EMUC-B202 / EGPC-B201

USB/M.2 to dual isolated CANbus 2.0B/J1939/CANopen

User Manual

Rev 1.8

Revision History

Revision	Date	Description
1.0	2017/08/18	Initial Release
1.1	2017/09/19	Modify "NOTE" of 3.2.3, 3.2.4 inactive to active.
1.2	2017/10/18	Modify 4.3 for SocketCAN driver version 2.1.
1.3	2018/07/16	1. Update Linux COM port support table in 6.1 COM Port Selection. (ttyS0-ttyS15 -> ttyCAN0-ttyCAN15) 2. Add new API functions. ● EMUCEnableSendQueue ● EMUCSetRecvBlock ● EMUCOpenSocketCAN ● EMUCGetBusError
1.4	2019/08/08	Modify "data_err" register link of 6.2.16 from 4.2 to 8.2.
1.5	2020/4/7	Add 7.3 CANopen Sample Code
1.6	2020/6/23	1. Add 4.3.3 boot up script 2. Modify canutils testing picture and add shell scripts in 4.3.2
1.7	2020/8/13	Modify 4.3 SocketCAN installation process
1.8	2021/3/4	1. Modify description of 1. Introduction 2. Add new API function ● EMUCOpenDeviceSCT 3. Add 6.2.15 queue size (10-10000) 4. Modify 7.1 description 5. Add 4.3.5 CAN Error Frame 6. Change the start.sh, run_emucd and help screensot in 4.3 7. 2. Hardware Installation and 3. Windows OS driver installation divide into EMUC-B202 and EGPC-B201

Table of Contents

1. Introduction

We provide basic CAN 2.0B, J1939 and CANopen API for application programming in Windows and Linux.

The following table shows the corresponding model to these API which can be used.

Part Number	Basic CAN 2.0B	J1939	CANopen
EMUC-B202-W1	Yes	No	No
EMUC-B202-W2	Yes	Yes	No
EMUC-B202-W3	Yes	No	Yes

Factory default setting

Baud Rate	500 Kbps
CANbus Mode	Normal mode
Filter Type	None
Filter ID	None
Filter mask	None
Error Setting	EEPROM only

Supported Operation System

Windows	XP(32bit), 7(32/64bit), 8/8.1(32/64bit), 10(32/64bit)
Linux (cdc-acm driver)	Kernel 2.6 and above, 32/64bit
Linux (SocketCAN driver)	Kernel 2.6.38 and above, 32/64bit
QNX	6.6, 7.1

2. Hardware Installation

2.1. EMUC-B202

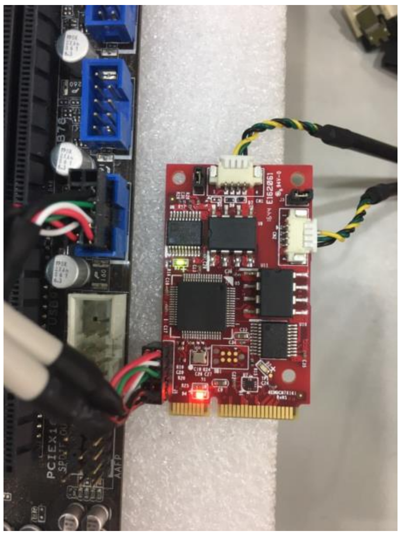
EMUC-B202 CANbus module uses USB 2.0 input interface, there are dual options to install the module.

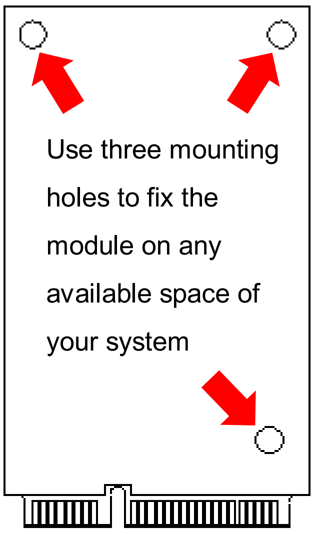
2.1.1. mPCIe Slot

Install the module to mPCIe slot which has USB 2.0 interface.

2.1.2. USB Pin Header

Don’t need to connect mPCIe golden finger, it can be connected through USB pin headers on the PCB to the motherboard. Then use three mounting holes to fix the module on any available space of your system.

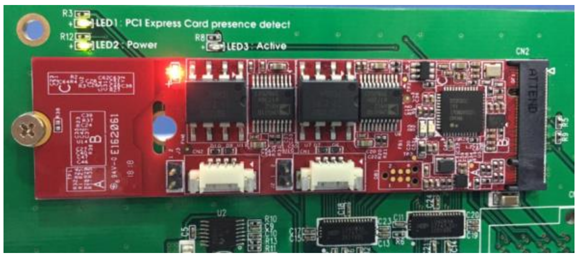



Use three mounting holes to fix the module on any available space of your system

NOTE: *This USB cable in the picture is not included in the package; you need to design your own USB cable.*

2.2. EGPC-B201

Install the module to M.2 B-M key slot which has PCIe interface.

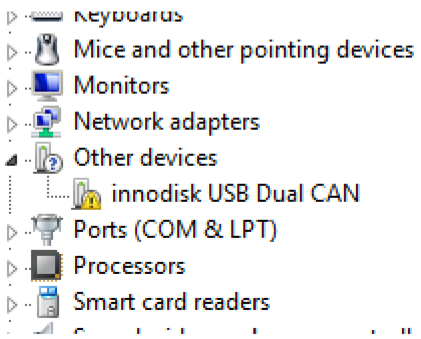


3. Windows OS

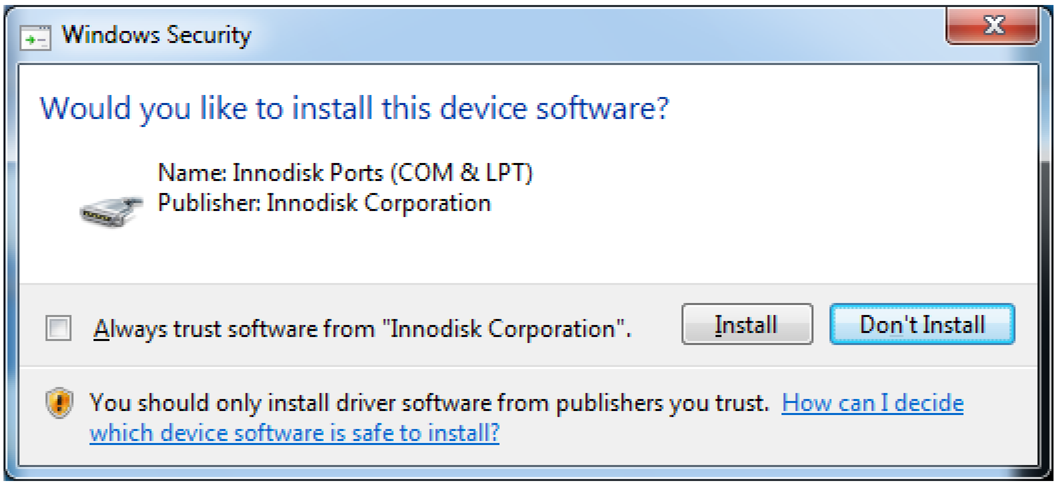
3.1. EMUC-B202 Driver Installation

Install EMUC-B202 either into mPCIe slot or with USB pin header. The device named "innodisk USB Dual CAN" can be found in "Device Manager".

Run the driver package as administrator.


Keyboards
Mice and other pointing devices
Monitors
Network adapters
Other devices
innodisk USB Dual CAN
Ports (COM & LPT)
Processors
Smart card readers

When prompt "Windows Security" dialog, click "Install".


Windows Security

Would you like to install this device software?

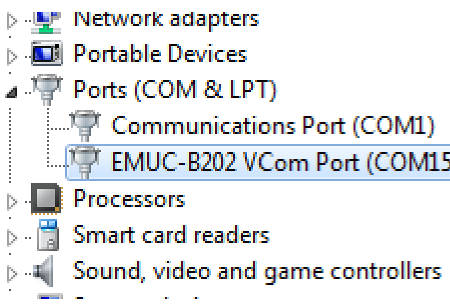
Name: Innodisk Ports (COM & LPT)
Publisher: Innodisk Corporation

Always trust software from "Innodisk Corporation".

Install | Don't Install

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

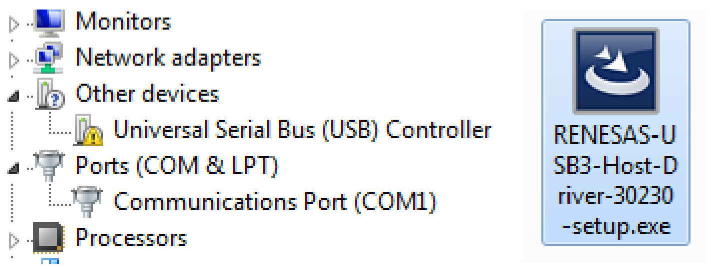
After installing the driver, device can be recognized as a COM port named "EMUC-B202 VCom Port".


Network adapters
Portable Devices
Ports (COM & LPT)
Communications Port (COM1)
EMUC-B202 VCom Port (COM15)
Processors
Smart card readers
Sound, video and game controllers

3.2. EGPC-B201 Driver Installation

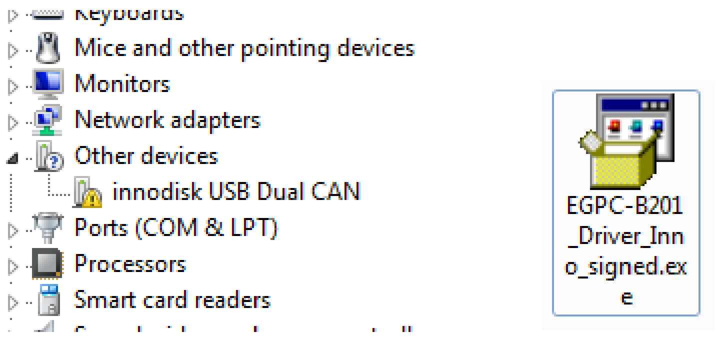
There are two drivers need to be installed.

The device named “Universal Serial Bus (USB) Controller” can be found in “Device Manager”, run the USB driver package first as administrator.

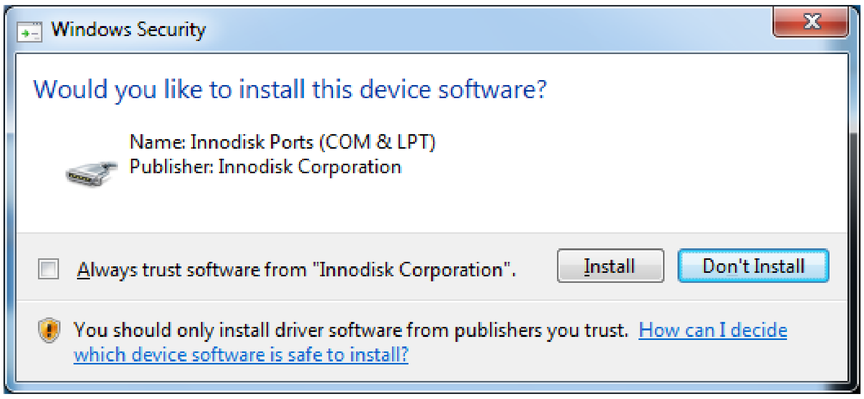

Monitors
Network adapters
Other devices
Universal Serial Bus (USB) Controller
Ports (COM & LPT)
Communications Port (COM1)
Processors
RENESAS-USB3-Host-Driver-30230-setup.exe

After installing USB driver, the device named “innodisk USB Dual CAN” can be found in “Device Manager”.

Run the CAN driver package as administrator.



When prompt “Windows Security” dialog, click “Install”.


Windows Security

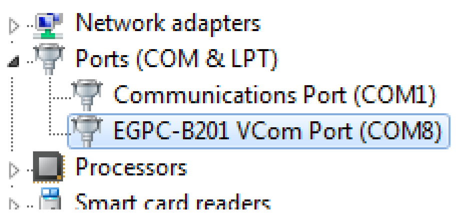
Would you like to install this device software?

Name: Innodisk Ports (COM & LPT)
Publisher: Innodisk Corporation

Always trust software from "Innodisk Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

After installing the two drivers, device can be recognized as a COM port named “EGPC-B201 VCom Port”.


