USB2CANFD-X2 User Manual

V1.0

Dual Channel CANFD For High Speed USB2.0

Version	Change History	Contact Information
V1.0	First Released 2021/12/22	
V2.0		

1. Introduction

1.1 Series Products

Product Name	CAN Version	Chanel	Bitrate (MAX)	Power Isolation	MAX FPS	Interface
USB2CANFD-X2	CAN2.0A/B CANFD 1.0	2	12Mbit/s	2500V	15000fps	USB 2*D-SUB 9PIN
USB2CANFDX2-MPCIE	CAN2.0A/B CANFD 1.0	2	12Mbit/s	2500V	15000fps	USB 2*3P Connector MINIPCIE
USB2CANFDX2-Core	CAN2.0A/B CANFD 1.0	2	12Mbit/s	2500V	15000fps	MINIPCIE

1.2 Description

The USB2CANFD-X2 is a plug and play high speed USB2.0 to CANFD adapter enables the connection of dual channel CANFD networks to a computer via USB. Each CAN FD channel is separately isolated against USB with a maximum of 2500V.

The new CAN FD standard (CAN with Flexible Data rate) is primarily characterized by higher bandwidth for data transfer. The maximum of 64 data bytes per CAN FD frame (instead of 8 so far) can be transmitted with bit rates up to 12 Mbit/s, USB2CANFD-X2 supports 120Ω termination resistor enable by software.

CAN bus connection via D-Sub, 9-pin (in accordance with CiA® 303-1)

Provide Drivers for Windows/Linux, Compatible with windows PCAN-View, Linux Socket CAN.
The monitor software PCAN-View and the programming interface PCAN-Basic for the development of applications with CAN connection are included in the scope of supply and support the new standard CAN FD.

1.3 Features

CAN operation properties

- Adapter for High-speed USB 2.0 (compatible to USB 1.1 and USB 3.0)
- Complies with CAN specifications 2.0A/B and FD
- CAN FD support for ISO and Non-ISO standard software switchable
- **CAN FD bit rates data field (64 bytes max.) from 25 kbit/s up to 12Mbit/s.**
- **Class CAN bit rates from 25 kbit/s up to 1 Mbit/s;**
- **Time stamp Resolution Up to 1 μs;**
- Each CAN FD Signal &Power Separately Isolated Up to 2500 Volts against USB;
- CAN 120Ω termination resistor Activated/Deactivated Through Software;
- Support CAN Clock Settings.
- Measurements of bus load including error frames and overload frames on the physical bus
- Induced error generation for incoming and outgoing CAN messages
- Extended operating temperature range -40 - 85 °C (-40 - 185 °F)

System Requirements

- Operating system Windows 10, 8.1, 7 (32/64-bit) or Linux (32/64-bit)
- A vacant USB port (USB 1.1, USB 2.0 or USB 3.0) at the computer or at a USB hub connected to the computer

Scope of Supply

- USB2CANFD-X2 DEVICE in aluminum casing CAN FD interface drivers for Windows 10, 8.1, 7 and Linux (32/64-bit)
- CAN monitor PCAN-View for Windows
- Dual DB9 To Terminal Adapter Board with 120Ω TERM RESISTOR Selectable By Jumper
- Programming interface PCAN-Basic for developing applications with CAN connection
- Programming interfaces for standardized protocols from the
- Manual in PDF format

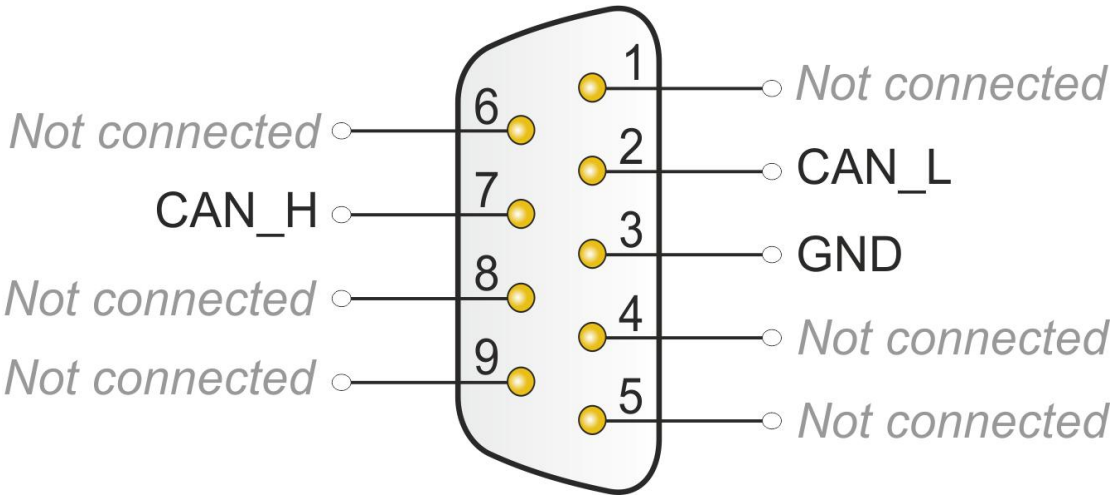
1.3 Technical Specification

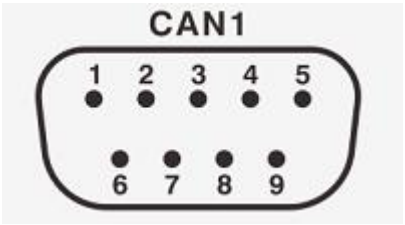
Connector	
CANFD	Dual Channel 9PIN D-SUB Connectors
USB	USB plug type A（Computer） USB plug type B（Devcie）
CAN Features	
Protocols	CAN 2.0A (standard format) CAN 2.0B (extended format) CAN FD ISO 11898-1:2015 CAN FD non-ISO
CAN bit rates	25 kbit/s up to 1 Mbit/s
CANFD bit rates	25 kbit/s up to 12Mbit/s
Galvanic isolation	Signal &Power Separately Isolated by 2500 Volts against USB
Micro controller	180MHZ Cortex-M4 MCU
Timestamp resolution	1 μs
Built In 120Ω Termination Resistor	Activated/Deactivated Through Software
Software	Windows PCAN-VIEW Linux PCAN-VIEW（Instruction） Linux SOCKET-CAN: ● CAN Utils（Instruction）， ● C（Source Code Instruction）， ● Python（Source Code Instruction）
PCAN BASIC API Windows 10, 8.1, 7(32/64-bit) Windows CE 6.x (x86/ARMv4) Linux (32/64-bit)	C#, C++/CLR, Delphi,VB.NET, Java, Phyton 2.6
Third Party Software	LabView, CodeSys, Matlab, BUSMASTER, EasyMotion Studio, CANmoon, XX-SCAN, PCAN-Explorer5
Others	
Temperature	-40°~ 85°
PCBA Size (L * W * H)	84x80x28 mm
Weight	190g

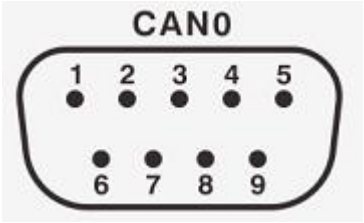
2 Hardware Connection LED Indication

Two High-speed CAN buses (ISO 11898-2) can be connected, one to each D-Sub connector. The pin assignment for CAN corresponds to the specification CiA® 303-1.

2.1 DB9 PINOUT Description



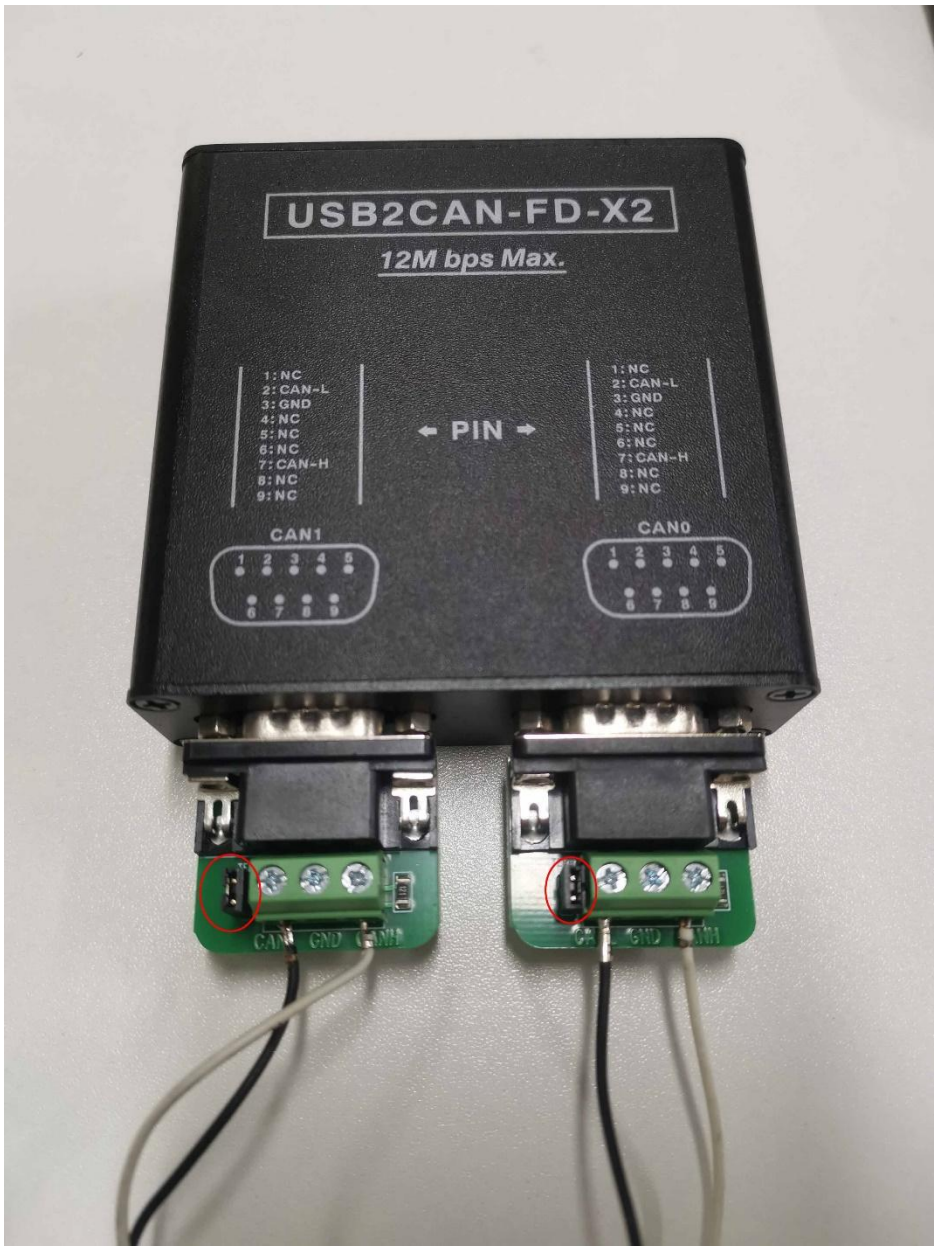
CAN1	PIN OUT Description	
	1	NC
	2	CANL bus line (dominant low)
	3	CAN_GND
	4	NC
	5	NC
	6	NC
	7	CANH bus line (dominant high)
	8	NC
	9	NC

CAN0（Normal）	PIN OUT Description	
	1	NC
	2	CANL bus line (dominant low)
	3	CAN_GND
	4	NC
	5	NC
	6	NC
	7	CANH bus line (dominant high)
	8	NC
	9	NC

Note1: Not connect GND do not affect normal communication, if cable with shielding suggest connect to GND;

2.2 Activated 120Ω Resistor

By Hardware



We provide 2PCS DB9 To Termination Board with 120Ω termination resistor, put on jumper as picture show（red circle）to enable.

