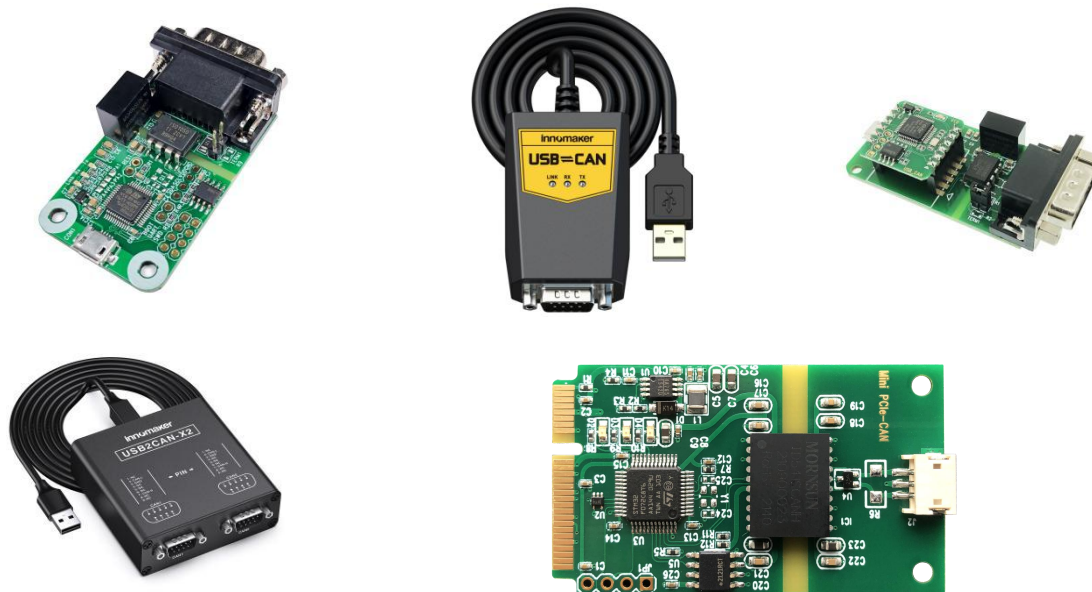


Innomaker USB2CAN Device UserManual



Menu

Innomaker USB2CAN Device UserManual	1
1. General Description:	4
2. Features	5
3. Technical Specification	6
3.1 USB2CAN modue and USB2CAN-Dev	6
3.2 USB2CAN-DEV	7
3.3 USB2CAN-C	8
3.4 USB2CAN-X2	9
3.5 MINIPCI-E-CAN	10
4. Hardware Description	11
4.1 USB2CAN modue and USB2CAN-Dev	11
4.2 USB2CAN-C	13
4.3 USB2CAN-X2	15
4.4 MINIPCI-E-CAN	19
5. Windows	22
5.1 The Directory Structure Of The WINDOWS folder	22
5.2 Windows Driver	23
6. Windows/Mac OS Ready-Made Tools	25
5.1 Open the USB2CAN tools	25
5.2 Scan For Devices	25
5.3 Setting The BandRate And Working Mode	27
5.4 Enable Device	28
5.5 Send/Receive Data	28
5.6 Continuous sending / Sending In Large Quantities	29
5.7 Windows Real Time Setting	29
5.8 Mac Os Double Open The USB2CAN Tool	30
7. BUSMASTER On Windows	31
7.1 Download and Install	31
7.2 Using The Software	34
8. Mac Os	36
8.1 Library	37
8.2 Document	37
9. Linux/Ubuntu/Raspbian	38
9.1 Run Linux Test Demo	38
9.1.1 Preparatory work	39
9.1.2 Run can-utils Tool	40
9.1.3 Run C Demo	42
9.1.4 Run Python3 Demo	43
9.2 Software Description	44
8.2.1 Programming in C	45
8.2.2 Programming in Python3	47
Support: support@inno-maker.com https://github.com/INNO-MAKER/usb2can	2
Bulk Price: sales@inno-maker.com	

10. Error Frame	49
11. User Manual Version Descriptions	51

1. General Description:

USB2CAN device is a 'plug and play' and bi-directional port powered USB/PCIE to CAN converter which realizes long-distance communication between your Computer/SBC/PC/laptop and other devices stably though CAN- Bus connection. With small size and convenient operation, It's a cost-effective solution that are safe and reliable for all your data-conversion / device-protection applications for any experienced engineer interfacing to expensive industrial equipment yet simple enough for home use by an amateur hobbyist.

Support Linux system , It's a socket-can device in Linux, not need to install any driver and fully compatible with other socket-can software in Linux such as can-utils.

Support Mac OS version equal or above 10.11 and provide development library for help customer develop own applications.

Support Win7/Win8/Win10 and provide C#/C++/Python demo and dynamic link libraries for help customer develop own applications.

USB2CAN can also be applied to obtain the data of car via the OBD connector, but you need to configured and secondary development by yourself.

We provide five kinds of USB2CAN device, the hardware appearance is different but the software ,firmware features are 100% same. Therefore, when describing the software usage, we will collectively refer to it as the USB2CAN Device or USB2CAN.



USB2CAN Module



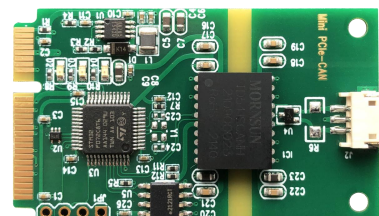
USB2CAN-C



USB2CAN-DEV



USB2CAN-X2




MINIPCIE-CAN

2. Features


1. The USB/MINIPCIE to CAN Converter Module compatible with all series Raspberry Pi/Jetson Nano/Tinker Board/any single board computer, desktop and laptop.
2. The USB CAN Module support Windows System, Mac OS and Linux, Raspbian(v5.4 kernel), Ubuntu.
3. Plug and Play USB CAN device. No external power required. Support wider CAN baud rate, from 20Kbps to 1Mbps can be programmed arbitrarily. Support for CAN bus 2.0A and 2.0B specification.
4. Provides 3000V/5000V voltage isolation and 2500V ESD isolated protection on signal pins. 120 Ohm resistor selectable jumper feature.
5. Provide C/Python and source code with Socket-CAN for Raspbian(Linux), Dynamic library and demo with IOUSBKit for Mac Os(Big Sur).

3. Technical Specification


3.1 USB2CAN module and USB2CAN-Dev

Product Part Number: USB2CAN Module	Features	USB2CAN PCBA, High performance cost and easy for install in your product shell
	CAN Port	D-SUB, 9 pins
	USB Port	USB 2.0 Full-Speed
	Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN
	CAN-BUS Baud rate	From 20kbps to 1Mbps can be programmed arbitrarily.
	Isolation Voltage	1.5K VDC/min, 3K VDC/1s
	Microcontroller	STM32F0
	Termination	120 Ohm resistor selectable jumper
	CAN Transceiver	ISO1050DUBR ,Texas Instruments
	Work Temperature	-40° ~ 85
	Power Consumption	5V/40mA = 0.2W
	Weight	15.5 g
	Certificate	CE / FCC SDOC
	Relative humidity	15-90%, not condensing


3.2 USB2CAN-DEV

Product Part Number: USB2CAN-DEV	Features	USB2CAN stamp-core module development board. You could use the core module in your circuit design to add USB to CAN function.
	CAN Port	D-SUB, 9 pins
	USB Port	USB 2.0 Full-Speed
	Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN
	CAN-BUS Baud rate	From 20kbps to 1Mbps can be programmed arbitrarily.
	Isolation Voltage	1.5K VDC/min, 3K VDC/1s
	Microcontroller	STM32F0
	Termination	120 Ohm resistor selectable jumper
	CAN Transceiver	ISO1050DUBR ,Texas Instruments
	Work Temperature	-40° ~ 85
	Power Consumption	5V/40mA = 0.2W
	Weight	15.5 g
	Certificate	USB2CAN-Core stamp-core module get CE / FCC SDOC/ UKAC/ICES
	Relative humidity	15-90%, not condensing

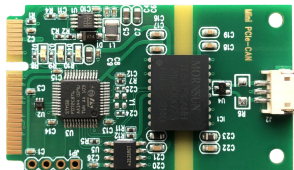
3.3 USB2CAN-C

Product Part Number: USB2CAN-C	Features:	USB2CAN module with shell and USB cable. Safer and more flexible.
	CAN Port	D-SUB, 9 pins
	USB Port	USB 2.0 Full-Speed
	Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN
	CAN-BUS Baud rate	From 20kbps to 1Mbps can be programmed arbitrarily.
	Isolation Voltage	1.5K VDC/min, 3K VDC/1s
	Microcontroller	STM32F0
	Termination	120 Ohm resistor by external added only
	CAN Transceiver	ISO1050DUBR ,Texas Instruments
	Work Temperature	-40° ~ 85
	Power Consumption	5V/40mA = 0.2W
	Weight	50 g
	Certificate	CE / FCC SDOC/ UKAC/ICES
	Relative humidity	15-90%, not condensing

3.4 USB2CAN-X2

Product Part Number: USB2CAN-X2	Features	Dual channels CAN via one USB ports only, Can be set up and communication independently. Higher voltage isolation and more sturdy metal casing.
	CAN Port	D-SUB, 9 pins x 2
	USB Port	USB 2.0 Full-Speed
	Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN Dual channels can be set up independently
		CAN-BUS Baud rate From 20kbps to 1Mbps can be programmed arbitrarily.
	Isolation Voltage	Bus-Pin ESD protection up to 15kV(HBM) High isolation up to 5000 VDC/1s
	Microcontroller	STM32F0
	Termination	120 Ohm resistor build-in setting and external added
	CAN Transceiver	TD541SCANH ,MORNSUN
	Work Temperature	-40° ~ 85
	Power Consumption	5V/100mA = 0.5W
	Weight	157 g
	Certificate	CE / FCC SDOC/ UKAC/ICES
	Relative humidity	15-90%, not condensing

