USB2CAN Device User Manual

Menu

1. General Description:

USB2CAN module is a ‘plug and play’ and bi-directional port powered USB to CAN converter which realizes long-distance communication between your Raspberry Pi/SBC/PC 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.
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.

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++ demo and dynamic link libraries for help customer develop own applications.

We provide four kinds of USB2CAN, the hardware appearance is different but the software ,firmware features are 100% same.

USB2CAN

USB2CAN-C

USB2CAN-DEV

USB2CAN-X2

2. Features

1. The USB 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 voltage isolation and 2500V ESD isolated protection on signal pins.120 Ohm resistor selectable jumper feature.

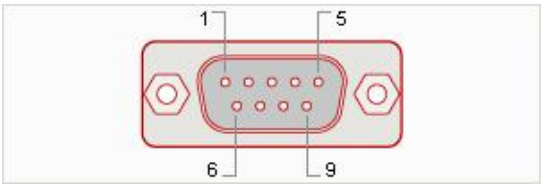
5. Provide C/Python and source code with Socket-CAN for Raspbian(Linux), Dynamic library and demo with IOUSBKit for Mac Os(Big Sur).

3. Technical Specification

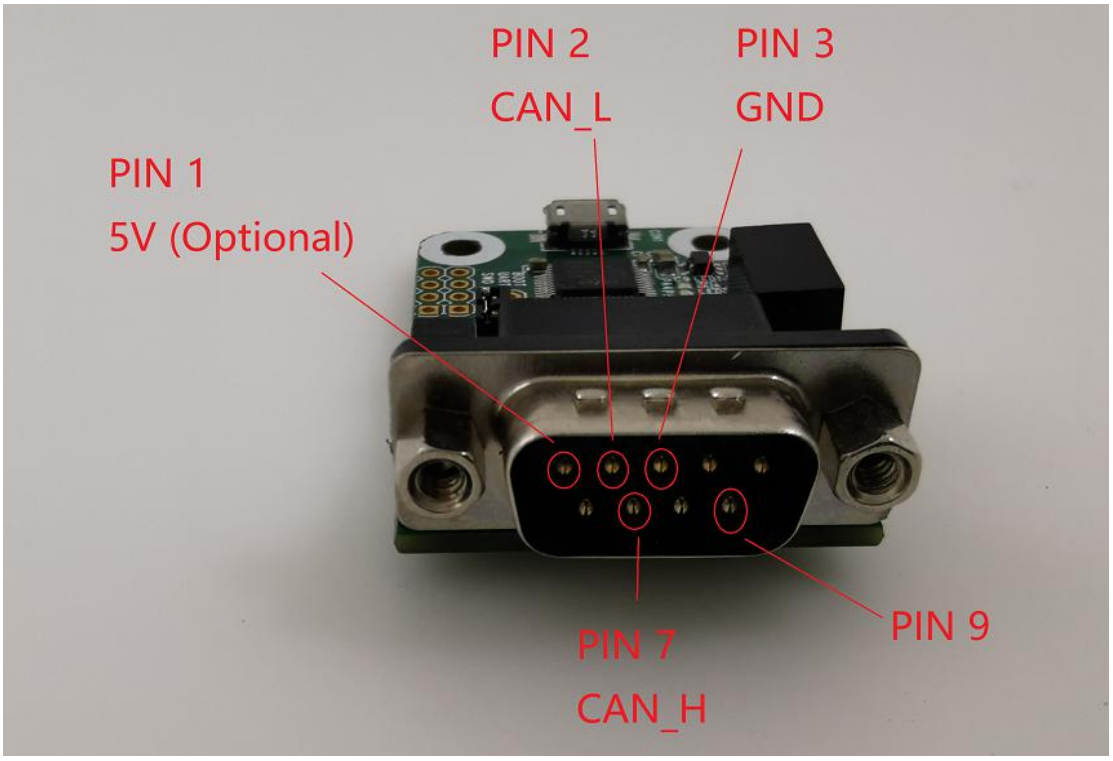
Connector	
CAN	D-SUB, 9 pins
USB	USB 2.0 Full-Speed, Micro USB
CAN Features	
Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN
Data Rate	From 20kbps to 1Mbps can be programmed arbitrarily.
Isolation Voltage	1.5K VDC/min, 3K VDC/1s
Microcontroller	STM32F0, 48MHz
Termination	120 Ohm resistor selectable jumper
CAN Transceiver	ISO1050DUBR ,Texas Instruments
Other	
Work Temperature	-40° ~ 85°
Relative humidity	15-90%, not condensing
Weight	15.5 g

4. Hardware Description

4.1 CAN Connector Pinout



Pin	Description
1	5V/150ma output . Weld 0Ω resistor on R9 to enable this function(close to the jumper).
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



4.2 120 Ohm Resistor Setting.

A High-speed CAN bus (ISO 11898-2) must be terminated on both ends with 120 Ohms. The USB2CAN module with a on-board 120Ω selectable jumper.

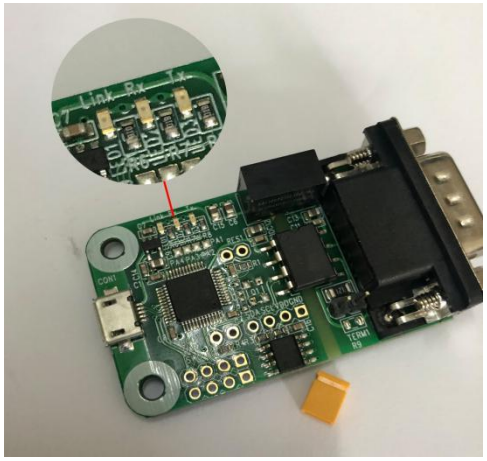
Disable 120 Ohm Resistor.

Enable120 Ohm Resistor.

Disable 120 Ohm Resistor.

Enable120 Ohm Resistor.

4.3 LED Indicate

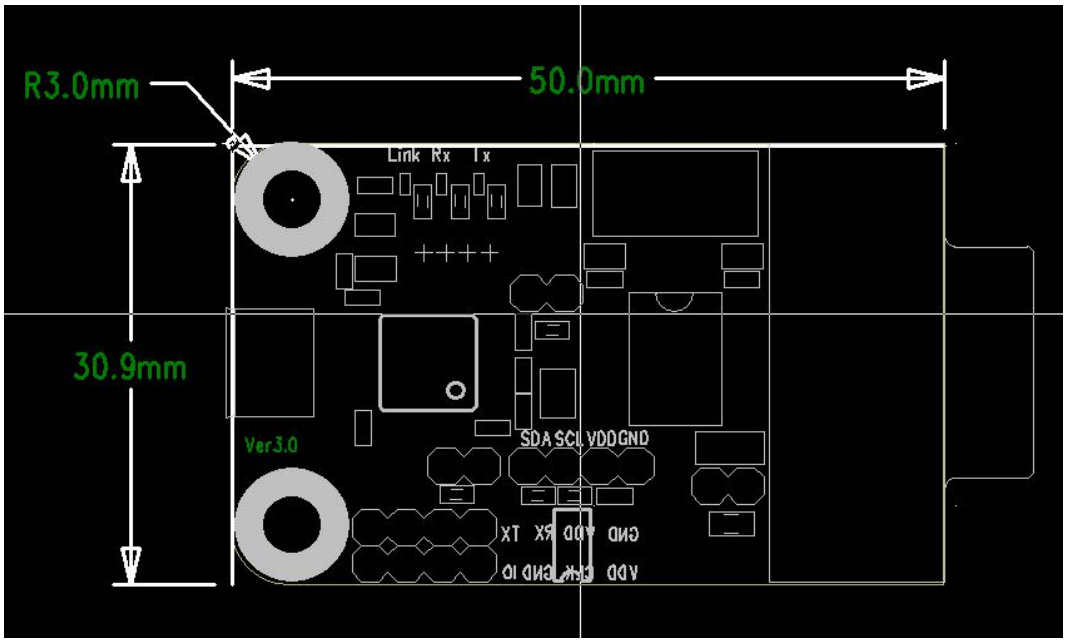


LED Name	Description
Link	Red led is normally on to indicate The module is came up without any errors
Tx	Red led flash to indicate send data.
Rx	Red led flash to indicate receive data.