Network adapters
Ports (COM & LPT)
Communications Port (COM1)
EGPC-B201 VCom Port (COM8)
Processors
Smart card readers

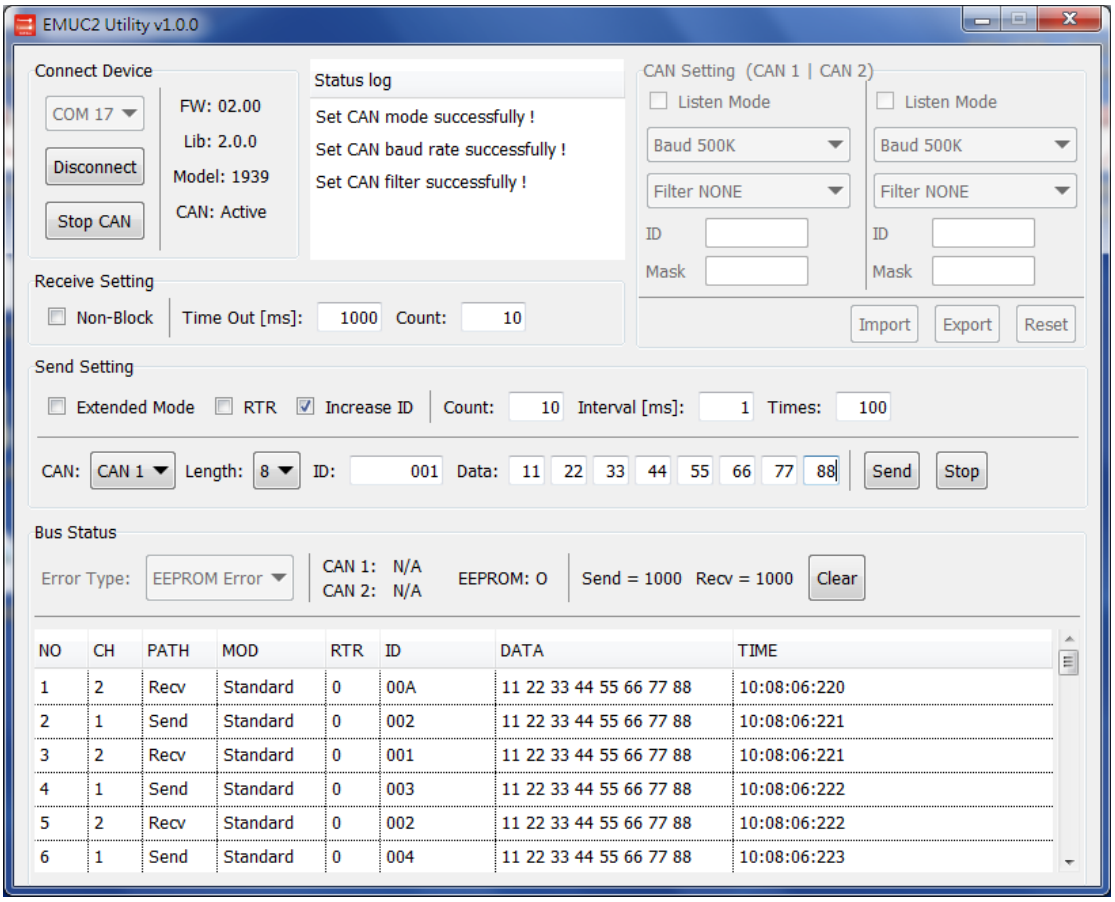
NOTE: *Drivers for Windows 64bit need to be signed with digital certificate. The driver is signed with SHA-2 certificate*

Windows 7 must be installed the hotfix KB3033929 to support SHA-2 code signing.

https://technet.microsoft.com/en-ie/library/security/3033929

3.3. Basic CAN 2.0B Test Utility

You can use this GUI utility to test EMUC-B202 for sending/receiving basic CAN frames.



Connect Device: COM 17 — FW: 02.00, Lib: 2.0.0, Model: 1939, CAN: Active

Status log:
```
Set CAN mode successfully !
Set CAN baud rate successfully !
Set CAN filter successfully !
```

CAN Setting (CAN 1 | CAN 2): Listen Mode ☐ / ☐; Baud 500K / Baud 500K; Filter NONE / Filter NONE; ID; Mask

Receive Setting: ☐ Non-Block, Time Out [ms]: 1000, Count: 10

Send Setting: ☐ Extended Mode, ☐ RTR, ☑ Increase ID, Count: 10, Interval [ms]: 1, Times: 100

CAN: CAN 1, Length: 8, ID: 001, Data: 11 22 33 44 55 66 77 88

Bus Status: Error Type: EEPROM Error, CAN 1: N/A, CAN 2: N/A, EEPROM: O, Send = 1000, Recv = 1000

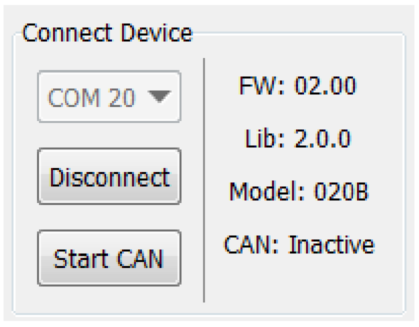
NO	CH	PATH	MOD	RTR	ID	DATA	TIME
1	2	Recv	Standard	0	00A	11 22 33 44 55 66 77 88	10:08:06:220
2	1	Send	Standard	0	002	11 22 33 44 55 66 77 88	10:08:06:221
3	2	Recv	Standard	0	001	11 22 33 44 55 66 77 88	10:08:06:221
4	1	Send	Standard	0	003	11 22 33 44 55 66 77 88	10:08:06:222
5	2	Recv	Standard	0	002	11 22 33 44 55 66 77 88	10:08:06:222
6	1	Send	Standard	0	004	11 22 33 44 55 66 77 88	10:08:06:223

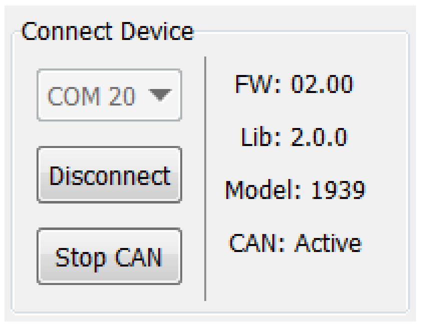
3.3.1. Connect Device

Select the COM port which is recognized as "EMUC VCom Port" in Device Manager, then click "Connect".

After connecting successfully, you will see the versions of firmware and library, and the model which can support J1939 or not.

Example:

 Connect Device: COM 20 — FW: 02.00, Lib: 2.0.0, Model: 020B, CAN: Inactive	Firmware version is v2.00
	CAN API version is v2.0.0
	This model only support basic CAN API
	CAN is inactive to configure CAN

 Connect Device COM 20 Disconnect Stop CAN FW: 02.00	Firmware version is v2.00
Lib: 2.0.0	CAN API version is v2.0.0
Model: 1939	This model can support basic CAN and J1939 API
CAN: Active	CAN is active to send/receive CAN frames

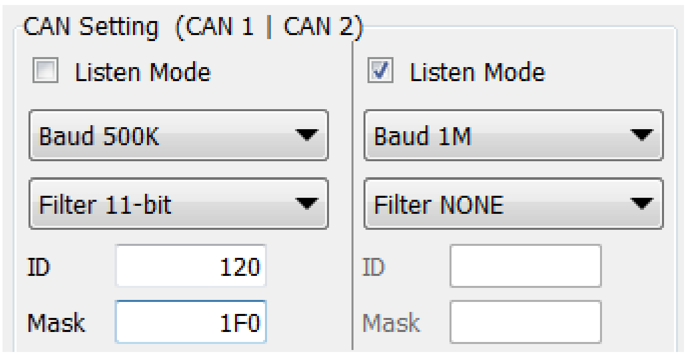
3.3.2. CAN Setting

NOTE: *Only can be used when CAN is inactive.*

In this section you can set CAN mode, baud rate, CAN acceptance filter, import/export CAN settings to a file, or reset all CAN settings to the default below.

Default Setting	
Baud Rate	500K
CANbus Mode	Normal Mode
Filter Type	None
Filter ID	None
Filter Mask	None
Error Setting	EEPROM only

Example:



CAN Setting (CAN 1 | CAN 2)

CAN 1	CAN 2
☐ Listen Mode	☑ Listen Mode
Baud 500K	Baud 1M
Filter 11-bit	Filter NONE
ID 120	ID
Mask 1F0	Mask

CAN1 is normal mode, baud rate is 500K, filter setting is 11bit, filtered id is 0x120, and filtered mask is 0x1F0. (Only receive CAN ID from 0x120 to 0x12F)

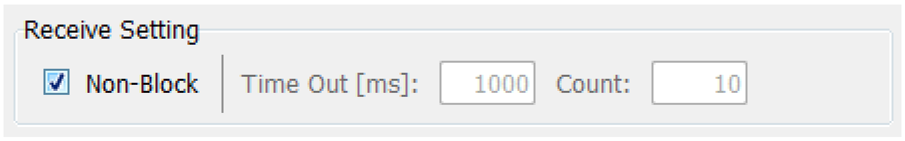
CAN2 is listen mode, baud rate is 1000K, and filtered setting is none.

3.3.3. Receive Setting

NOTE: *Only can be used when CAN is active.*

Enable non-block function to receive CAN frames. You can set the received conditions of "Time Out" or "Count". As long as one of the conditions is reached, the CAN frames are returned.

Example:



Receive Setting

☑ Non-Block | Time Out [ms]: 1000 Count: 10

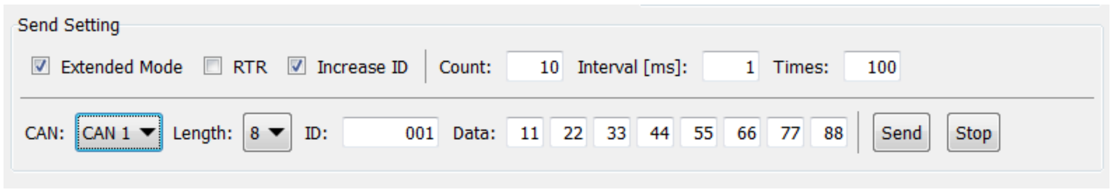
Non-block is enabled. Time Out is 1000ms (1 sec.), data count is 10. It means if receive 10 frames less then 1000ms, it will return 10 frames; if 1000ms time out but only receive 5 frames, it will return 5 frames.

3.3.4. Sending Setting

***NOTE:** Only can be used when CAN is active.*

Extended Mode: Check this checkbox to send EID (29bit) frames.
RTR: Check this checkbox to send RTR frames.
Increase ID: Check this check box to increase ID when "Count" setting > 1.
Count: Amount of CAN frames you want to send. Leave blank to send one frame.
Interval: Sending interval of each CAN frame when "Count" setting > 1.
Times: Amount of repetitions you want to send CAN frames.

Example:

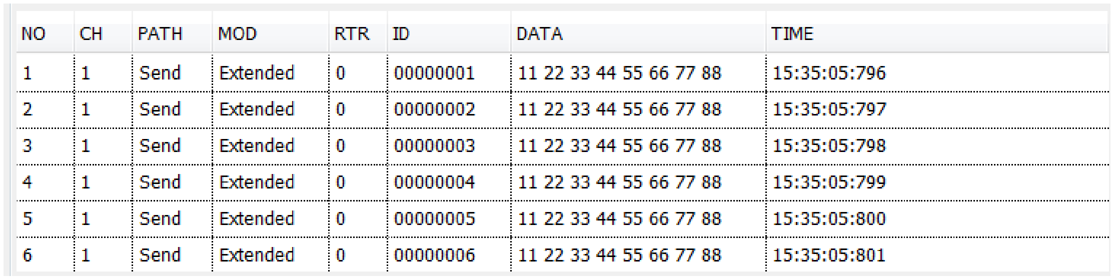


Send Setting

☑ Extended Mode ☐ RTR ☑ Increase ID | Count: 10 Interval [ms]: 1 Times: 100

CAN: CAN 1 Length: 8 ID: 001 Data: 11 22 33 44 55 66 77 88

Set 29bit ID without RTR and increased ID when sending next frame.
Send 10 frames with interval 1ms for each frame and repeat 100 times. It will send is 1000 frames totally.