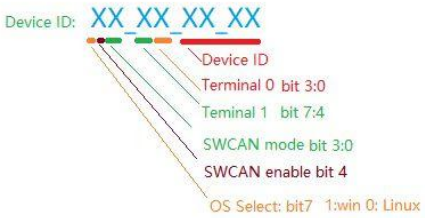
By Software

Built-in 120Ω Termination Resistor design, enable USB2CANFD-X2 Activated or Deactivated 120Ω Termination Resistor by software.

Note: Remove jumper from the DB9 To Termination Board.

Device serial number byte 2 used to control 2 channels of terminal resister.
BYTE3 bit0—3 control channel0--- 1: resister on 0 :terminal resister off
BYTE3 bit7—4 control channel1--- 1: resister on 0 :terminal resister off

Example: setting Device serial number to 0x11xxxxxx will enable 120 Ohm terminal on both channel .

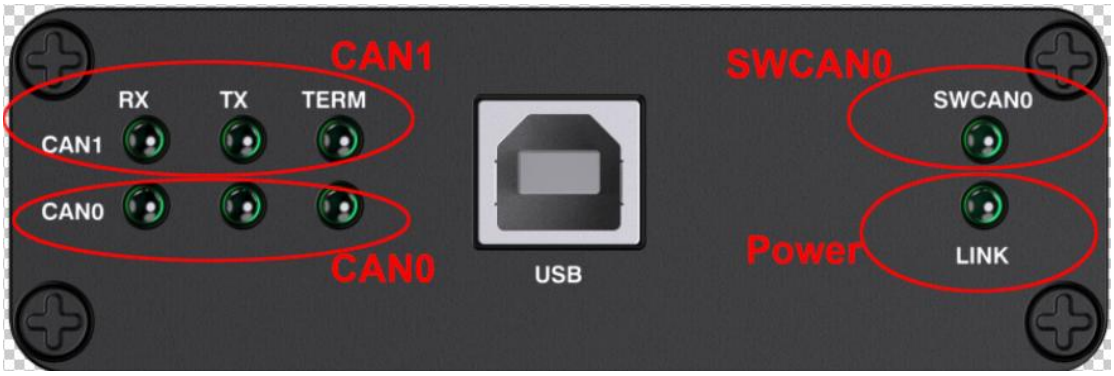


Note: re plug the Device to USB Host then take effect.

Figure2 shows CAN0/CAN1 LED Status Turns Green When Enable CAN0/CAN1 (In Red Circle)

Figure2

2.2 LED Indicate Description

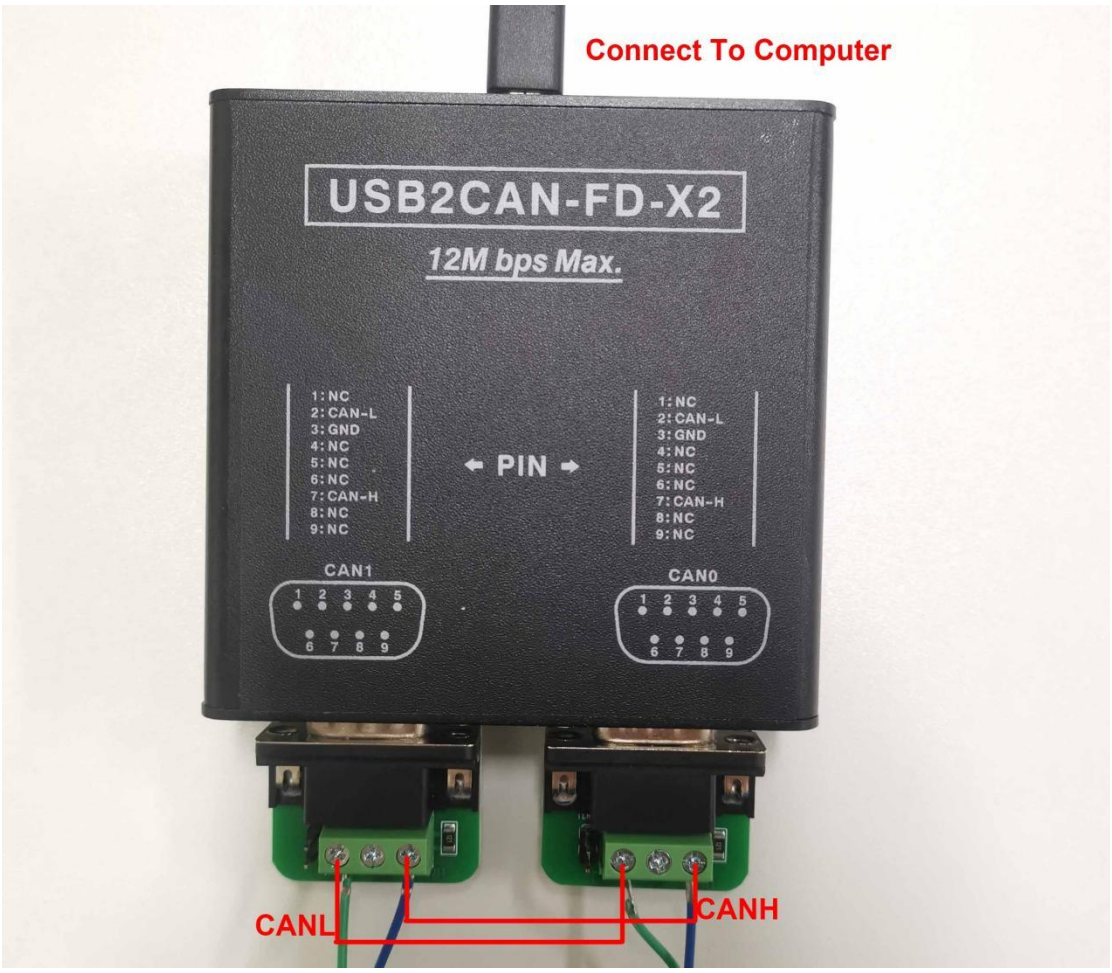


When plug usb2canfd-x2 device to Computer All lights are flashing for one second.
Then TERM LED And LINK LED turns to be green.

Channel	Description
Power LED Indicator	When USB2CANFD-X2 Power Up LINK LED turns to be blinking.
CAN1 LED Indicator	TX LED Blinking, Sending Data; RX LED Blinking, Receiving Data; TERM LED Green,120Ω Activated;
CAN0 LED Indicator	TX LED Blinking, Sending Data; RX LED Blinking, Receiving Data; TERM LED Green,120Ω Activated;

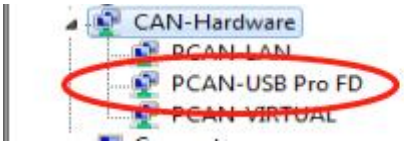
3 PCANVIEW For Windows

This part is for USB2CANFD-X2 Windows Software PCANVIEW. Connect as below:



3.1 Installing drivers

- Connect the USB2CANFD-X2 Device to PC, drivers will recognize automatically;
- If not, Unzip PEAK-System_Driver-Setup.zip and install PeakOemDrv.exe accordingly.
- After the driver is successfully recognized, the usb2canfd-x2 device can be viewed in device manager as shown in the following figure.



LED Indicate of Link is BLINKING.

3.2 Start and Initialize PCAN-View

Step1, Open 2 of PCAN-View windows. The Connect dialog box appears

Step2, Select an interface from the list. (Channe1, Channel2 Setting as follows)

Step3, From the drop-down menu, choose a Clock Frequency. The selectable bit rates in the following are based on this setting

Step4, From the drop-down list, select a Nominal Bit rate, which is used for the arbitration phase (max. 1Mbit/s).

Step5. Enable the Data Bit rate checkbox.

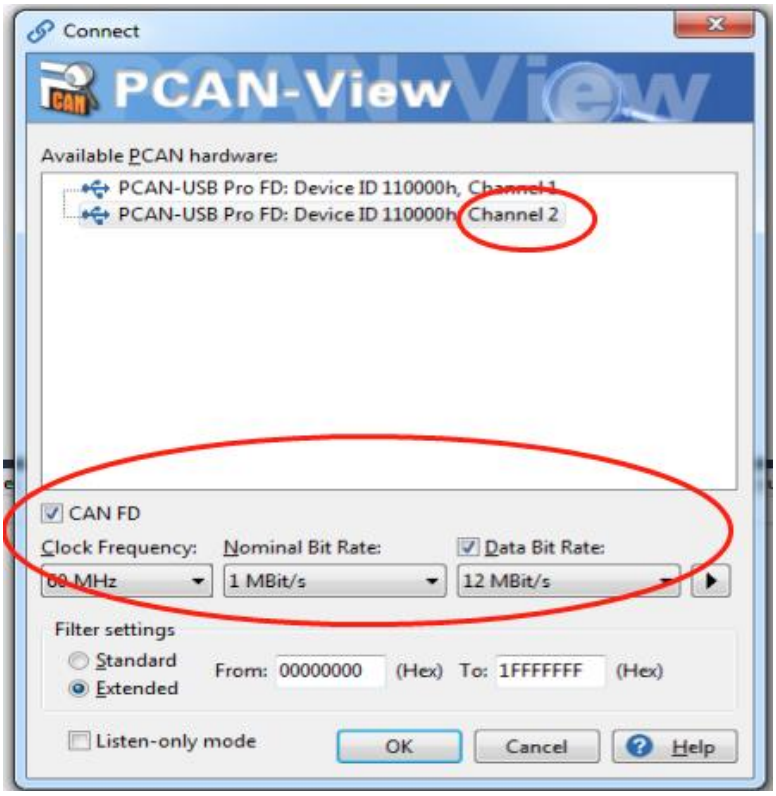
Step6, From the drop-down menu, choose an additional Data Bit rate for the CAN FD bus. The bit rate selected here is used to transfer the data fields of a CAN FD frame with a higher bit rate.

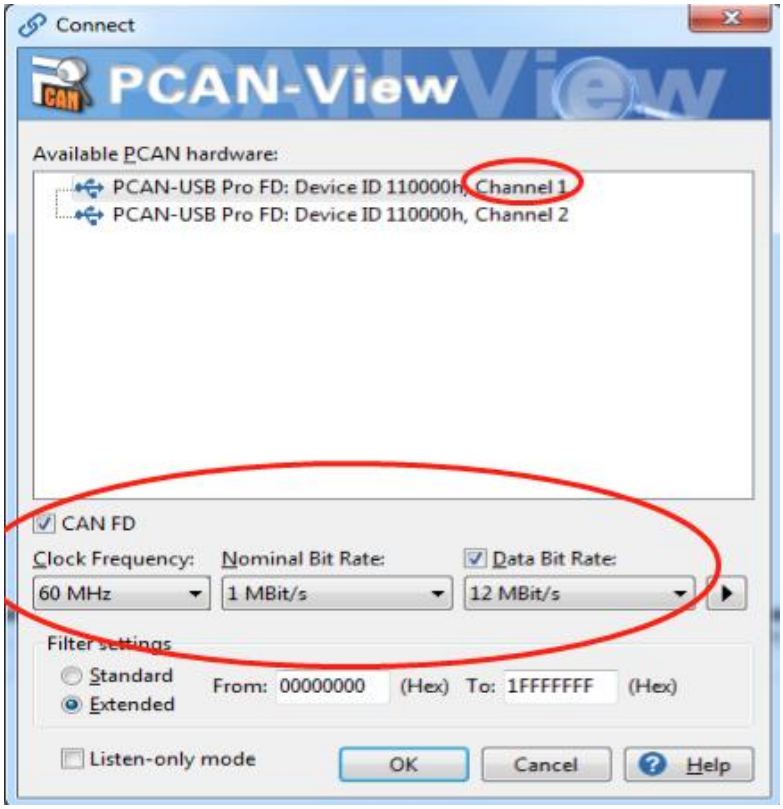
Step7. Under Filter settings you can limit the range of CAN IDs to be received, either for standard frames (11-bit IDs) or for extended frames (29-bit IDs).

