

PiCAN FD and SAE J2716 SENT USER GUIDE V1.0

Product name	PiCAN FD and SENT Board for Raspberry Pi
Model number	RSP-PICANFDSENT
Manufacturer	SK Pang Electronics Ltd

Contents

Table of Contents

1. Introduction	3
1.1. CAN Features.....	3
1.2. SAE J2716 SENT Features.....	3
2. Hardware Installation	3
1.3. Screw Terminal Plugs.....	4
1.4. CAN-BUS 120Ω Terminator.....	4
1.5. LEDs	4
1.6. Optional	4
3. Software Installation	5
1.7. Installing CAN Utils.....	5
1.8. Bring Up the Interface	5
4. Python Installation and Use.....	6
1.9. SENT Usage	7
1.10. ASCII Command Set.....	8
1.11. SENT GUI Demo.....	10

1. Introduction

This PiCAN FD board with SAE J2716 SENT. Classic CAN and CAN FD is provided by the Microchip MCP2518FD IC.

SAE J2716 SENT is provided by a dsPIC33 micro-controller. Communication to the Pi is over UART on ttyS0 using ASCII text commands. Example SENT GUI app is available written in Python3 and tkinter.

The firmware is updatable using the Microchip UnifiedHost java app. This requires the Raspberry Pi running in GUI mode.

Easy to install SocketCAN driver. Programming can be done in Python.

Optional 3A SMPS module which can power the PiCAN FD SENT board and Raspberry Pi from 7 to 24v external supply.

1.1. CAN Features

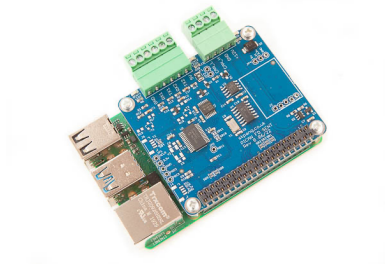
- Arbitration Bit Rate upto 1Mbps
- Data Bit Rate up to 8Mbps
- CAN FD Controller modes
- Mixed CAN2.0B and CANFD mode
- Conforms to ISO11898-1:2015
- High speed SPI Interface
- CAN connection via 4way screw terminal
- 120Ω terminator ready
- LED indicator (GPIO04)
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 and can1 to application
- Interrupt RX on GPIO25

1.2. SAE J2716 SENT Features

- Two independent SAE J2716 SENT channels
- Each channel can be configured for Tx or Rx
- Configurable Tick Time and Frame Time
- Status LED for each channel
- 5v DC output to power small sensor
- Powered by Microchip dsPIC33 micro-controller with updatable firmware
- Communicate to the Pi via ASCII text commands on ttyS0
- Python3 demo GUI app on Raspberry Pi

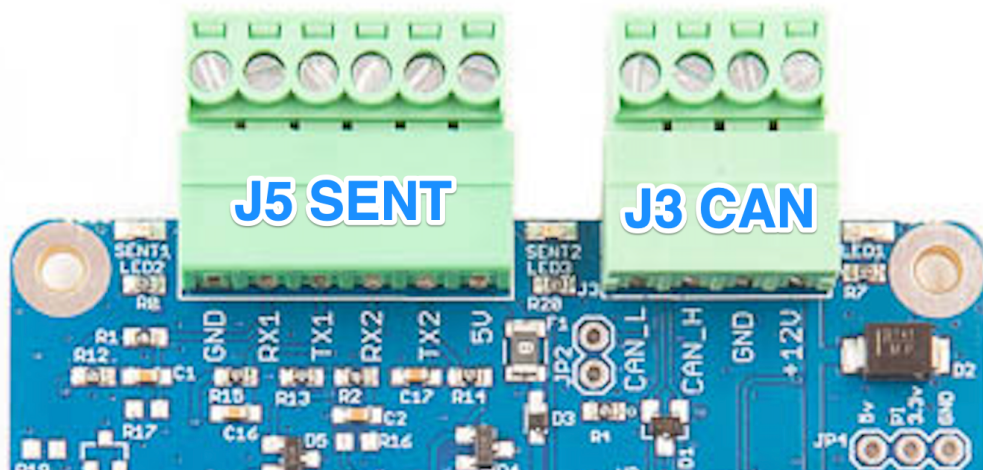
2. Hardware Installation

Before installing the board make sure the Raspberry Pi is switched off. Carefully align the 40way connector on top of the Pi. Use spacer and screw (optional items) to secure the board.