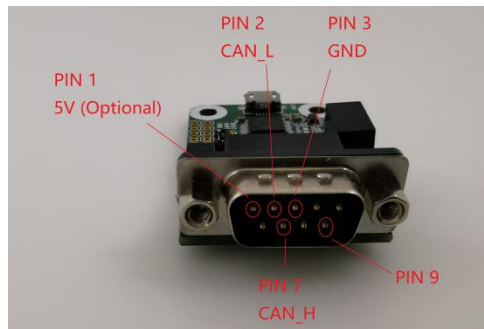
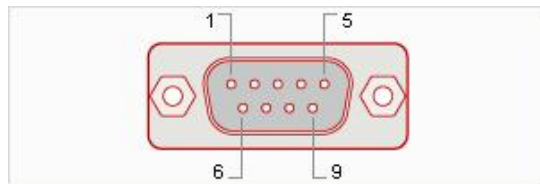
3.5 MINIPCIE-CAN

Product Part Number: MINIPCIE-CAN	Features	Standard MINI PCIE port to CAN bus. Fully suitable for the MINI PCIE port of PC and ARM chip.
	CAN Port	3 pins 1.5mm separation distance
	PCIE Port	Full-Mini PCIE , 52Pins
	Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN
	CAN-BUS Baud rate	From 20kbps to 1Mbps can be programmed arbitrarily.
	Isolation Voltage	Bus-Pin ESD protection up to 15kV(HBM) High isolation up to 5000 VDC/1s
	Microcontroller	STM32F0
	Termination	120 Ohm resistor build-in setting and external added
	CAN Transceiver	TD541SCANH ,MORNSUN
	Power Consumption	3.3V/35mA = 0.115W
	Work Temperature	-40° ~ 85
	Weight	15 g
	Relative humidity	15-90%, not condensing

4. Hardware Description

4.1 USB2CAN module and USB2CAN-Dev

(1) CAN Connector Pinout



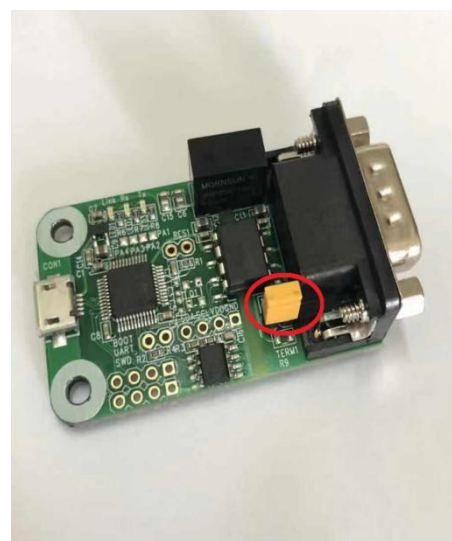
Pin	Description
1	5V/150ma output . Weld 0 Ω resistor on R9 to enable this function(close to the jumper).
2	CANL bus line (dominant low)
3	CAN_GND
4	NC
5	NC
7	CANH bus line (dominant high)
8	NC
9	NC

(2) 120 Ohm Resistor Setting.

A High-speed CAN bus (ISO 11898-2) must be terminated on both ends with 120 Ohms. There are three ways to enable the 120 Ω resistance.

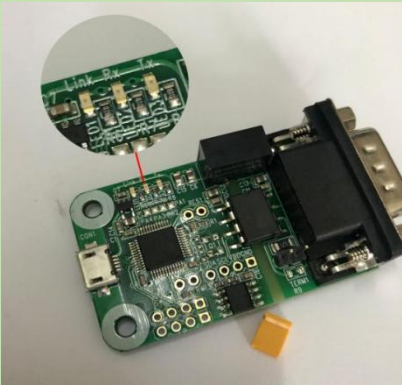


Disable 120 Ohm Resistor

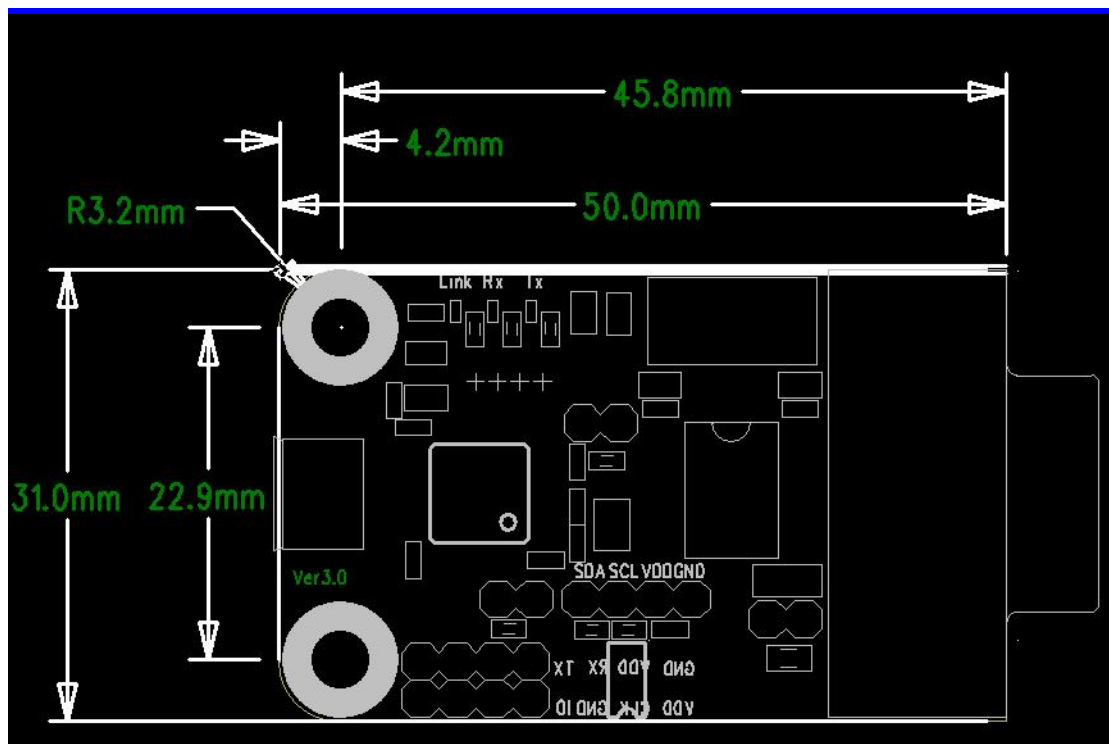


Enable 120 Ohm Resistor

(3) LED Indicate

	LED Name	Description
	Link	Red led is normally on to indicate The module is came up without any errors
	Tx	Red led flash to indicate send data.
	Rx	Red led flash to indicate receive data.

(4) Dimension Figure



4.2 USB2CAN-C

(1) CAN Connector Pinout

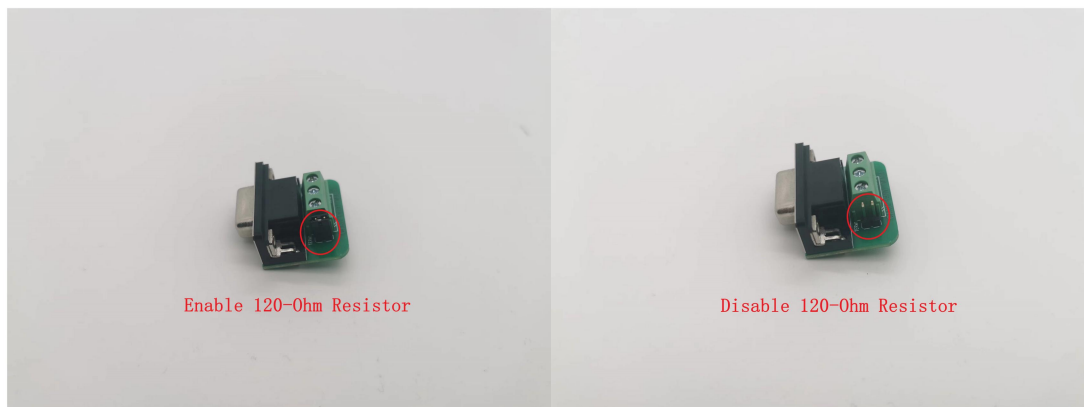


Pin	Description
1	NC
2	CAN_L bus line (dominant low)
3	CAN_GND
4	NC
5	NC
7	CAN_H bus line (dominant high)
8	NC
9	NC


(2) 120 Ohm Resistor Setting.

A High-speed CAN bus (ISO 11898-2) must be terminated on both ends with 120 Ohms. There are three ways to enable the 120 Ω resistance.

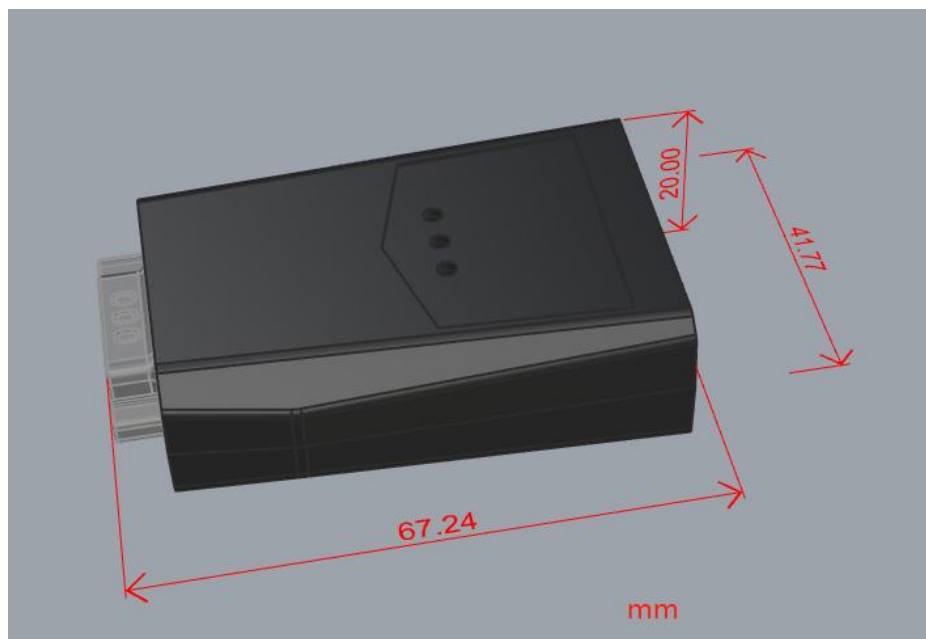
There is a DB9 to 3pins terminal comes with goods. they have the 120 Ω resistance selectable feature.