Step8. Activate the Listen-only mode if you do not actively take part in the CAN traffic and just want to observe. This also avoids an unintended disruption of an unknown CAN environment (e.g. due to different bit rates).

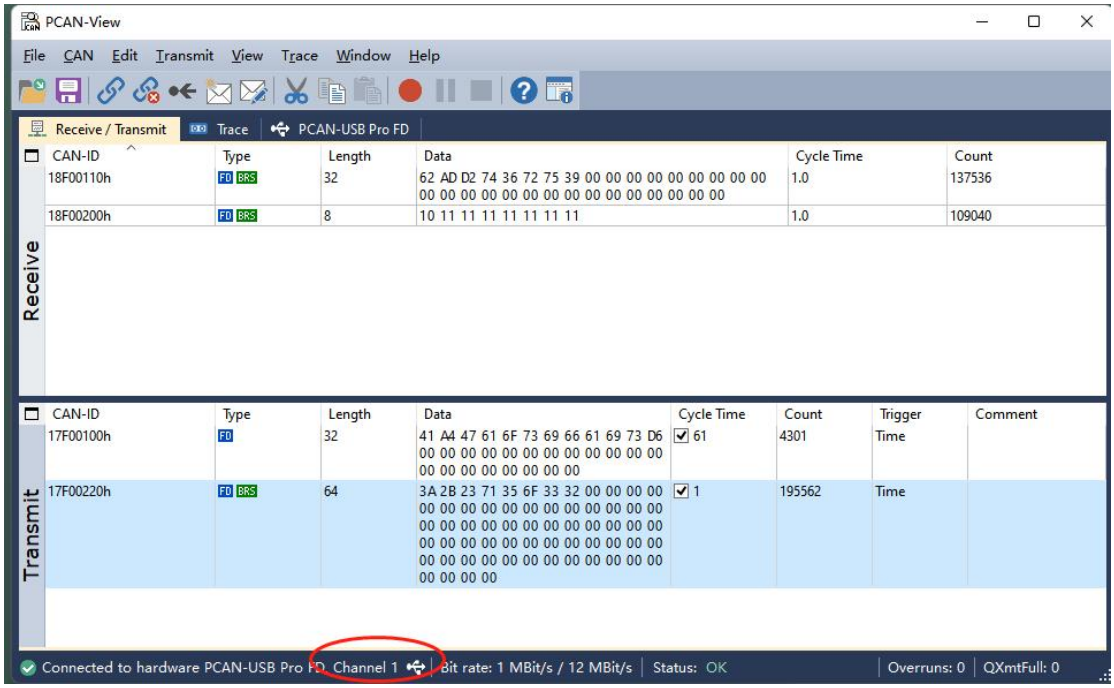
Step9. Confirm the settings in the dialog box with OK. The main window of PCAN-View appears (see Figure 14)

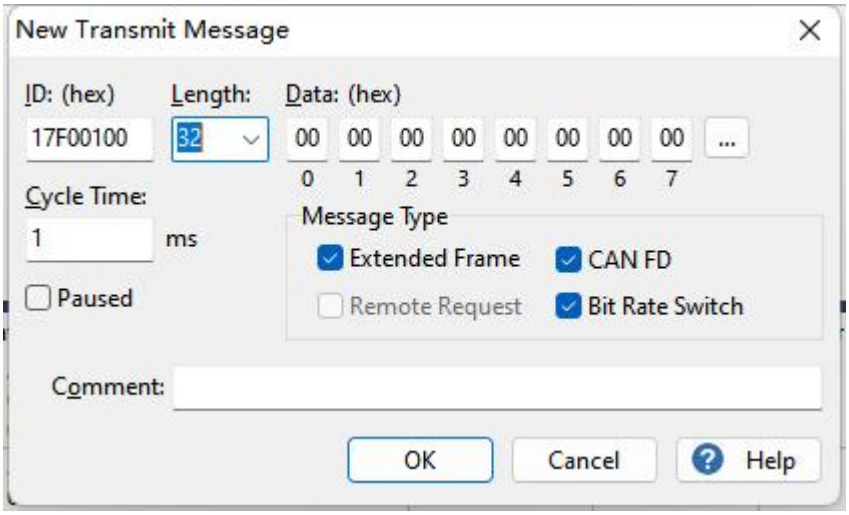
Set as below pictures for channel1 and channel 2.
Setting Value:
CANFD,Clock Frequency 60MHz,Norminal Bit Rate 1MBit/s,Data Bit Rate 12MBit/s


Connect
PCAN-View
Available PCAN hardware:
PCAN-USB Pro FD: Device ID 110000h, Channel 1
PCAN-USB Pro FD: Device ID 110000h, Channel 2
CAN FD
Clock Frequency:
Nominal Bit Rate:
Data Bit Rate:
60 MHz
1 MBit/s
12 MBit/s
Filter settings
Standard
Extended
From: 00000000 (Hex) To: 1FFFFFFF (Hex)
Listen-only mode
OK
Cancel
Help


Connect
PCAN-View
Available PCAN hardware:
PCAN-USB Pro FD: Device ID 110000h, Channel 1
PCAN-USB Pro FD: Device ID 110000h, Channel 2
CAN FD
Clock Frequency:
Nominal Bit Rate:
Data Bit Rate:
60 MHz
1 MBit/s
12 MBit/s
Filter settings
Standard
Extended
From: 00000000 (Hex) To: 1FFFFFFF (Hex)
Listen-only mode
OK
Cancel
Help

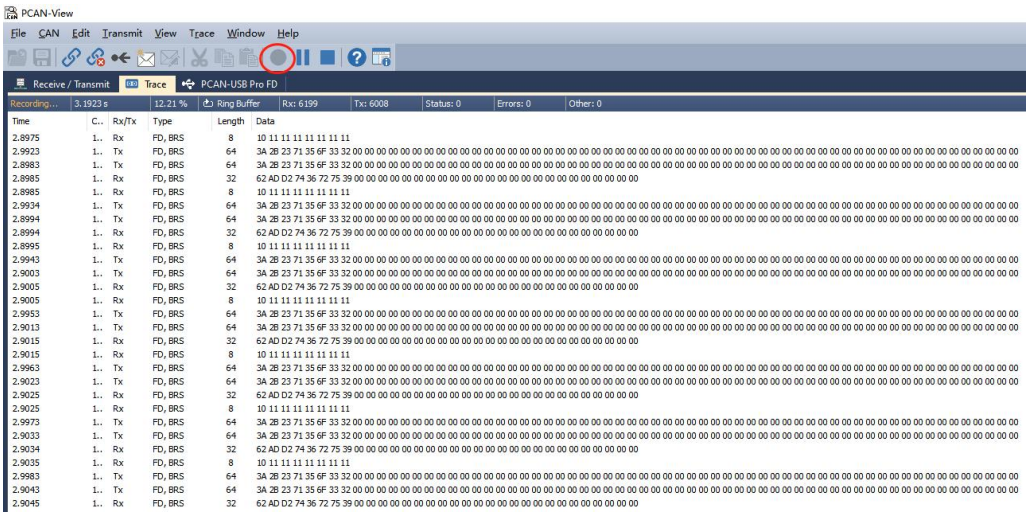
3.3 Receive/Transmit Tab


PCAN-View
File CAN Edit Transmit View Trace Window Help
Receive / Transmit
Trace
PCAN-USB Pro FD
Receive
CAN-ID Type Length Data Cycle Time Count
18F00110h FD BRS 32 62 AD D2 74 36 72 75 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 1.0 137536
18F00200h FD BRS 8 10 11 11 11 11 11 11 11 1.0 109040
Transmit
CAN-ID Type Length Data Cycle Time Count Trigger Comment
17F00100h FD 32 41 A4 47 61 6F 73 69 66 61 69 73 D6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 61 4301 Time
17F00220h FD BRS 64 3A 2B 23 71 35 6F 33 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00



2. Enable the CAN FD checkbox to define a CAN FD message with a maximum Length of 64 data bytes.
3. Enter the ID, the data Length, and the CAN message Data. With a length of more than 8 bytes, click on and enter the data bytes into the editor.
4.Enter a value into the Cycle Time field to choose manually or periodically message transmission. Enter a value greater than 0 to transmit periodically. Enter the value 0 to transmit only manually.
5. Enable the Bit Rate Switch checkbox, that the data of a CAN FD message is transmitted with the selected Data Bit rate.
6. Confirm the entries with OK. The created transmit message appears on the Receive/Transmit tab.
7. Trigger selected transmit messages manually with the menu command Transmit > Send (alternatively Space bar). The manual transmission for CAN messages being transmitted periodically is carried out additionally.

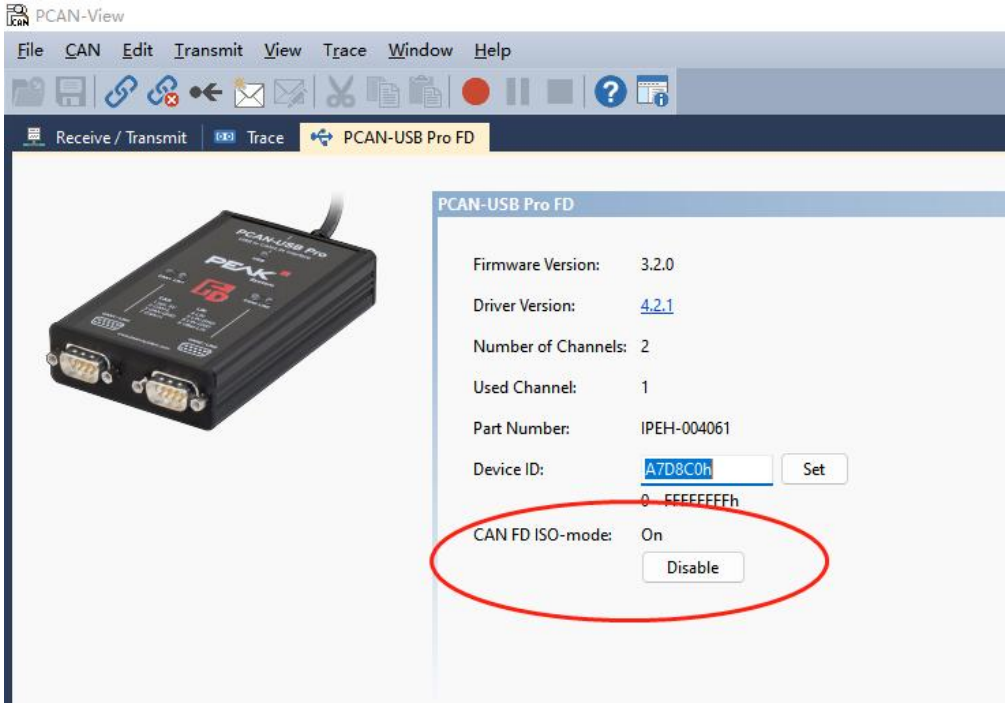
3.4 Trace Tab



On the Trace tab, the data tracer (data logger) of PCAN-View is used for logging the communication on a CAN bus. During this process the messages are cached in the working memory of the PC. Afterwards they can be saved to a file.