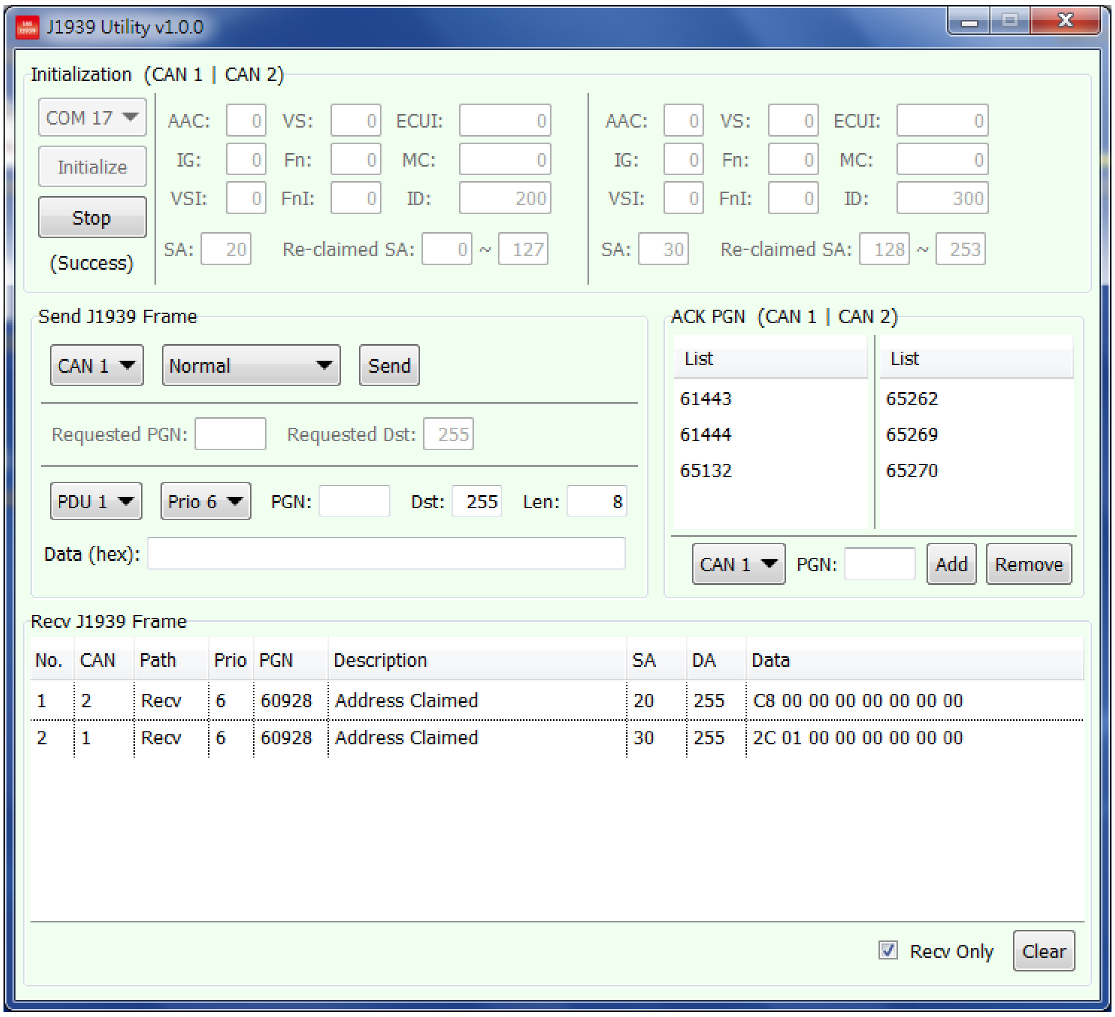


NO	CH	PATH	MOD	RTR	ID	DATA	TIME
1	1	Send	Extended	0	00000001	11 22 33 44 55 66 77 88	15:35:05:796
2	1	Send	Extended	0	00000002	11 22 33 44 55 66 77 88	15:35:05:797
3	1	Send	Extended	0	00000003	11 22 33 44 55 66 77 88	15:35:05:798
4	1	Send	Extended	0	00000004	11 22 33 44 55 66 77 88	15:35:05:799
5	1	Send	Extended	0	00000005	11 22 33 44 55 66 77 88	15:35:05:800
6	1	Send	Extended	0	00000006	11 22 33 44 55 66 77 88	15:35:05:801

3.4. J1939 Test Utility

You can use this GUI utility to test EMUC-B202 for sending/receiving normal J1939 frames and functions of "Address claimed", "Commanded Address", "Request PGN" and "Transport protocol".

Select the COM port which is recognized as "EMUC VCom Port" in Device Manager, then click "Initialize".



Initialization (CAN 1 | CAN 2)

COM 17 (Success)

	CAN 1		CAN 2
AAC:	0	AAC:	0
VS:	0	VS:	0
ECUI:	0	ECUI:	0
IG:	0	IG:	0
Fn:	0	Fn:	0
MC:	0	MC:	0
VSI:	0	VSI:	0
FnI:	0	FnI:	0
ID:	200	ID:	300
SA:	20	SA:	30
Re-claimed SA:	0 ~ 127	Re-claimed SA:	128 ~ 253

Send J1939 Frame

CAN 1 | Normal

Requested PGN: Requested Dst: 255

PDU 1 | Prio 6 | PGN: | Dst: 255 | Len: 8

Data (hex):

ACK PGN (CAN 1 | CAN 2)

List	List
61443	65262
61444	65269
65132	65270

CAN 1 | PGN:

Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	2	Recv	6	60928	Address Claimed	20	255	C8 00 00 00 00 00 00 00
2	1	Recv	6	60928	Address Claimed	30	255	2C 01 00 00 00 00 00 00

☑ Recv Only

NOTE: *Only frame data is Hexadecimal, the other values are all Decimal.*

3.4.1. Initialization

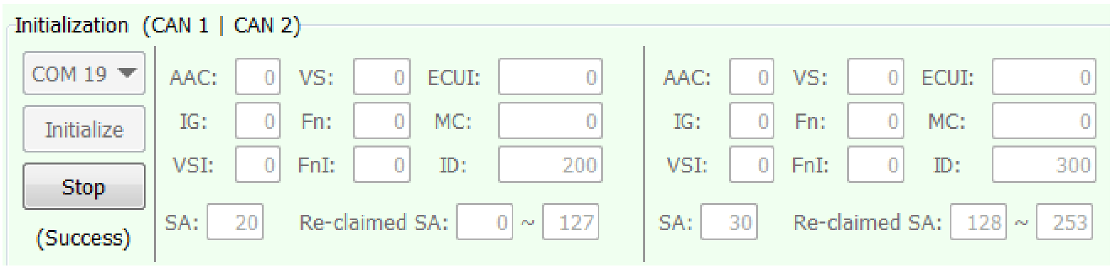
Set NAME and source address of CAN1 and CAN2 before initializing J1939 protocol. All ECUs must claim an address on the network. Initialized procedure set CANbus baud rate to 250 Kbps and sends PGN 60928 with the source address and NAME to claim the address which you want to use.

If another ECU claims the same address, the ECU with the lower value NAME field wins. NAME field is 64 bits long and is placed in the data field of the address claimed message. If an ECU loses, it can attempt another source address to reclaim.

The following table describes definitions of the fields.

AAC	1 bit Arbitrary Address Capable
IG	3 bits Industry Group
VSI	4 bits Vehicle System Instance
VS	7 bits Vehicle System
Fn	8 bits Function
FnI	5 bits Function Instance
ECUI	3 bits ECU Instance
MC	11 bits Manufacturer Code
ID	21 bits Identity Number
SA	8 bits Source Address
Re-claimed SA	Source address of the range 0-253 which are used for reclaiming address.

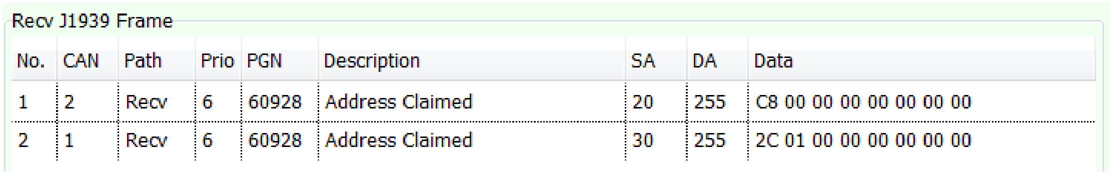
Example:


Initialization (CAN 1 | CAN 2)

COM 19, Initialize, Stop, (Success)

CAN 1: AAC: 0 VS: 0 ECUI: 0 IG: 0 Fn: 0 MC: 0 VSI: 0 FnI: 0 ID: 200 SA: 20 Re-claimed SA: 0 ~ 127

CAN 2: AAC: 0 VS: 0 ECUI: 0 IG: 0 Fn: 0 MC: 0 VSI: 0 FnI: 0 ID: 300 SA: 30 Re-claimed SA: 128 ~ 253


Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	2	Recv	6	60928	Address Claimed	20	255	C8 00 00 00 00 00 00 00
2	1	Recv	6	60928	Address Claimed	30	255	2C 01 00 00 00 00 00 00

3.4.2. Normal J1939 Frame

You can select CAN1 or CAN2 to send normal J1939 frame.

PDU1: PDU format < 240, PDU specific is destination address.

PDU2: PDU format >= 240, PDU specific is group extension.

Prio: Message priority.

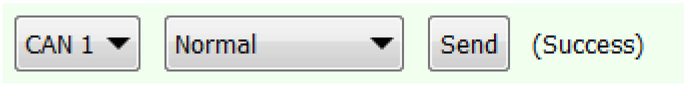
PGN (Dec): Parameter group number. When PDU format (PF) is PDU1, the second bytes of PGN must be 0x00 such as 61184 (0xEF00), 60928 (0xEE00), 60672 (0xED00)…

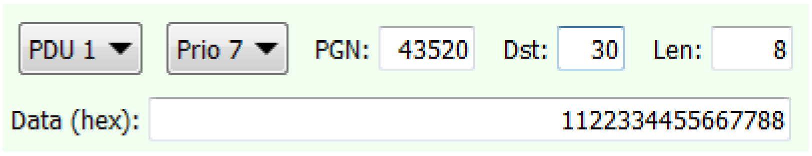
Dst (Dec): Destination address. If you select PDU1, destination address can be specific of global address (255); if you select PDU2, destination address must be global address (255).

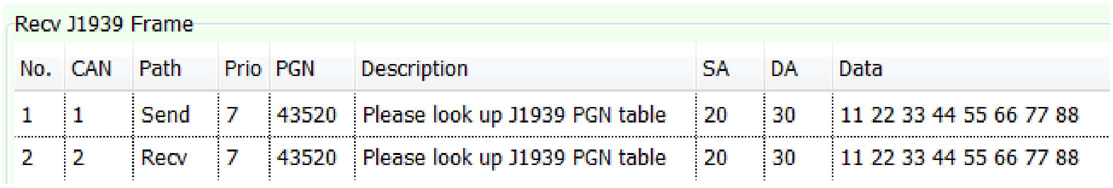
Len: Data length. Only PGN 59904 can have 3 bytes data, others PGN must have 8 bytes of more than 8 bytes data. If data bytes are 9 to 1785, it will use J1939 transport protocol to send the frame.
Data (Hex): J1939 data. It must match with data length.

Example 1: PDU1

CAN1 (SA=20) sends normal J1939 frame of PDU1 to CAN2 (SA=30), priority is 7, PGN is 43520 (0xAA00), destination is 30, data length is 8, data is 0x1122334455667788.

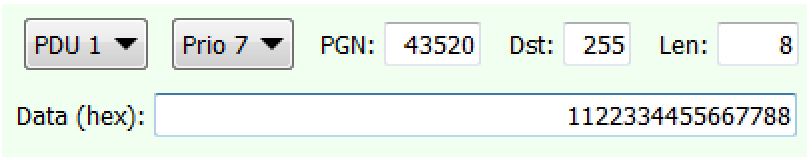


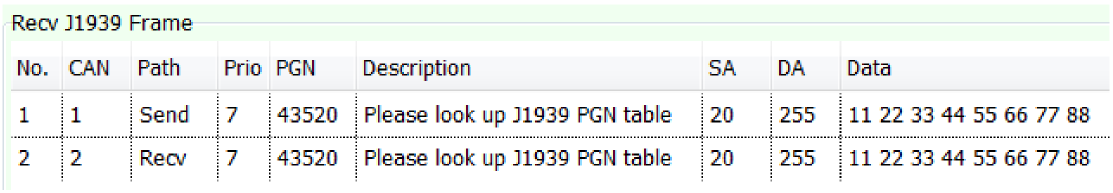

PDU 1 Prio 7 PGN: 43520 Dst: 30 Len: 8
Data (hex): 1122334455667788


Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	7	43520	Please look up J1939 PGN table	20	30	11 22 33 44 55 66 77 88
2	2	Recv	7	43520	Please look up J1939 PGN table	20	30	11 22 33 44 55 66 77 88

If your destination set to global address (255), this frame will be a broadcast, so CAN2 still can receive this frame.

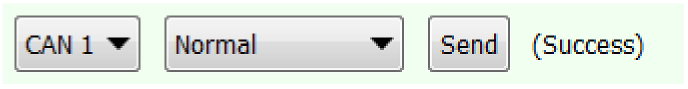

PDU 1 Prio 7 PGN: 43520 Dst: 255 Len: 8
Data (hex): 1122334455667788

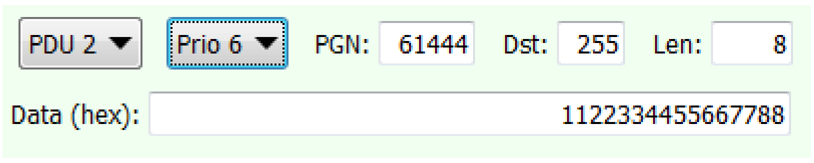

Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	7	43520	Please look up J1939 PGN table	20	255	11 22 33 44 55 66 77 88
2	2	Recv	7	43520	Please look up J1939 PGN table	20	255	11 22 33 44 55 66 77 88

Example 2: PDU2

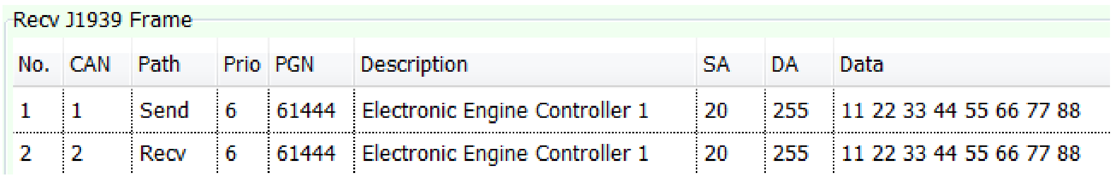
CAN1 (SA=20) sends normal J1939 frame of PDU2, priority is 6, PGN is 61444 (0xF004), destination must be global address (255), data length is 8, data is 0x1122334455667788.