(3) LED Indicate

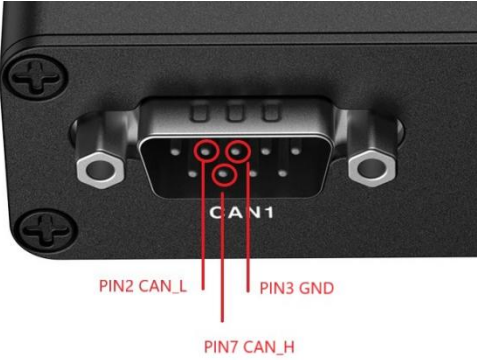
Product	LED Name	Description
	Link	Green led is normally on to indicate the module is came up with out any errors
	Tx	Green led is normally on to indicate the usb2can is Green led flash to indicate send data.
	Rx	Green led flash to indicate receive data.

(4) Dimension Figure

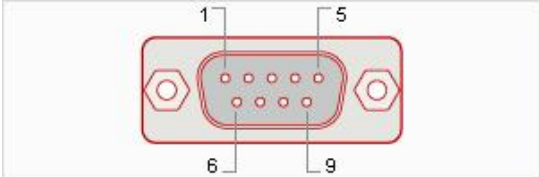


4.3 USB2CAN-X2

(1) CAN Connector Pinout



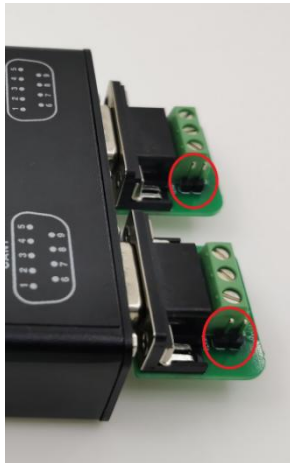
Pin	Description
1	NC
2	CANL bus line (dominant low)
3	CAN_GND
4	NC
5	NC
6	NC
7	CANH bus line (dominant high)
8	NC
9	NC



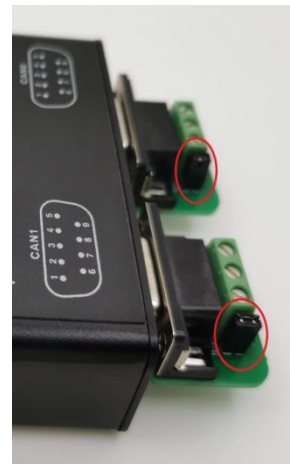
(2) 120 Ohm Resistor Setting.

A High-speed CAN bus (ISO 11898-2) must be terminated on both ends with 120 Ohms. There are three ways to enable the 120 Ω resistance.

There are two DB9 to 3pins terminal comes with goods. they have the 120 Ω resistance selectable feature.

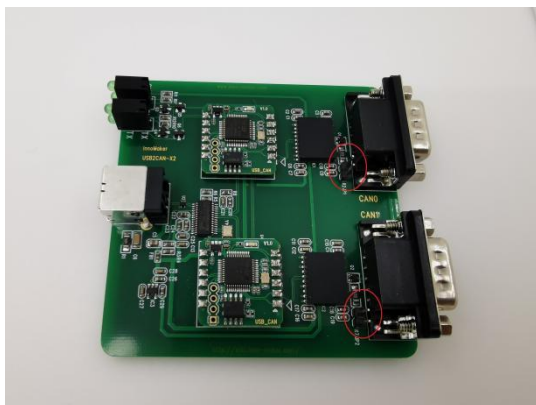


Disable the 120 Ω resistance

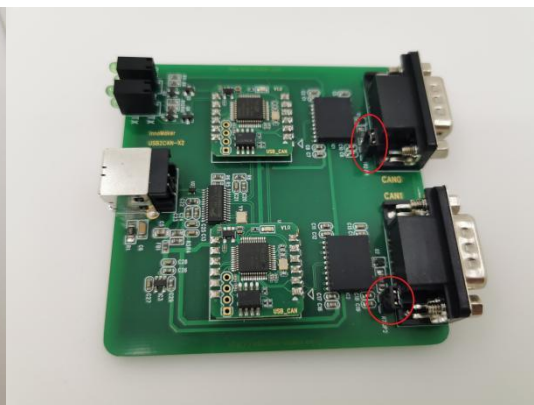


Enable the 120 Ω resistance

If you want to enable the 120 Ω resistance all the time. You can open the shell and enable the build-in resistance.





Disable the 120 Ω resistance



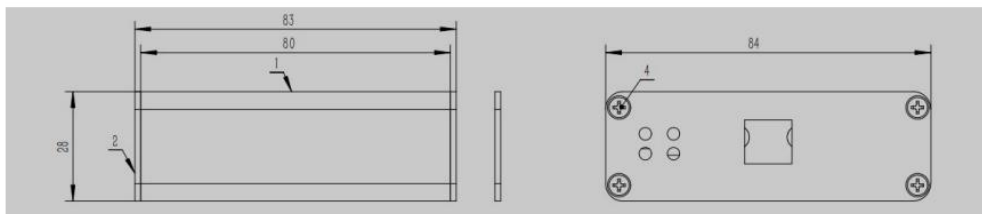
Enable the 120 Ω resistance

(3) LED Indicate

Operation	Appearance
	When you plug the device into the USB port of the computer, All lights are flashing for one second and then all LED goes off.
	Enable CAN0 only , The TX/RX LED of CAN0 should be on.
	Enable CAN1 only , The TX/RX LED of CAN0 should be on.
	Enable CAN0 and CAN1 both, All LED should be on.

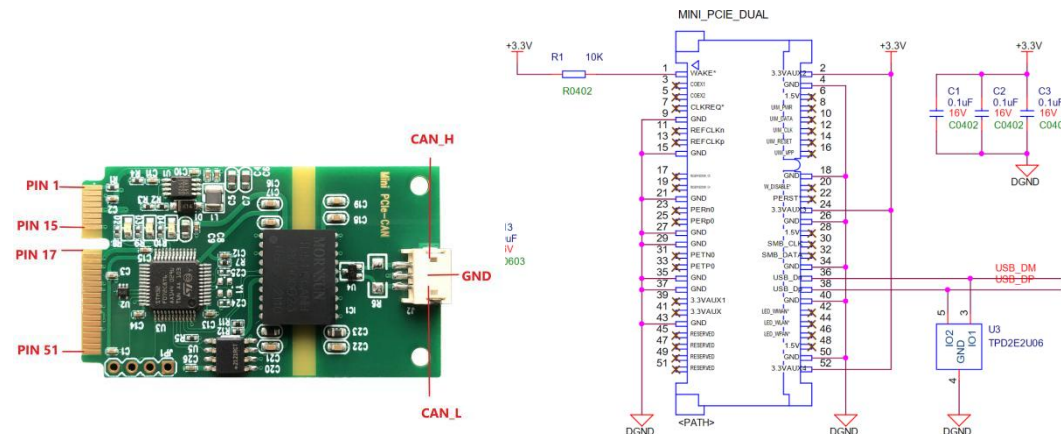
	<p>RX led flashing to indicate the data is being Received.</p>
	<p>TX led flashing to indicate the data is being sent.</p>

(4) Dimension Figure



4.4 MINIPCI-E-CAN

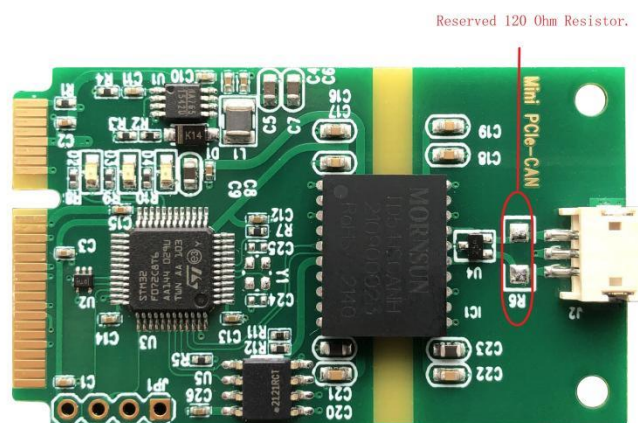
(1) CAN Connector Pinout



(2) 120 Ohm Resistor Setting.

A High-speed CAN bus (ISO 11898-2) must be terminated on both ends with 120 Ohms. There are three ways to enable the 120 Ω resistance.


There is a on-board reserved 120 Ohm Resistor with 1206 package.