The Tracer runs either in linear or in ring buffer mode. The linear buffer mode stops the Tracer as soon as the buffer is full. The ring buffer mode overwrites the oldest messages by new ones as soon as the buffer is full.

3.5 PCAN-USB Pro FD Tab



The PCAN-USB FD Pro tab contains some detailed information about the hardware and driver. In addition, you can assign a Device ID to the adapter. Thus, it can be uniquely identified while operating several PCAN-USB Pro FD adapters on a computer at the same time.

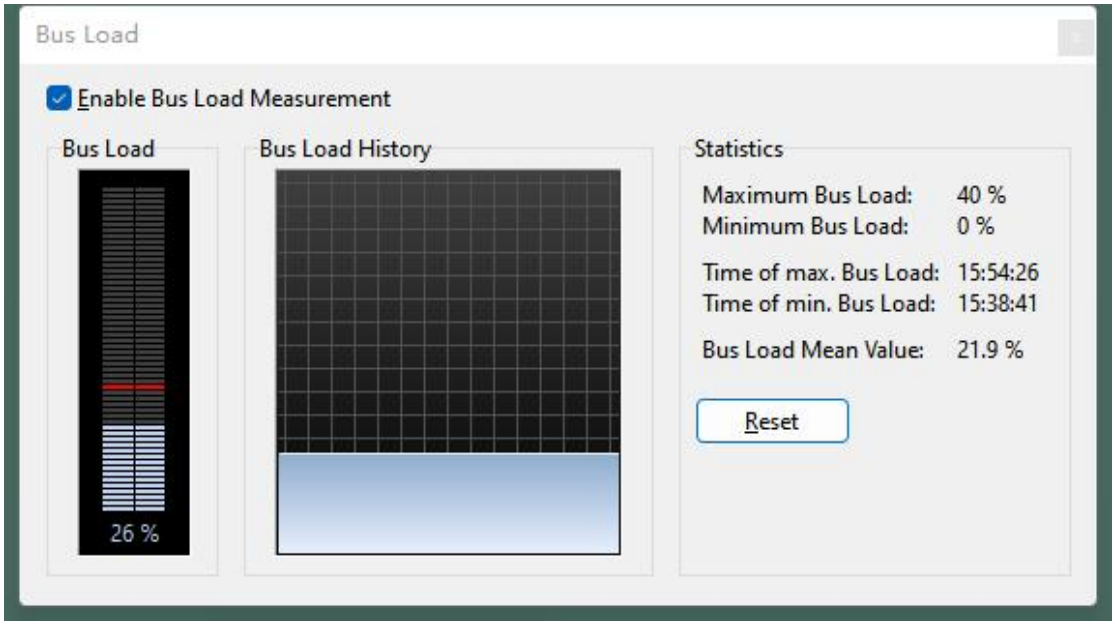
To identify a PCAN-USB Pro FD adapter, you first go to the dialog box for selecting the hardware of PCAN-View. In the list "Available PCAN hardware and PCAN-nets", you can perform a right-click on every USB adapter and execute the command "identify". Thereby, the LED of the corresponding adapter flashes shortly.

CAN FD ISO-mode

The defined in the ISO 11898-standard is not compatible with the original protocol. PEAK-System takes this into account by supporting both protocol versions with their CAN FD interfaces.

If required, the user can switch to the CAN FD protocol used in the environment with the **Enable / Disable** button ("Non-ISO" and "ISO")

3.6 Bus Load Tab



On the Bus Load tab, the current bus load, time course, and statistical information of the CAN channel are displayed. The CAN bus load reflects the utilization of transmission capacity.

3.7 Status Bar

The status bar shows information about the current CAN connection, about error counters (Overruns, QXmtFull) and shows error messages.
You can find further information about the use of PCAN-View in the help which you can invoke in the program via the Help menu or with the **F1** key

3.8 PCAN-Basic API

You can find files of the programming interface PCAN-Basic in the directory branch Develop. This API provides basic functions for linking own programs to CAN and CAN FD interfaces by PEAK-System and can be used for the following
operating systems:

- Windows 10, 8.1, 7(32/64-bit)
- Windows CE 6.x (x86/ARMv4)
- Linux (32/64-bit)

The API is designed for cross-platform use. Therefore software projects can easily ported between platforms with low efforts. For all common programming languages examples are available. Beginning with version 4, PCAN-Basic supports the new CAN FD standard (CAN with Flexible Data Rate) which is primarily characterized by higher bandwidth for data transfer.

More details please refer to: https://www.peak-system.com

3.8.1 Features of PCAN-Basic

- API for developing applications with CAN and CAN FD connection
- Access to the CAN channels of a PCAN-Gateway via the new PCAN-LAN device type
- Supports the operating systems Windows 10, 8.1, 7 (32/64-bit), Windows CE 6.x, and Linux (32/64-bit)
- Multiple PEAK-System applications and your own can be operated on a physical channel at the same time
- Use of a single DLL for all supported hardware types
- Use of up to 16 channels for each hardware unit (depending on the PEAK CAN interface used)
- Simple switching between the channels of a PEAK CAN interface
- Driver-internal buffer for 32,768 messages per CAN channel
- Precision of time stamps on received messages up to 1 µs (depending on the PEAK CAN interface used)
- Supports PEAK-System‘s trace formats version 1.1 and 2.0 (for CAN FD applications)
- Access to specific hardware parameters, such as listen-only mode Notification of the application through Windows events when a message is received
- Extended system for debugging operations
- Multilingual debugging output
- Output language depends on operating systems
- Debugging information can be defined individually
- Thread-safe API

3.8.2 Principle Description of the API

The PCAN-Basic API is the interface between the user application and device driver. In Windows operating systems this is a DLL (Dynamic Link Library).
The sequence of accessing the CAN interface is divided into three phases:
1. Initialization
2. Interaction
3. Completion

Initialization

A channel must be initialized before using it. This is done by the simple call of the function CAN_Initialize for CAN and CAN_InitializeFD for CAN FD. Depending on the type of the CAN hardware, up to 16 CAN channels can be opened at the same time. After a successful initialization the CAN channel is ready. No further configuration steps are required.

Interaction

For receiving and transmitting messages, the functions CAN_Read and CAN_Write as well as CAN_ReadFD and CAN_WriteFD are available. Additional settings can be made, e.g. setting up message filters for specific CAN IDs or the listen-only mode for the CAN controller.

When receiving CAN messages, events are used for an automatic notification of an application (client). This offers the following advantages:

- The application no longer needs to check for received messages periodically (no polling).
- The response time at reception is reduced.

Completion

To end the communication the function CAN_Uninitialize is called in order to release the reserved resources for the CAN channel, among others. In addition the CAN channel is marked as "Free" and is available to other applications.

4 PCANVIEW For Linux (Use AS PCAN)

Our DEMO is for **Ubuntu 18.04 64bits system**, For other System,

Please Refer To: https://www.peak-system.com/fileadmin/media/linux/index.htm

4.1 Driver Install

Step1, Install the Necessary Package First

```
sudo apt-get install gcc
sudo apt-get install g++
sudo apt-get install libpopt-dev
```

Step2, Download Driver form below link, we use V8.13.0
https://www.peak-system.com/fileadmin/media/linux/version-history.html

Download Driver v8.13.0 (tar.gz) | Download Manual (PDF)

Step3, make and install drivers

```
tar -xzf peak-linux-driver-8.13.0.tar.gz
cd peak-linux-driver-8.13.0/
make clean
make
sudo make install
```

4.2 PCAN-View for Linux

Software for Displaying CAN and CAN FD Messages PCAN-V⌯◯◡ ⌯Ⓢ ∠ Ⓢ⌯ⓂⓅⓁ◯ CAN Ⓜ□∅⌯⬀⬀□◎ Ⓢ□//⬀⬀◡∠◎◯ //□◎ ◎◯▱◯⌯⊔⌯∅/∅/ ∠∅⌓ ⬀⬀◎∠∅ⓈⓂ⌯⬀⬀⬀⌯∅/∅/ CAN ∠∅⌓ CAN FD Ⓜ◯ⓈⓈ∠/∅/◯Ⓢ. PCAN-V⌯◯◡ //□◎ L⌯∅—⊽ ⌯Ⓢ ⊥∠Ⓢ◯⌓ □∅ ⬀⬀⬀◯ NC—◎Ⓢ◯Ⓢ Ⓛ⌯⊥◎∠◎▷.

4.2.1System requirements:

T⬀⌯Ⓢ Ⓢ□//⬀⬀◡∠◎◯ ◎◯¢—⌯◎◯Ⓢ ⬀⬀⬀◯ ▱⬀∠◎⌓◯⊔ ⌓◎⌯⊔◯◎. PⓁ◯∠Ⓢ◯ —Ⓢ◯ ⬀⬀⬀◯ D◎⌯⊔◯◎ P∠▱⌒∠/∅/◯ //□◎ P◎□Ⓟ◎⌯◯⬀⬀∠◎▷ P—◎Ⓟ□Ⓢ◯Ⓢ.

4.2.2 Install PCAN-View via repository

Installing software through repository needs first to register the repository only once. Next to the first installation of the software, there is nothing you have to do, except installing available updates when prompted by your system.

S⊿⊿○Ⓟ1, D□∨∅Ⓛ□∠⌓ ∠∅⌓ ⊟∅Ⓢ⊿⊿∠ⓁⓁ ⊿⊿⊿○

//□ⓁⓁ□∨⊟∅/∅/ //⊟Ⓛ○ **peak-system.list** //◎□Ⓜ ⊿⊿⊿○

PEAK−S▷Ⓢ⊿⊿○Ⓜ ∨○⊥Ⓢ⊟⊿⊿○:

∨/∅/○⊿⊿ −¢

⊿⊿⊿⊿Ⓟ://∨∨∨.Ⓟ○∠⌒−Ⓢ▷Ⓢ⊿⊿○Ⓜ.▱□

Ⓜ/⌓○⊥⊟∠∅/⌓⊟Ⓢ⊿⊿Ⓢ/±ⓁⓈ⊥_◎○Ⓛ○∠Ⓢ○

−▱Ⓢ±/Ⓟ○∠⌒−Ⓢ▷Ⓢ⊿⊿○Ⓜ.Ⓛ⊟Ⓢ⊿⊿ −0− |

Ⓢ—⌓□ ⊿⊿○○

/○⊿⊿▱/∠Ⓟ⊿⊿/Ⓢ□—◎▱○Ⓢ.Ⓛ⊟Ⓢ⊿⊿.⌓/Ⓟ○∠

⌒−Ⓢ▷Ⓢ⊿⊿○Ⓜ.Ⓛ⊟Ⓢ⊿⊿

```
innomaker@innomaker:~/Downloads/peak-linux-driver-8.13.0$  wget -q http://w
ww.peak-system.com/debian/dists/`lsb_release -cs`/peak-system.list -O- | su
do tee /etc/apt/sources.list.d/peak-system.list

deb http://www.peak-system.com/debian bionic non-free
#deb-src http://www.peak-system.com/debian bionic non-free
```