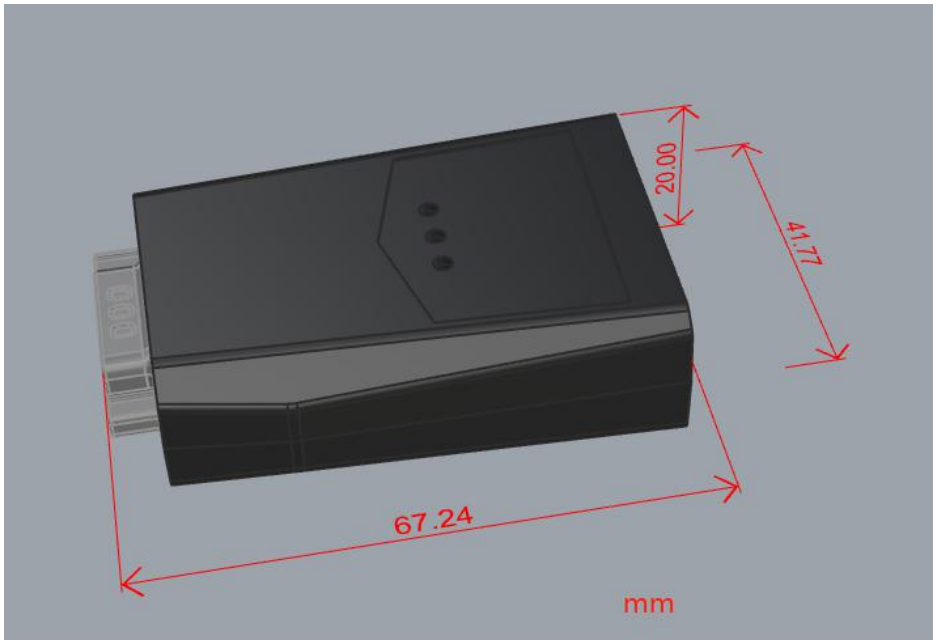
LED Name	Description
Link	Green led is normally on to indicate the module is came up with out any errors
Tx	Green led flash to indicate send data.
Rx	Green led flash to indicate receive data.

4.4 Dimension Figure

USB2CAN：



USB2CAN-C：

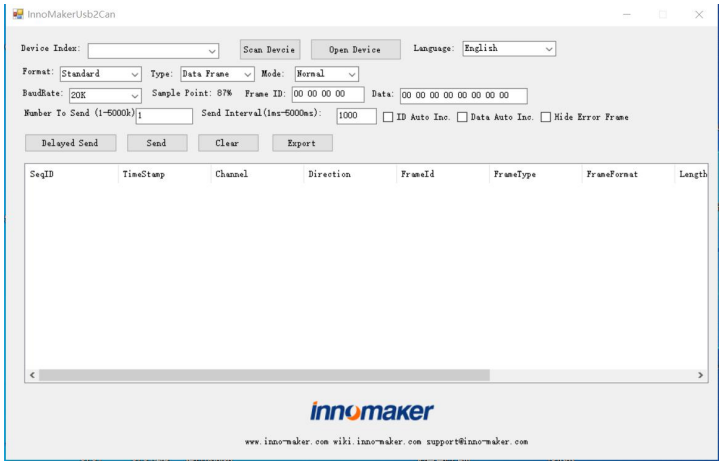


5. Windows/Mac OS Ready-Made Tools

Download the tools installation package from below link:
https://github.com/INNO-MAKER/usb2can

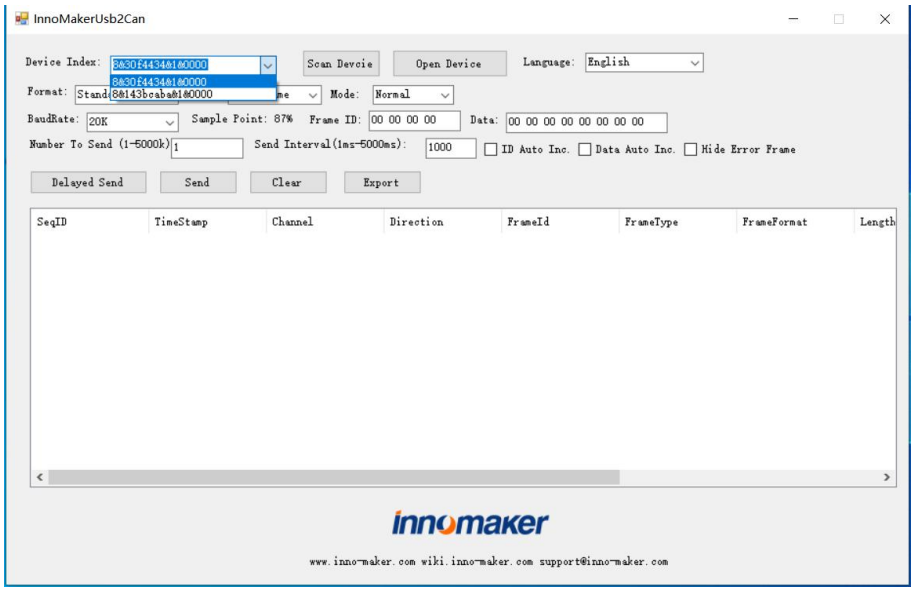
5.1 Open the USB2CAN tools.

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. You’ll Find USB2CAN device and named with different ID.



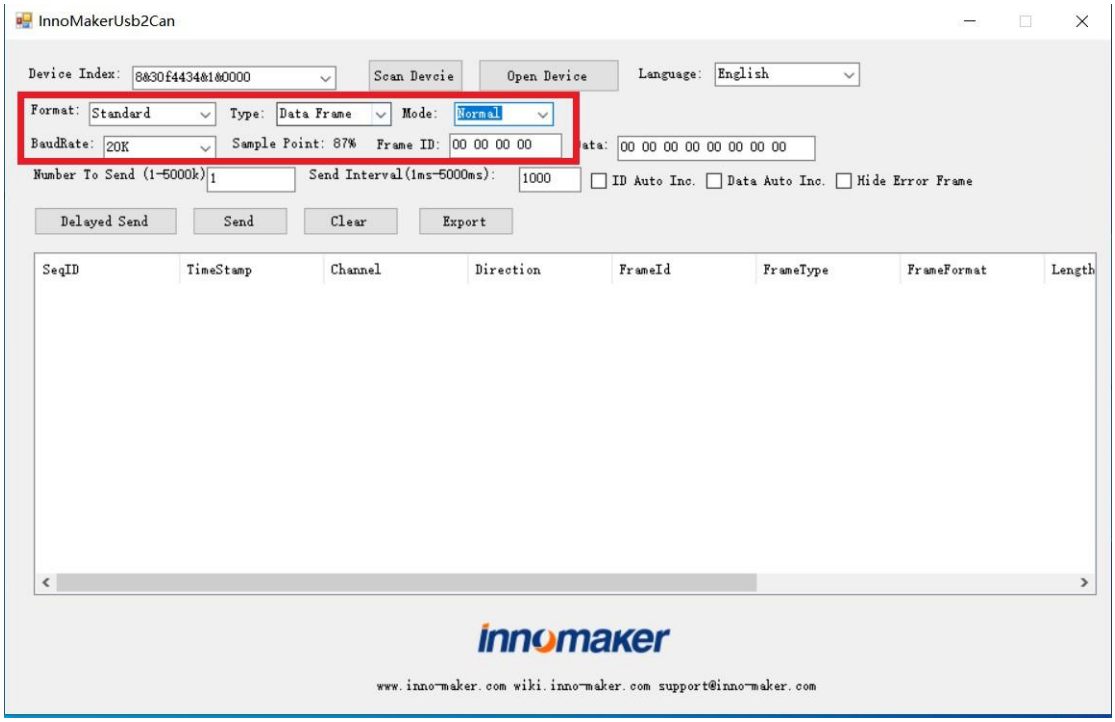
5.3 Setting The BandRate And Working Mode.

Setting below parameters in the red box before open device.

Normal mode: The CAN module will appear on the CAN-bus, and it can send and receive CAN messages， communication with other CAN devices directly.