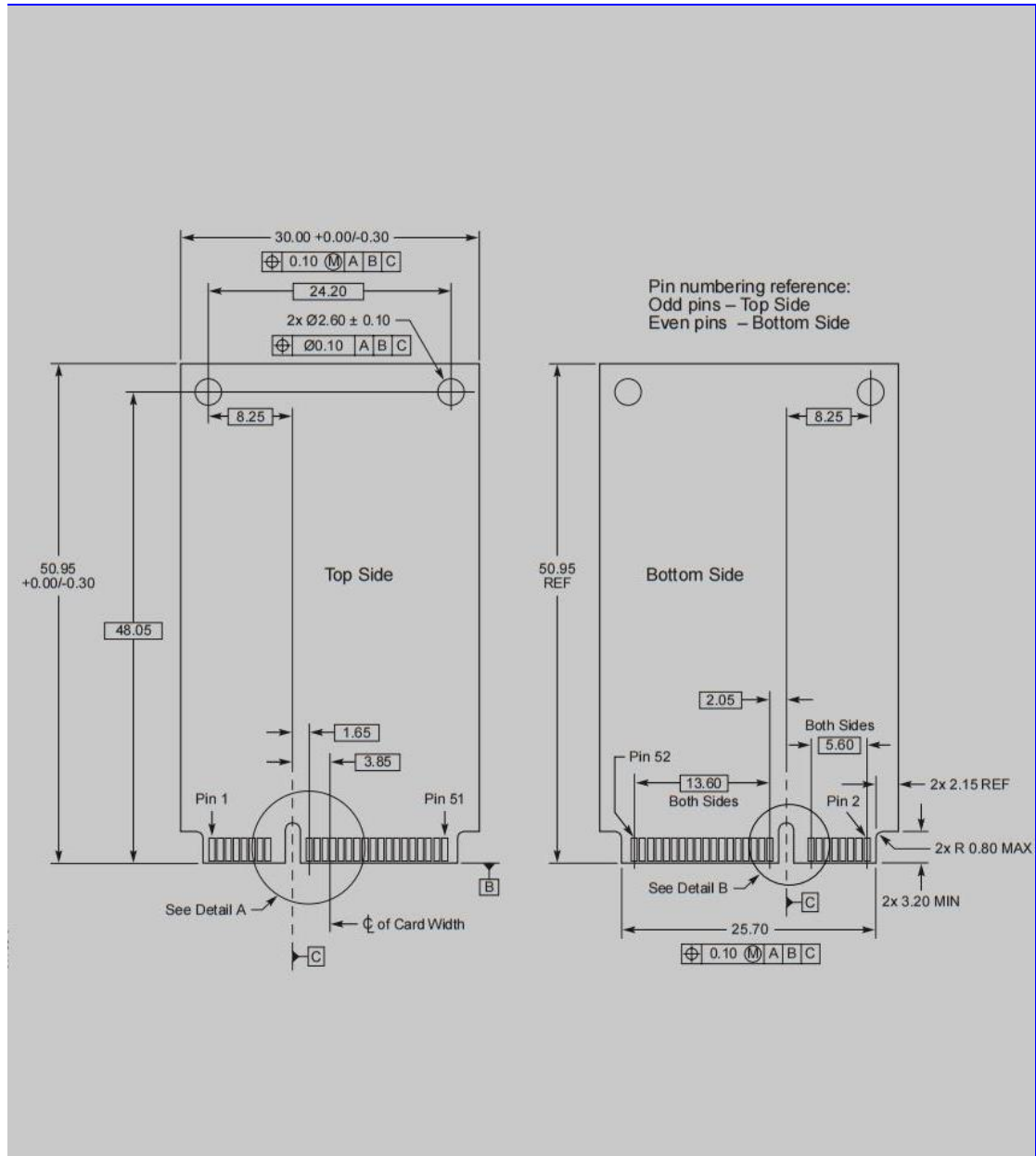
There is a DB9 to 3pins terminal comes with goods. they have the 120 Ω resistance selectable feature.



(3) LED Indicate

	LED Name	Description
	Link	Red led is normally on to indicate The module is came up without any errors
	Tx	Red led flash to indicate send data.
	Rx	Red led flash to indicate receive data.

(4) Dimension Figure



5. Windows

All related materials for WINDOWS are located at the following path:

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Windows>

5.1 The Directory Structure Of The WINDOWS folder.

Name	Last commit message
..	
Busmaster	Create BUSMASTER_HELP_ORIGINAL.pdf
C#-V1.3.0	Create C# Application Administrator Privileges.pdf
C++ (Visual Studio)	Change C++ folder name
Provided By Customer	Update C++ DEMO
Python	update doc
QT	update doc
SavvyCAN-InnoMaker	Create CANdevStudio.zip
VB	update doc
Windows Driver Tool	Added python on windows
WindowsReady-MadeApp	update folder name
C# API.pdf	update doc
C++ API.pdf	update doc
FAQ: Innomaker Echo and buffer.pdf	update doc
Python .pdf	update doc
VB.pdf	update doc

In the WINDOWS folder, we have categorized the files based on development tools and environments.

If you are using it for the first time, you can download the software tools from the "WindowsReady-Made APP" folder to use our application.

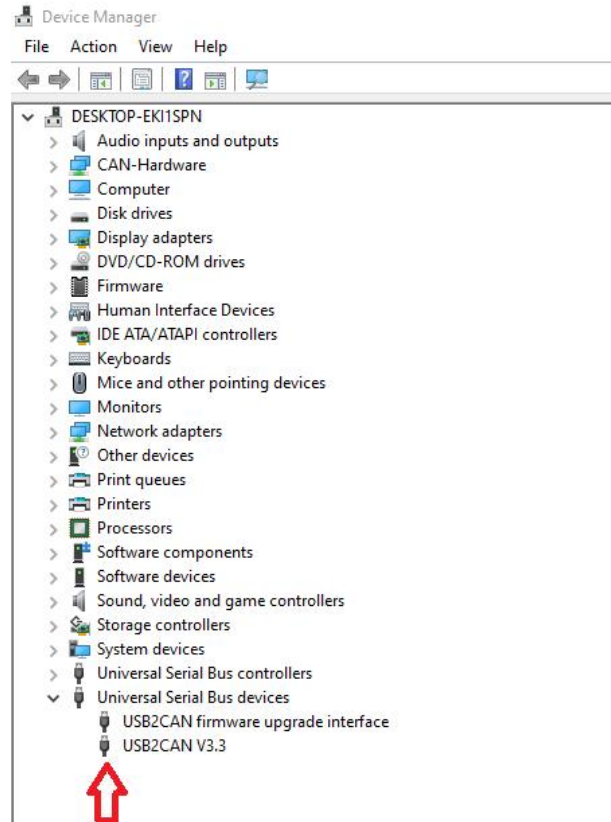
BUSMASTER and SAVVYCAN-INNOMAKER are well-known third-party CAN software provided by other companies.

C#, C++, Python, QT, and VB are for engineers who need to perform secondary development.

Windows Driver Tools is a tool for installing CAN device drivers on Windows. However, for Windows 8 and above, the system comes with built-in drivers, so installation is not required.

5.2 Windows Driver

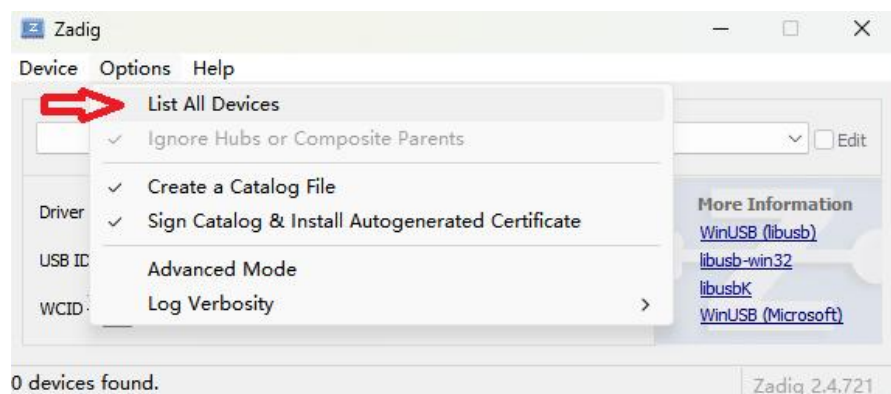
The USB2CAN device is a WINUSB/LIBUSB device on Windows, not the COM to CAN port. Insert the USB2CAN device into the computer's USB port. Open the Device Manager on the computer, and you will see the INNOMAKER USB2CAN device.



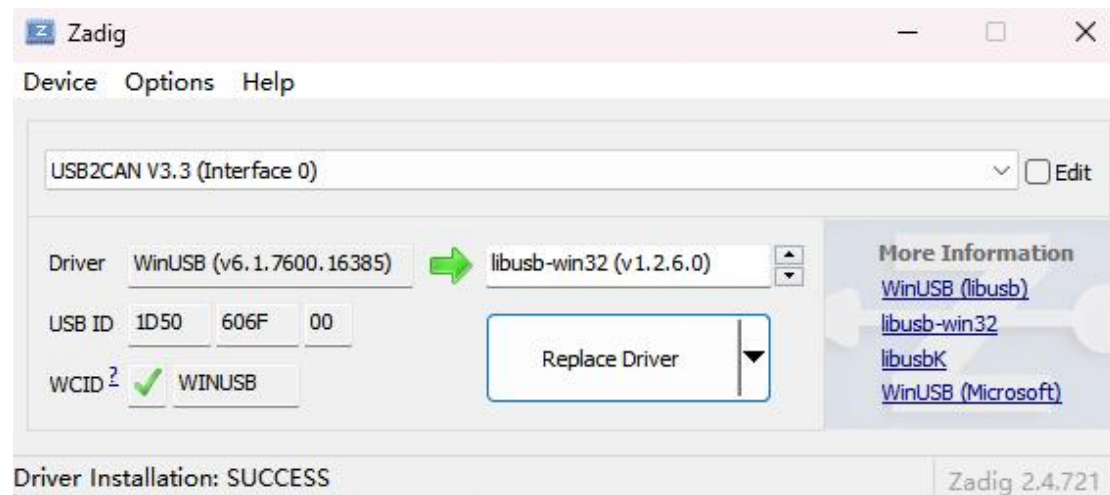
Some systems may lose the driver, or if you want to switch to the LIBUSB driver, use the Zadig tool in the Windows Driver Tool folder.

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Windows/Windows%20Driver%20Tool>

(1) Open zadig.exe and select Options → List All Devices.



(2) Most systems will default to the WinUSB driver. If it appears blank, you can choose to install the WinUSB driver or replace it with the libusb-win32 driver. These two drivers are not significantly different, but it is recommended to install and use the libusb driver.



6. Windows/Mac OS Ready-Made Tools

Download the ready-made tools installation package from below link:

Windows:

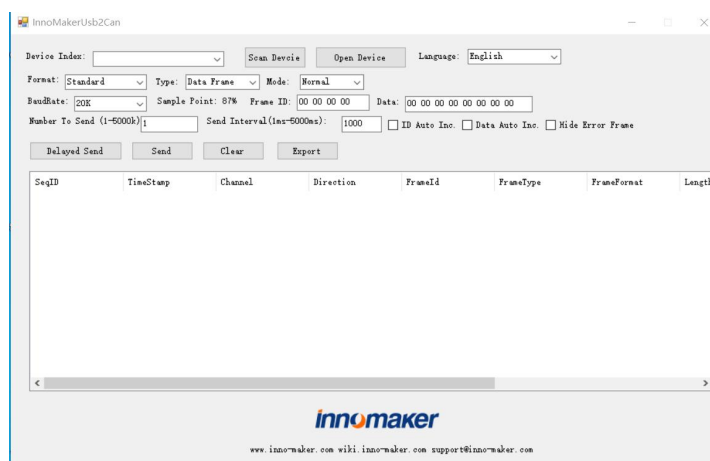
<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Windows/WindowsReady-MadeApp>

Mac:

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Mac%20Os/MAC-V1.2.0/InnoMaker%20USB2CAN%20Tools>

5.1 Open the USB2CAN tools.

Download and install the ready-made tool. One tool interface is only for one device. If you want to use two devices at the same time, Please open two interfaces.



