µCAN & miCAN CAN-USB Device

USER'S GUIDE version 1.1

Software version 1.0.0.14

Published on 19.4.2017

Table of Contents

Illustration Index

Introduction

This user's guide describes how to get started with Voblox ***µCAN*** and ***miCAN*** Controller Area Network to Universal Serial Bus devices.
The two products cover a wide spectrum of users ranging from academic to professional use. Devices are widely used for their excellent price/performance ratio. Hardware has flexible range of settings which in pair with easy-to-use software offers a quick learning experience.

This user's manual and software application is available on web address www.voblox.com/downloads. Please check our web page periodically for updates.

Hardware

µCAN

µCAN is a small form factor CAN interface with no enclosure. Overall dimensions are 66mm * 17mm * 17mm. USB type A to type Mini B cable is required to connect the device to a host. CAN connection is established through a RJ45 connector. The pinout can be seen on the picture *Illustration 2*.

A RJ45 connector has been chosen due to a general availability of LAN cables. It offers stable latching connection with proven reliability. LAN cables have twisted pair wires and STP cable type is shielded for extra electromagnetic noise immunity. The Device has a header for a jumper wire which enables on-board 120 ohm CAN termination resistor. Two LEDs (TX and RX) act as a traffic monitor. They are tied to microcontroller's CAN_TX and CAN_RX data lines, therefore they act as a direct indication of the traffic on the CAN bus.

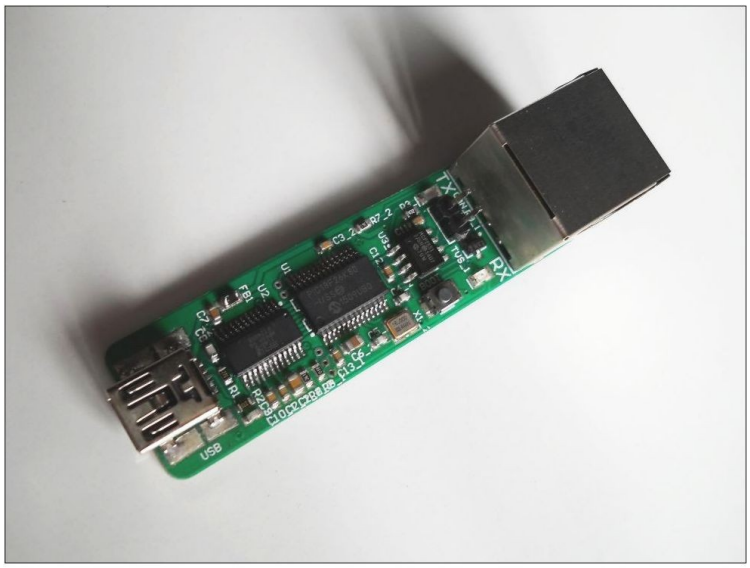

Illustration 1: µCAN

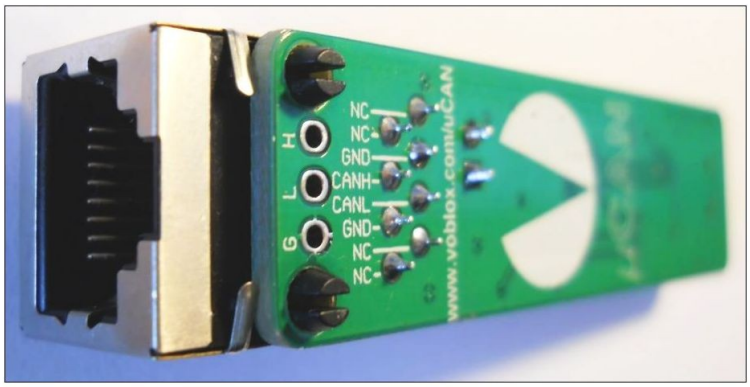

Illustration 2: µCAN pinout

NC (pins 1, 2, 7 and 8) – not connected
GND (pins 3 and 6) and pin G – system ground
CANL (pin 4) and pin L – CAN low differential signal line
CANH (pin 5) and pin H – CAN high differential signal line

miCAN

miCAN is a hand-held form factor CAN interface with overall dimensions of 26mm * 51mm * 90mm. The connection to a host is established via a USB type A to Type B cable. CAN connector and pinout is the same as on the µCAN.