PGN: 61444 Dst: 255 Len: 8

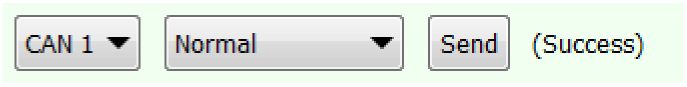
Data (hex): 1122334455667788

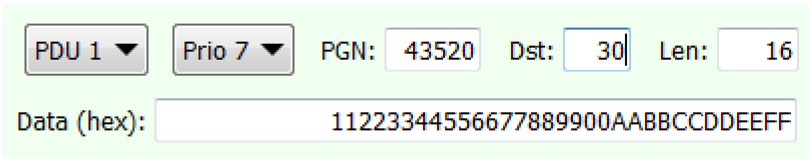

Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	6	61444	Electronic Engine Controller 1	20	255	11 22 33 44 55 66 77 88
2	2	Recv	6	61444	Electronic Engine Controller 1	20	255	11 22 33 44 55 66 77 88

Example 3: Transport protocol

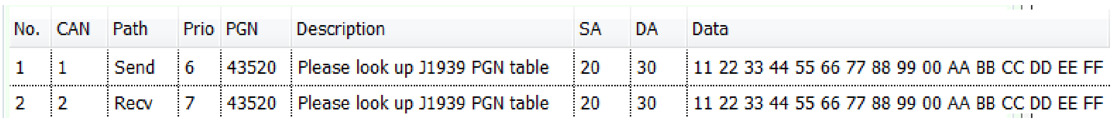
CAN1 (SA=20) sends normal J1939 frame of PDU1 data > 8 to CAN2 (SA=30), priority is 7, PGN is 43520 (0xAA00), destination is 30, data length is 8, data is 0x112233445566778899 00AABBCCDDEEFF.





PGN: 43520 Dst: 30 Len: 16

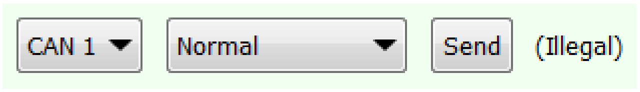
Data (hex): 112233445566778899 00AABBCCDDEEFF



No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	6	43520	Please look up J1939 PGN table	20	30	11 22 33 44 55 66 77 88 99 00 AA BB CC DD EE FF
2	2	Recv	7	43520	Please look up J1939 PGN table	20	30	11 22 33 44 55 66 77 88 99 00 AA BB CC DD EE FF

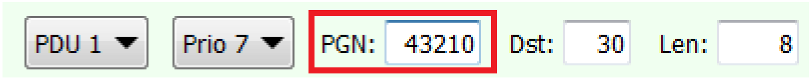
Example 4: Illegal

If input values don't comply with J1939 standard; the utility will not send the frame because of illegal values.



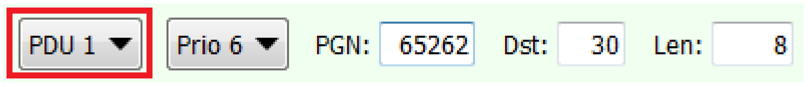
PDU format of PDU1 < 240, PGN must equal to or lower than 61184 (0xEF00, PF=EF_{16}=239_{10}), and the second bytes of PGN must be 0x00 such as 61184 (0xEF00), 60928 (0xEE00), 60672 (0xED00)…

PGN 43210 is 0xA8CA, PF=0xA8=168. It is PDU1; the second bytes of PGN cannot have value, so it is illegal. Correct the value from 43210 to 43008 (0xA800).

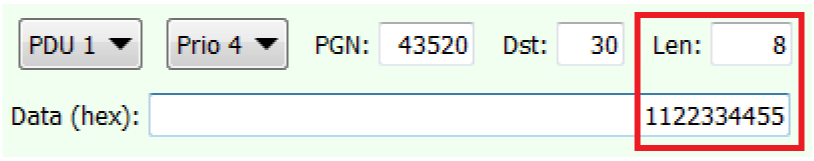


PGN: 43210 Dst: 30 Len: 8

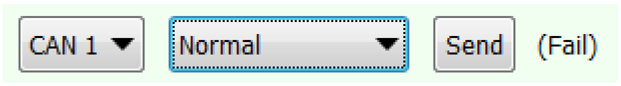
PDU format of PDU2 >=240, PGN must equal to or higher than 61440 (0xF000, PF=0xF0=240).
PGN 65262 (0xFEEE, PF=0xFE=254) is higher than 240, so it is illegal. Correct the option from PDU1 to PDU2



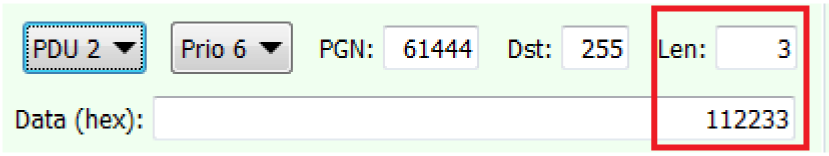
Data length is 8, but there are only 5 bytes data, so it is illegal. Fill the data to 8 bytes.



Example 5: Fail



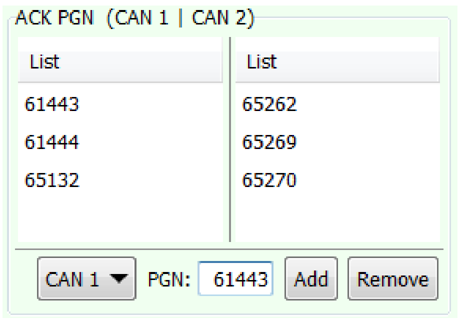
Only PGN 59904 can have 3 bytes data, others PGN must have 8 bytes of more than 8 bytes data. Correct the value of data length from 3 to 8 and fill the data to 8 bytes.



3.4.3. Request (PGN 59904)

You can select CAN1 or CAN2 to send request PGN.

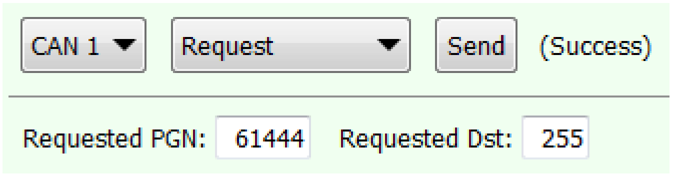
Requested PGN (Dec): The PGN which you want to request.
Requested Dst (Dec): The destination address you want to send this request, it can be specific of global address (255).
ACK PGN (Dec): The PGNs of CAN1 and CAN2 which will send “Positive ACK” if receive PGN 50094 and requested PGN is in the list. You can select CAN1 or CAN2 to add/remove PGN.



List	List
61443	65262
61444	65269
65132	65270

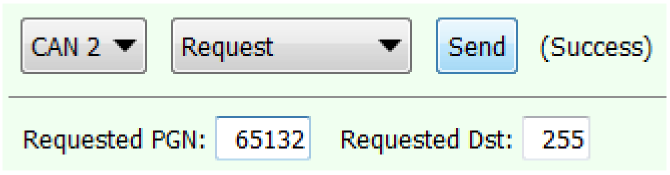
Example 1: Send Request

CAN1 send requested PGN 61444 to global address (255).



Requested PGN: 61444 Requested Dst: 255

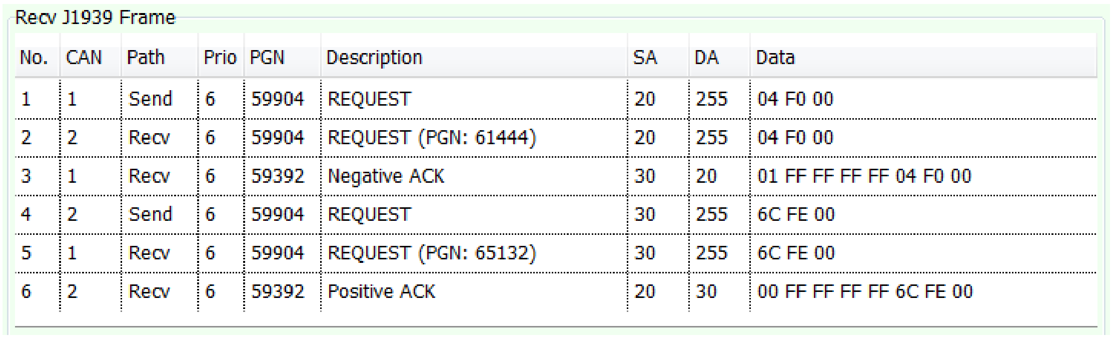
CAN2 send requested PGN 65132 to global address (255).



Requested PGN: 65132 Requested Dst: 255

CAN2 receives the request then returns PGN 59392 with Negative ACK to CAN1.

CAN1 receives the request then returns PGN 59392 with Positive ACK to CAN2.

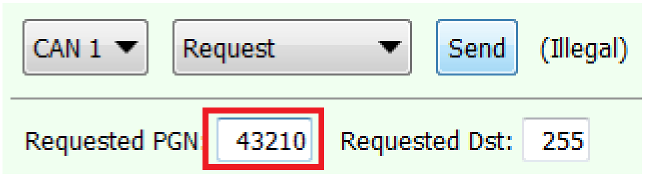


Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	6	59904	REQUEST	20	255	04 F0 00
2	2	Recv	6	59904	REQUEST (PGN: 61444)	20	255	04 F0 00
3	1	Recv	6	59392	Negative ACK	30	20	01 FF FF FF FF 04 F0 00
4	2	Send	6	59904	REQUEST	30	255	6C FE 00
5	1	Recv	6	59904	REQUEST (PGN: 65132)	30	255	6C FE 00
6	2	Recv	6	59392	Positive ACK	20	30	00 FF FF FF FF 6C FE 00

Example 2: Illegal

PGN 43210 is 0xA8CA, PF=0xA8=168. It is PDU1; the second bytes of PGN cannot have value, so it is illegal. Correct the value from 43210 to 43008 (0xA800).



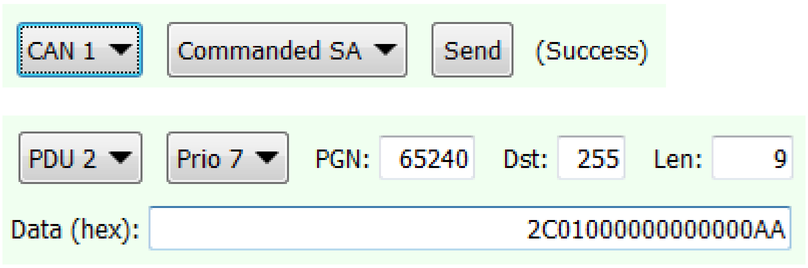
Requested PGN 43210 Requested Dst: 255

3.4.4. Commanded Source Address (PGN 65240)

If ECU receives the J1939 frame of commanded address (PGN 65240), and the NAME is the same as ECU owns, the 9th byte of data is the source address which is used to set the ECU to this specific address.

Example:

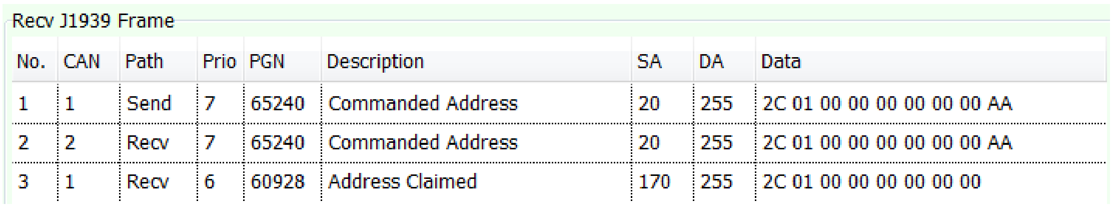
CAN1 send a commanded address to ask CAN2 to change source address to 170 (0xAA).



PGN: 65240 Dst: 255 Len: 9

Data (hex): 2C01000000000000AA

After CAN2 receive the command, it changes its source address from 30 to 170 and claims address again.



Recv J1939 Frame

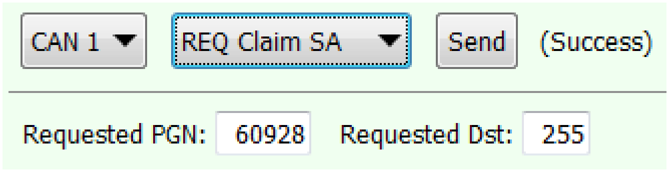
No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	7	65240	Commanded Address	20	255	2C 01 00 00 00 00 00 AA
2	2	Recv	7	65240	Commanded Address	20	255	2C 01 00 00 00 00 00 AA
3	1	Recv	6	60928	Address Claimed	170	255	2C 01 00 00 00 00 00 00

3.4.5. Request Claim Source Address

Send PGN 59904 with requested PGN 60928 to retrieve information about addresses being used by other devices on the network.

Example:

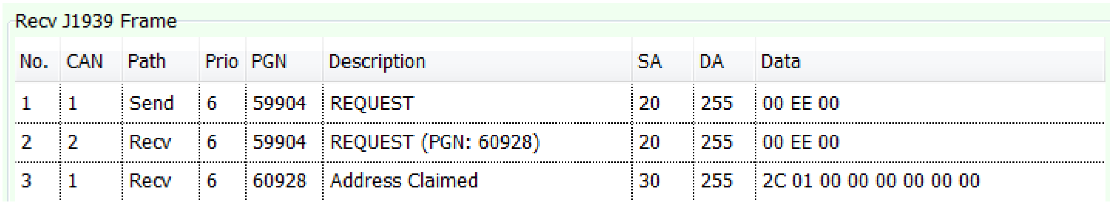
CAN1 sends a request for address claimed to global address.