N□⊿⊿○: I// ⊿⊿⊿○ **lsb_release** ⊿⊿□□Ⓛ ⊟Ⓢ ∅□⊿⊿

⊟∅Ⓢ⊿⊿∠ⓁⓁ○⌓ □∅ ▷□—◎ L⊟∅—↧

Ⓢ▷Ⓢ⊿⊿○Ⓜ ⊿⊿⊿○∅ ◎○ⓅⓁ∠▱○ **`lsb_release**

-cs` ⊥▷ ⊿⊿⊿○ ∅∠Ⓜ○ □// ▷□—◎ L⊟∅—↧

⌓⊟Ⓢ⊿⊿◎⊟⊥—⊿⊿⊟□∅. F□◎ ○↧∠ⓂⓅⓁ○:

∨/∅/○⊿⊿ −¢

⊿⊿⊿⊿Ⓟ://∨∨∨.Ⓟ○∠⌒−Ⓢ▷Ⓢ⊿⊿○Ⓜ.▱□

Ⓜ/⌓○⊥⊟∠∅/⌓⊟Ⓢ⊿⊿Ⓢ/∨⊿○○▷▷/Ⓟ○

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
innomaker@innomaker:~/Downloads/peak-linux-driver-8.13.0$ wget -q http://ww
w.peak-system.com/debian/peak-system-public-key.asc -O- | sudo apt-key add
-
OK
```

[illegible]

[illegible]

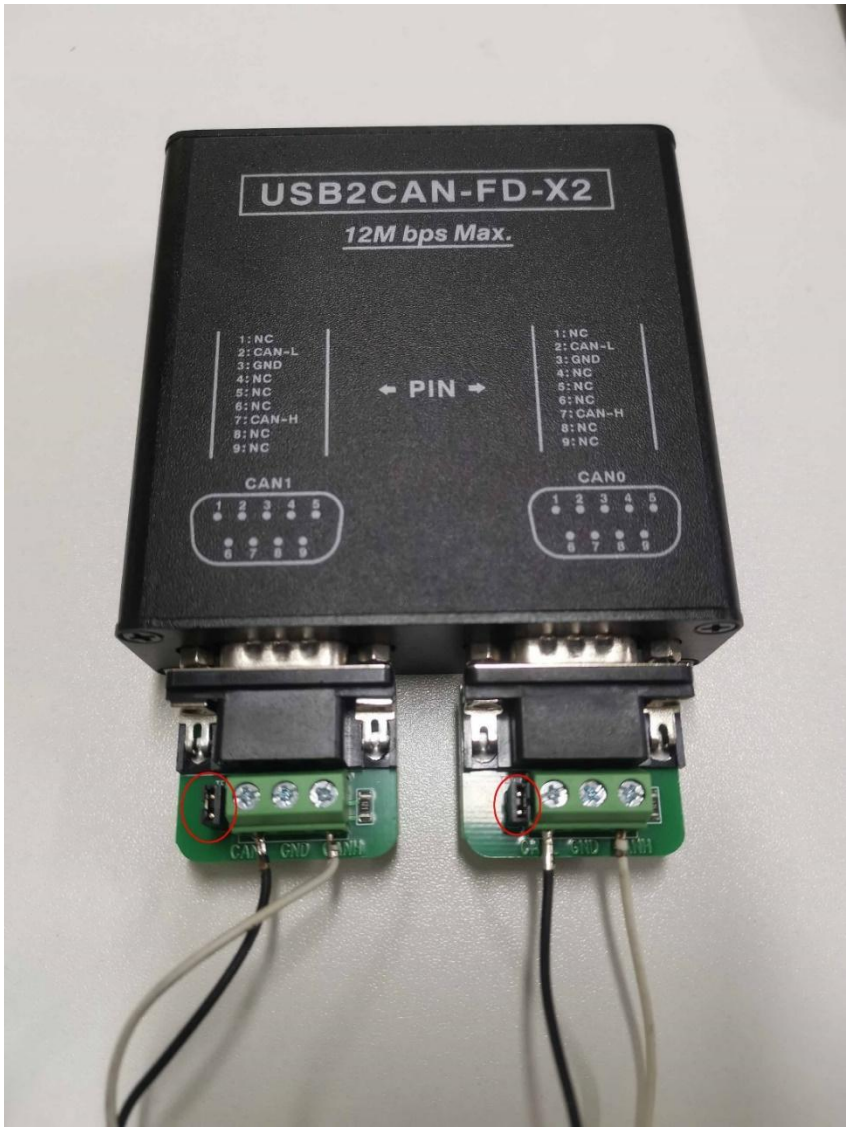
[illegible]