This hardware has a full galvanic isolation therefore it is suitable for environments where equality of potentials is not guarantied (industrial, automotive, home automation...).

A jumper can be set internally to bridge the 120 ohm termination resistor if necessary.

LEDs (TX and RX) found at the sides of the CAN connector act as a traffic monitor. They are tied to CAN_TX and CAN_RX microcontroller signals to show a direct activity on the CAN bus.

Illustration 3: miCAN

miCAN is aimed at professional users like field operators and CAN product developers. It is the tool of choice for many companies who use it for testing of their own products, as a connectivity device in their end products or as a service tool for troubleshooting industrial and automotive CAN networks.

Software

Driver installation

In newer operating systems most of these steps are performed automatically (after following the procedure of automatic internet based driver installation). If for whatever reason this is not your case, manual installation procedure follows.

Go to http://www.ftdichip.com/Drivers/VCP.htm and download the latest drivers for your Windows OS version. Unpack the content of the zip file to a preferred location and plug the device into a free USB port on your computer. When asked for the drivers, point the installer to a location of the unzipped driver files and install.

Open Device Manager and find entry *USB Serial Converter* under *Universal Serial Bus controllers.* Double-click to open Properties window and click *Advanced* tab. **Make sure the *Load VCP* check box is checked!**

Another driver installation will emerge. Install it with the same procedure as the first. In Device Manager verify that you have a new entry called *USB Serial Port (COMxx)* under *Ports (COM & LPT)* list. Remember the assigned COM port number.

In case of problems with installing the drivers for your OS, refer to the FTDI's installation guide: http://www.ftdichip.com/Support/Documents/InstallGuides.htm.

Software installation

Download and install the application *CAN-USB basic tool* from www.voblox.com/downloads. The installation is straightforward, no special guide is provided for this step. The newly installed software should be accessible under entry called VBL in *Programs* list from the *Start* menu.

Application is divided in groups assembled in one main window, all the parameters are visible at all times. CAN filtering, send and received frames are displayed in hexadecimal format.