Requested PGN: 60928 Requested Dst: 255

CAN2 receives the request then claims the source address again.

CAN1 receives address claimed from CAN2



Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	1	Send	6	59904	REQUEST	20	255	00 EE 00
2	2	Recv	6	59904	REQUEST (PGN: 60928)	20	255	00 EE 00
3	1	Recv	6	60928	Address Claimed	30	255	2C 01 00 00 00 00 00 00

4. Linux OS

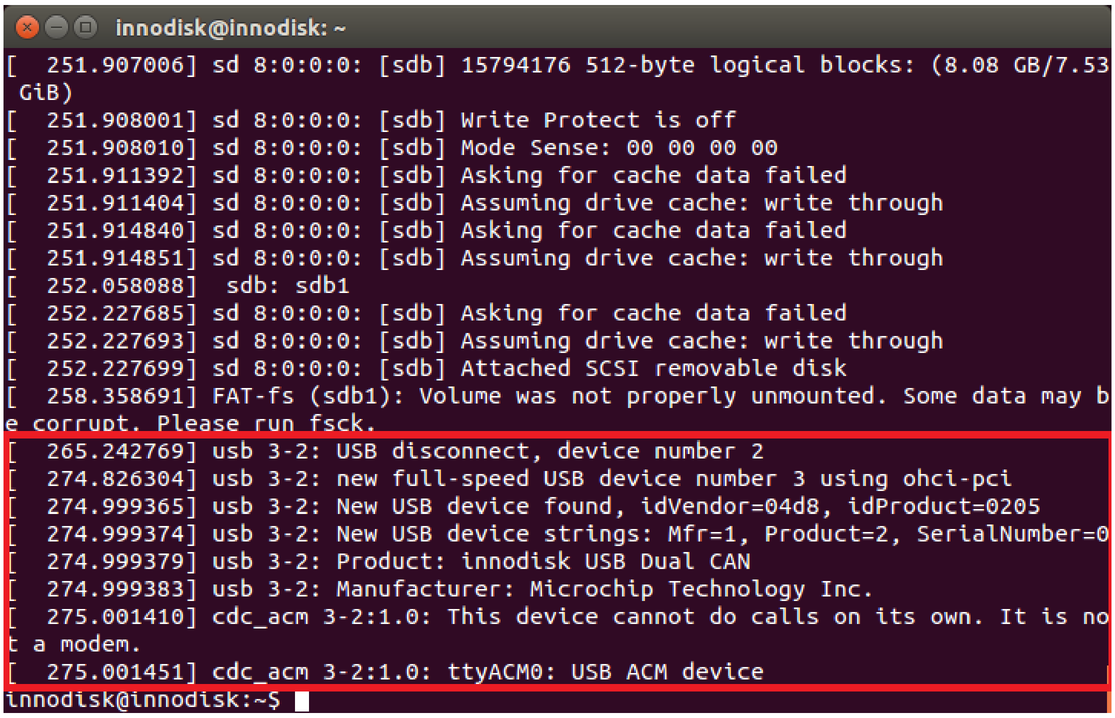
The following sections use Ubuntu 14.04.

4.1. Driver Installation

The device will be recognized as ttyACM% (%=0, 1…) by using CDC-ACM kernel driver.

Note: *Linux kernel 2.6 and above have native CDC-ACM kernel driver. Some Linux OS may need to add CDC-ACM configuration manually in building process. In different Linux OS may have different tty name.*

Type command "dmesg" to see messages below.
Generally the name would be ttyACM0 or ttyACM1 in Linux.

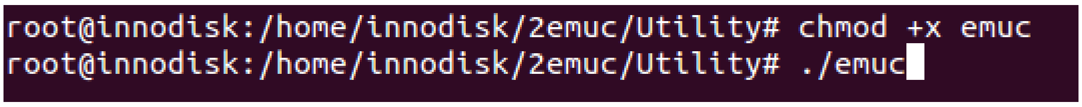


```
innodisk@innodisk: ~
[  251.907006] sd 8:0:0:0: [sdb] 15794176 512-byte logical blocks: (8.08 GB/7.53
 GiB)
[  251.908001] sd 8:0:0:0: [sdb] Write Protect is off
[  251.908010] sd 8:0:0:0: [sdb] Mode Sense: 00 00 00 00
[  251.911392] sd 8:0:0:0: [sdb] Asking for cache data failed
[  251.911404] sd 8:0:0:0: [sdb] Assuming drive cache: write through
[  251.914840] sd 8:0:0:0: [sdb] Asking for cache data failed
[  251.914851] sd 8:0:0:0: [sdb] Assuming drive cache: write through
[  252.058088]  sdb: sdb1
[  252.227685] sd 8:0:0:0: [sdb] Asking for cache data failed
[  252.227693] sd 8:0:0:0: [sdb] Assuming drive cache: write through
[  252.227699] sd 8:0:0:0: [sdb] Attached SCSI removable disk
[  258.358691] FAT-fs (sdb1): Volume was not properly unmounted. Some data may b
e corrupt. Please run fsck.
[  265.242769] usb 3-2: USB disconnect, device number 2
[  274.826304] usb 3-2: new full-speed USB device number 3 using ohci-pci
[  274.999365] usb 3-2: New USB device found, idVendor=04d8, idProduct=0205
[  274.999374] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[  274.999379] usb 3-2: Product: innodisk USB Dual CAN
[  274.999383] usb 3-2: Manufacturer: Microchip Technology Inc.
[  275.001410] cdc_acm 3-2:1.0: This device cannot do calls on its own. It is no
t a modem.
[  275.001451] cdc_acm 3-2:1.0: ttyACM0: USB ACM device
innodisk@innodisk:~$
```

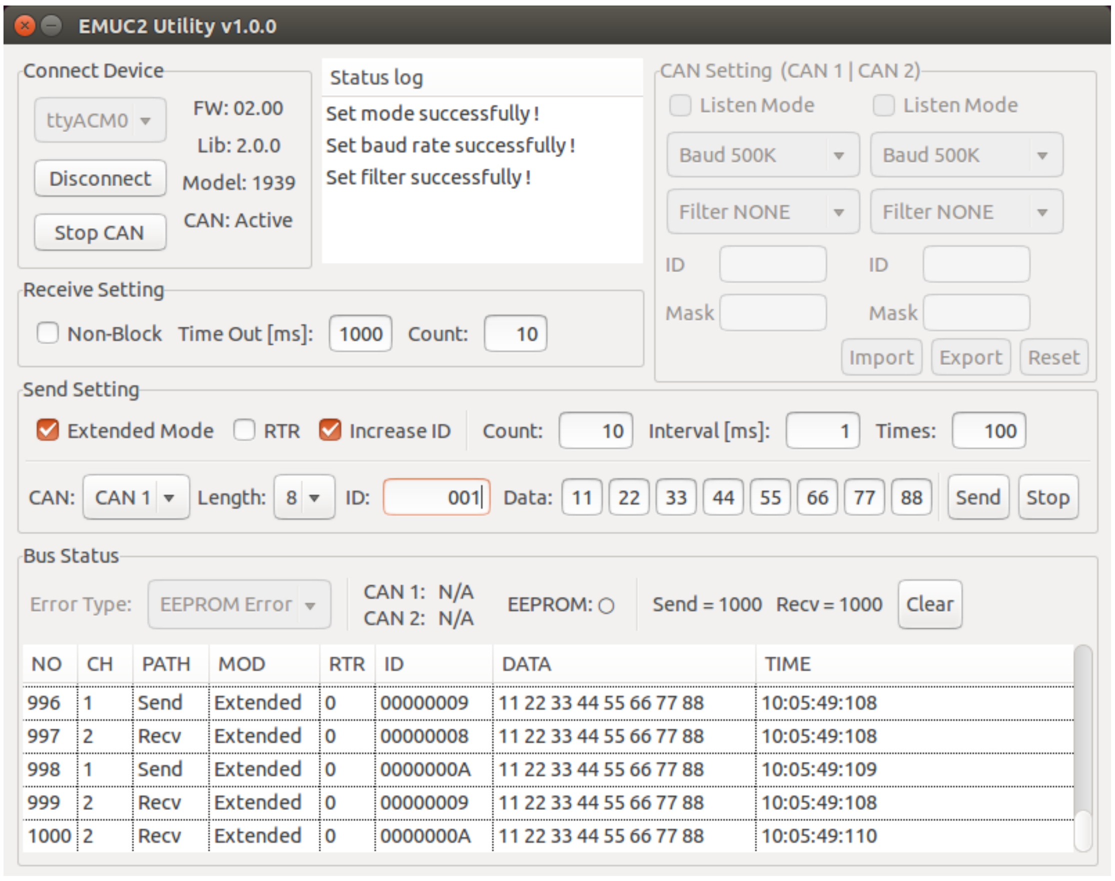
4.2. Basic CAN 2.0B Test Utility

All operations and configurations are the same as Windows version. Please refer to *3.2 EMUC-B202 Test Utility*

Before running the utility, you need to use command "chmod +x" to give executable permission to it.



```
root@innodisk:/home/innodisk/2emuc/Utility# chmod +x emuc
root@innodisk:/home/innodisk/2emuc/Utility# ./emuc
```



ttyACM0 FW: 02.00 Lib: 2.0.0 Model: 1939 CAN: Active

Set mode successfully !
Set baud rate successfully !
Set filter successfully !

Listen Mode | Listen Mode
Baud 500K | Baud 500K
Filter NONE | Filter NONE
ID | ID
Mask | Mask

Non-Block Time Out [ms]: 1000 Count: 10

Extended Mode ✓ RTR Increase ID ✓ Count: 10 Interval [ms]: 1 Times: 100

CAN: CAN 1 Length: 8 ID: 001 Data: 11 22 33 44 55 66 77 88

Error Type: EEPROM Error CAN 1: N/A CAN 2: N/A EEPROM: ○ Send = 1000 Recv = 1000

NO	CH	PATH	MOD	RTR	ID	DATA	TIME
996	1	Send	Extended	0	00000009	11 22 33 44 55 66 77 88	10:05:49:108
997	2	Recv	Extended	0	00000008	11 22 33 44 55 66 77 88	10:05:49:108
998	1	Send	Extended	0	0000000A	11 22 33 44 55 66 77 88	10:05:49:109
999	2	Recv	Extended	0	00000009	11 22 33 44 55 66 77 88	10:05:49:108
1000	2	Recv	Extended	0	0000000A	11 22 33 44 55 66 77 88	10:05:49:110

4.3. SocketCAN

EMUC-B202 can support SocketCAN by additional driver and user space tool on Linux kernel 2.6.38 and above.

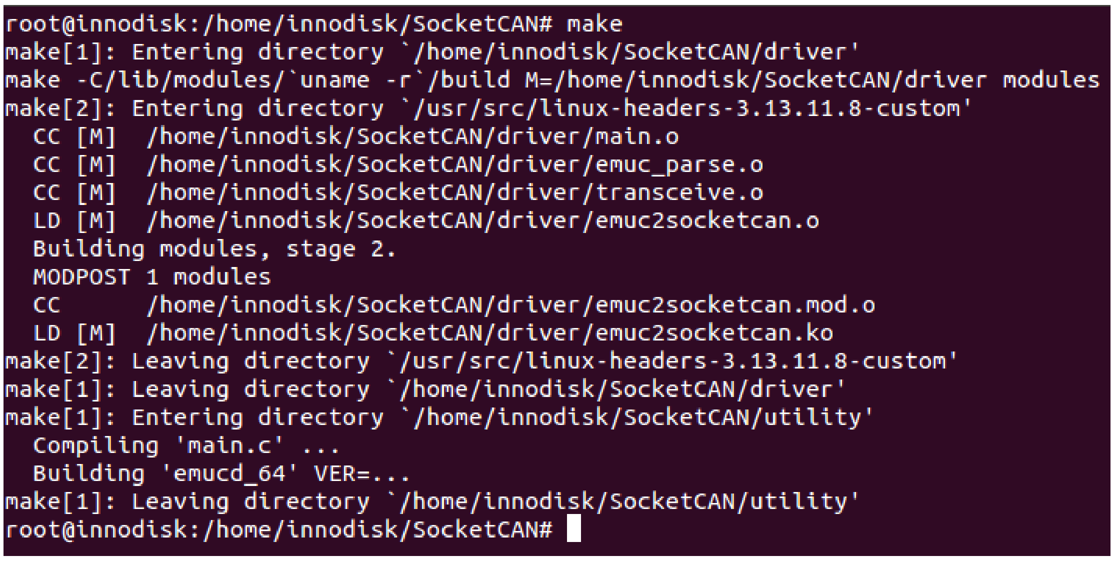
Before installing SocketCAN driver, you must confirm that the Linux Kernel include SocketCAN kernel module and recognize EMUC-B202 as ttyACM%(%=0,1,…) by using native CDC-ACM driver.

4.3.1. Build driver and user-space tool

Please copy kernel development packages into your system and type "make" command in root folder of this package.