5.2 Scan For Devices

Plug the device into the USB port of your computer, click 'Scan Device' button.

If you are using the winusb driver (default on Windows 8 and above), you could find USB2CAN device and named with different ID. Clicking on the device name will complete the selection.

If you are using the libusb driver, you could see the libusb0-001, libusb0-002, libusb0-00X... in the list. Clicking on the 'libusb0-00X' will complete the selection.

If you need to operate multiple devices, open multiple windows.

InnoMakerUsb2Can

Device Index: 843044646160000 Scan Device Open Device Language: English

Format: Stand(84143baba9160000) Mode Normal

BaudRate: 20K Sample Point: 87% Frame ID: 00 00 00 00 Data: 00 00 00 00 00 00 00

Number To Send (1-5000k) 1 Send Interval (1ms-5000ms): 1000 ☐ ID Auto Inc. ☐ Data Auto Inc. ☐ Hide Error Frame

Delayed Send Send Clear Export

SeqID	TimeStamp	Channel	Direction	FrameId	FrameType	FrameFormat	Length
-------	-----------	---------	-----------	---------	-----------	-------------	--------

www.inno-maker.com wiki.inno-maker.com support@inno-maker.com

Device Index: 9&36057906&2&0000

Format: Standby 9&346b833b&0&0000

BaudRate: 20K \\.\libusb0-0001

Setting below parameters in the red box before open device.

Silent mode: The module will appear on the CAN-bus, but in a passive state. It can receive CAN messages, but cannot transmit CAN messages or answer. This mode can be used as a bus monitor because it does not affect CAN-bus communications but can observe the CAN-bus states.

Loopback mode: For USB2CAN self-test, CAN module receives its own messages. In this mode, the send part of the CAN module is connected internally with the reception one.

InnoMakerUsb2Can

Device Index: 8&30f4434&1&0000 Scan Device Open Device Language: English

Format: Standard Type: Data Frame Mode: Normal

BaudRate: 20K Sample Point: 87% Frame ID: 00 00 00 00 Data: 00 00 00 00 00 00 00 00

Number To Send (1-5000k): 1 Send Interval(1ms-5000ms): 1000 ☐ ID Auto Inc. ☐ Data Auto Inc. ☐ Hide Error Frame

Delayed Send Send Clear Export

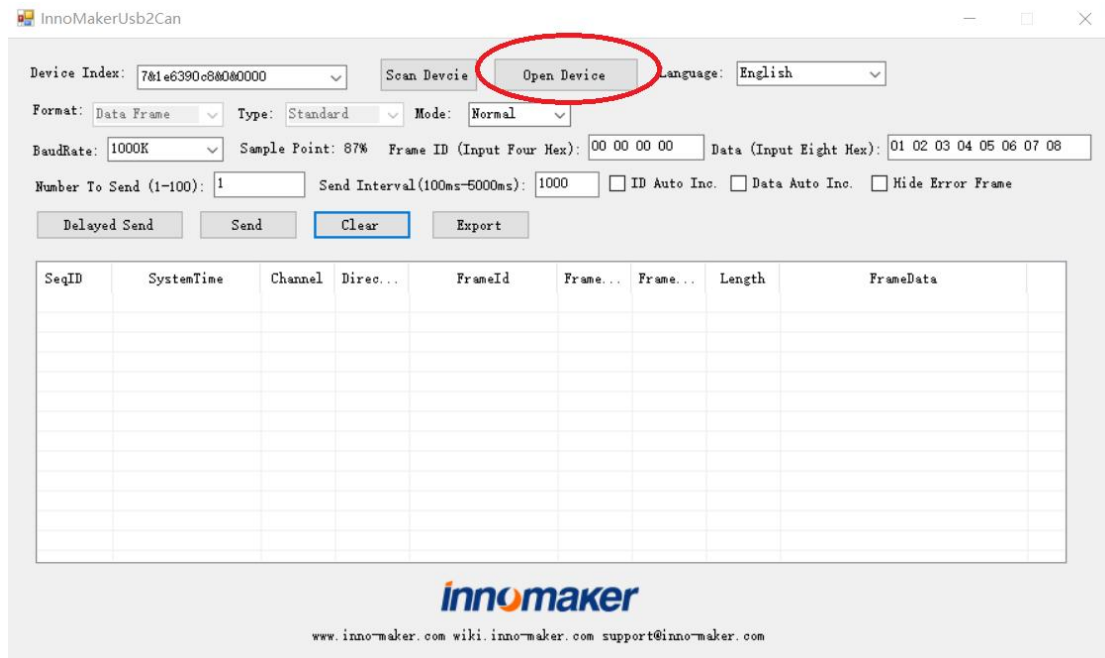
SeqID	TimeStamp	Channel	Direction	FrameId	FrameType	FrameFormat	Length

innomaker

www.inno-maker.com wiki.inno-maker.com support@inno-maker.com

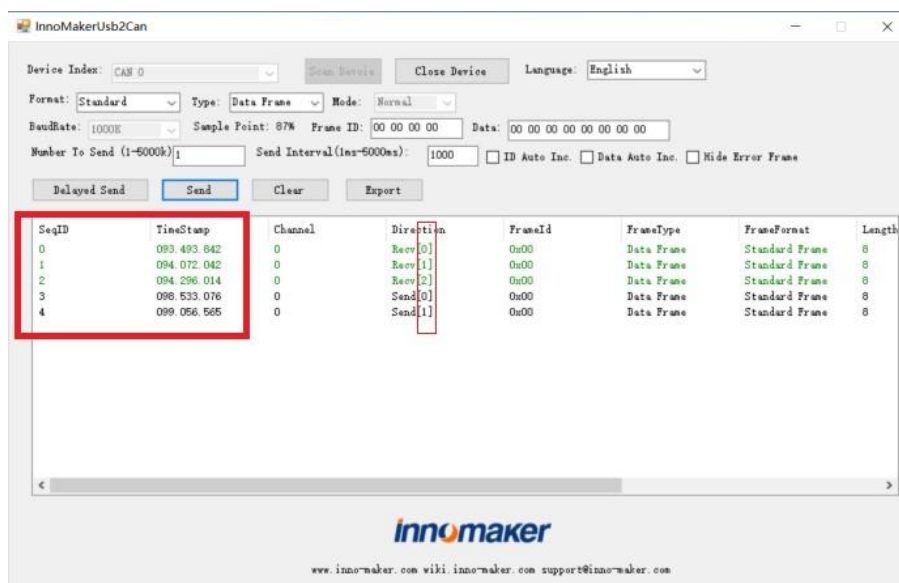
5.4 Enable Device.

Click 'Open device', the TX/RX LED light up.

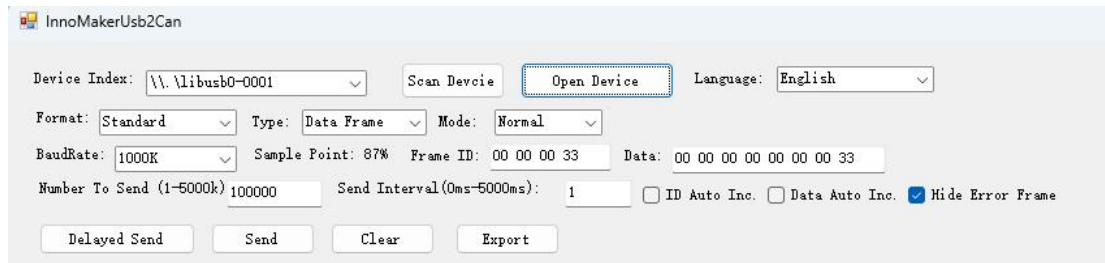


5.5 Send/Receive Data

After Open device, the device will automatic received data. The send data will be marked in black, Receive data will be marked in green. Error Frame will be marked in red. Every message will comes with the time stamp and serial number.



5.6 Continuous sending / Sending In Large Quantities



The software supports timed sending and continuous large data transmission.

In the "**Send Interval input**" input box allows you to set the time interval for continuous sending. When set to 0, it will use the current maximum sending capacity to transmit data. On computers with different configurations, unknown exceptions may occur, potentially causing software or computer lag. It is only reserved for use during special tests. Therefore, it is generally recommended to set the minimum value to 1ms.

In the "**Number To send**" input box, you can enter the amount of data you want to send, ranging from 1 frame to 5,000,000 frames. Note that when sending large amounts of data, please ensure your computer has enough memory to buffer it.

After setting the parameters for Send Interval and Number To send, click "**Delayed Send**" to automatically continue sending data. If "**Data Auto Inc**" and "**ID Auto Inc**" are checked, the sent frame data or frame ID will automatically increment by 1 for each frame.

5.7 Windows Real Time Setting

If you want to test the USB2CAN module on at high speed and mass data mode (such as send/receive one million frames with 1 millisecond interval), please change the base priority level, That would be very helpful.

I take Windows 10 for example.