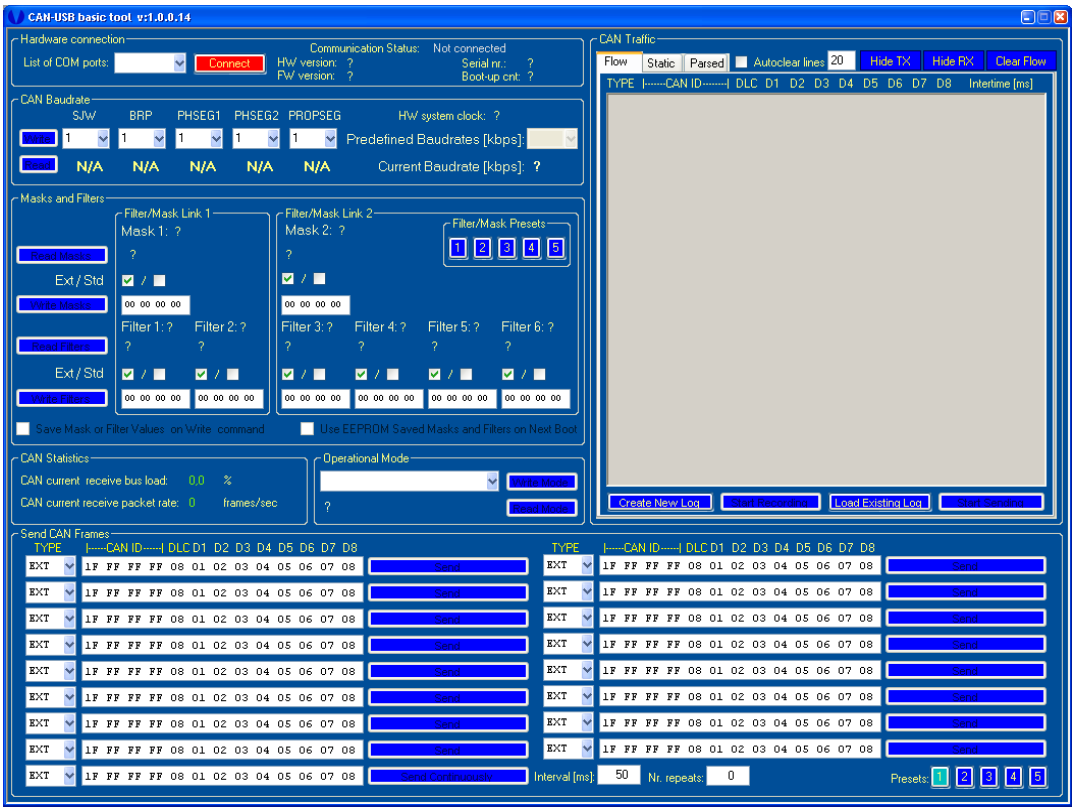

Illustration 4: Application main window

Quick application setup

CAN-USB basic tool application offers an easy to use form, which is divided in several groups according to the functionality:

- Hardware connection group box contains a drop-down list of available COM ports, handles connection to the device and displays connected hardware specific information. **Choose the correct COM port (as assigned by the system in Device Manager) and click the *Connect* button.** If the connection succeeds, the button renames itself to *Disconnect* and hardware information is displayed.

Illustration 5: Hardware connection

- CAN Baudrate group box lets you set the appropriate baudrate. **Choose the desired baudrate from the drop-down list named *Predefined Baudrates* to match the baudrate of the CAN bus you wish to monitor.** If a non listed baudrate is required, go to the chapter “Setting custom baudrate”, to get acquainted with how to set the baudrate with bit timing registers.

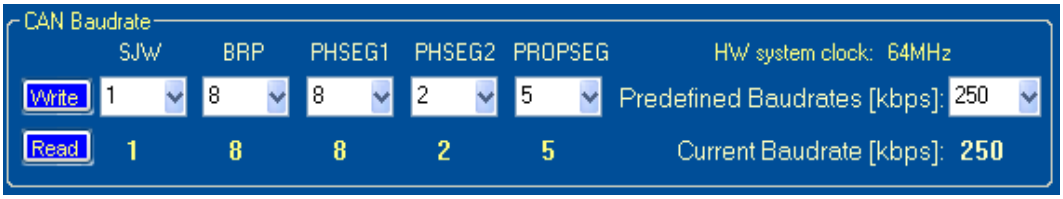

Illustration 6: CAN Baudrate

- Masks and Filters group box lets you set hardware receive filtering. **By default the filtering is disabled (all masks and filters set to *0x000000*).** In case filtering is needed, go to the chapter “Filtering”, to find out how to effectively use this feature.