1.3. Screw Terminal Plugs

The CAN connections is made via the 4way screw terminal plug on J3 and SENT on J5.



J5 SENT	Function
1	GND
2	RX1
3	TX1
4	RX2
5	TX2
6	+5v

J3 CAN-Bus	Function
1	CAN_L
2	CAN_H
3	GND
4	+12v

1.4. CAN-BUS 120Ω Terminator

There is a 120Ω fitted to the board. To use the terminator solder a 2way header pin to JP2 then insert a jumper.

1.5. LEDs

There is a red LED (LED1) fitted to the board. This is connected to GPIO04. LED2 and LED3 are the SENT status light which is controlled by the dsPIC33.

1.6. Optional

SMPS. Switch mode power supply module option, this is a 5v module that can power the board and the Raspberry Pi. It has an input voltage range of 7v to 24v.

3. Software Installation

It is best to start with a brand new Raspbian image. Download the latest from:

After first time boot up, do an update and upgrade first.

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

Add these lines to the end of file:

```
enable_uart=1
dtparam=spi=on
dtoverlay=mcp251xfd,spi0-0,interrupt=25
```

Reboot Pi:

```
sudo reboot
```

1.7. Installing CAN Utils

Install the CAN utils by:

```
sudo apt-get install can-utils
```

1.8. Bring Up the Interface

You can now bring the CAN interface up with CAN 2.0B at 500kbps:

```
sudo /sbin/ip link set can0 up type can bitrate 500000
```

or CAN FD at 500kbps / 2Mbps. Use copy and paste to a terminal.

```
sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on sample-point .8 dsample-point .8
```

Connect the PiCAN FD LIN board to your CAN network.

To send a CAN 2.0 message use :

```
cansend can0 7DF#0201050000000000
```

This will send a CAN ID of 7DF. Data 02 01 05 – coolant temperature request.

To send a CAN FD message with BRS use :

```
cansend can0 7df##155555555555555555
```

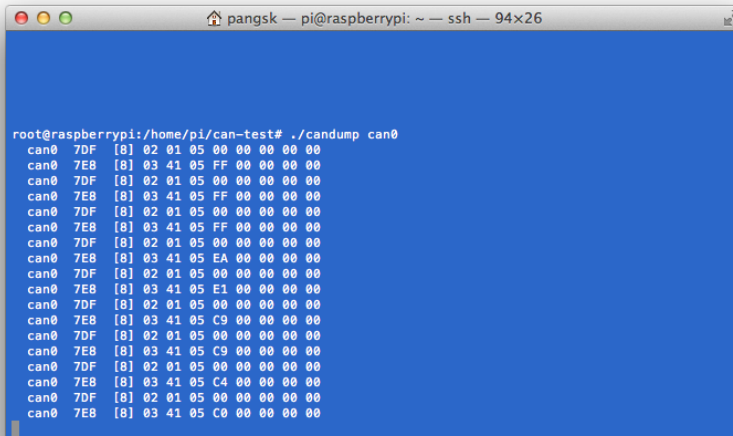
To send a CAN FD message with no BRS use :

```
cansend can0 7df##055555555555555555
```

Connect the PiCAN to a CAN-bus network and monitor traffic by using command:

```
candump can0
```

You should see something like this:



```
pangsk — pi@raspberrypi: ~ — ssh — 94x26
root@raspberrypi:/home/pi/can-test# ./candump can0
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 FF 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 FF 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 FF 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 EA 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 E1 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 C9 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 C9 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 C4 00 00 00 00
can0 7DF [0] 02 01 05 00 00 00 00 00
can0 7E8 [0] 03 41 05 C0 00 00 00 00
```

4. Python Installation and Use

Ensure the driver for PiCAN FD is installed and working correctly first.