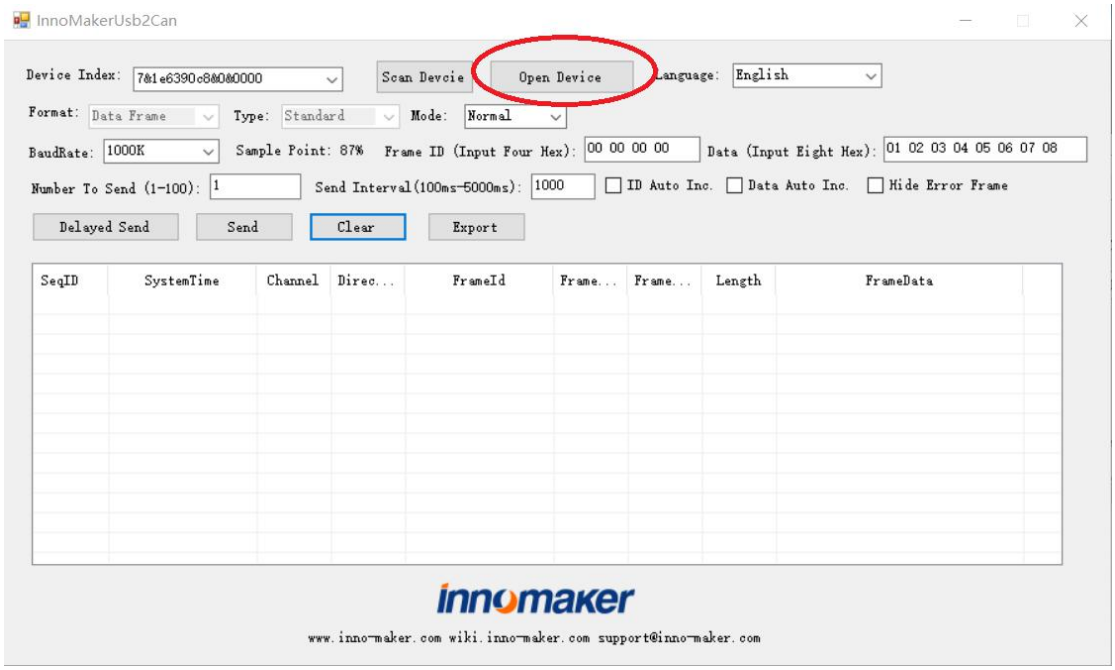
Silent mode: The module will appear on the CAN-bus, but in a passive state. It can receive CAN messages, but cannot transmit CAN messages or answer. This mode can be used as a bus monitor because it does not affect CAN-bus communications but can observe the CAN-bus states.

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.



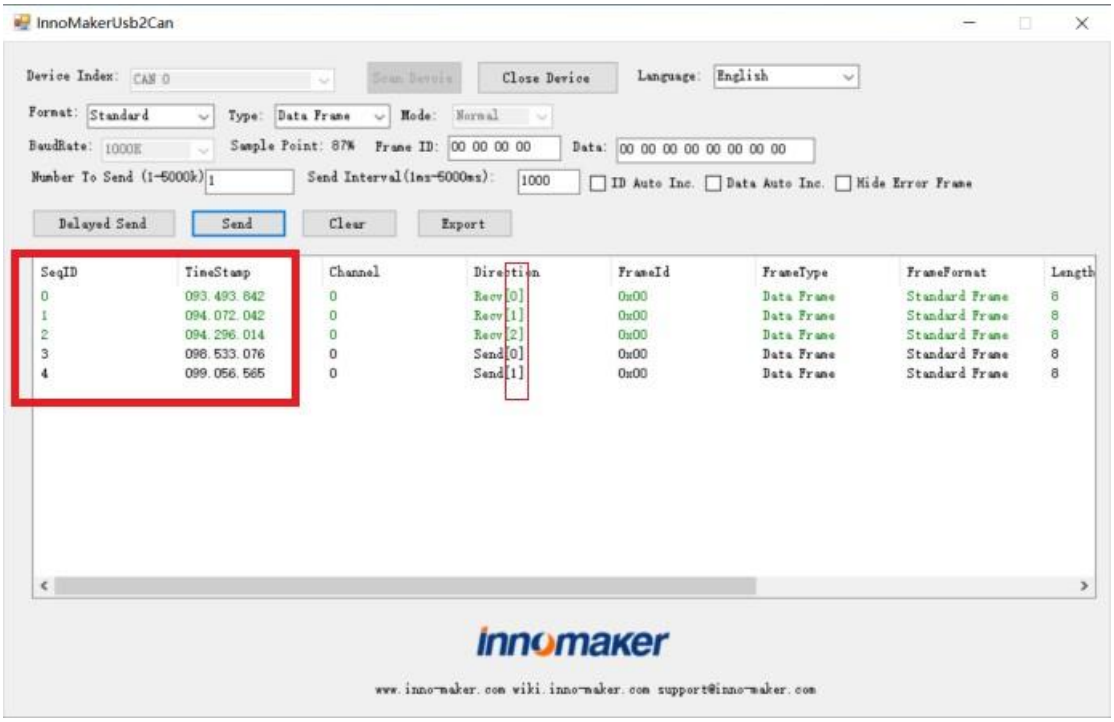
5.4 Enable Device.

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



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.



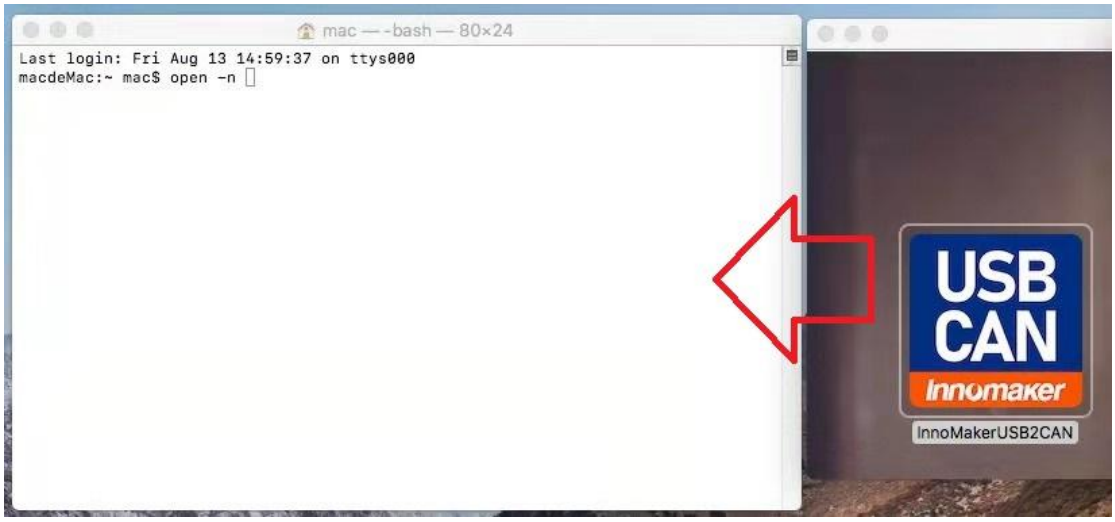
5.6 Windows Real Time Setting

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 Manger
→ right click on innomaker application process
→ left click on 'go to details'
→ right click on innomaker application process
→ left click on 'set priority'
→left click on 'realtime'

5.7 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.

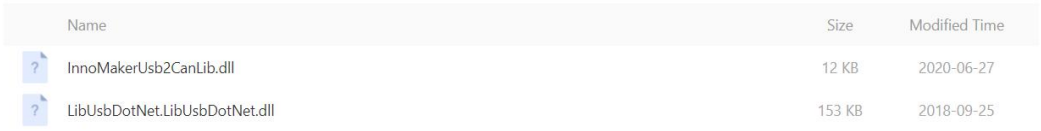


6. Windows API

If you are not familiar with the CAN communication and WINDOWS software development, it is strongly recommended you that use the ready-made application we provide or entrust us with the development.

Released Date: 2021-07-16
Version No: 1.2.0

6.1 Dynamic Link Libraries



InnoMakerUsb2CanLib.dll: USB2CAN function dynamic link libraries.

LibUsbDotNetLibUsbDotNet.dll : WINDOWS USB Universal Interface dynamic link libraries.