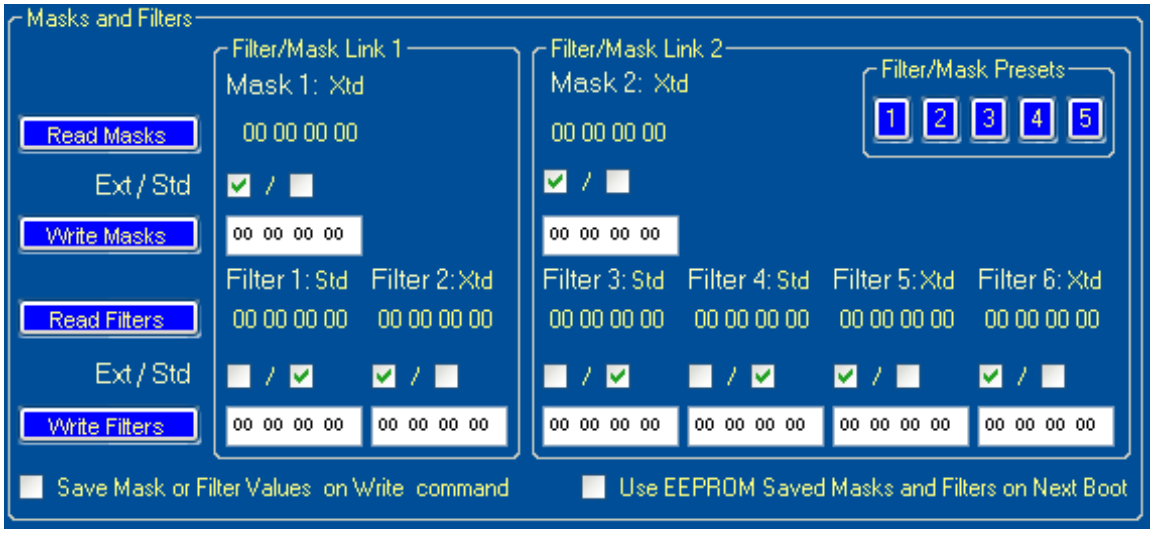

Illustration 7: Masks and Filters

- CAN Statistics shows CAN bus load as a percentage of time CAN bus is active and a counter of received packets in the last second.

  Group box turns red for a predefined period of time when a write to CAN transmit buffers is attempted, while all transmit buffers are still full. This is an indication for one of the three possible causes; feeding the packets to CAN bus is to fast, wrong baudrate is set or the wiring to the bus is not correct.

  Group box turns orange for a set period of time if application receive buffer gets overflowed. In theory this can occur on slow PCs that can't handle the incoming packets fast enough. It can also indicate the transmitting device doesn't receive a confirmation bit from any receiving nodes. In this case verify µCAN/miCAN is not configured in LISTEN ONLY mode.

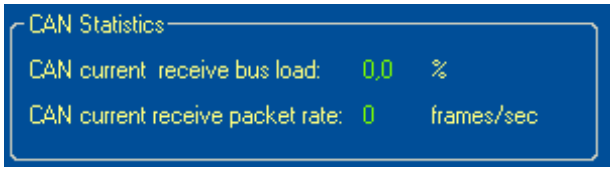

Illustration 8: CAN Statistics

- Operational Mode lets you set the desired mode. **If you are connecting to an existing CAN bus, make sure it is set to NORMAL.** In case you don't have a CAN network to connect to, you can test the device by setting the mode to LOOPBACK. Don't forget to set it back to normal, once you connect to a real network.

Illustration 9: Operational Mode

- CAN Traffic group box has three tabs; *Flow*, *Static* and *Parsed,* to show CAN data in different ways.

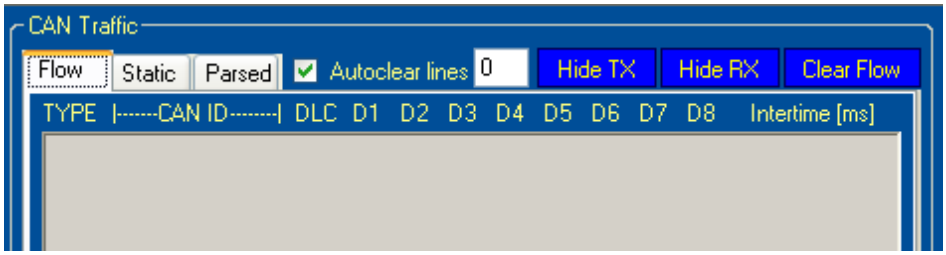

Illustration 10: CAN Traffic

Flow view allows the user to create and save current displayed traffic, record, load an existing log and send a saved CAN traffic log.

Illustration 11: CAN logging

- Send CAN Frames group box offers presetting of 17 different CAN packets in 5 presets (in total 85 different CAN message cells), that can be sent to CAN bus by clicking the buttons to the right of the written message structure. In each preset the last packet at the bottom of the group box is a continuously transmitted packet, which can be sent repeatedly in specified time interval in milliseconds. Number of repeats specifies how many times the packet is transmitted after pressing *Send Continuously* button. Value 0 is defined as indefinite sending.