Clone the pythonCan repository by:

```
git clone https://github.com/hardbyte/python-can
```

```
cd python-can
```

```
sudo python3 setup.py install
```

Check there is no error been displayed.

Bring up the can0 interface:

```
sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrates 2000000 fd on sample-point .8 dsample-point .8
```

Now start python3 and try the transmit with CAN FD and BRS set.

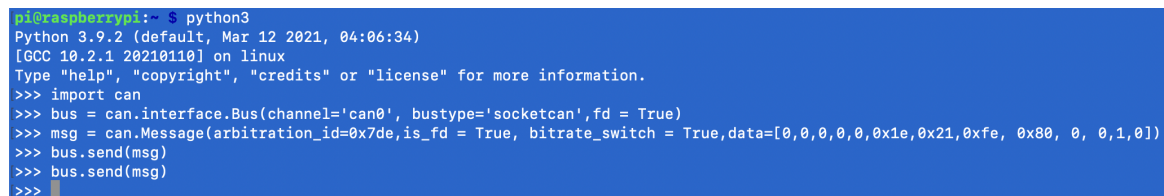
```
python3
```

```
import can
```

```
bus = can.interface.Bus(channel='can0', bustype='socketcan', fd = True)
```

```
msg = can.Message(arbitration_id=0x7de, is_fd = True, bitrate_switch = True, data=[0,0,0,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])
```

```
bus.send(msg)
```



```
pi@raspberrypi:~$ python3
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import can
>>> bus = can.interface.Bus(channel='can0', bustype='socketcan', fd = True)
>>> msg = can.Message(arbitration_id=0x7de, is_fd = True, bitrate_switch = True, data=[0,0,0,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])
>>> bus.send(msg)
>>> bus.send(msg)
>>>
```

To received messages and display on screen type in:

```
notifier = can.Notifier(bus, [can.Printer()])
```

```
>>> msg = can.Message(arbitration_id=0x7de,is_fd = True, bitrate_switch = True,data=[0,0,0,0,0,0x1e,0x21,0xfe,
0x80, 0, 0,1,0])
>>> bus.send(msg)
>>> bus.send(msg)
>>> notifier = can.Notifier(bus, [can.Printer()])
>>> Timestamp: 1653164390.561713      ID: 0400      S Rx      F BS      DL: 64      44 64 56 35 74 56 74 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 56 74 65 70
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69      Channel: can0
Timestamp: 1653164393.817720      ID: 0400      S Rx      F BS      DL: 64      44 64 56 35 74 56 74 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 56 74 65 70
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69      Channel: can0
Timestamp: 1653164398.897447      ID: 0400      S Rx      DL: 4      02 00 56 34      Chanr
el: can0
Timestamp: 1653164403.721645      ID: 0400      S Rx      DL: 8      00 10 00 00 00 00 00 00 00      Chanr
el: can0
Timestamp: 1653164406.113598      ID: 0100      S Rx      F BS      DL: 64      44 64 56 35 74 56 74 00 00 00 00
00 00 00 00 00 00 0d fa 23 53 25 ad ad 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 5a df a5 70 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69      Channel: can0
Timestamp: 1653164406.825656      ID: 0100      S Rx      F BS      DL: 64      44 64 56 35 74 56 74 00 00 00 00
00 00 00 00 00 00 0d fa 23 53 25 ad ad 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 5a df a5 70 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69      Channel: can0
>>>
```

Documentation for python-can can be found at :