6.2 Structure

```
public struct innomaker_host_frame
 {
        public  UInt32  echo_id;//The echo-id identifies the frame from (echos the id from a
                                //previous UCAN_OUT_TX message).

       public UInt32 can_id;          //CAN ID    Reserved for dual CAN Deivce
       public Byte can_dlc;           // data len      0~8
       public Byte channel;          // channel number，
       public Byte flags;              //additional flags
       public Byte reserved;       //reserved / padding
       [MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
       public Byte[] data;              //CAN data
       public UInt32 timestamp_us;    //times stamp
    }
```

```
public struct innomaker_device_bittming
{
    public UInt32 prop_seg;          //propagation Segment
    public UInt32 phase_seg1;        //phase segment 1 (1~15)
    public UInt32 phase_seg2;        //phase segment 2 (1~8)
    public UInt32 sjw;               //synchronization segment (1~4)
    public UInt32 brp;               //clock divider, USB2CAN moduel clock    is 48M
}
```

```
public enum UsbCanMode
{
    UsbCanModeNormal,          //Normal working mode
    UsbCanModeLoopback,        // Loopback mode
    UsbCanModeListenOnly,      // Listen only, not ACK
}
```

6.3 CallBack

(1) public delegate void AddDeviceNotifyDelegate() ;

-Summary: If Device Plug In, it will call the delegate

(2) RemoveDeviceNotifyDelegate();

public delegate void RemoveDeviceNotifyDelegate();

-Summary: If Device Plug Out, it will call the delegate

6.4 Function

(1) GetDllVersion

public String GetDllVersion()
-Summary: Return Current Dll Version
-Return: Current Dll Version

(2) scanInnoMakerDevices

public bool scanInnoMakerDevices()
-Summary: Scann Inno Maker Devices
-Return: Scan success return true , else return false

(3) getInnoMakerDeviceCount

public int getInnoMakerDeviceCount()
-Summary: Get Device Count
-Return: Device count

(4) getInnoMakerDevice

public InnoMakerDevice getInnoMakerDevice(int devIndex)
-Summary: Get Inno Maker device by device index
-devIndex: Device index
-return: Inno Maker Device Instance

(5) openInnoMakerDevice

public bool openInnoMakerDevice(InnoMakerDevice device)
-Summary: Open Device
-param: device
-return: if open success return true,else return false

(6) closeInnoMakerDevice

public bool closeInnoMakerDevice(InnoMakerDevice device)
-Summary: Close Device
-param: device
-return: if Close success return true, else return false

(7) asyncGetInnoMakerDeviceBuf

public bool asyncGetInnoMakerDeviceBuf(InnoMakerDevice device,
Byte[] buf,
int size,
int transferredIn,
int timeout)
-Summary: Read buffer from device async
-param: device
-param: buf, buffer reads in
-param: size, buffer size
-param: transferredIn, actually buffer length reads
-param: timeout, read buffer timeout,This is specified in milliseconds
-return: if read device success,return true,else return false

(8) syncGetInnoMakerDeviceBuf

public bool syncGetInnoMakerDeviceBuf(InnoMakerDevice device,
Byte[] buf,
int size,
int transferredIn,
int timeout)
-Summary: Read buffer from device sync
-param: device
-param: buf, buffer reads in
-param: size, buffer size
-param: transferredIn, actually buffer length reads
-param: timeout, read buffer timeout, This is specified in milliseconds.
-return: if read device success,return true,else return false

(9) asyncGetInnoMakerDeviceBuf

public bool asyncGetInnoMakerDeviceBuf(InnoMakerDevice device,
Byte[] buf,
int size,
int timeout,int transferredOut)

-Summary: write buffer to device async
-param: device
-param: buf, buffer writes out
-param: size, buffer size
-param: transferredOut, actually buffer length writes
-param: timeout, write buffer timeout,This is specified in milliseconds.
-return: if write device success,return true,else return false

(10) syncGetInnoMakerDeviceBuf

public bool syncGetInnoMakerDeviceBuf(InnoMakerDevice device,
Byte[] buf,
int size,
int timeout,
int transferredOut)

-Summary: write buffer to device sync
-param: device
-param: buf, buffer writes out
-param: size, buffer size
-param: transferredOut, actually buffer length writes
-param: timeout, write buffer timeout
-return: if worte success,return true,else return false

(11) UrbResetDevice

public bool UrbResetDevice(InnoMakerDevice device)
-Summary: Reset Device
-param: Device Instance
-return: If reset device success return true,else return false

(12) UrbSetupDevice

public bool UrbSetupDevice(InnoMakerDevice device,
UsbCanMode canMode,
innomaker_device_bittming bittming)

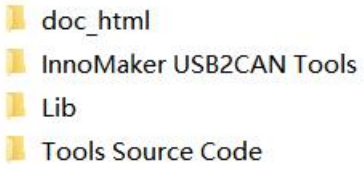
-Summary: Setup device
-param: Device Instance
-param: canMode, usbCanMode
-param: bittming, usb bittming params
-return: if setup device success return true,else return false

7. Mac Os API

Download the tools and SDK of Mac Os from below link:
https://www.jianguoyun.com/p/DVvY6VIQpdSrBxjyqKUB

https://github.com/INNO-MAKER/usb2can

There are consists of four parts:



Folder name	Description
InnoMaker USB2CAN Tools	InnoMaker USB2CAN 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.

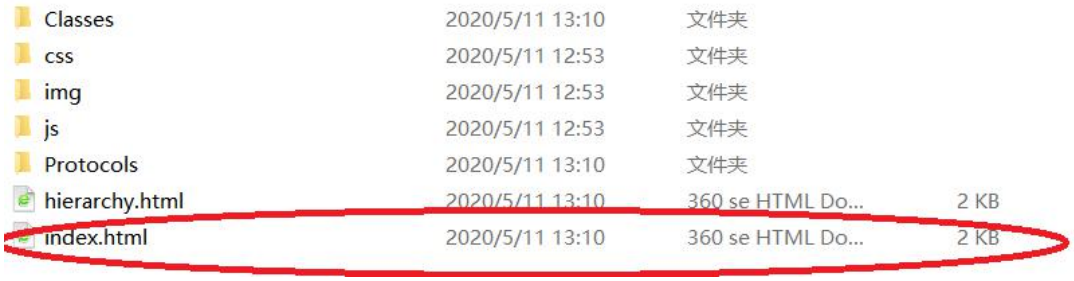
7.1 Library

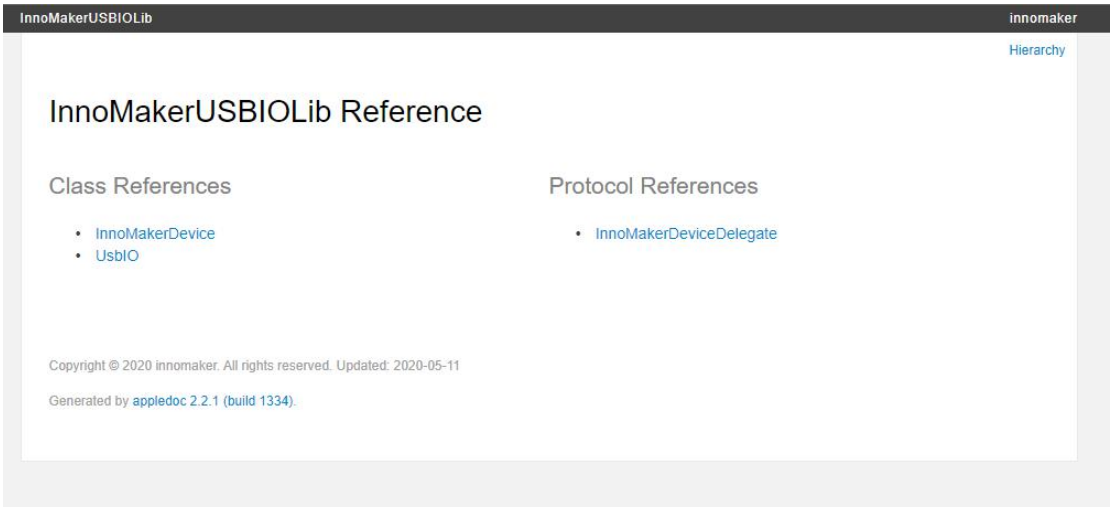
Open the Lib folder, There are three files in it:

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

7.2 Document

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




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).