Open Task Manager

→ right click on inno-maker application process

→ left click on 'go to details'

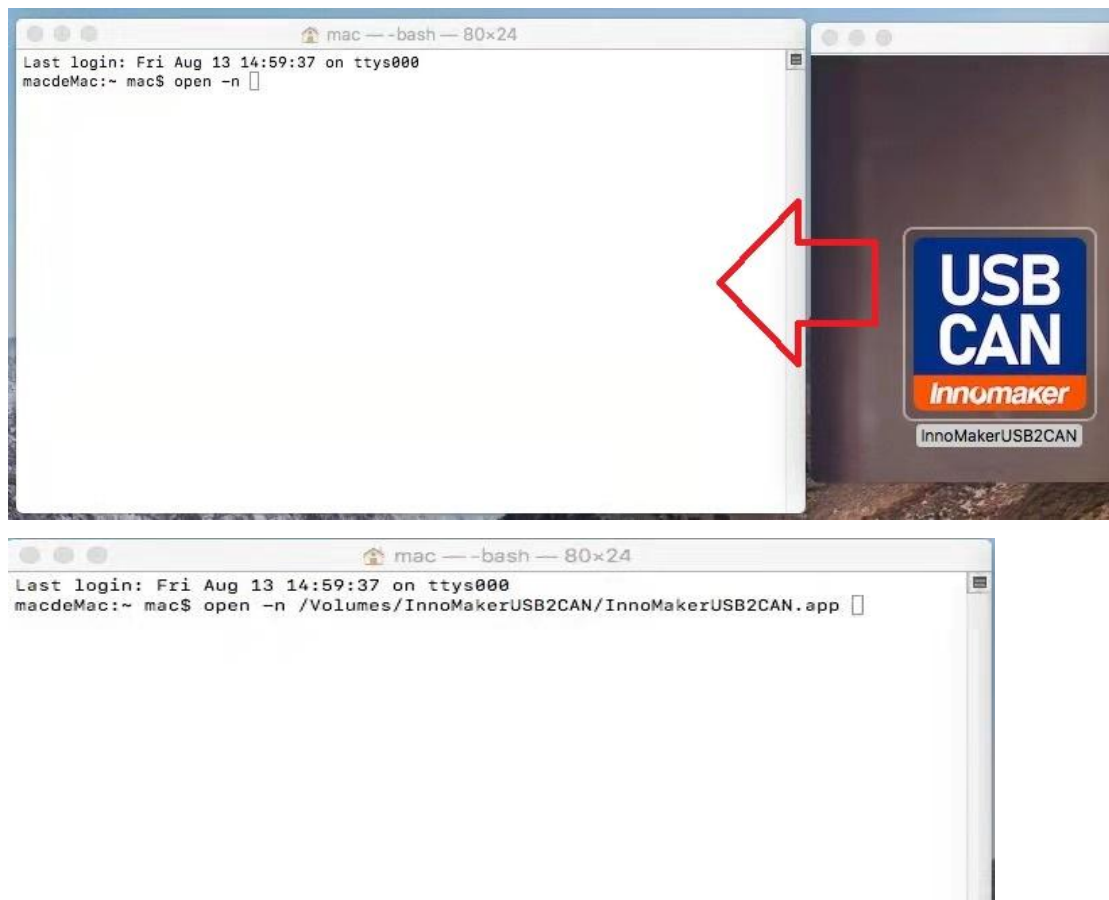
→ right click on inno-maker application process

→ left click on 'set priority'

→ left click on 'realtime'

5.8 Mac Os Double Open The USB2CAN Tool

Open the terminal of MacOS, if you can't find it, please googling it. Type 'open -n' + the software path. If you don't know the path, drag and drop the application icon.



7. BUSMASTER On Windows

BUSMASTER is an Open Source Software tool to Simulate, Analyze and Test data bus systems such as CAN. It seem stop to update now, but it also a great tools for reference.

Home Page:

<https://rbei-etas.github.io/busmaster/>

Project:

<https://github.com/rbei-etas/busmaster>

Video for BUSMASTER:

<https://www.youtube.com/user/busmasteropensource/featured>

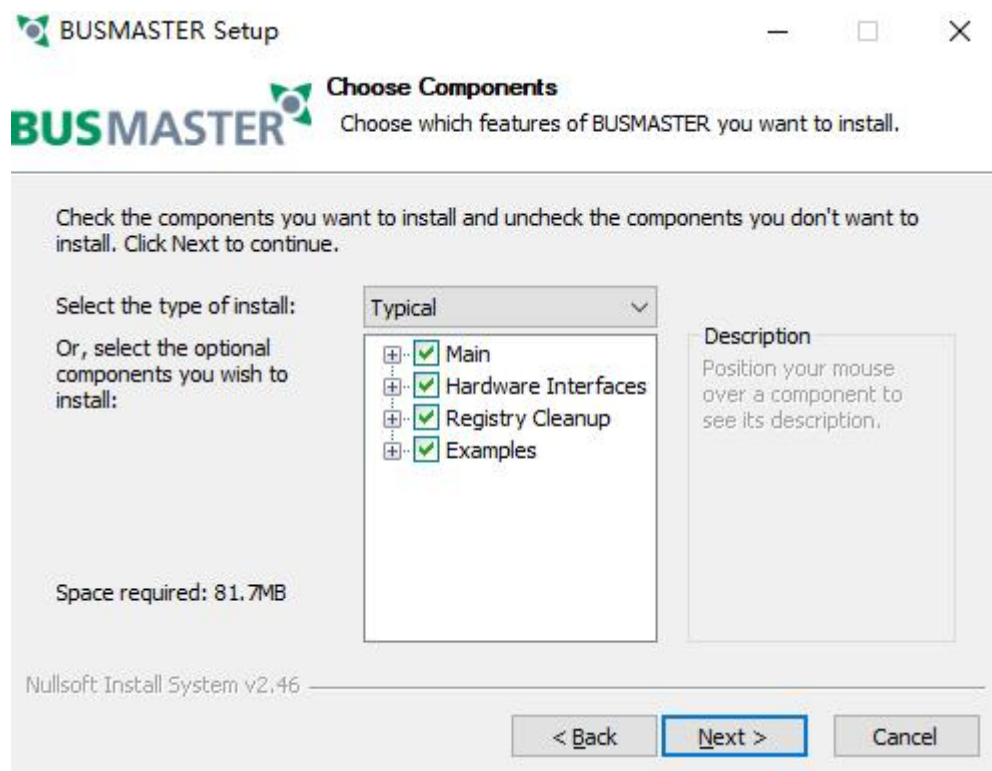
7.1 Download and Install

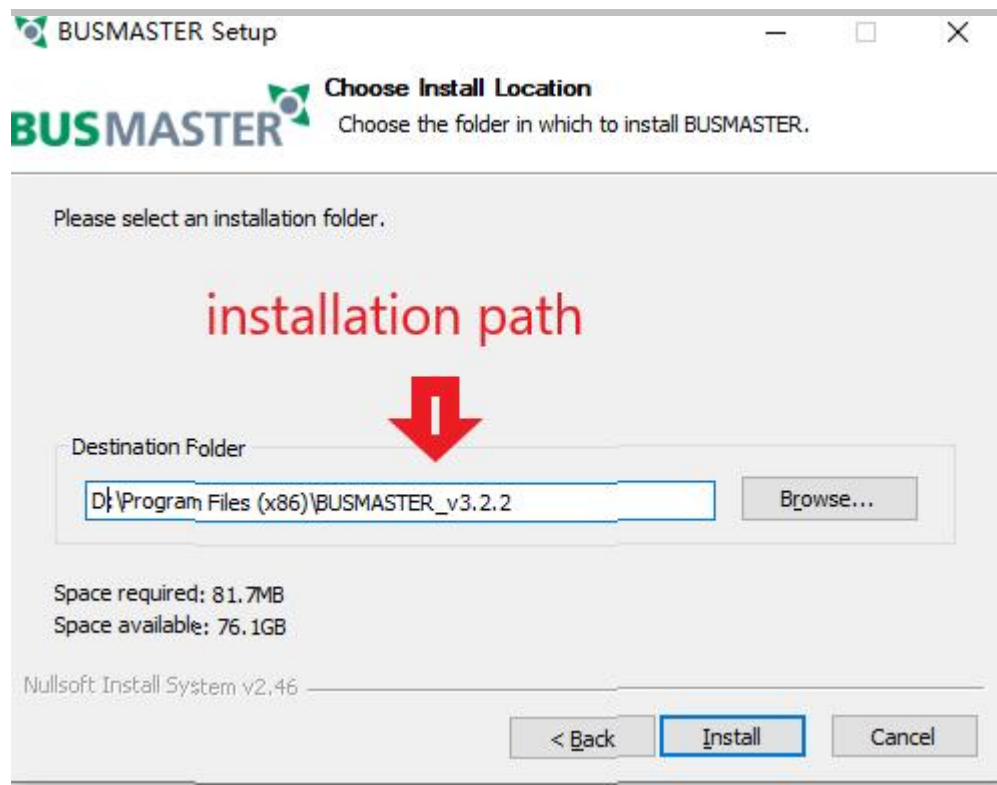
(1)Download the from our github:

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Windows/Busmaster>

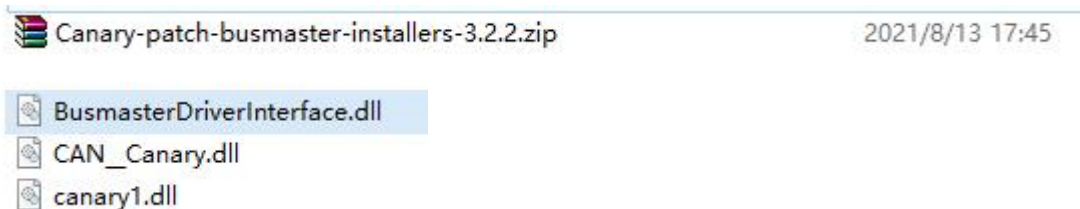
INNO-MAKER Added Coreboard hardware usermanual		1f56b1a 1 hour ago	🕒 22 commits
	Busmaster	add discription	6 months ago