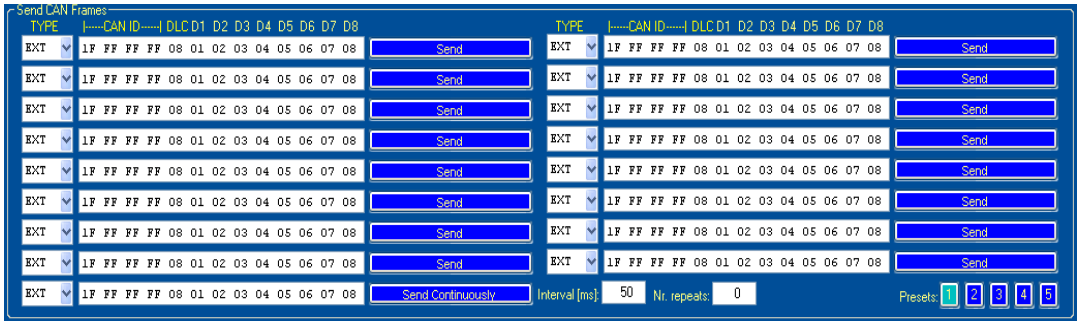

Illustration 12: Send CAN Frames

Advanced features

Setting custom baudrate

In rare occasions there is a need to set a custom CAN baudrate. For this purpose a direct manipulation of CAN bit timing registers is available.

To calculate bit timing values SJW, BRP, PHSEG1, PHSEG2 and PROPSEG for a desired baudrate, a general knowledge of CAN bit timing is necessary. We strongly suggest the user gets to know the specifics by reading online resources e.g.
http://ww1.microchip.com/downloads/en/AppNotes/00754.pdf

Some online CAN baudrate calculators are also available to facilitate the task.

To get the desired baudrate the following equation is used:

BAUDRATE = HW_system_clock /(2*BRP*(1+PHSEG1 + PHSEG2 + PROPSEG))

where

BAUDRATE presents calculated bits per second,

HW_system_clock is a connected hardware clock displayed in CAN baudrate group box

BRP is a prescaler with valid settings between 1 and 64

PHSEG1, ***PHSEG2*** and ***PROPSEG*** are phase and propagation values with valid settings between 1-8

SJW presents a value CAN hardware uses to synchronize with other nodes on the CAN bus.

Example:

If we wish to set a CAN baudrate of 800kbps we first must divide the HW_system_clock with the desired baudrate.

*(2*BRP*(1+PHSEG1 + PHSEG2 + PROPSEG)) = 64000000Hz/800000bps = 80*

Now divide by *2*BRP* so the remaining value is less or equal to 25. If we choose B*RP* = 2;

*(1+PHSEG1 + PHSEG2 + PROPSEG) = 80 /(2*2) = 20*

Now we can divide the value of 19 (20-1) between the three values in respect to their limits. E.g. *PROPSEG* = 5, *PHSEG1* = 6 and *PHSEG2* = 8.

We can also calculate the time point in the bit where sampling is executed. If

1+PHSEG1 + PHSEG2 + PROPSEG = 100%

1+PHSEG1 + PROPSEG = SAMPLE POINT%

*SAMPLE POINT = ((1+PHSEG1 + PROPSEG) * 100)/(1+PHSEG1 + PHSEG2 + PROPSEG)*

in our case

*SAMPLE POINT = ((1 + 6 + 5) * 100)/(1 + 6 + 8 + 5) = 60%*

As you can see multiple combinations of settings are possible to achieve the same baudrate. Different register settings can produce different sampling points.

Filtering

To enable filtering, both filter/mask links must be enabled by setting appropriate mask and filter values.

By setting mask bits, CAN hardware peripheral will compare the corresponding bits in CAN ID with the filters. Mask 1 is tied to Filter 1 and 2, mask 2 to filters 3 to 6.

For example if we'd like to receive only messages with extended CAN ID *0x01A3F218,* we must set both masks to *1F FF FF FF* and all six filters to *01 A3 F2 18*.