There should be two output files:

- **emuc2socketcan.ko**: Kernel driver of EMUC SocketCAN
- **emucd_32** or **emucd_64**: User-space tool for enabling EMUC SocketCAN



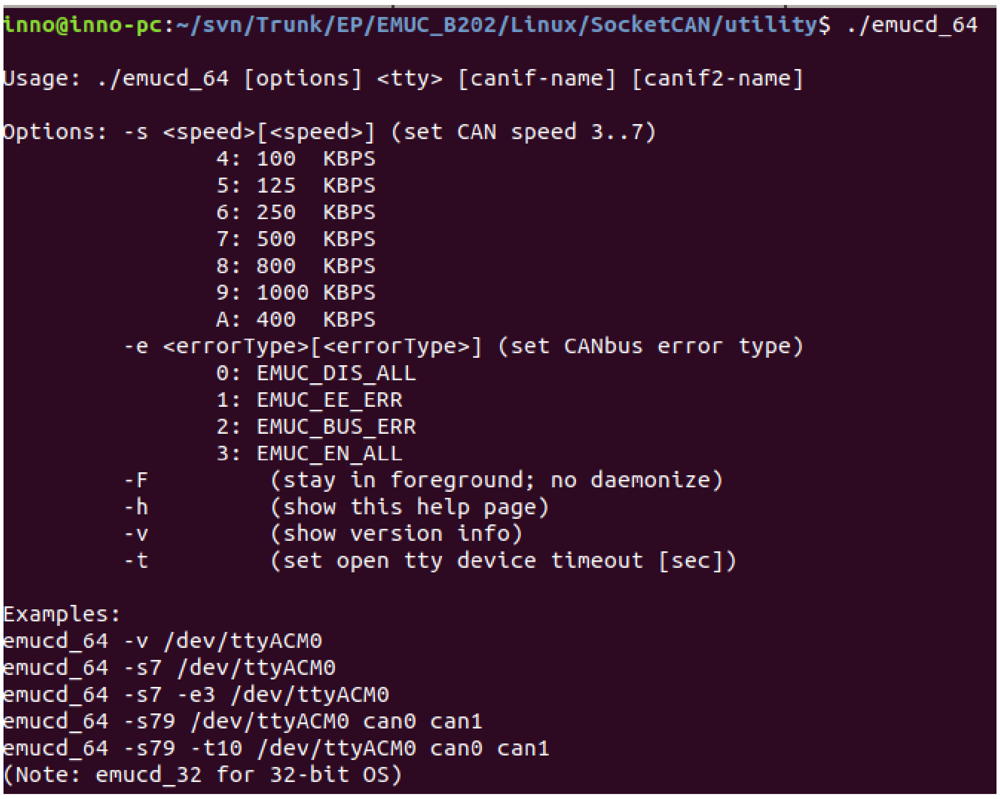
```
root@innodisk:/home/innodisk/SocketCAN# make
make[1]: Entering directory `/home/innodisk/SocketCAN/driver'
make -C/lib/modules/`uname -r`/build M=/home/innodisk/SocketCAN/driver modules
make[2]: Entering directory `/usr/src/linux-headers-3.13.11.8-custom'
  CC [M]  /home/innodisk/SocketCAN/driver/main.o
  CC [M]  /home/innodisk/SocketCAN/driver/emuc_parse.o
  CC [M]  /home/innodisk/SocketCAN/driver/transceive.o
  LD [M]  /home/innodisk/SocketCAN/driver/emuc2socketcan.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/innodisk/SocketCAN/driver/emuc2socketcan.mod.o
  LD [M]  /home/innodisk/SocketCAN/driver/emuc2socketcan.ko
make[2]: Leaving directory `/usr/src/linux-headers-3.13.11.8-custom'
make[1]: Leaving directory `/home/innodisk/SocketCAN/driver'
make[1]: Entering directory `/home/innodisk/SocketCAN/utility'
  Compiling 'main.c' ...
  Building 'emucd_64' VER=...
make[1]: Leaving directory `/home/innodisk/SocketCAN/utility'
root@innodisk:/home/innodisk/SocketCAN# 
```

You can type "emucd_64 -h" for help.

./emucd_64 -s7 /dev/ttyACM0   (500 KBPS on both channel)

./emucd_64 -s79 /dev/ttyACM0 (500 KBPS on ch1, 1000 KBPS on ch2)

NOTE: *If you don't specify interface name, default name will be "emuccan0" and "emuccan1"*



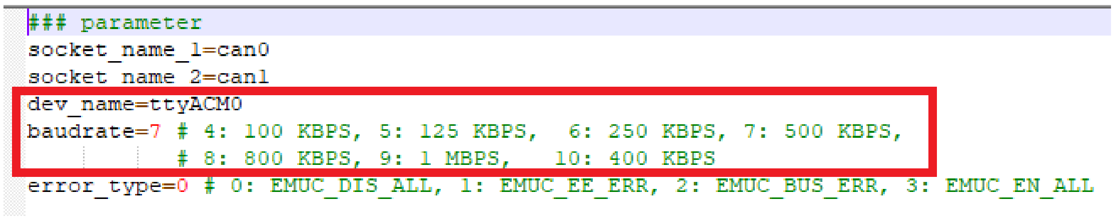
```
inno@inno-pc:~/svn/Trunk/EP/EMUC_B202/Linux/SocketCAN/utility$ ./emucd_64

Usage: ./emucd_64 [options] <tty> [canif-name] [canif2-name]

Options: -s <speed>[<speed>] (set CAN speed 3..7)
                4: 100  KBPS
                5: 125  KBPS
                6: 250  KBPS
                7: 500  KBPS
                8: 800  KBPS
                9: 1000 KBPS
                A: 400  KBPS
         -e <errorType>[<errorType>] (set CANbus error type)
                0: EMUC_DIS_ALL
                1: EMUC_EE_ERR
                2: EMUC_BUS_ERR
                3: EMUC_EN_ALL
         -F         (stay in foreground; no daemonize)
         -h         (show this help page)
         -v         (show version info)
         -t         (set open tty device timeout [sec])

Examples:
emucd_64 -v /dev/ttyACM0
emucd_64 -s7 /dev/ttyACM0
emucd_64 -s7 -e3 /dev/ttyACM0
emucd_64 -s79 /dev/ttyACM0 can0 can1
emucd_64 -s79 -t10 /dev/ttyACM0 can0 can1
(Note: emucd_32 for 32-bit OS)
```

4.3.2. SocketCAN Driver Installation

There are shell scripts "start.sh" and "end.sh" to install the driver and enable SocketCAN interface.

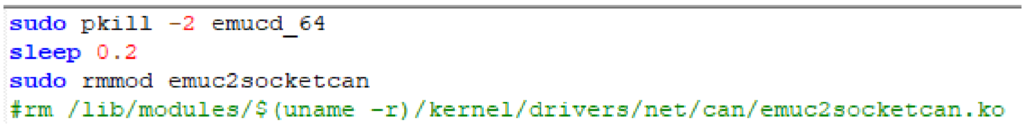
start.sh

Please modify the baud rate and tty port setting depend on the environment needs.



```
### parameter
socket_name_1=can0
socket_name_2=can1
dev_name=ttyACM0
baudrate=7 # 4: 100 KBPS, 5: 125 KBPS,  6: 250 KBPS, 7: 500 KBPS,
           # 8: 800 KBPS, 9: 1 MBPS,   10: 400 KBPS
error_type=0 # 0: EMUC_DIS_ALL, 1: EMUC_EE_ERR, 2: EMUC_BUS_ERR, 3: EMUC_EN_ALL
```

end.sh

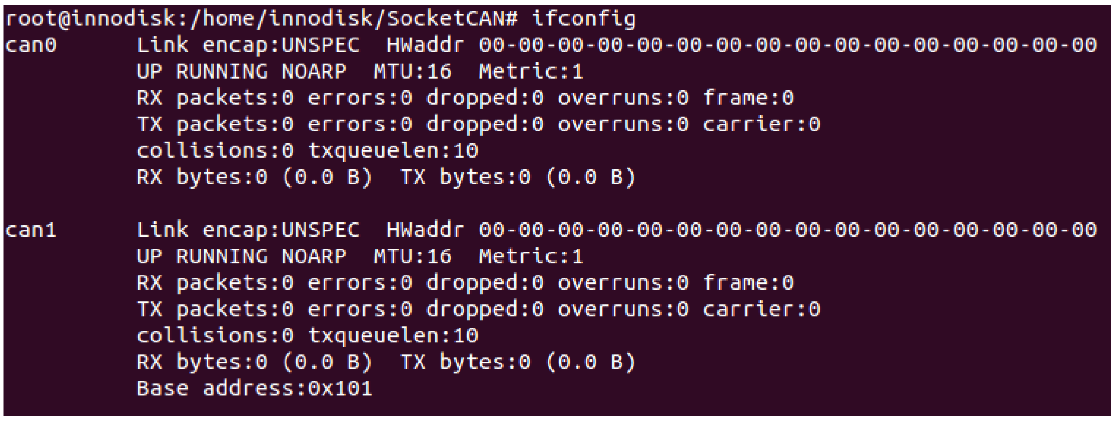


```
sudo pkill -2 emucd_64
sleep 0.2
sudo rmmod emuc2socketcan
#rm /lib/modules/$(uname -r)/kernel/drivers/net/can/emuc2socketcan.ko
```

You can start/end SocketCAN interface simply by using the scripts.

```
-$ chmod +x start.sh
-$ ./start.sh
```

You can see the CAN interface name by "ifconfig" command.



```
root@innodisk:/home/innodisk/SocketCAN# ifconfig
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

can1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Base address:0x101
```

4.3.3. CAN-utils

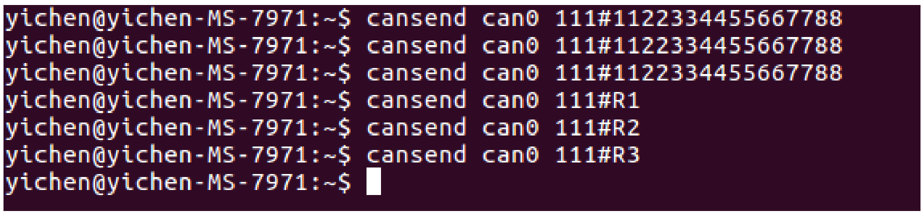
After SocketCAN setup is finished, you can use open source project “can-utils” to test by “cansend” and “candump”.

(https://github.com/linux-can/can-utils).

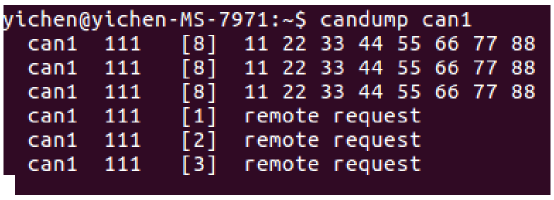
- Install CAN-utils

    - $ apt-get install can-utils

- use can0 to send and can1 to receive.



```
yichen@yichen-MS-7971:~$ cansend can0 111#1122334455667788
yichen@yichen-MS-7971:~$ cansend can0 111#1122334455667788
yichen@yichen-MS-7971:~$ cansend can0 111#1122334455667788
yichen@yichen-MS-7971:~$ cansend can0 111#R1
yichen@yichen-MS-7971:~$ cansend can0 111#R2
yichen@yichen-MS-7971:~$ cansend can0 111#R3
yichen@yichen-MS-7971:~$ 
```



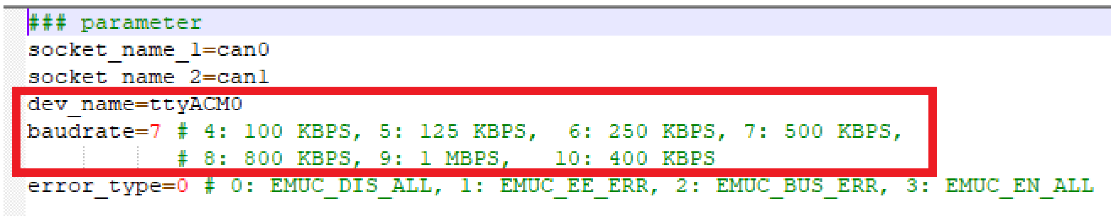
```
yichen@yichen-MS-7971:~$ candump can1
  can1  111   [8]  11 22 33 44 55 66 77 88
  can1  111   [8]  11 22 33 44 55 66 77 88
  can1  111   [8]  11 22 33 44 55 66 77 88
  can1  111   [1]  remote request
  can1  111   [2]  remote request
  can1  111   [3]  remote request
```

4.3.4. Boot Up Script

We provide Linux boot up script to initial SocketCAN interface automatically after system boot up.

run_emucd

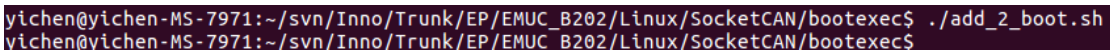
Please modify the baud rate and tty port setting depend on the environment needs.



```
### parameter
socket_name_1=can0
socket_name_2=can1
dev_name=ttyACM0
baudrate=7 # 4: 100 KBPS, 5: 125 KBPS,  6: 250 KBPS, 7: 500 KBPS,
           # 8: 800 KBPS, 9: 1 MBPS,    10: 400 KBPS
error_type=0 # 0: EMUC_DIS_ALL, 1: EMUC_EE_ERR, 2: EMUC_BUS_ERR, 3: EMUC_EN_ALL
```

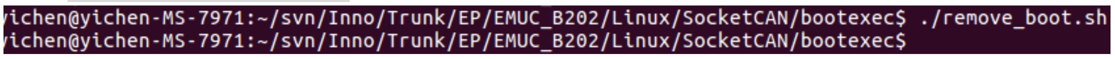
Run the following command in the “release” folder to add/remove boot up script.