(2) unzip the installation package and setup



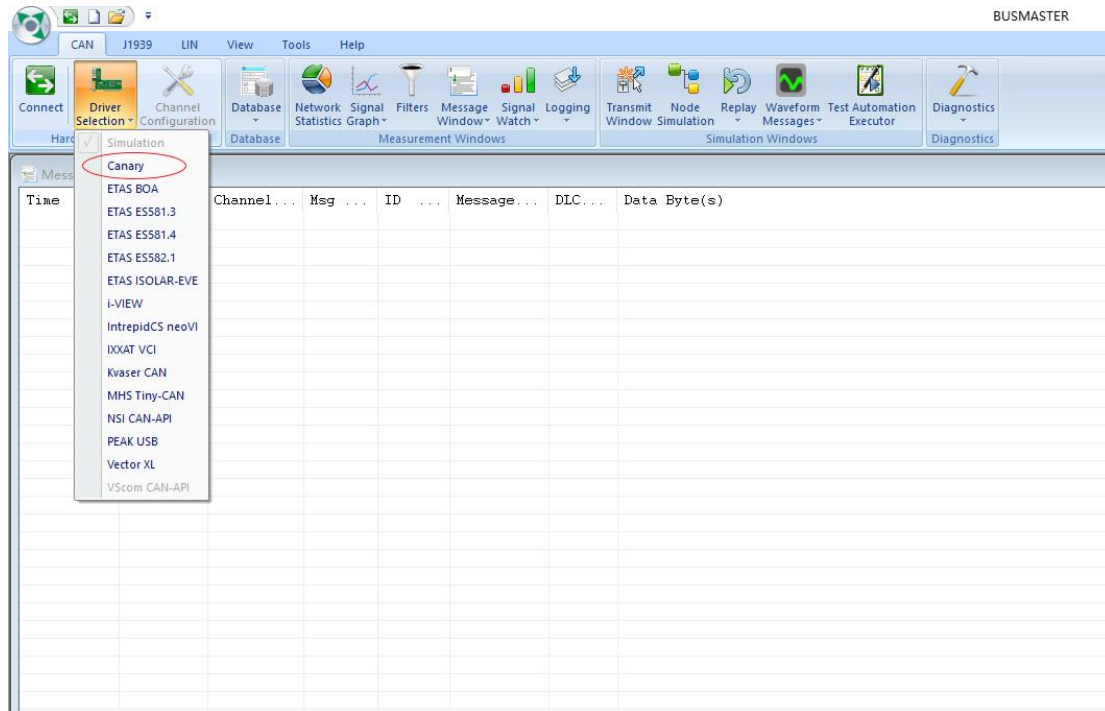


(3) unzip below service pack and past these three files into the installation path of BUSMASTER v3.2.2. Replace the files in the target.

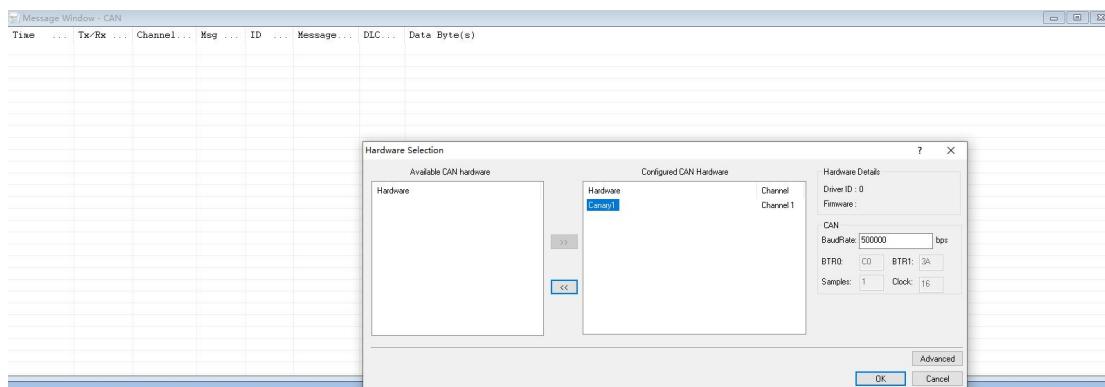


7.2 Using The Software

- (1) Open the BUSMASTER v3.2.2 and plug the usb2can into the usb port of your computer.
- (2) Click the 'driver selection' and select the 'Canary'



- (3) Setting the Baudrate, and press OK



(4) Click connect, start to work.



8. Mac Os

Download the tools and SDK of Mac Os from below link:

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Mac%20Os/MAC-V1.2.0>

There are consists of four parts:

- 📁 doc_html
- 📁 InnoMaker USB2CAN Tools
- 📁 Lib
- 📁 Tools Source Code

Folder name	Description
InnoMaker USB2CAN Tools	InnoMaker USB2CAN ready-made test tools, You can run natively on Mac OS version equal or above 10.11. If Mac os
Tools Source Code	The source of InnoMaker USB2CAN test tools, to show you how to use the SDK to develop a usb to can application.
Lib	The Library function for develop USB2CAN applications. These libraries are not open source. If you have any problem and suggestion, feel free to contract us.
doc_html	Simple document for library description.

8.1 Library








Open the Lib folder, There are three files in it:

 .DS_Store	2020/5/10 14:58
 libInnoMakerUSBIOLib.dylib	2020/5/10 14:57
 UsbIO.h	2020/5/10 14:55
 USBIO+USBCAN.h	2020/5/10 14:50

Filename	Description
UsbIO.h	USB Common Interface
USBIO+USBCAN.h	USB Special For CAN Interface
libInnoMakerUSBIOLib.dylib	Dynamic library file

8.2 Document

Open the doc_html folder and open the index.html files. You can see InnoMakerUSB IO LIB reference.

 Classes	2020/5/11 13:10	文件夹	
 css	2020/5/11 12:53	文件夹	
 img	2020/5/11 12:53	文件夹	
 js	2020/5/11 12:53	文件夹	
 Protocols	2020/5/11 13:10	文件夹	
 hierarchy.html	2020/5/11 13:10	360 se HTML Do...	2 KB
 index.html	2020/5/11 13:10	360 se HTML Do...	2 KB

InnoMakerUSBIOLib

innomaker

Hierarchy

InnoMakerUSBIOLib Reference

Class References

- InnoMakerDevice
- UsbIO

Protocol References

- InnoMakerDeviceDelegate

Copyright © 2020 innomaker. All rights reserved. Updated: 2020-05-11

Generated by [appledoc 2.2.1 \(build 1334\)](#).

9. Linux/Ubuntu/Raspbian

9.1 Run Linux Test Demo

The USB2CAN is a SOCKET-CAN device, so it works properly on all Linux systems (e.g., Ubuntu, Debian, Fedora, Raspberry Pi OS etc.) starting from version 3.9 without requiring additional drivers. If you encounter driver issues on older versions, you will need to reconfigure the kernel drivers. Enable `gs_usb.c` and install `gs_usb.ko` into the system. Please note that simply compiling these drivers may result in a failure to load them. To avoid this, ensure you perform a full compilation with the updated configuration. If you have any questions, feel free to contact our support team at support@inno-maker.com.

You can use the USB2CAN device on any Linux-based computer, laptop, or ARM board. For convenience, I will use a Raspberry Pi 4 and two USB2CAN devices to demonstrate how to run our C and Python demos, as well as how to use the `can-utils` tool.

Please download the codes and tools from below links:

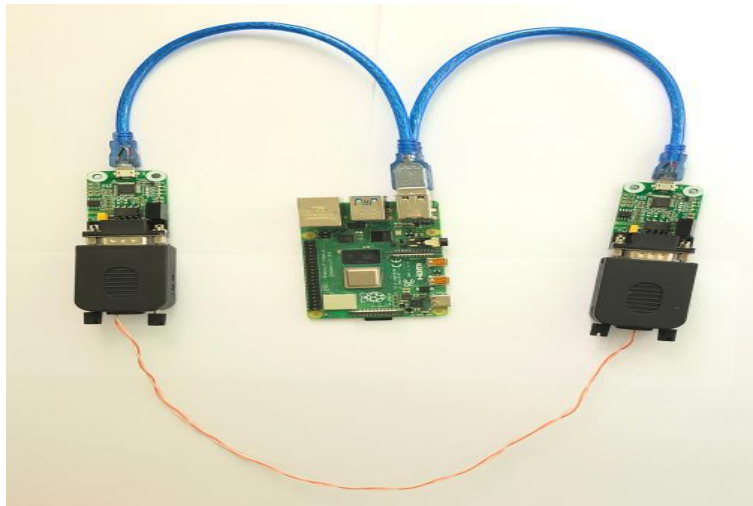
<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Linux%20Raspbian%20Ubuntu/software>

9.1.1 Preparatory work

9.1.1.1 Connection

Plug the USB2CAN device into the USB Host of Raspberry Pi. And then connect the CAN_H pin and CAN_L pin to each other. No GND pin connection requirement.

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



9.1.1.2 ifconfig -a

Type command 'ifconfig -a' to check 'can0' and 'can1' device is available in system.

```

pi@raspberrypi:~$ 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

eth0: flags=4096<UP,BROADCAST,MULTICAST> mtu 1500
    ether dc:a6:32:0b:85:32 txqueuelen 1000 (Ethernet)
    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

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 18 bytes 1655 (1.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 18 bytes 1655 (1.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.115 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::e073:8291:5834:4dc3 prefixlen 64 scopeid 0x20<link>
    ether dc:a6:32:0b:85:33 txqueuelen 1000 (Ethernet)

```

9.1.1.3 dmesg

You can type command 'dmesg' for see more information about USB2CAN module at the bottom. Here, you can see the Serial Number of the USB2CAN. This Serial Number is unique and can be used to identify the device in certain situations.

```
3.100730] usb 1-1.1.1: New USB device found, idVendor=1d50, idProduct=606f, bcdDevice= 0.00
3.100755] usb 1-1.1.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
3.100773] usb 1-1.1.1: Product: USB2CAN v1
3.100790] usb 1-1.1.1: Manufacturer: InnoMaker
3.100807] usb 1-1.1.1: SerialNumber: 002C00325353530620313737
3.184277] i2c /dev entries driver
3.303669] usb 1-1.1.2: new full-speed USB device number 6 using xhci_hcd
3.542779] usb 1-1.1.2: New USB device found, idVendor=1d50, idProduct=606f, bcdDevice= 0.00
3.542803] usb 1-1.1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
3.542822] usb 1-1.1.2: Product: USB2CAN v1
3.542839] usb 1-1.1.2: Manufacturer: InnoMaker
3.542856] usb 1-1.1.2: SerialNumber: 002B00315353530620313737
```

9.1.2 Run can-utils Tool

This tool is a very easy way to test USB2CAN module communication. There is only a simple use instruction. For more details, please refer to can-utils usermanual and source code.