<https://python-can.readthedocs.io/en/stable/index.html>

More expamples in github:

<https://github.com/skpang/PiCAN-FD-Python-examples>

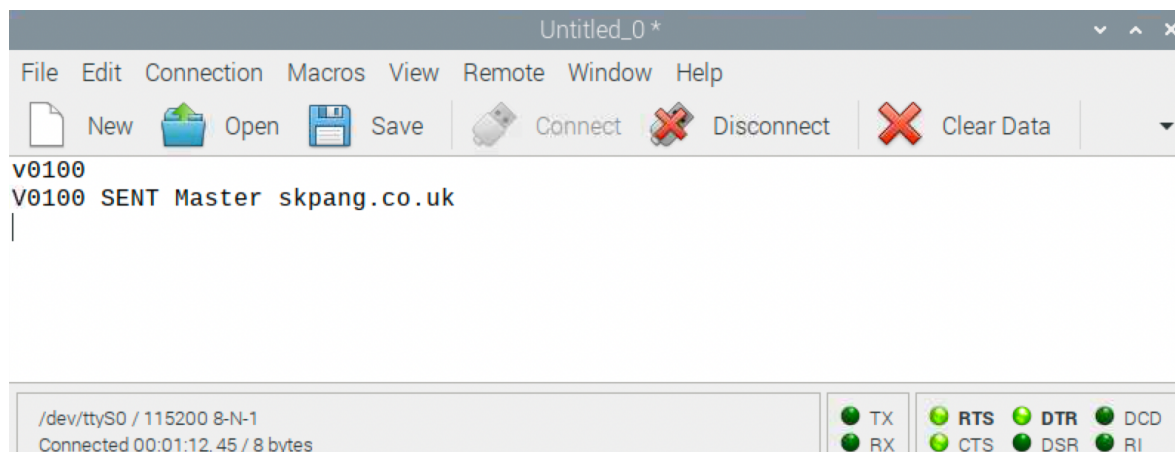
1.9.SENT Usage

The SENT controller is communicated to the Pi over the UART on ttyS0 port at 115200 8-N-1 setting.

Using Coolterm for Raspberry Pi or a text base terminal. Set the port to /dev/ttyS0 / 115200 8-N-1 and connect.

On the terminal type in 'v' and press enter. Type in 'V' and press enter.

You should see a reply like this screen:



1.10. ASCII Command Set

To control the SENT port a simple ASCII command is sent from the Pi to the PiCAN FD LIN board.

Commands from Pi to the PiCAN FD SENT board

The command requires a carriage return character (0x0D). All values are in hex.

V	Get hardware version
v	Get firmware version
ox	Open SENT port. x:1 port 1. x:2 port. x:3 both ports
cx	Close SENT port. x:1 port 1. x:2 port. x:3 both ports
dxsiiiiii	Transmit a SENT frame with 6 nibble dxsiiiiii x :1 port1. x:port2 x:3 both ports s :Status nibble 0 to 0xf iiiiii :6 nibble of data. 0 to 0xf each
txii	Set Tick Time 03 to 90 (decimal) txii x :1 port1. x:port2 x:3 both ports ii :00 to 90
fxiii	Set Frame Time 284 to 920 (decimal) fxiii x :1 port1. x:2 port2 iii :284 to 920
px	Set Pause Pulse Present px x :1 port1. x:2 port2 x:3 both ports
mxii	Set operation mode mxii x :1 port1. x:port2 x:3 both ports i :r receive. i:t transmit

Data from PiCAN FD SENT board to Pi

After each command is received, an acknowledge character is sent back to the Pi.

z Command is ok.

E Error. Command is not ok or command format error.

dxsiiiiic

x:1 port1. x:2 port2

s: Status nibble

i: Six nibbles of data

c: Checksum nibble

Example 1. Port 1 to receive at 3us tick rate. Device such as Microchip LXE3302AR001 - LX3302A Inductive Position sensor.