```
innomaker@innomaker:~/Downloads/peak-linux-driver-8.13.0$ sudo apt-get install pcanview-ncurses
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libegl1-mesa libwayland-egl1-mesa
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
 Terminal ew-ncurses
0 upgraded, 1 newly installed, 0 to remove and 8 not upgraded.
Need to get 80.7 kB of archives.
After this operation, 169 kB of additional disk space will be used.
Get:1 http://www.peak-system.com/debian bionic/non-free amd64 pcanview-ncurses amd64 0.9.1-0 [80.7 kB]
Fetched 80.7 kB in 1s (66.4 kB/s)
Selecting previously unselected package pcanview-ncurses:amd64.
(Reading database ... 170478 files and directories currently installed.)
Preparing to unpack .../pcanview-ncurses_0.9.1-0_amd64.deb ...
Unpacking pcanview-ncurses:amd64 (0.9.1-0) ...
Setting up pcanview-ncurses:amd64 (0.9.1-0) ...
```

4.3 Transmit/Received Data

Step1，

Connect hardware to your pc As below, please add on the jumper for 120Ω jumper.

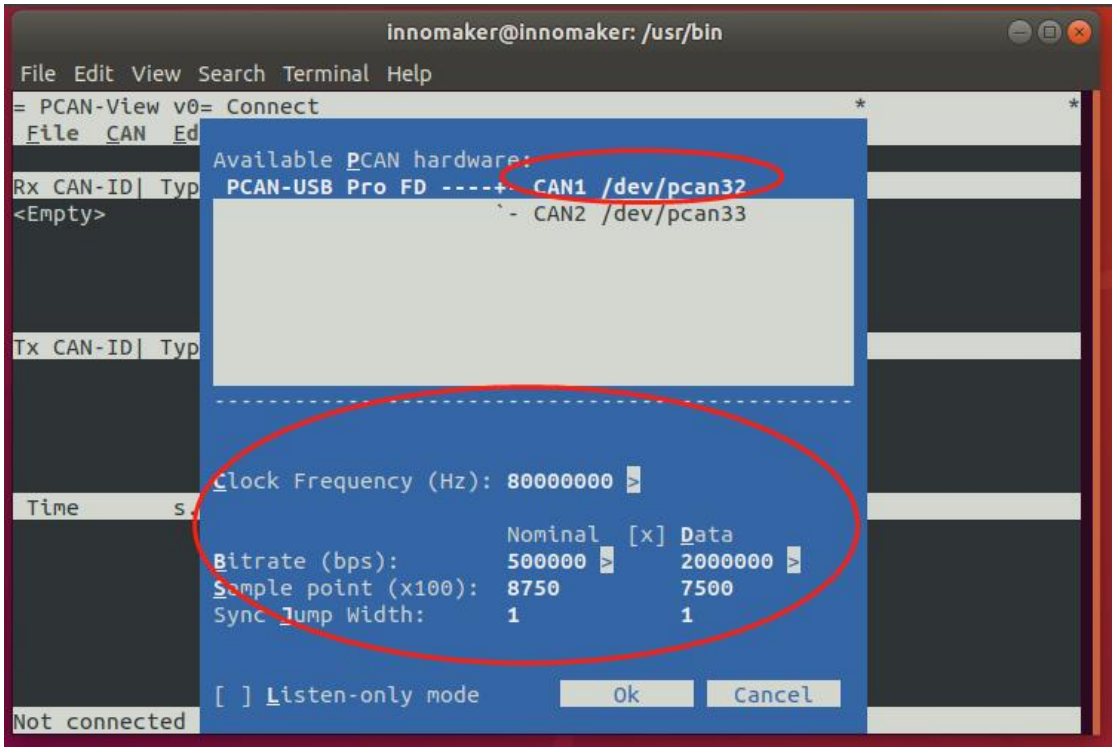


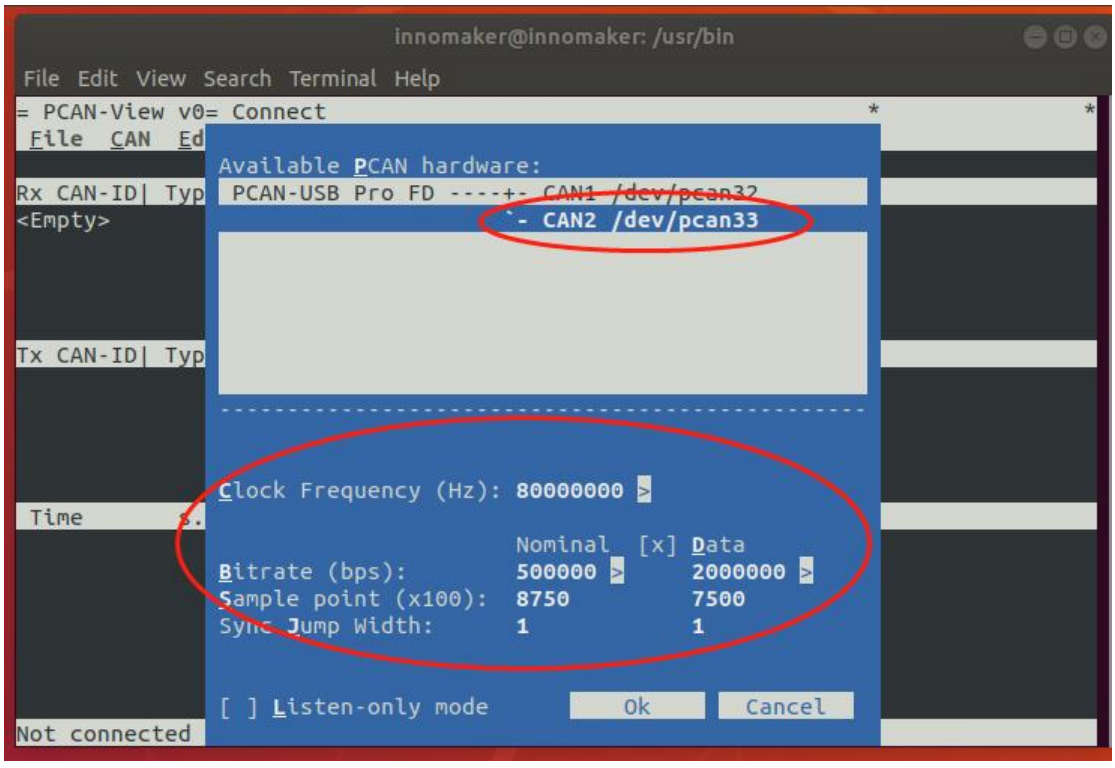
,

Step2，Open 2 termination window ,One for can0, One for can1

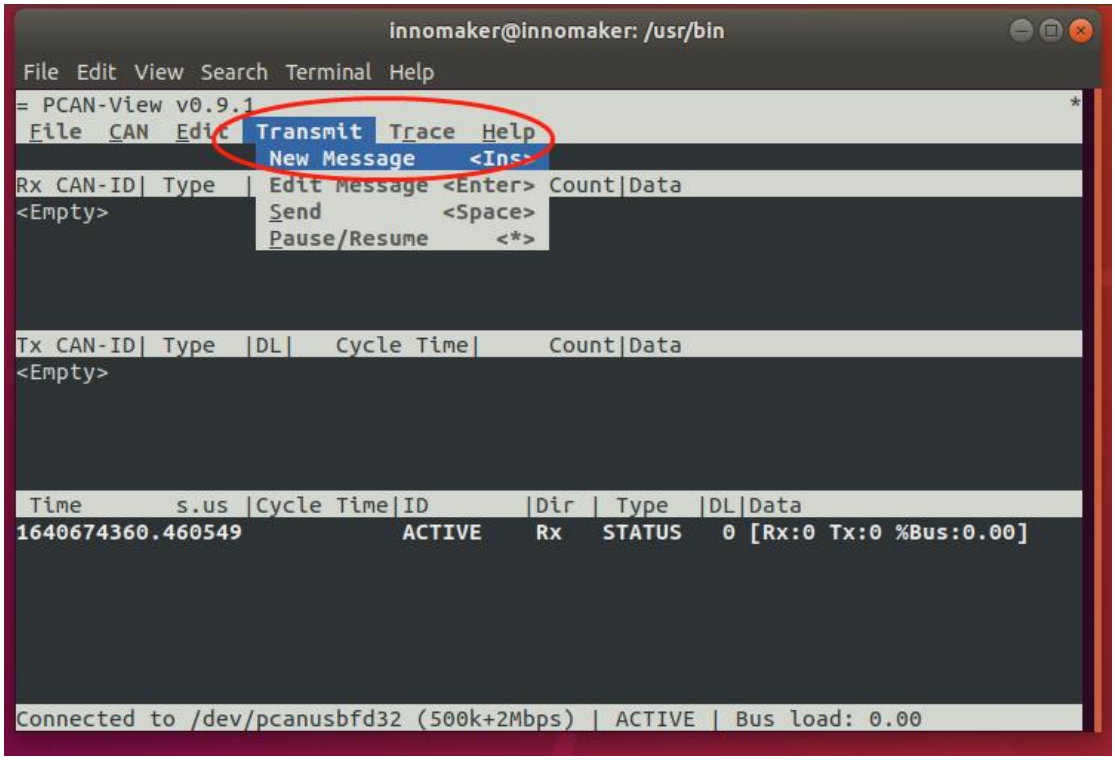
```
cd /usr/bin
./pcanview
```

choose the same Nominal/Data Bitrate

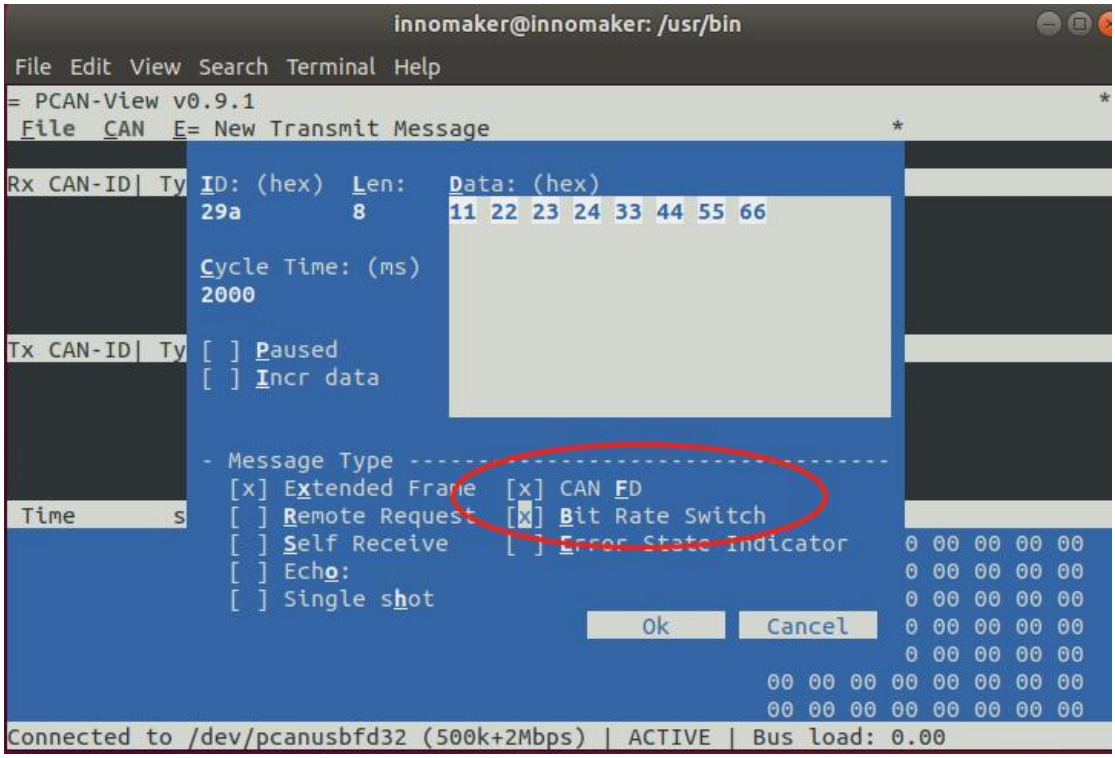




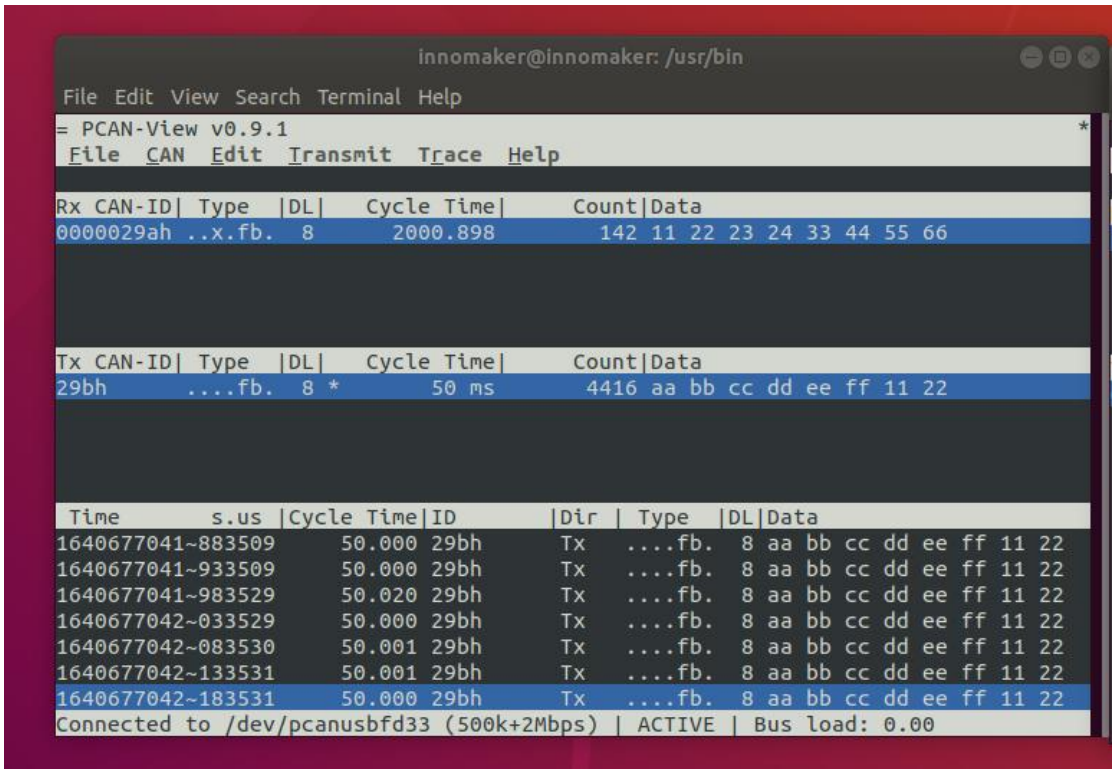
Step3，Create New Message for can0 and can1



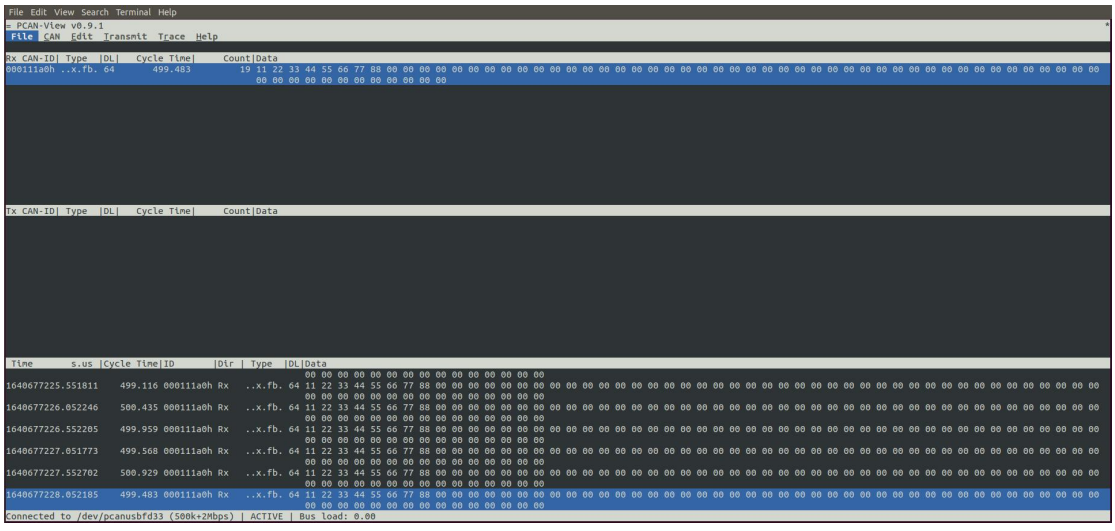
Remember choose bit rate switch



Effect: (Note:)



To show DATA Length 64bit, Must Maxmum Preview window



5 CAN-UTILS/C/Python For Linux (USE AS SOCKET CAN)

5.1 Linux Support List

USB2CANFD-X2 device can run properly without any additional driver request on all Linux

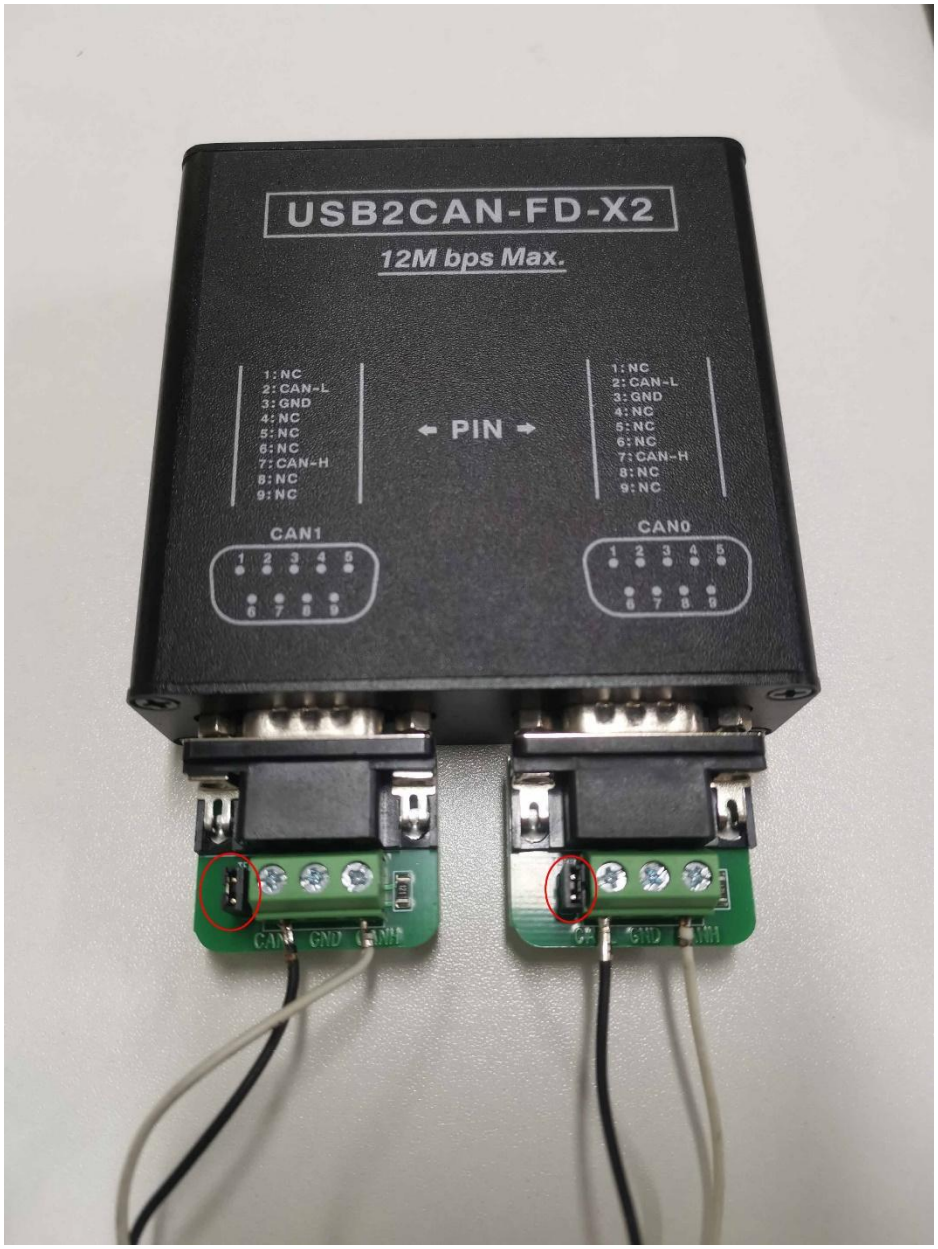
system as below.

	amd64	i386	arm64	armhf	ppc64el
Ubuntu:					
Trusty 14.04 LTS	■	■	■	■	
Xenial 16.04 LTS	■	■	■	■	■
Bionic 18.04 LTS	■	■	■	■	■
Cosmic 18.10	■	■	■	■	
Disco 19.04	■	■	■	■	
Eoan 19.10	■	■	■	■	
Focal 20.04 LTS	■	■	■	■	■
Groovy 20.10	■	■	■	■	■
Hirsute 21.04	■	■	■	■	■
OpenSUSE Tumbleweed	see Xenial				
Debian:					
Wheezy 7.11	■	■	■	■	
Jessie 8.11	■	■	■	■	
Stretch 9.9	■	■	■	■	
Buster 10	■	■	■	■	■
Bullseye 11	■	■	■	■	■

5.2 Hardware Connection

Connect device to your Linux computer As below picture and follow chapter 2.2 to activated 120Ω resistor by hardware, use the 2pcs db9 to termination board we provide and **put on jumper in red circle**.

CAN 0 Channel	Connection	CAN 1 Channel
CAN_L(pin 2)	----------	CAN_L(pin 2)
CAN_H(pin 7)	----------	CAN_H(pin 7)



LED Indication should be as below picture:

5.3 CAN-UTILS DEMO

5.3.1 Prepare

Type command ifconfig -a to check 'can0' and 'can1'device is available in system, if you can not find the command ifconfig, use command sudo apt-get install net-tools