<https://github.com/linux-can/can-utils/>

9.1.2.1 -Install Tools

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

9.1.2.2 -Send And Receive Test

(1) Initialize CAN port, After below command, you could see the the TX/RX light on for CAN0 and CAN1 channels.

```
sudo ip link set can0 down
sudo ip link set can0 type can bitrate 125000
sudo ip link set can0 up
```

```
sudo ip link set can1 down
sudo ip link set can1 type can bitrate 125000
sudo ip link set can1 up
```

(2)Set CAN0 as receiver

```
candump can0
```

(3)Set CAN1 as sender

```
cansend can1 123#1234567890
```



```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~$ sudo ip link set can0 down
pi@raspberrypi:~$ sudo ip link set can0 type can bitrate 125000
pi@raspberrypi:~$ sudo ip link set can0 up
pi@raspberrypi:~$ candump can0
can0 123 [5] 12 34 56 78 90

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~$ sudo ip link set can1 down
pi@raspberrypi:~$ sudo ip link set can1 type can bitrate 125000
pi@raspberrypi:~$ sudo ip link set can1 up
pi@raspberrypi:~$ cansend can1 123#1234567890
pi@raspberrypi:~$
  
```

9.1.2.3 -Loopback Mode Test

Loopback mode is only send and receive data by itself. Will not send the data to the CAN-BUS.
Loop back mode only use for test the device when only have one device in hand. Do not use this mode to communication with other CAN device.

```

sudo ip link set can0 down
sudo ip link set can0 type can bitrate 125000 loopback on
sudo ip link set can0 up
candump can0 & cansend can0
  
```

```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~$ sudo ip link set can0 down
pi@raspberrypi:~$ sudo ip link set can0 type can bitrate 125000 loopback on
pi@raspberrypi:~$ sudo ip link set can0 up
pi@raspberrypi:~$ candump can0 & cansend can0 123#1234567890
[1] 1551
can0 123 [5] 12 34 56 78 90
can0 123 [5] 12 34 56 78 90
pi@raspberrypi:~$
  
```

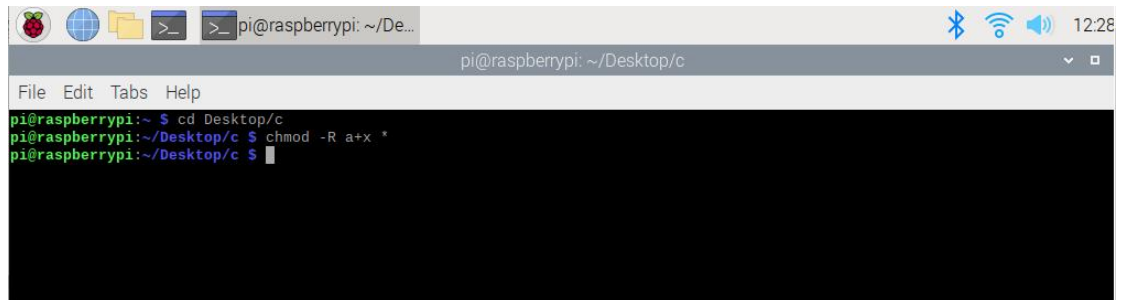
9.1.3 Run C Demo

(1) Load C Demo named from our github and up-zip it to the desktop of Raspbian.

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Linux%20Raspbian%20Ubuntu/software/c>

(2) Go to folder named 'c' and change the permissions.

```
chmod -R a+rw * *
```



(3) Set one CAN channel as receiver, execute following commands in serial terminal. Now this Raspberry pi is blocked.

```
./can0_receive
```

(4) Set the other CAN channel as sender, execute following commands.

```
./can1_send
```

```

pi@raspberrypi: ~/Desktop/usb2cantest
File Edit Tabs Help
pi@raspberrypi:~$ cd Desktop
pi@raspberrypi:~/Desktop$ cd usb2cantest
pi@raspberrypi:~/Desktop/usb2cantest$ chmod -R a+x *
pi@raspberrypi:~/Desktop/usb2cantest$ ls
can0_receive  can1_receive  can1_test  can_send  cantool  can-utils-master  rs485_test  setup
can0_send     can1_send     can_receive can_server cantool-sh can-utils-master.zip rs485_test-sh
pi@raspberrypi:~/Desktop/usb2cantest$ ./can0_receive
This is a can receive demo, can0 with 1Mbps!
Received standard frame!
can_id = 0x123
can_dlc = 8
data[0] = 1
data[1] = 2
data[2] = 3
data[3] = 4
data[4] = 5
data[5] = 6
data[6] = 7
data[7] = 8
pi@raspberrypi:~/Desktop/usb2cantest$

pi@raspberrypi:~/Desktop/usb2cantest
File Edit Tabs Help
pi@raspberrypi:~$ cd Desktop
pi@raspberrypi:~/Desktop$ cd usb2cantest
pi@raspberrypi:~/Desktop/usb2cantest$ ls
can0_receive  can1_send  can_send  cantool-sh  rs485_test
can0_send     can1_test  can_server can-utils-master  rs485_test-sh
can1_receive  can_receive cantool    can-utils-master.zip  setup
pi@raspberrypi:~/Desktop/usb2cantest$ ./can1_send
This is a socket can transmit demo program ,can1 with 1Mbps baud rate
Transmit standard frame!
can_id = 0x123
can_dlc = 8
data[0] = 1
data[1] = 2
data[2] = 3
data[3] = 4
data[4] = 5
data[5] = 6
data[6] = 7
data[7] = 8
pi@raspberrypi:~/Desktop/usb2cantest$

```

9.1.4 Run Python3 Demo

Download Python Demo from our github and up-zip it to Desktop(or wherever you want put it).

<https://github.com/INNO-MAKER/usb2can/tree/master/For%20Linux%20Raspbian%20Ubuntu/software/python3>

There are three files in the folder, send.py and receive.py is for testing send and receive function separately and the test.py program is used to demonstrate sending different frame types via can0, while can1 prints the received content.

(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

```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~$ python3 -V
Python 3.7.3
pi@raspberrypi:~$

```

(2) If you can't find the Python3 in system. Install the Python3

```
sudo apt-get install python3-pip
```

(3) If you are testing on Ubuntu system, It may can't use the 'ifconfig' command. Please install the net tools firstly.

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

(4) Install Python CAN library.

```
sudo pip3 install python-can
```

(5) Run test code. One usb2can will send data and the other will print the result

```
sudo python3 test.py
```

(6) You will see the data received.

```
pi@raspberrypi:~/python3 $ sudo python3 test.py
Received base frame:
Timestamp: 1659947408.906498 ID: 00000123 X Rx DL: 8 00 01 02 03 04 05 06 07 Channel: can1
Received extended frame:
Timestamp: 1659947408.907386 ID: 1fff6666 X Rx DL: 8 07 06 05 04 03 02 01 00 Channel: can1
Received remote frame:
Timestamp: 1659947408.907977 ID: 00000321 X Rx R DL: 0 Channel: can1
```

9.2 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(Controllor 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

8.2.1 Programming in C

8.2.1.1- 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);
```

8.2.1.2 -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 filter 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));
```

8.2.2 Programming in Python3

8.2.2.1- 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
```

8.2.2.2 -Simple common functions

(1) Install Python CAN library.

```
sudo pip3 install python-can
```

(2)Set bitrate and start up CAN device. Shut down the CAN down before setting the it.Otherwise the setting will be failed.

```
os.system('sudo ifconfig can0 down')
os.system('sudo ip link set can0 type can bitrate 1000000')
os.system("sudo ifconfig can0 txqueuelen 100000")
os.system('sudo ifconfig can0 up')
```

```
os.system('sudo ifconfig can1 down')
os.system('sudo ip link set can1 type can bitrate 1000000')
os.system("sudo ifconfig can1 txqueuelen 100000")
os.system('sudo ifconfig can1 up')
```

(3)Bind the socket to can0 and can1

```
can0 = can.interface.Bus(channel = 'can0', bustype = 'socketcan')
can1 = can.interface.Bus(channel = 'can1', bustype = 'socketcan')
```

(4)Assembly data to send.

```
sff frame = can.Message(arbitration_id=0x123, data=[0,1,2,3,4,5,6,7])
```

(5)Can0 send data.

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

(6)CAN1 receive data. May I draw your attention that you should open a new thread to received.

```
msg = can1.recv(30.0)
```

(7)Close CAN device

```
os.system('sudo ifconfig can0 down')
os.system('sudo ifconfig can1 down')
```


10. Error Frame

You may receive some error frame marked in red when you use the USB2CAN device. They will let you know what problem does the USB2CAN device meet on your CAN Bus.

Some people would say why didn't they meet the error frame with other tool or USB to CAN device 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 system.

To understand why error frames are used and to define the statuses of CAN bus, you can refer to the document **ISO 11898-1 2015.12.15** for detailed reading.

<https://github.com/INNO-MAKER/usb2can/tree/master/Document%EF%BC%88read%20me%20first%EF%BC%89/Advanced%20Document>

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 USB2CAN 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.

11. User Manual Version Descriptions

Version	Description	Date	E-mail
V1.1	Initial version	2019.02.02	calvin@inno-maker.com
V1.2	Add Python Add can-utils	2019.08.26	calvin@inno-maker.com
V1.3	Correct description	2019.08.26	calvin@inno-maker.com
V1.4	Added DB9 Description Added CAN utils tools loopback mode description	2020.10.11	calvin@inno-maker.com
V1.5	Added description for how to install net tools on Ubuntu system	2021.05.01	calvin@inno-maker.com
V1.6	Added USB2CAN-DEVKIT ,USB2CAN-Cable,USB description Merged Mas Os , WINDOWS, LINUX description	2021.10.10	calvin@inno-maker.com
V1.7	Added Dimension Figure Added Chapter BUSMASTER in WINDOWS	2022.01.06	calvin@inno-maker.com
V1.8	Merge USB2CAN-x2 doc Update Python3 description Add MINIPCIE-CAN description	2022.08.08	calvin@inno-maker.com
V2.0	The directory structure has been reorganized, and the descriptions have been modified and improved.	2025.03.22	calvin@inno-maker.com

If you have any suggestions, ideas, codes and tools please feel free to email to me. I will update the user manual.