To setup port 1 to receive at 3us tick rate.

Command string:

m1r

t103

o1

You should now see a series of data returned like shown below:

d10E66E66F

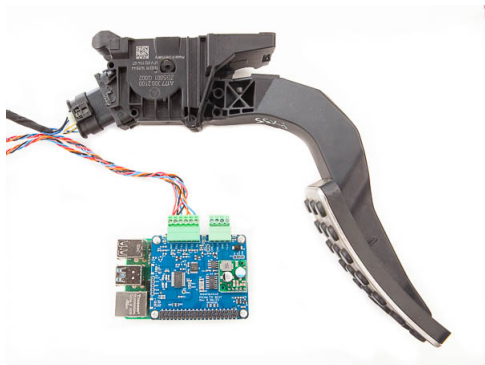
d10E66E66F

d10E66E66F

d10E66E66F

d10E66E66F

Example 2. To read the output of a vehicle accelerator pedal. This SENT device has two outputs.



To setup port 1 and 2 to receive at 3us tick rate.

Command string:

m3r

t103

t203

o3

You should now see a series of data returned like shown below:

d140EFFCFE

d200776CF6

d140EE15F8

d2007685F6

1.11. SENT GUI Demo

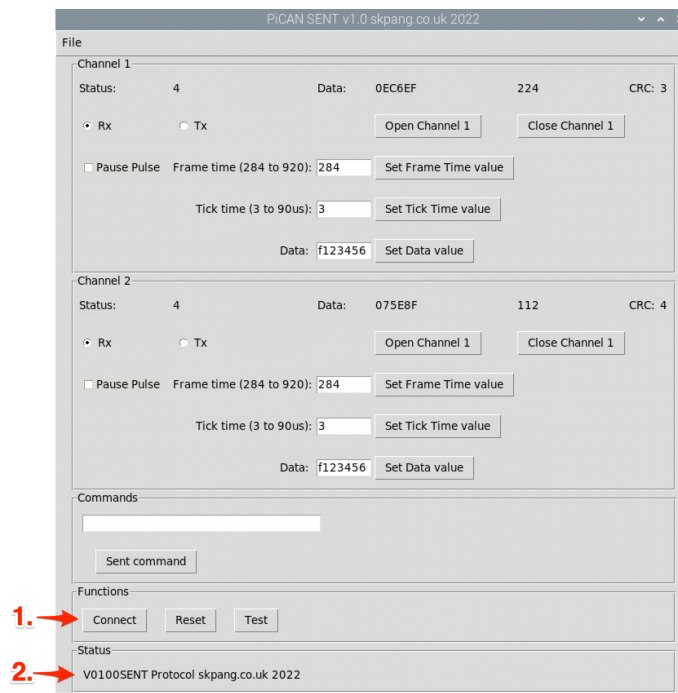
A simple GUI interface is available to download from github:

https://github.com/skpang/PiCAN_SENT_GUI_demo

To run the program start a terminal and type in:

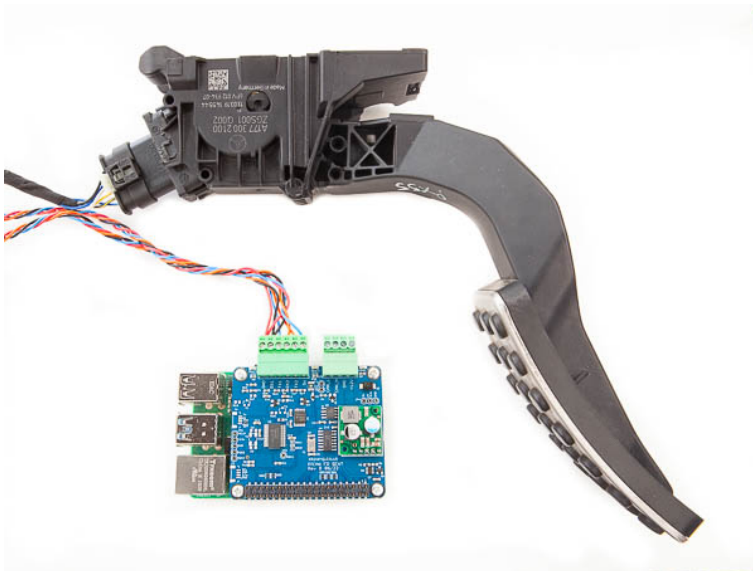
```
python3 pican-sent.py
```

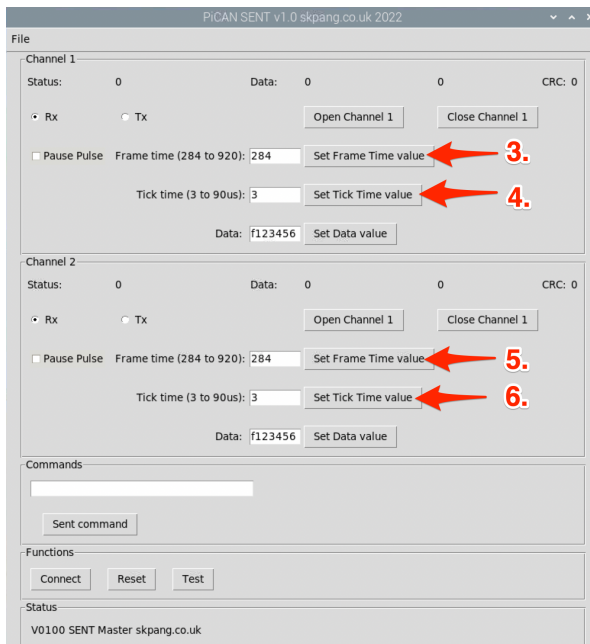
You should see a screen shown below:



1. Click the Connect button to connect to the PiCAN FD SENT board.
2. Check the Status box is updated.

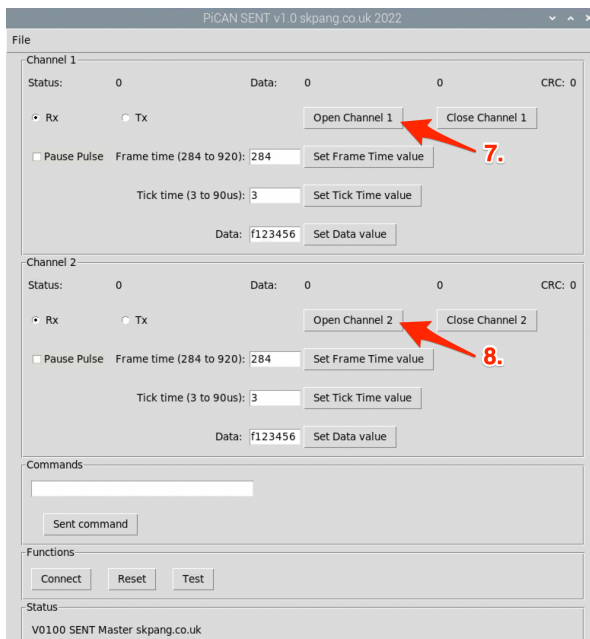
Example 1. Connect an accelerator pedal to the board.





Now click.