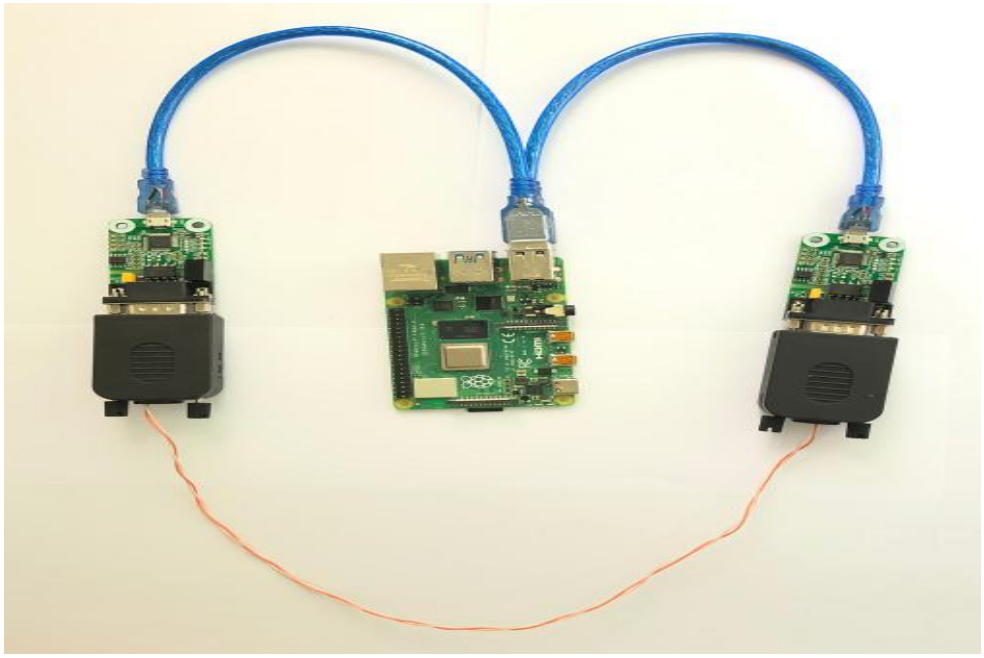
8. Linux/Ubuntu/Raspbian

8.1 Run Test Demo

USB2CAN device can run properly without any additional driver request on all Linux system since version 3.9. such as Ubuntu, Debian and Raspbian. If you meet driver problem on the old version You need to reconfigure the kernel drivers. Enable 'gs_usb.c' and install 'gs_usb.ko' into system. So notice that if you only compile this drivers, It may fail to load in system. At this time, compile fully with new configures. If you have any question, feel free to e-mail to our support team(support@inno-maker.com).

You can test the USB2CAN device with all single board computer or PC with the right Linux version. We take Raspberry Pi 4 as an example to show you how to run the C/Python and can-utils demo.

For demonstration purposes, I take two USB2CAN device to show the communication.



Please download the codes and tools from below links:
https://github.com/INNO-MAKER/usb2can

8.1.1 Preparatory work

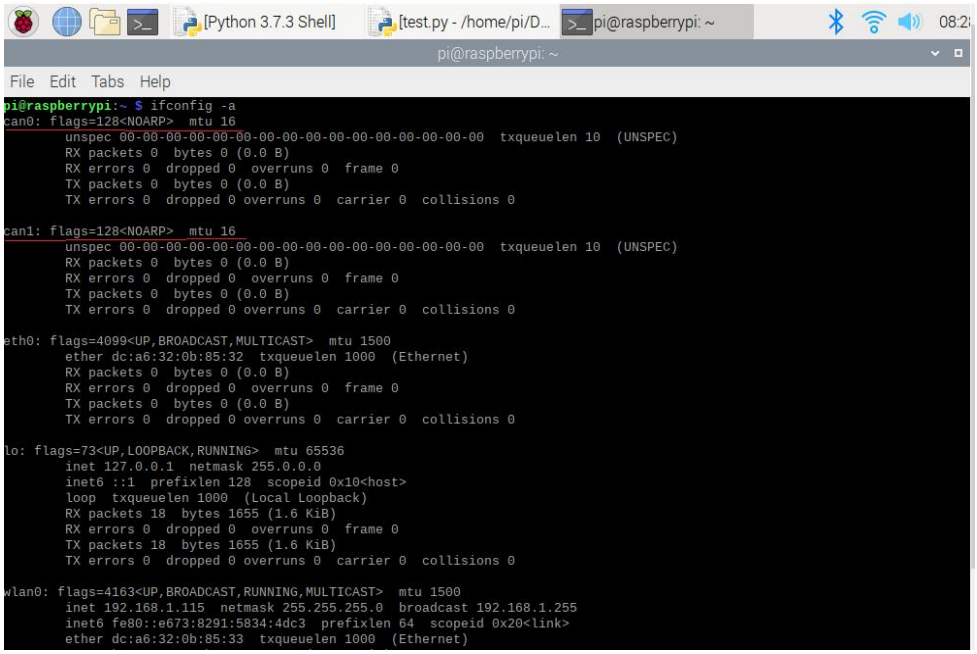
8.1.1.1 Connection

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)

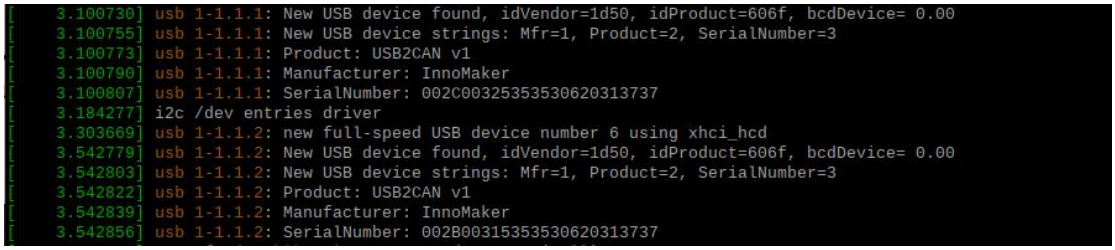
8.1.1.2 ifconfig -a

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



8.1.1.3 dmesg

You can type command 'dmesg' for see more information about USB2CAN module at the bottom.



8.1.2 Use 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/

8.1.2.1 -Install Tools

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

8.1.2.2 -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.

