As an example, index 0 corresponds to 12000 rpm. Index 7 corresponds to 1588.6 rpm.
Inside software, Fuelino breakpoints are not using rpm. Instead, the time ticks between one injection and the following (2 engine rotations) is used. The difference between one breakpoint and the following is a power of 2. These constraints have been used, in the software, because they reduce the computation time needed for interpolation process, which includes multiplication, sum, and division (since ATmega328P ALU does not support HW division, I used power of 2, so that the division can be simplified as bits shifting).

Index	Rpm	Hz	Time [us]	Ticks	Delta	Shifting
0	12000	100	10000	2500	256	8
1	10885.34	90.71118	11024	2756	256	8
2	9960.159	83.00133	12048	3012	512	9
3	8513.053	70.94211	14096	3524	1024	10
4	6596.306	54.96922	18192	4548	2048	11
5	4548.211	37.90176	26384	6596	4096	12
6	2805.836	23.38197	42768	10692	8192	13
7	1588.646	13.23872	75536	18884		

These breakpoints can be modified by changing the software array called *incrementi_rpm_brkpts[]*.

4.2 Appendix 2: “incrementi_thr[]” map breakpoints

The relation between map 1 indexes (from 0 to 15) and throttle % breakpoints is as following.

Throttle sensor signal (0 = 0V, 1023 = 5V) is converted into the map index by performing a division by 64 (6 bits shifting to the right). Doing so, 0V corresponds to index 0, while 5V corresponds to index 15.

4.3 Appendix 3: “incrementi_tim[]” map breakpoints

Not used at the moment.