If we'd like to receive all CAN ID messages that have the same first three bytes, but the value of the fourth byte doesn't matter, we must set the masks to *1F FF FF 00* and filters to *01 A3 F2 XX,* where *XX* byte value does not matter since the mask doesn't check it. In this case CAN IDs ranging from *01 A3 F2 00* to *01 A3 F2 FF* will be received. Masking can be applied to whichever bit in the CAN ID mask.

You can set entirely different masks for Mask 1 and Mask 2 settings. Bear in mind that received CAN ID gets checked by both mask/filter links in parallel, so if any of them lets them through, the application will receive it.

Five Mask/Filter Presets allow easier reconfiguration of hardware filtering by storing and recalling different filtering presets. Left-clicking a Mask/Filter Preset button updates mask and filter windows with saved values. By clicking to *Write Masks* and *Write Filters* buttons the filtering is applied. Right-clicking any of the Mask/Filter Preset buttons saves current filtering settings to that button.

Monitoring CAN traffic

Flow tab in CAN Traffic group box displays CAN packets from top to bottom according to the time they were sent or received.

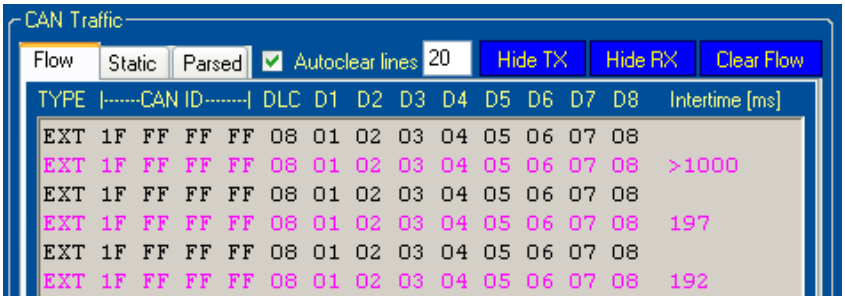

Illustration 13: CAN Traffic - TAB Flow

Sent packets are colored black and don't contain a time stamp value, whereas received packets have a magenta color and display the time elapsed between the last received packet and the current packet in milliseconds. If the time difference between the two packets exceeds 1 second, a *>1000* indication is displayed. The displayed resolution of intertime is 16 microseconds.

Clear flow button clears the window, *Hide TX/Show TX* and *Hide RX/Show RX* toggles displaying transmitted and received packets respectively.

Autoclear check box enables automatic clearing of the window, where the entry box to the right presents the number of lines at which the window is cleared. This feature comes handy with slower PCs, where a high packet rate for a prolonged time can cause the PC to become less responsive.

Logging is a powerful feature that allows real time storage of CAN data for later analysis. It is extremely useful to field operators who want to make an on-the-field record of the traffic for later simulation or analysis in a controlled environment.

Create new log button offers a dialog to create and save a new file in which CAN messages are saved. Upon saving, any text contained in flow text box is saved. This is useful if the user wants to save past traffic. This feature saves all packets, no matter received or sent, as long as they are displayed in the text box. If no data is present in text box an empty file is created in which data is saved during recording.

Start recording button triggers recording of the incoming data in a newly created log file. Only received data is recorded. Sent packets during recording are ignored. After starting the process the button turns red and the button label is "*Stop Recording*". By pressing the button again the recording operation is stopped.

Load Existing Log button loads a presaved log file into memory and displays the content of that log in flow text box. It is possible to append new traffic data in an existing log by starting recording.

Start Sending button is used to send a previously recorded data log. After starting the process the button turns red and the button label is "*Stop Sending*". By pressing the button again the sending operation is stopped. Sending is also stops when the process completes sending of the loaded log file.

Static tab displays CAN packets from top to bottom statically according to the order they were received. Packets with the same type and CAN ID get written to the same position as the previous packet with the same characteristics did and increment the Frame count value.