    - $ chmod +x add_2_boot.sh
    - $ ./add_2_boot.sh



```
yichen@yichen-MS-7971:~/svn/Inno/Trunk/EP/EMUC_B202/Linux/SocketCAN/bootexec$ ./add_2_boot.sh
yichen@yichen-MS-7971:~/svn/Inno/Trunk/EP/EMUC_B202/Linux/SocketCAN/bootexec$
```

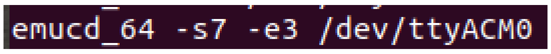
    - $ chmod +x remove_boot.sh
    - $ ./ remove_boot.sh



```
yichen@yichen-MS-7971:~/svn/Inno/Trunk/EP/EMUC_B202/Linux/SocketCAN/bootexec$ ./remove_boot.sh
yichen@yichen-MS-7971:~/svn/Inno/Trunk/EP/EMUC_B202/Linux/SocketCAN/bootexec$
```

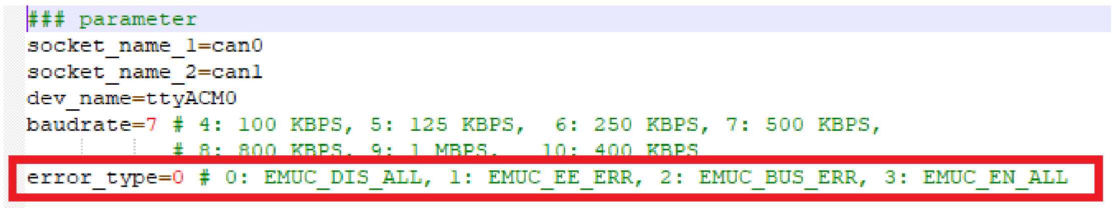
4.3.5. CAN Error Frame

CAN error frame can be dumped by adding the parameter “-e” when running the emucd_32 or emucd_64 utility.



```
emucd_64 -s7 -e3 /dev/ttyACM0
```

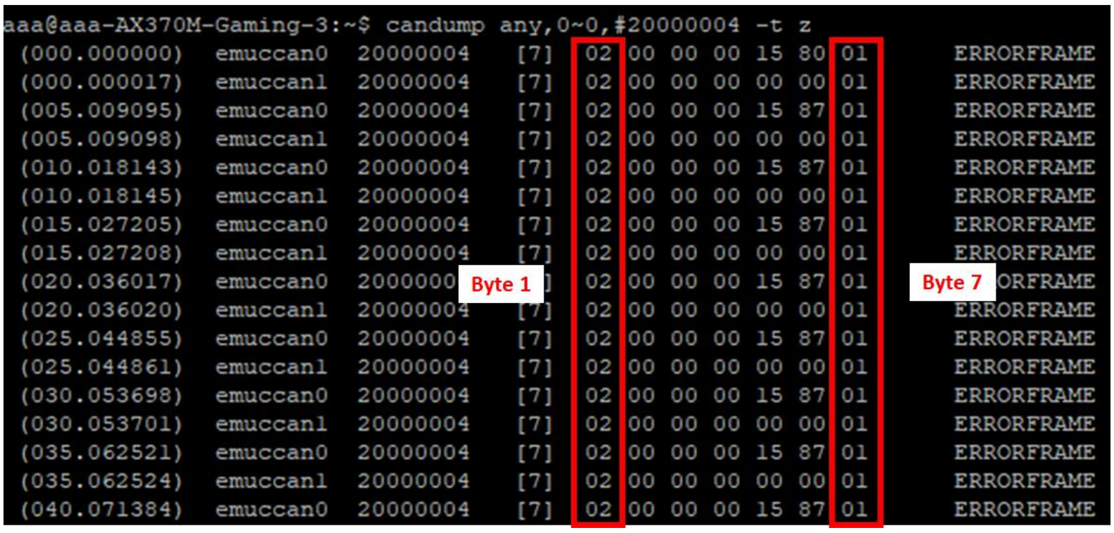
It can be simply set the error type by editing “start.sh”.
“run_emucd” of boot up script has this parameter as well.



```
### parameter
socket_name_1=can0
socket_name_2=can1
dev_name=ttyACM0
baudrate=7 # 4: 100 KBPS, 5: 125 KBPS,  6: 250 KBPS, 7: 500 KBPS,
           # 8: 800 KBPS, 9: 1 MBPS,   10: 400 KBPS
error_type=0 # 0: EMUC_DIS_ALL, 1: EMUC_EE_ERR, 2: EMUC_BUS_ERR, 3: EMUC_EN_ALL
```

0: EMUC_DIS_ALL: disable all error frame.
1: EMUC_EE_ERR: enable EEPROM error only.
2: EMUC_BUS_ERR: enable CAN bus error only.
3: EMUC_EM_ALL: enable both EERPOM and CAN bus error.

CAN error frame can be dumped through the following command of CAN-utils.



```
aaa@aaa-AX370M-Gaming-3:~$ candump any,0~0,#20000004 -t z
 (000.000000)  emuccan0  20000004   [7]  02 00 00 00 15 80 01        ERRORFRAME
 (000.000017)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (005.009095)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
 (005.009098)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (010.018143)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
 (010.018145)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (015.027205)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
 (015.027208)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (020.036017)  emuccan0  2000000         ]  02 00 00 00 15 87 01           ORFRAME
 (020.036020)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (025.044855)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
 (025.044861)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (030.053698)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
 (030.053701)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (035.062521)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
 (035.062524)  emuccan1  20000004   [7]  02 00 00 00 00 00 01        ERRORFRAME
 (040.071384)  emuccan0  20000004   [7]  02 00 00 00 15 87 01        ERRORFRAME
```

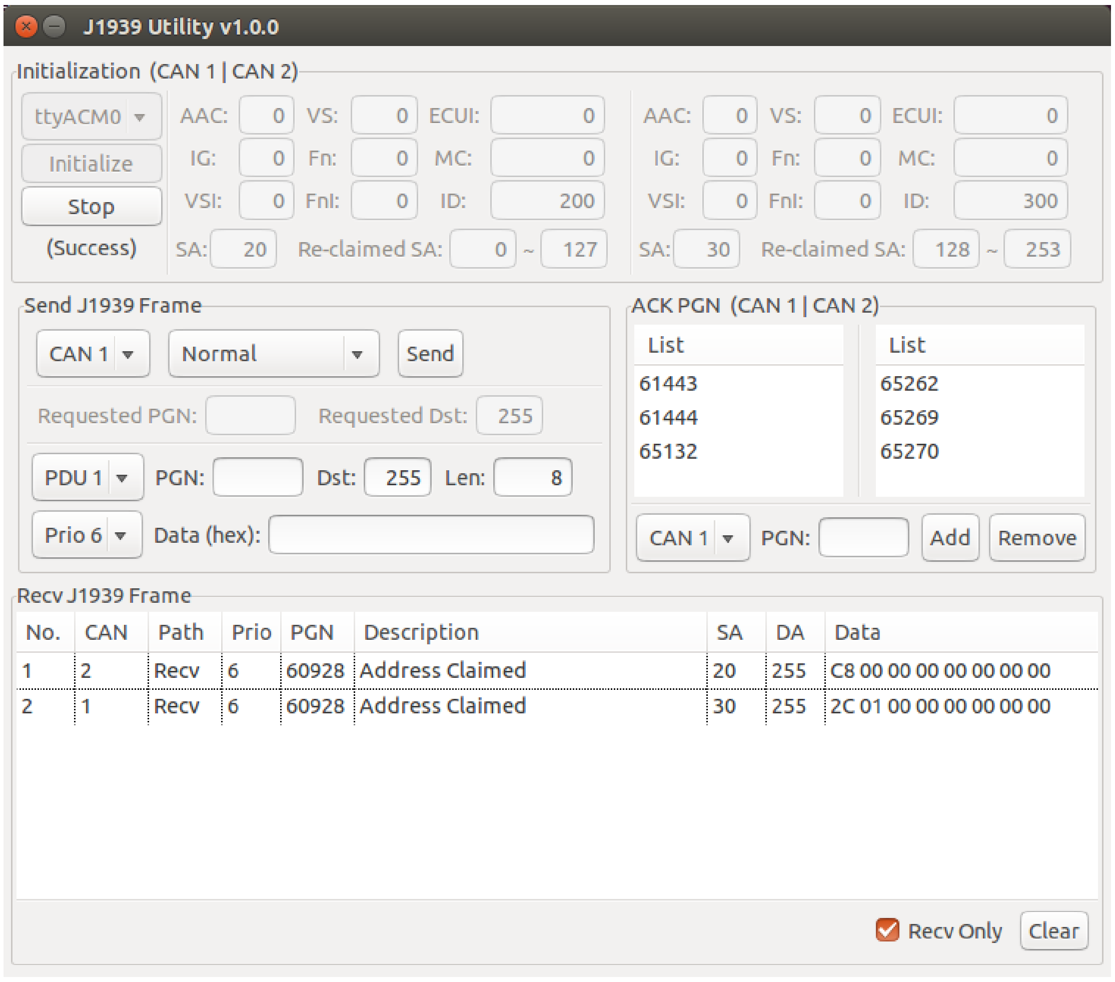
Byte1: Error Type, 0x01=EEPROM Error, 0x02=Bus Error
Byte2~Byte7: Bus Error Register, please refer to *3.2.Register mapping table of CAN error status.*

4.4. J1939 Test Utility

All operations and configurations are the same as Windows version, please refer to *3.3 J1939 Test Utility*

Before running the utility, you need to use command "chmod +x" to give executable permission to it.

```
root@innodisk:/home/innodisk/2emuc/Utility_J1939# chmod +x j1939
root@innodisk:/home/innodisk/2emuc/Utility_J1939# ./j1939
```



J1939 Utility v1.0.0

Initialization (CAN 1 | CAN 2)

ttyACM0 | Initialize | Stop | (Success)

AAC: 0	VS: 0	ECUI: 0	AAC: 0	VS: 0	ECUI: 0
IG: 0	Fn: 0	MC: 0	IG: 0	Fn: 0	MC: 0
VSI: 0	FnI: 0	ID: 200	VSI: 0	FnI: 0	ID: 300
SA: 20	Re-claimed SA: 0 ~ 127		SA: 30	Re-claimed SA: 128 ~ 253	

Send J1939 Frame

CAN 1 | Normal | Send

Requested PGN: Requested Dst: 255

PDU 1 | PGN: | Dst: 255 | Len: 8

Prio 6 | Data (hex):

ACK PGN (CAN 1 | CAN 2)

List	List
61443	65262
61444	65269
65132	65270

CAN 1 | PGN: | Add | Remove

Recv J1939 Frame

No.	CAN	Path	Prio	PGN	Description	SA	DA	Data
1	2	Recv	6	60928	Address Claimed	20	255	C8 00 00 00 00 00 00 00
2	1	Recv	6	60928	Address Claimed	30	255	2C 01 00 00 00 00 00 00

✓ Recv Only | Clear

5. Loop Back Test Program

We provide a loop back test program with source code in Windows and Linux to verify the module.

Please connector CAN1 and CAN2 with each other by using an adapter (MINI GENDER CHANGER).

When the program is running, CAN1 sends a frame to CAN2, after CAN2 receives the frame CAN2 will check if the frame is correct or not. Then turn to CAN2 sends and CAN1 receives.

If the received CAN port doesn't receive the frame or the received frame is incorrect, the program will terminate and show the result is failed.

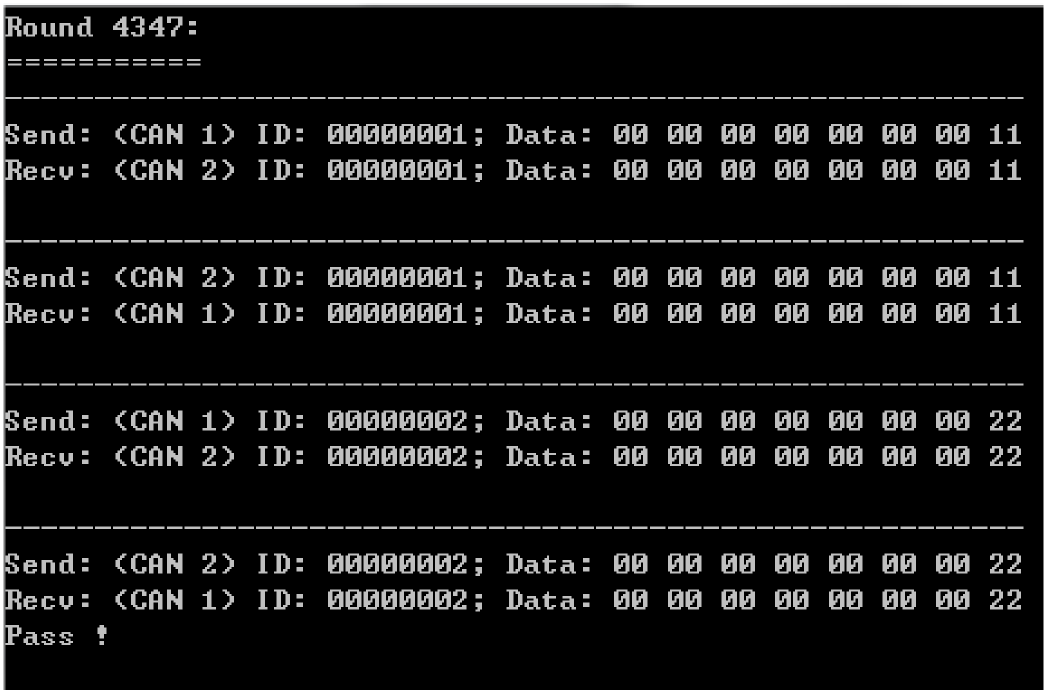
Before running the program, you can modify the "setup.ini" to set your test conditions.