```
/irtual-machine:~$ ifconfig -a
can0: flags=128<NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

can1: flags=128<NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Type command dmesg to check more information

```
[ 4334.851804] usb 1-1: new high-speed USB device number 6 using ehci-pci
[ 4335.120375] usb 1-1: New USB device found, idVendor=0c72, idProduct=0011, bcdDevice= 0.00
[ 4335.120380] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 4335.120384] usb 1-1: Product: PCAN-USB Pro FD
[ 4335.120385] usb 1-1: Manufacturer: PEAK-System Technik GmbH
[ 4335.137143] peak_usb 1-1:1.0: PEAK-System PCAN-USB Pro FD v2 fw v3.2.0 (2 channels)
[ 4335.143995] peak_usb 1-1:1.0 can0: attached to PCAN-USB Pro FD channel 0 (device 0)
[ 4335.151388] peak_usb 1-1:1.0 can1: attached to PCAN-USB Pro FD channel 1 (device 0)
```

Type command sudo apt-get install can-utils to install can-utils.

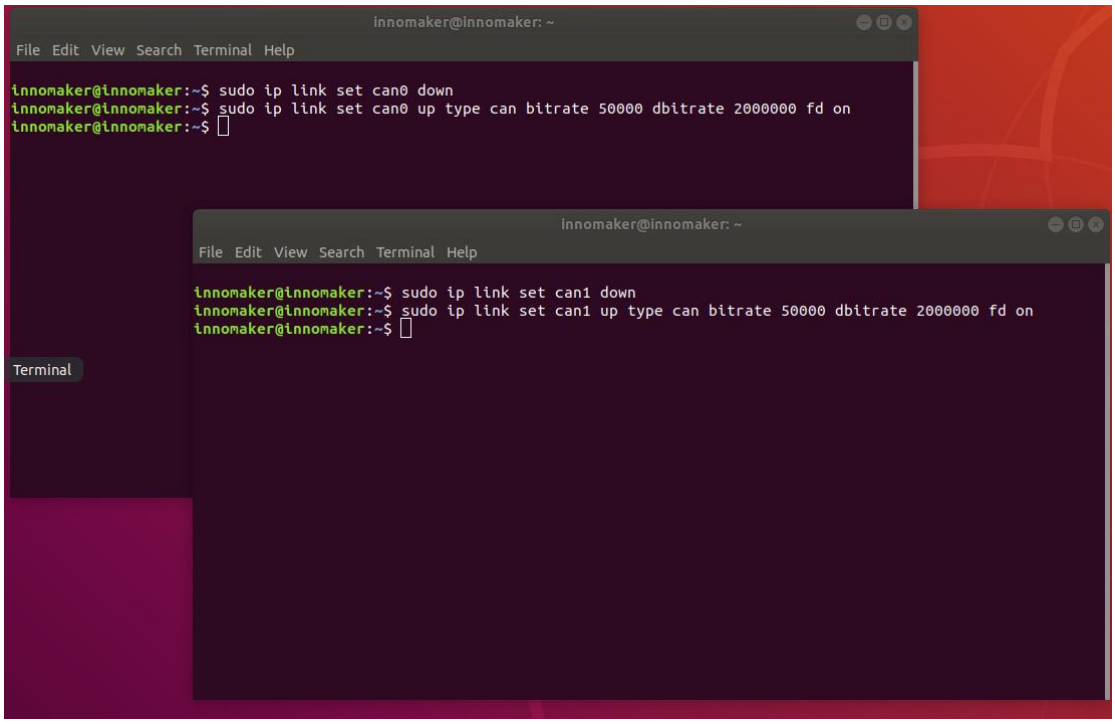
Note:
This tool is a very easy way to test USB2CANFD-X2 module communication. There is only a simple use instruction. For more details, please refer to can-utils user manual and source code. https://github.com/linux-can/can-utils/

5.3.2 Send/Receive

Initialize CAN port, Open two termination command for can0 and can1.
sudo ip link set can0 down
sudo ip link set can0 up type can bitrate 50000 dbitrate 2000000 fd on