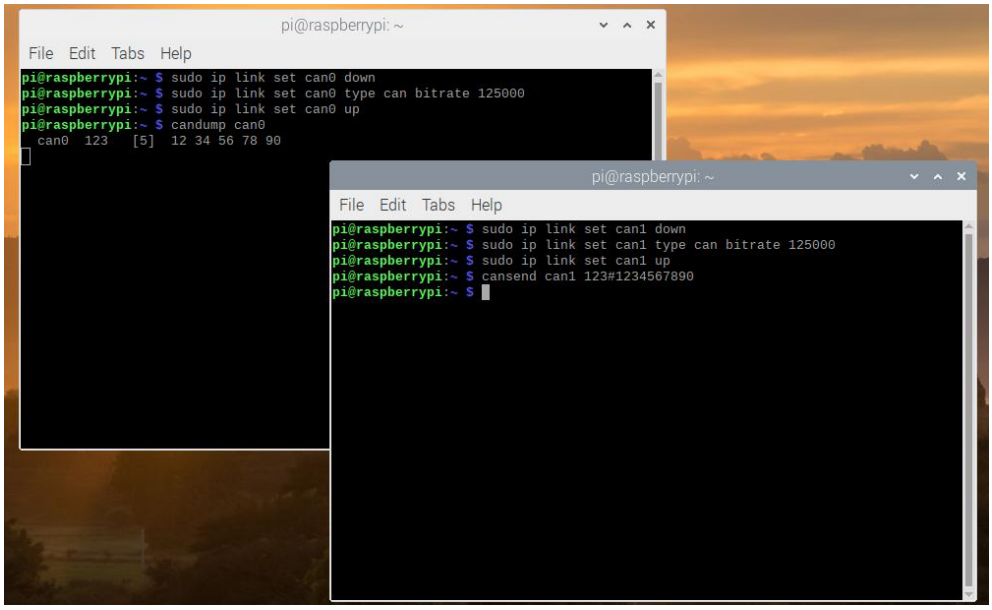
```
sudo ip link set can0 down
sudo ip link set can0 type can bitrate 125000
sudo ip link set can0 up

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



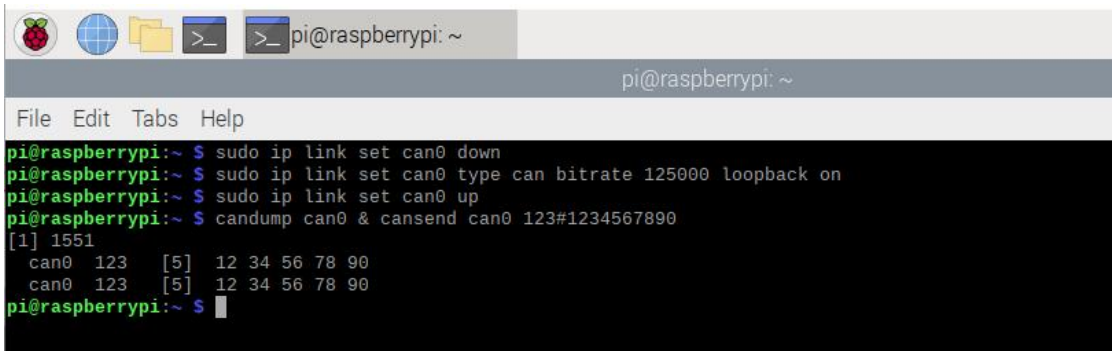
8.1.2.3 -Loopback Mode Test

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



8.1.3 Run C Demo

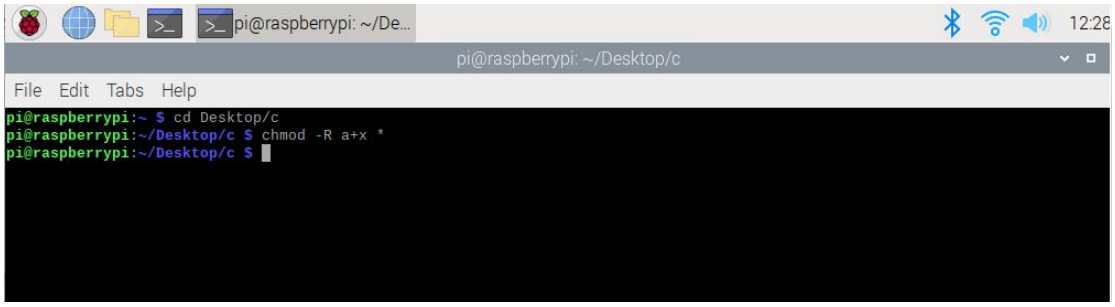
(1) Load C Demo named 'usb2cantest' from our Wiki and up-zip it to the desktop of Raspbian.

http://www.inno-maker.com/wiki/doku.php?id=usb_can

Or **http://www.inno-maker.com/wiki/doku.php**

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

```
chmod -R a+rwx *
```

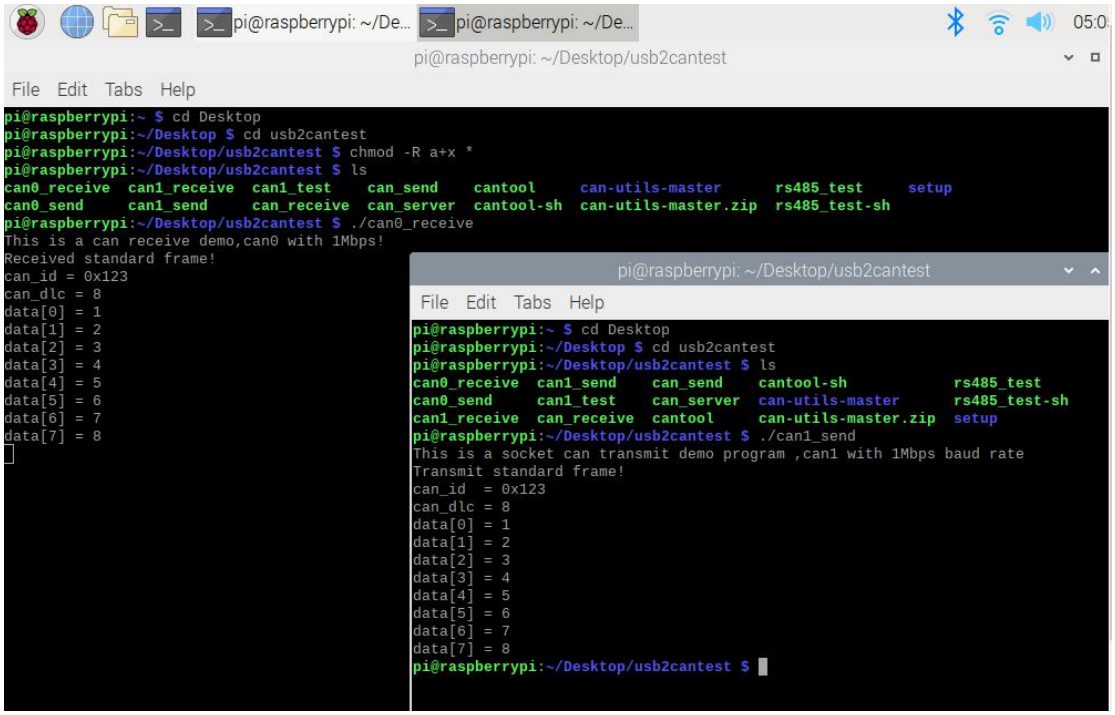


(3) Set one as receiver, execute following commands in serial terminal. Now this Raspberry pi is blocked.

```
./can0_receive
```

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

```
./can1_send
```



8.1.4 Run Python3 Demo

Download Python Demo named ‘python3’ from our Wiki and up-zip it to Desktop(or wherever you want put it).

http://www.inno-maker.com/wiki/doku.php?id=usb_can

Or **http://www.inno-maker.com/wiki/doku.php**

There are three files in the ‘c’ folder. send.py and receive.py is for testing send and receive function separately and test.py is use for automatic testing.

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

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

(4) Install Python CAN library.

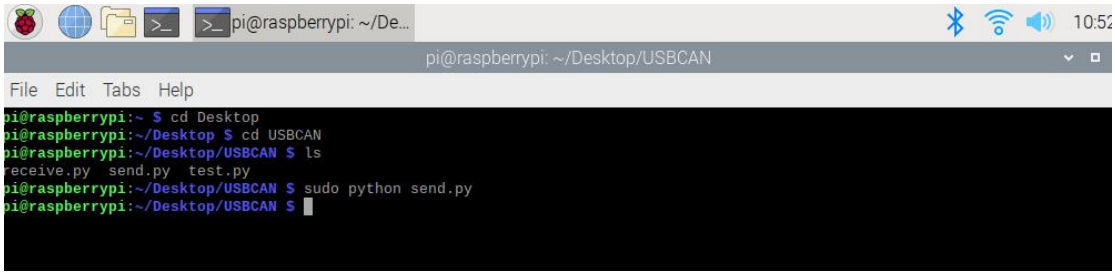
```
sudo pip3 install python-can
```

(5) Set one Raspberry Pi as receiver.

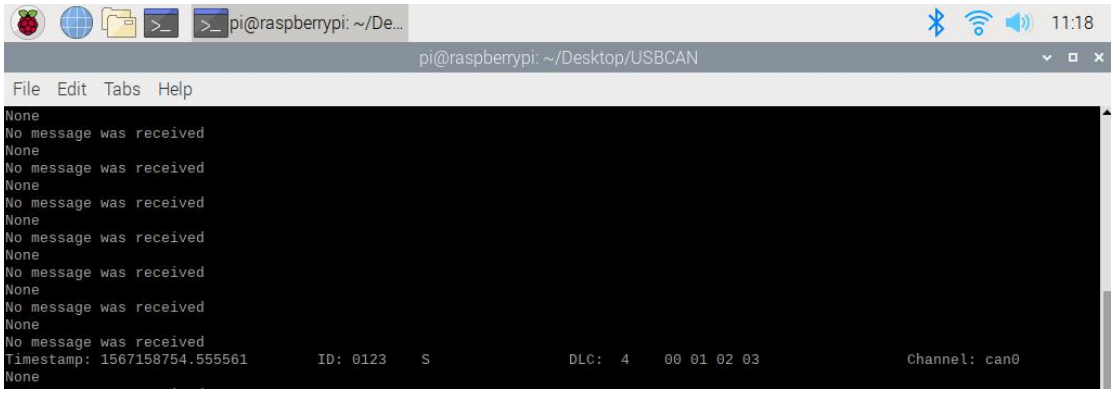
```
sudo python3 receive.py
```

(6) Set the other as sender.