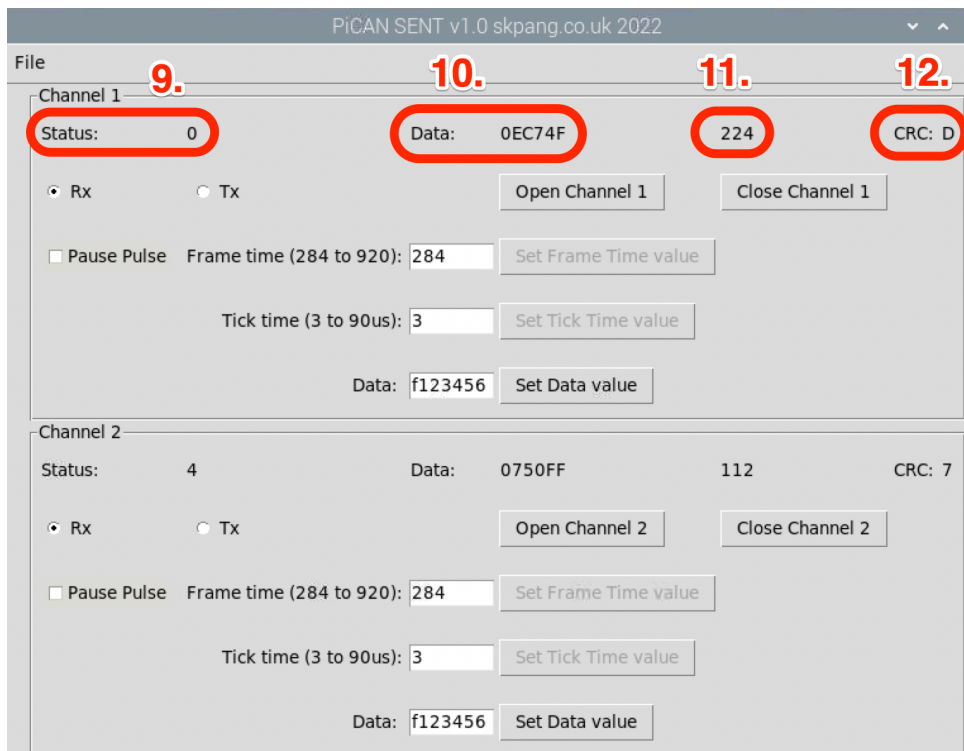
3. Set Frame Time value for Channel 1
4. Set Tick Time value for Channel 1
5. Set Frame Time value for Channel 2
6. Set Tick Time value for Channel 1



To open the channels click.

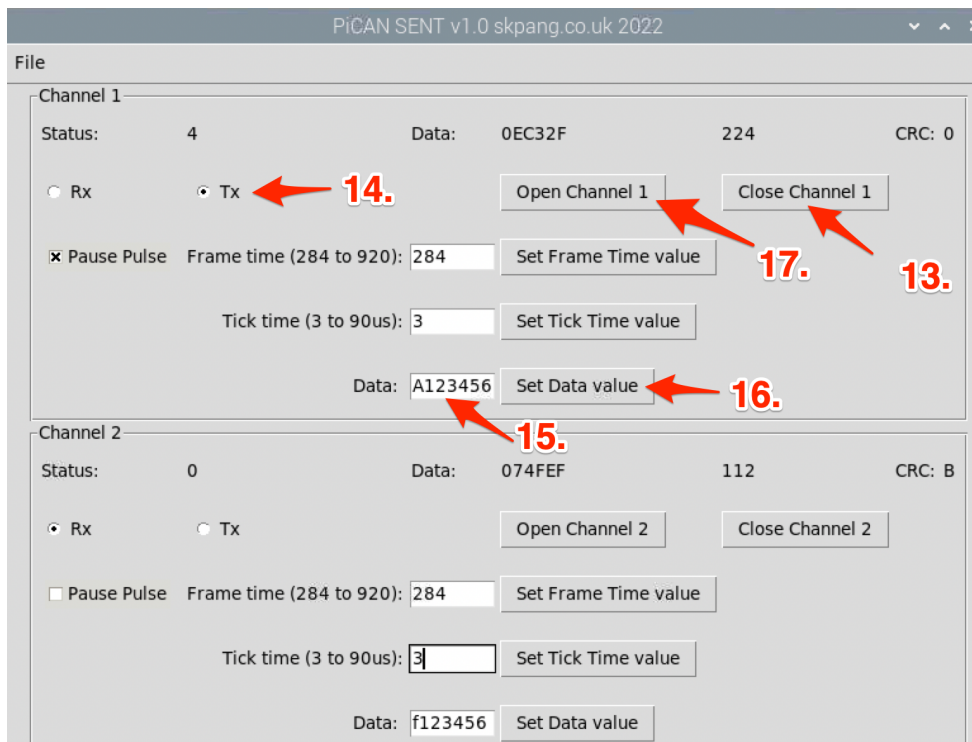
7. Click Open Channel 1
8. Click Open Channel 2

You should now see data from both channels.