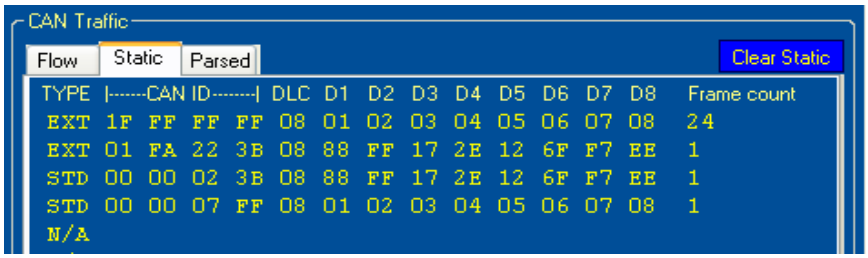

Illustration 14: CAN Traffic - TAB Static

Clear Static button clears all entries and respective frame counters. Up to 16 different CAN packets can be tracked in static view.

Parsed tab shows parametric representation of received data. It allows setting up 14 different parameters to be displayed in a human readable form.

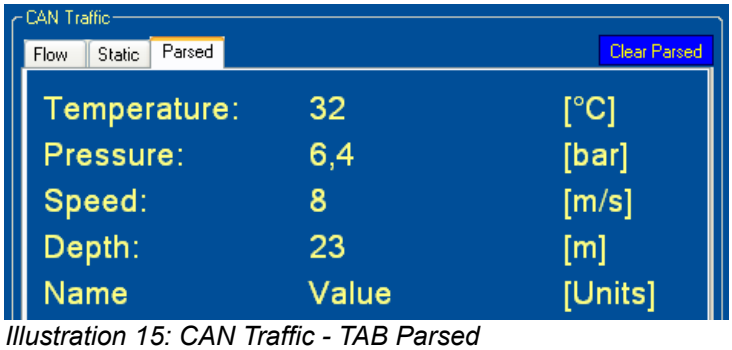

Illustration 15: CAN Traffic - TAB Parsed

The *Name* and *Units* labels (left and right column) can be renamed freely by right-clicking. A rename dialog pops up where you can input a name or units for the parameter.

Illustration 16: Rename dialog

By right-clicking any of the Value labels (center column) a Parse Data dialog window gets opened.

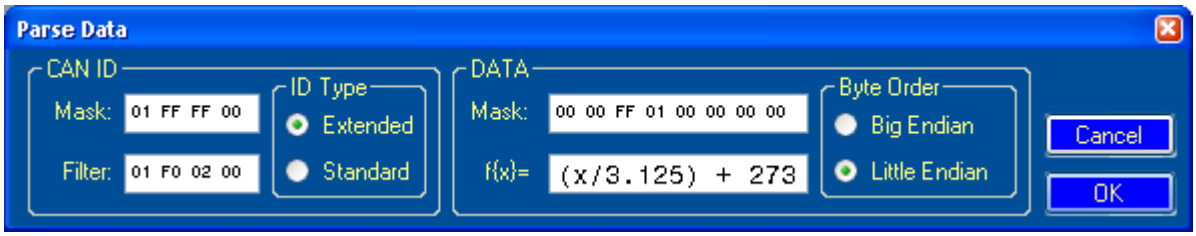

Illustration 17: Parse Data dialog

Set CAN ID type, mask and filter values to receive a specific CAN ID. The logic is the same as setting hardware filtering. Refer to chapter “Filtering” for more info about setting this up. In the above example all packets matching CAN ID bits from 8 to 24 (17 bits in total) with the set filter value will be accepted.

Bit acceptance examples:

Nr.	CAN ID	Accepted
1	01 F0 02 00	YES
2	01 F0 02 FF	YES
3	03 F0 02 00	YES
4	1F F0 02 49	YES
5	02 F0 02 00	NO
6	1E F0 02 00	NO
7	01 F0 03 00	NO

Examples 5 (0x02 = BIN00000010) and 6 (0x1E = BIN00011110) are not accepted because bit 24 is not set as required by the filter.

Example 7 (0x03 = BIN00000011) is not accepted because bit 8 is set and not cleared as required by the filter.