```
sudo python3 receive.py
```

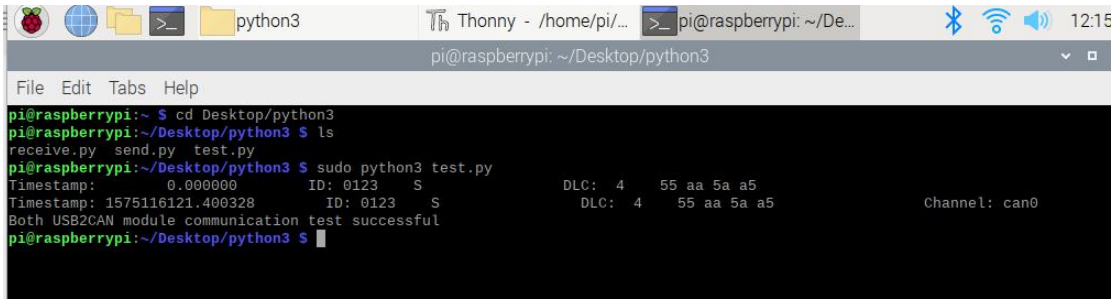


(7) You will see the data received.



(8) If you use one Raspberry Pi and two USB2CAN modeule for testing. Run 'test.py' and check the result.

```
sudo python3 test.py
```


python3
Thonny - /home/pi/...
pi@raspberrypi: ~/De...
12:15
pi@raspberrypi: ~/Desktop/python3
File Edit Tabs Help
pi@raspberrypi:~ $ cd Desktop/python3
pi@raspberrypi:~/Desktop/python3 $ ls
receive.py send.py test.py
pi@raspberrypi:~/Desktop/python3 $ sudo python3 test.py
Timestamp: 0.000000 ID: 0123 S DLC: 4 55 aa 5a a5
Timestamp: 1575116121.400328 ID: 0123 S DLC: 4 55 aa 5a a5 Channel: can0
Both USB2CAN module communication test successful
pi@raspberrypi:~/Desktop/python3 $

8.2 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(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

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 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));
```