- 9. Status nibble
- 10. Six data nibble in hex
- 11. First 3 data nibble converted to decimal
- 12. CRC nibble

Example 2. Transmitting SENT frames



To transmit SENT frames, click.

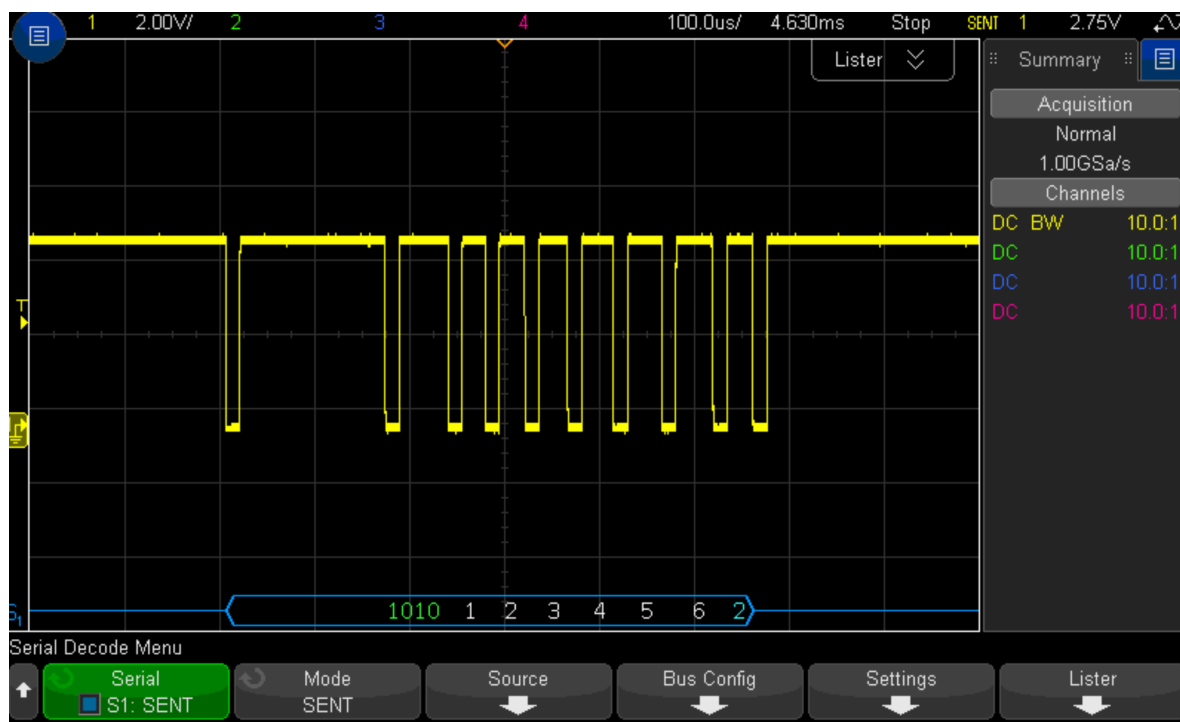
13. Close Channel button
14. Tx radio button
15. Enter status and data values
16. Set Data value button
17. Open Channel 1 button

The example data values are

Status nibble : A

Data nibble : 123456

From a scope you should see a waveform like shown below:



This should a status nibble of 0xA. Six data nibble and a CRC of 0x2.