Proceed with masking data by setting mask bits in data mask fields to indicate where the needed information is located. Please note that with the little endian byte order, the LSB byte is the most left one, but the bit order always remains from right (LSb) to left (MSb). In the above example bits 16 to 24 (9 bits) are masked. Select byte order in accordance to the protocol specifications. Set the formula *f{x}*. Supported operands are +, -, /, * and ().

Upon reception of a filter matched CAN ID, bits 16 to 24 are retrieved from data portion of the packet in the following manner (little endian – least significant byte is first):

*x = ((DATA4 & 0x01) * 256) + (DATA3 & 0xFF)*

Note that mask settings for little and big endian configurations can differ. In case of a big endian byte order the above mask value of FF 01 is invalid because adjacent byte bits are not contiguous with each other like they are for little endian byte order. A few examples of big endian mask values where mask crosses byte boundaries are; 01 FF, FF 80, 0F F0, 3F FF 80.

Similarly a few examples for valid little endian mask settings are; F0 0F, 80 03, FE FF 7F.

In most cases full bytes are used for conveying data, therefore common masks are FF, FF FF, FF FF FF...

The received value *x* is further processed by *f{x}* function and the result is displayed in the corresponding *Value* label.

If *x* already presents the final value to be displayed, leave *f{x}* as *x*. If the value *x* is for example pressure value received in *Pascals*, you can set *f{x}* as *x/100000* to display the value in *bars* (*100000Pa = 1bar*). Another example is transforming *Kelvins* into *degrees Celsius* by setting *f{x}* as *x – 273* (*0°C = -273K*).

Example of setting up parsing of a J1939 front axle speed data parameter:

J1939 protocol uses PGN 65215 to send wheel speed information. PGN occupies bits 8 to 24 in CAN ID so we set the CAN ID mask to *01 FF FF 00*. We must set CAN ID filter to *00 FE BF 00* (DEC65215 = 0x0FEBF). The lower CAN ID byte (bits 0-7) presents source address and bits 25-28 present packet priority which we don't care about therefore those bits are not masked.

Front axle speed occupies DATA1 and DATA2 bytes (bits 0-15) so we set our data mask to *FF FF 00 00 00 00 00 00*. In J1939 LSB is sent first (little endian) so we select byte order accordingly.

By definition of PGN65215 from J1939 protocol each bit in the retrieved value *x* presents *1/256 km/h* starting from 0 (no offset). Therefore we must set the equation value to:

f{x} = x/256

if we want to receive the value in *km/h.* After receiving a valid packet for example;

EXT 1E FE BF 89 08 23 64 FF FF FF FF FF FF

by the means of mask and endian settings wheel speed value *x* is retrieved. In our case value *x* is *0x6423* hexadecimal (*25635* decimal). After this step *f{x}* is calculated by dividing *x* by *256* and the result (*25635/256 = 100,137 km/h*) is displayed in the corresponding *Value* label.

Sending CAN packets

There are 85 cells to enter your predefined CAN packets to be sent to CAN bus. They are structured by 17 cells in 5 presets. By clicking Preset buttons different presets of 17 cells are shown. Each cell contains a CAN message type dropdown list, an entry box with predefined CAN packet structure (CAN ID, DLC, data) and a corresponding send button.

Message type can be 29 bit extended (EXT) or 11 bit standard (STD). First four bytes of the CAN packet structure present CAN ID. In case of extended CAN ID bits *1F FF FF FF* are to be filled, in case of standard CAN ID bits *00 00 07 FF* are to be filled. DLC presents the number of CAN data bytes in the CAN packet and can be from 0 to 8. D1 to D8 represent data bytes to be sent in the packet. By clicking the cell button a packet will be sent to CAN bus.

Interval entry box lets you set a value which represents an interval time in milliseconds between sent packets from the bottom most packet cell. Number of repeats lets window you specify how many consecutive packets are to be sent by the continuous send cell. If value zero is entered, packets are being sent indefinitely until the user clicks the send button again.

By right-clicking any of the buttons a rename dialog window *(Illustration 16)* pops out which serves as an entry for renaming the specific button. By entering the text in the window and pressing OK the button will display a new name.