8.2.2 Programming in Python

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) 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')
```

9. 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.

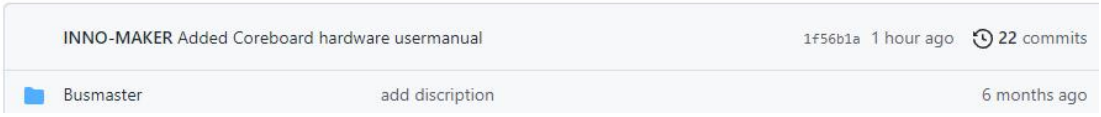
Home Page:
https://rbei-etas.github.io/busmaster/

Project:
https://github.com/rbei-etas/busmaster

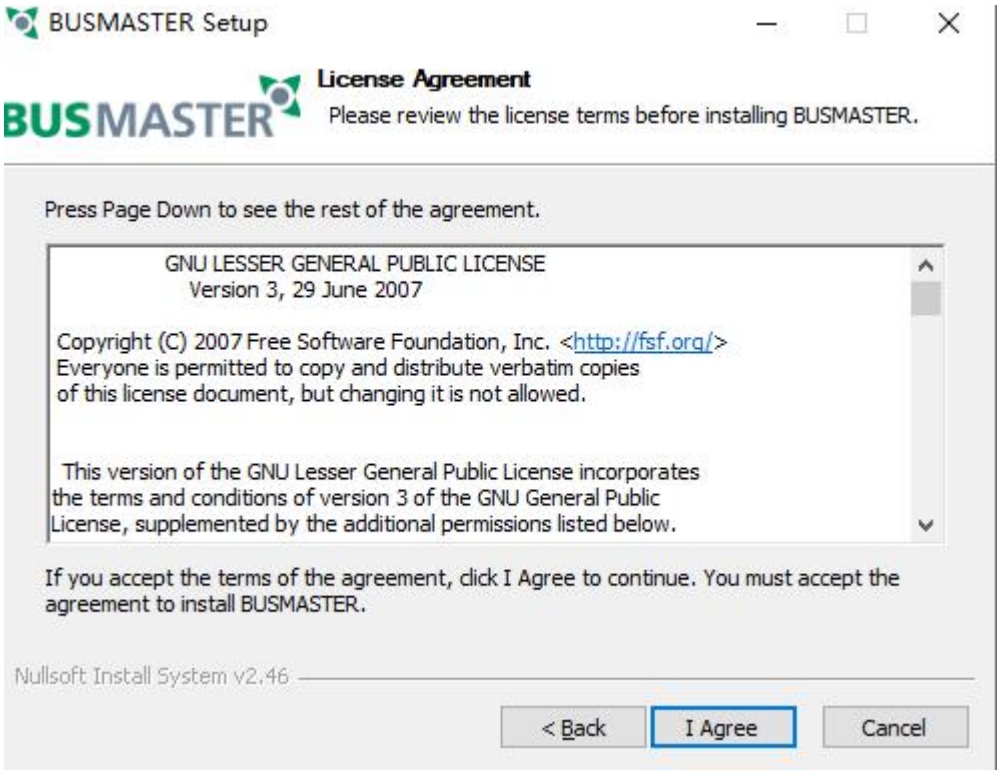
Video for BUSMASTER:
https://www.youtube.com/user/busmasteropensource/featured

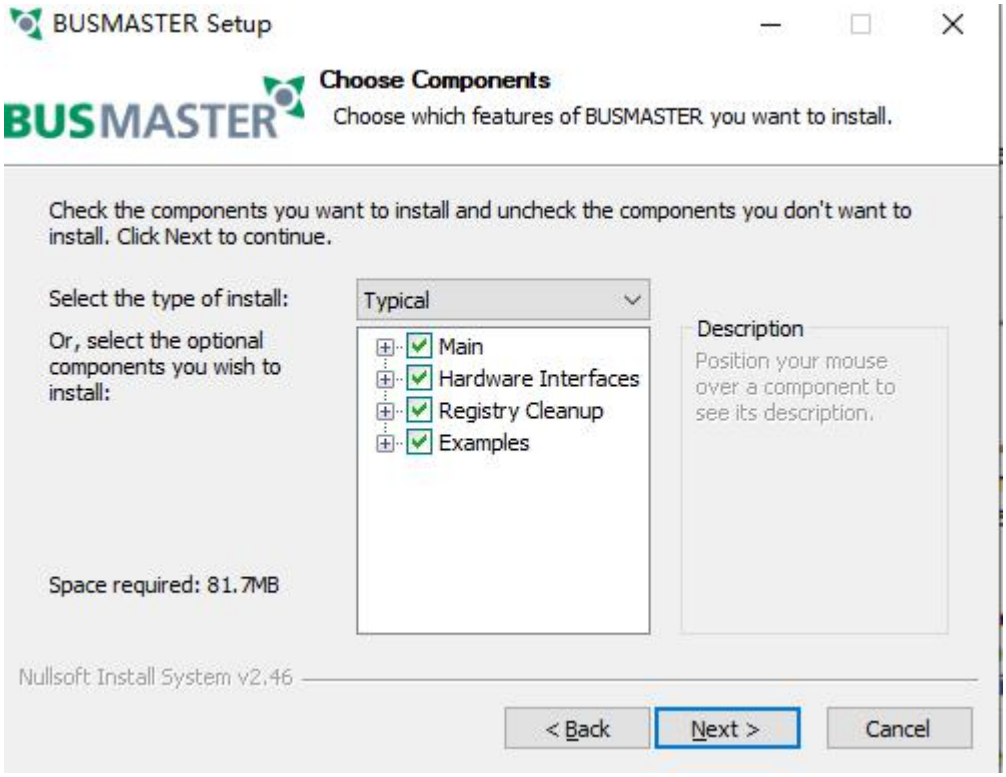
9.1 Download and Install

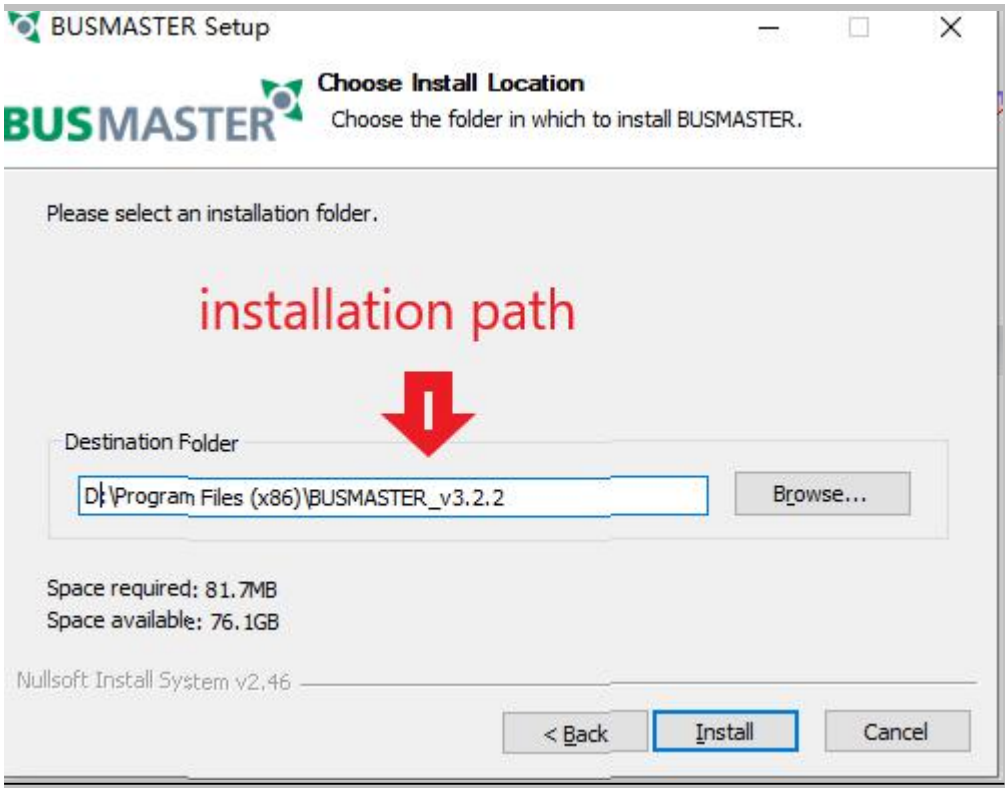
(1)Download the from our github:
https://github.com/INNO-MAKER/usb2can



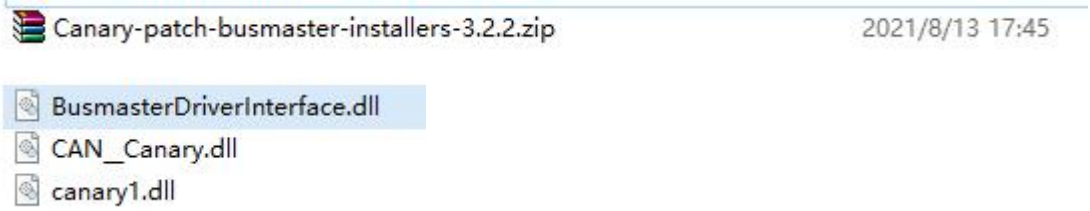
(2) unzip the installation package and setup


BUSMASTER Setup
BUSMASTER
License Agreement
Please review the license terms before installing BUSMASTER.
Press Page Down to see the rest of the agreement.
GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install BUSMASTER.
Nullsoft Install System v2.46
< Back
I Agree
Cancel


BUSMASTER Setup
BUSMASTER
Choose Components
Choose which features of BUSMASTER you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select the type of install:
Typical
Or, select the optional components you wish to install:
Main
Hardware Interfaces
Registry Cleanup
Examples
Description
Position your mouse over a component to see its description.
Space required: 81.7MB
Nullsoft Install System v2.46
< Back
Next >
Cancel



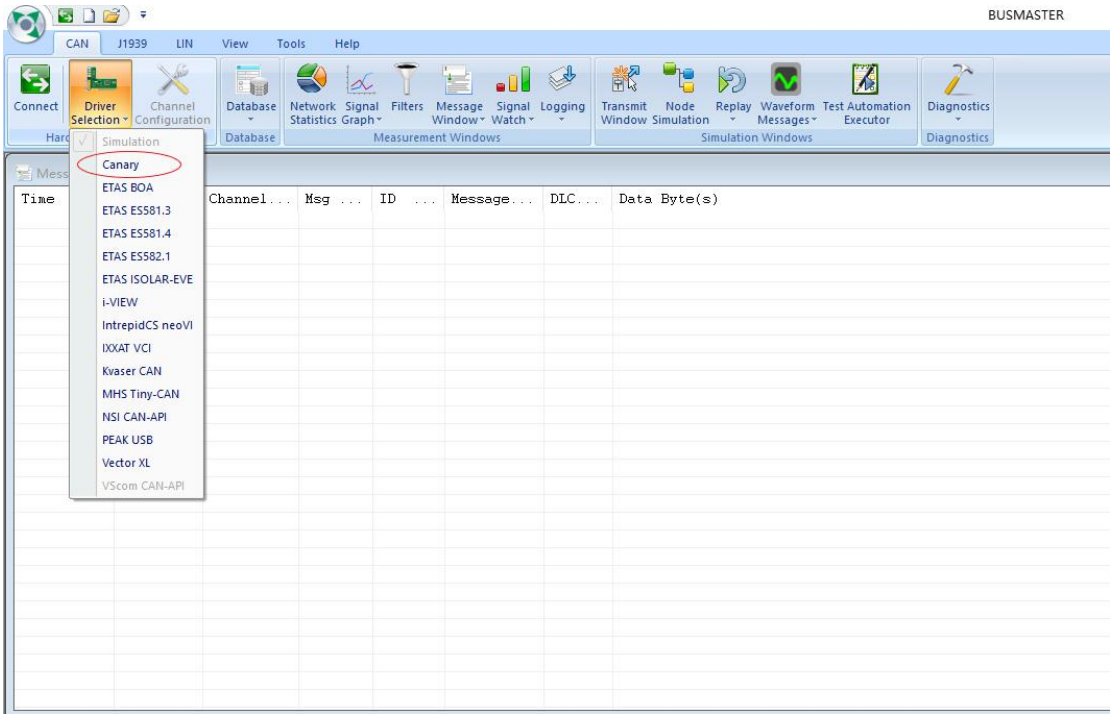
(3) unzip below service pack and past these three files into the installation path of BUSMASTER v3.2.2. Replace the files in the target.



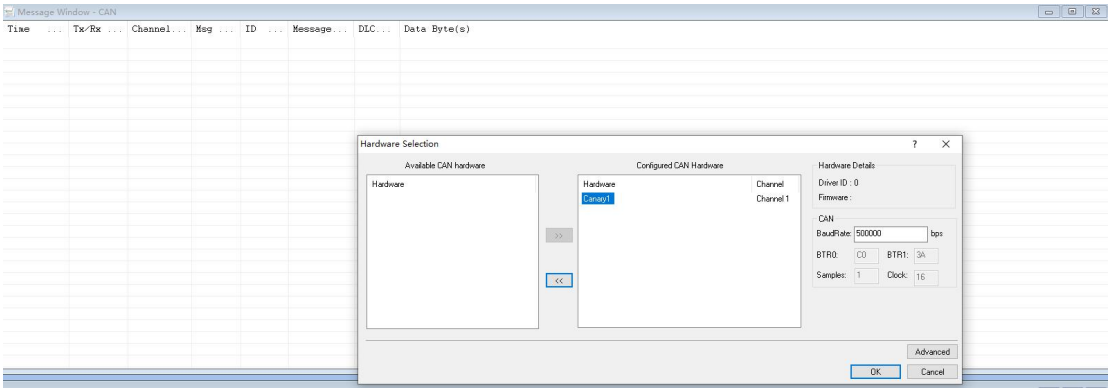
9.2 Using The Software

(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.

10. Error Frame

You may receive some error frame marked in red when you use the USB2CAN module. They will Let you know swhat problem does the USB2CAN 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 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		2019.02.02	support@inno-maker.com sales@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 ON WINDOWS	2022.01.06	**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 and record your name and E-mail in list. Look forward to your letter and kindly share.