sudo ip link set can1 down

sudo ip link set can1 up type can bitrate 50000 dbitrate 2000000 fd on

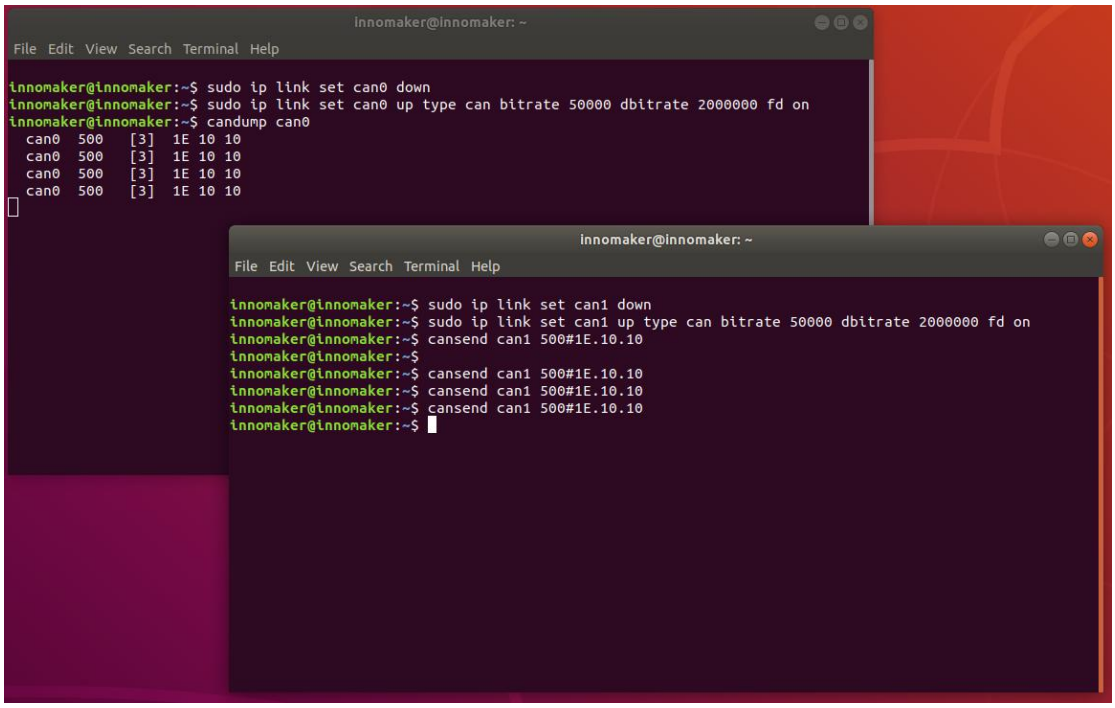


<1>Set can0 as receiver

candump can0

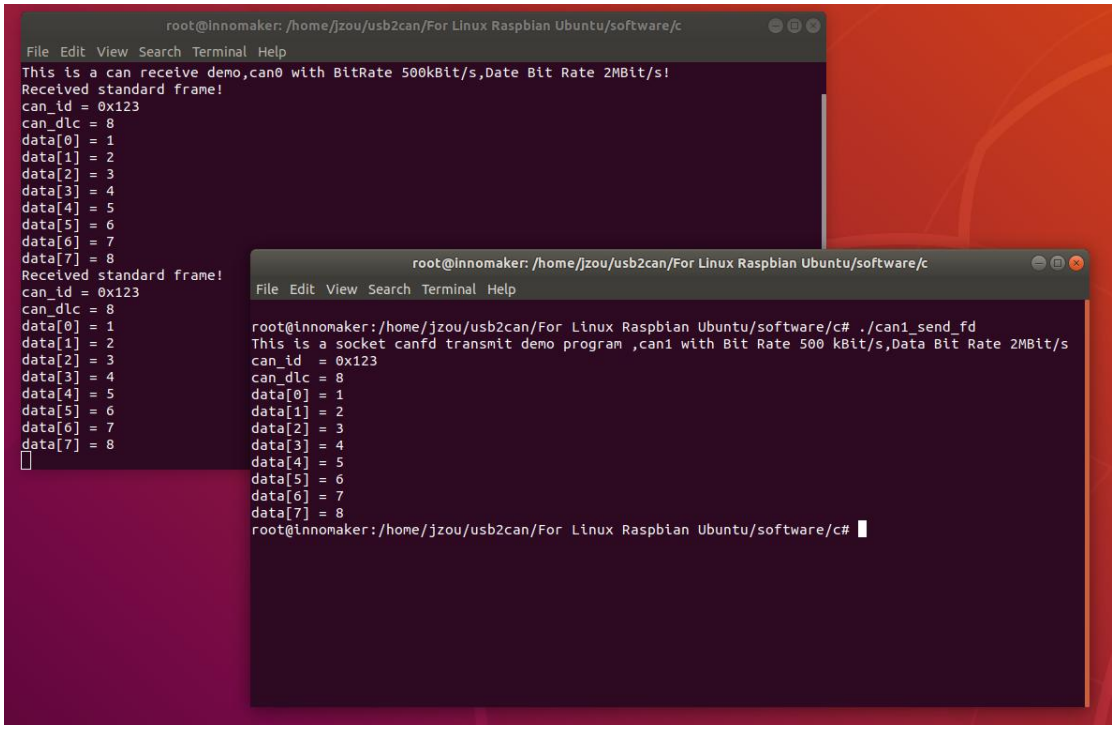
<2>Set can1 as sender

cansend can1 500#1E.10.10



5.4 C Demo

<1>Send CAN0 As Receiver,
sudo ./can0_receive_fd
<2>Set CAN1 As Sender
sudo ./can1_send_fd



5.5 Python3 Demo

(1) Check the Python version of your Raspbian. Python 3.7.3 default in 2019-09-26-Raspbian.img. Our Demo can run on any Python3 version.
python3 -V

(2) If you can't find the Python3 in system. Install the Python3
sudo apt-get install python3-pip
(3) Install Python CAN library.
sudo pip3 install python-can

(4) Set CAN0 as receiver
sudo python3 receive.py

(5) Set CAN1 as sender
sudo python3 send.py

5.6 Software Description

Now with previous demo's code to show you how to program socket can in Raspbian with C and Python . The socket can is an implementation of CAN protocols(Controller Area Network) for Linux. CAN is a networking technology which has widespread use in automation, embedded devices, and automotive fields. While there have been other CAN implementations for Linux based on character devices, Socket CAN uses the Berkeley socket API, the Linux network stack and implements the CAN device drivers as network interfaces. The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow programmers, familiar with network programming, to easily learn how to use CAN sockets. For more Socket CAN detail please refer to below link:
https://www.kernel.org/doc/Documentation/networking/can.txt
https://elinux.org/CAN_Bus

5.6.1 Programming in C

For Sender's codes

(1): Create the socket, If an error occurs then the return result is -1.

```
/*Create socket*/
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
if (s < 0) {
    perror("Create socket PF_CAN failed");
    return 1;
}
```

(2): Locate the interface to "can0" or other name you wish to use. The name will show when you execute "./ifconfig –a".

```
/*Specify can0 device*/
strcpy(ifr.ifr_name, "can0");
ret = ioctl(s, SIOCGIFINDEX, &ifr);
if (ret < 0) {
    perror("ioctl interface index failed!");
    return 1;
}
```

(3): Bind the socket to “can0”.

```
/*Bind the socket to can0*/
addr.can_family = PF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ret = bind(s, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0) {
    perror("bind failed");
    return 1;
}
```

(4): Disable sender’s filtering rules,this program only send message do not receive packets.

```
/*Disable filtering rules,this program only send message do not receive packets */
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
```

(5): Assembly data to send.

```
/*assembly  message data! */
frame.can_id = 0x123;
frame.can_dlc = 8;
frame.data[0] = 1;
frame.data[1] = 2;
frame.data[2] = 3;
frame.data[3] = 4;
frame.data[4] = 5;
frame.data[5] = 6;
frame.data[6] = 7;
frame.data[7] = 8;
//if(frame.can_id&CAN_EFF_FLAG==0)
if(!(frame.can_id&CAN_EFF_FLAG))
    printf("Transmit standard frame!\n");
else
    printf("Transmit extended frame!\n");
```

(6): Send message to the can bus.You can use the return value of write() to check whether all data has been sent successfully .

```
/*Send message out */
nbytes = write(s, &frame, sizeof(frame));
if(nbytes != sizeof(frame)) {
    printf("Send  frame incompletely!\r\n");
    system("sudo ifconfig can0 down");
}
```

(7): Close can0 device and disable socket.

```
/*Close can0 device and destroy socket!*/
close(s);
```

For Receiver's codes

(1)step 1 and (2) is same as Sender's code.
(3):It's different from Sender's.

```
/*Bind the socket to can0*/
addr.can_family = PF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ret = bind(s, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0) {
    perror("bind failed");
    return 1;
}
```

(4): Define receive filter rules,we can set more than one filters rule.

```
/*Define receive filter rules,we can set more than one filter rule!*/
struct can_filter rfilter[2];
rfilter[0].can_id = 0x123;//Standard frame id !
rfilter[0].can_mask = CAN_SFF_MASK;
rfilter[1].can_id = 0x12345678;//extend frame id!
rfilter[1].can_mask = CAN_EFF_MASK;
```

(5): Read data back from can bus.

```
nbytes = read(s, &frame, sizeof(frame));
```

5.6.2 Programming in Python

Import

import os
The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux. We usually use os.system() function to execute a shell command to set CAN.

import can
The python-can library provides Controller Area Network support for Python, providing common abstractions to different hardware devices, and a suite of utilities for sending and receiving messages on a CAN bus.
For more information about python-can, please to below link:
https://python-can.readthedocs.io/en/stable/index.html

ifconfig
If you are use Ubuntu system, It may can't use the 'ifconfig' command. Please install the net tools.

```
sudo apt install net-tools
```

Simple common functions

(1) Set bitrate and start up CAN device.

```
os.system('sudo ip link set can0 type can bitrate 1000000')
os.system('sudo ifconfig can0 up')
```

(2) Bind the socket to 'can0'.

```
can0 = can.interface.Bus(channel = 'can0', bustype = 'socketcan_ctypes')
```

(3) Assembly data to send.

```
msg = can.Message(arbitration_id=0x123, data=[0, 1, 2, 3], extended_id=False)
```

(4) Send data.

```
can0.send(msg)
```

(5) Receive data.

```
msg = can0.recv(30.0)
```

(6) Close CAN device

```
os.system('sudo ifconfig can0 down')
```

5.6.3 Error Frame

You may receive some error frame marked in red when you use the USB2CANX2-FD module. They will tell you what problem does the USB2CANX2-FD module meet on your CAN Bus.

Some people would say why didn't they meet the error frame with other tool or USB to CAN module before. The truth is that most of the tool filter out the error frame to avoid controversy and support. They just show nothing when there are some error on the CAN Bus. We want to show the all raw data to help you to analyze your CAN BUS. Some error can be ignored, but some error maybe the hidden danger for your CAN BUS.

For the error frame ID description, please refer to below link:

https://github.com/linux-can/can-utils/blob/master/include/linux/can/error.h

Now we take a simple case to show you how to analyze the error frame ID. I made the incorrect connection between the USB2CAN module and the CAN Bus, to see what happens.

CAN 0 Channel	ERRO Connection	CAN 1 Channel
CAN_L(pin 2)	----------	CAN_H(pin 7)
CAN_H(pin 7)	----------	CAN_L(pin 2)

SeqID	SystemTime	Channel	Direc...	FrameId	Frame...	Frame...	Length	FrameData
4	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
5	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
6	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
7	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
8	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
9	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
10	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
11	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 00 00 00 00 00 00 00
12	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 0C 00 00 00 00 00 00
13	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 0C 00 00 00 00 00 00
14	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 0C 00 00 00 00 00 00
15	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 0C 00 00 00 00 00 00
16	2020/6/29 14:44:08	0	Recv	0x20000024	Data ...	Stand...	8	0x|00 30 00 00 00 00 00 00

As Above, We received error frame Id: 0x20000024 and 2 set of 8 byte Frame Data:
data[0]=0x00, data[1]=0x0C,data[3] to data[7] are all 0x00 .
data[0]=0x00, data[1]=0x30,data[3] to data[7] are all 0x00 .

According the above error frame ID description link:

```
/* error class (mask) in can_id */
#define CAN_ERR_TX_TIMEOUT   0x00000001U /* TX timeout (by netdevice driver) */
#define CAN_ERR_LOSTARB      0x00000002U /* lost arbitration    / data[0]    */
#define CAN_ERR_CRTL         0x00000004U /* controller problems / data[1]    */
#define CAN_ERR_PROT         0x00000008U /* protocol violations / data[2..3] */
#define CAN_ERR_TRX          0x00000010U /* transceiver status  / data[4]    */
#define CAN_ERR_ACK          0x00000020U /* received no ACK on transmission */
#define CAN_ERR_BUSOFF       0x00000040U /* bus off */
#define CAN_ERR_BUSERROR     0x00000080U /* bus error (may flood!) */
#define CAN_ERR_RESTARTED    0x00000100U /* controller restarted */
```

This Error frame ID = 0x200000000 | 0x00000020|0x00000004
= 0x200000000 | CAN_ERR_ACK|CAN_ERR_CRTL

So the USB2CANX2-FD meet two problem ‘received no ACK on transmission’ and ‘controller problems’.

For problem ‘received no ACK on transmission’ may case by the not CAN-BUS or other module on the CAN BUS are only listen mode(No ACK).

For problem 'controller problems', refer to the data[1] description:

```
/* error status of CAN-controller / data[1] */
#define CAN_ERR_CRTL_UNSPEC      0x00 /* unspecified */
#define CAN_ERR_CRTL_RX_OVERFLOW 0x01 /* RX buffer overflow */
#define CAN_ERR_CRTL_TX_OVERFLOW 0x02 /* TX buffer overflow */
#define CAN_ERR_CRTL_RX_WARNING  0x04 /* reached warning level for RX errors */
#define CAN_ERR_CRTL_TX_WARNING  0x08 /* reached warning level for TX errors */
#define CAN_ERR_CRTL_RX_PASSIVE  0x10 /* reached error passive status RX */
#define CAN_ERR_CRTL_TX_PASSIVE  0x20 /* reached error passive status TX */
                                      /* (at least one error counter exceeds */
                                      /* the protocol-defined level of 127)  */
#define CAN_ERR_CRTL_ACTIVE      0x40 /* recovered to error active state */
```

data[1] = 0x0C = 0x04|0x08 = CAN_ERR_CRTL_RX_WARNING|CAN_ERR_CRTL_TX_WARNING
It means the USB2CAN module can't send/receive data properly and reached warning level.

data[1] = 0x30 = 0x10|0x20 = CAN_ERR_CRTL_RX_PASSIVE | CAN_ERR_CRTL_TX_PASSIVE
It means the USB2CAN module can't send/receive data too much, USB2CAN module into error status.

Summing up the above, the error frame tell us, USB2CAN module can't get ACK from CAN BUS and can't send data to the CAN Bus. So the CAN Bus may not inexistence or the connection error.