COM Port	0 = auto scan (Windows), -1 = auto scan (Linux)
Baud rate	4=100K, 5=125K, 6=250K, 7=500K, 8=800K, 9=1M
Interval	1, 2, ..., 1000 [ms], sending interval between each frame
Test time	0=once, 1, 2, ..., 60 [min] Length of time you want to run the testing.
Test file	Pattern.txt The file includes ID and Data used for sending test frames.
Log file	Log.txt Used for saving the test result.

Example:

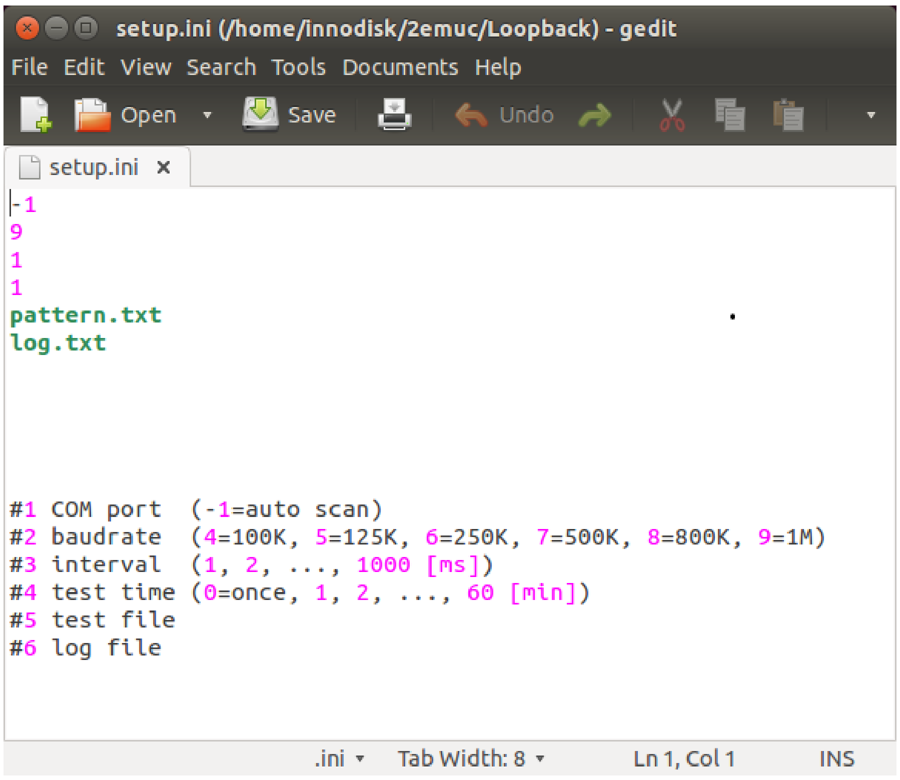
Use baud rate 1M to keep testing 1 min in Windows.

```
0
9
1
0
pattern.txt
log.txt





#1 COM port  (0=auto scan)
#2 baudrate  (4=100K, 5=125K, 6=250K, 7=500K, 8=800K, 9=1M)
#3 interval  (1, 2, ..., 1000 [ms])
#4 test time (0=once, 1, 2, ..., 60 [min])
#5 test file
#6 log file
```

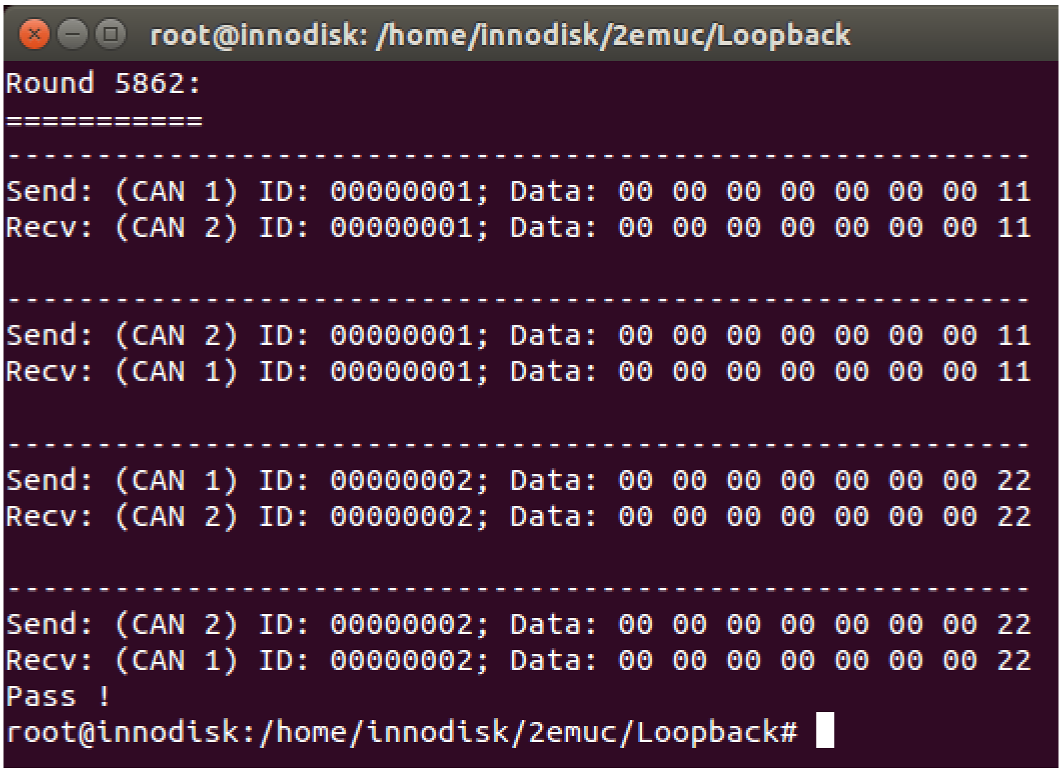


```
Round 4347:
===========
----------------------------------------------------------
Send: <CAN 1> ID: 00000001; Data: 00 00 00 00 00 00 00 11
Recv: <CAN 2> ID: 00000001; Data: 00 00 00 00 00 00 00 11

----------------------------------------------------------
Send: <CAN 2> ID: 00000001; Data: 00 00 00 00 00 00 00 11
Recv: <CAN 1> ID: 00000001; Data: 00 00 00 00 00 00 00 11

----------------------------------------------------------
Send: <CAN 1> ID: 00000002; Data: 00 00 00 00 00 00 00 22
Recv: <CAN 2> ID: 00000002; Data: 00 00 00 00 00 00 00 22

----------------------------------------------------------
Send: <CAN 2> ID: 00000002; Data: 00 00 00 00 00 00 00 22
Recv: <CAN 1> ID: 00000002; Data: 00 00 00 00 00 00 00 22
Pass !
```

Use baud rate 1M to keep testing 1 min in Linux.


setup.ini (/home/innodisk/2emuc/Loopback) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
setup.ini
.ini
Tab Width: 8
Ln 1, Col 1
INS

```
-1
9
1
1
pattern.txt
log.txt




#1 COM port  (-1=auto scan)
#2 baudrate  (4=100K, 5=125K, 6=250K, 7=500K, 8=800K, 9=1M)
#3 interval  (1, 2, ..., 1000 [ms])
#4 test time (0=once, 1, 2, ..., 60 [min])
#5 test file
#6 log file
```



```
Round 5862:
===========
------------------------------------------------------------
Send: (CAN 1) ID: 00000001; Data: 00 00 00 00 00 00 00 11
Recv: (CAN 2) ID: 00000001; Data: 00 00 00 00 00 00 00 11

------------------------------------------------------------
Send: (CAN 2) ID: 00000001; Data: 00 00 00 00 00 00 00 11
Recv: (CAN 1) ID: 00000001; Data: 00 00 00 00 00 00 00 11

------------------------------------------------------------
Send: (CAN 1) ID: 00000002; Data: 00 00 00 00 00 00 00 22
Recv: (CAN 2) ID: 00000002; Data: 00 00 00 00 00 00 00 22

------------------------------------------------------------
Send: (CAN 2) ID: 00000002; Data: 00 00 00 00 00 00 00 22
Recv: (CAN 1) ID: 00000002; Data: 00 00 00 00 00 00 00 22
Pass !
root@innodisk:/home/innodisk/2emuc/Loopback#
```

6. Software API

EMUC API is based on a dynamic library (DLL) in Windows and static library (.a) in Linux to control EMUC-B202.

There are basic CAN 2.0B and J1939 API.

The following table shows the corresponding model to these API which can be used.

Part Number	Basic CAN 2.0B API	J1939 API
EMUC-B202-W1	Yes	No
EMUC-B202-W2	Yes	Yes

6.1. COM Port Selection

EMUC-B202 is connected by virtual COM port using CDC-ACM driver.

COM port parameter of API must be given an “int” value instead of a real port name or port number in the OS.

Windows

Real COM port number-1 would be the “int” value for API.

Example: 0=COM1, 1=COM2, 2=COM3…254=COM255, 255=COM256

Linux

EMUC-B202 supports the following COM names in the path /dev. The port mapping to the following “int” values start from 0. Generally the name would be ttyACM0 or ttyACM1 in Linux.

Example: 24=ttyACM0, 25=ttyACM1

Index	Port	Index	Port	Index	Port
0	ttyCAN0	1	ttyCAN1	2	ttyCAN2
3	ttyCAN3	4	ttyCAN4	5	ttyCAN5
6	ttyCAN6	7	ttyCAN7	8	ttyCAN8
9	ttyCAN9	10	ttyCAN10	11	ttyCAN11
12	ttyCAN12	13	ttyCAN13	14	ttyCAN14
15	ttyCAN15	16	ttyUSB0	17	ttyUSB1
18	ttyUSB2	19	ttyUSB3	20	ttyUSB4
21	ttyUSB5	22	ttyAMA0	23	ttyAMA1
24	ttyACM0	25	ttyACM1	26	ttyACM2
27	ttyACM3	28	ttyACM4	29	ttyACM5
30	ttyACM6	31	ttyACM7	32	ttyACM8
33	ttyACM9	34	ttyACM10	35	ttyACM11

36	ttyACM12	37	ttyACM13	38	ttyACM14
39	ttyACM15	40	rfcomm0	41	Rfcomm1
42	Ircomm0	43	Ircomm1	44	cuau0
45	cuau1	46	cuau2	47	cuau3
48	cuaU0	49	cuaU1	50	cuaU2
51	cuaU3	52	serusb0	53	serusb1
54	serusb2	55	serusb3	56	serusb4
57	serusb5	58	serusb6	59	serusb7
60	serusb8	61	serusb9	62	serusb10
63	serusb11	64	serusb12	65	serusb13
66	serusb14	67	serusb15		

6.2. Basic CAN 2.0B Function Description

This chapter describes basic CAN 2.0B API functions and parameters.

Header file (lib_emuc_2.h) includes declaration and data structure requested for programming.

CAN status is inactive after the module is power on. The module is in configuration mode by default. In configuration mode you can use functions relate to CAN settings.

After initializing CAN status to be active, the module can start to send or receive frames. In CAN active mode, all setting functions cannot be used.

The following table shows which functions can be used in CAN inactive or active mode.

Function Name	CAN is inactive	CAN is active
EMUCShowVer	Yes	No
EMUCOpenDevice	Yes	Yes
EMUCCloseDevice	Yes	Yes
EMUCResetCAN	Yes	No
EMUCClearFilter	Yes	No
EMUCInitCAN	Yes	Yes
EMUCSetBaudRate	Yes	No
EMUCSetMode	Yes	No
EMUCSetFilter	Yes	No
EMUCSetErrorType	Yes	No
EMUCGetCfg	Yes	No

EMUCExpCfg	Yes	No
EMUCImpCfg	Yes	No
EMUCSend	No	Yes
EMUCReceive	Yes	Yes
EMUCReceiveNonblock	Yes	Yes
EMUCEnableSendQueue	Yes	Yes
EMUCGetBusError	Yes	Yes
EMUCSetRecvBlock	Yes	Yes
EMUCOpenSocketCAN	Yes	Yes

6.2.1. EMUCShowVer

Description: Get firmware and library version.

SYSTAX:

```
EMUCShowVer(int com_port, VER_INFO *ver_info)
```

VER_INFO struct:

```
typedef struct
{
    char fw[VER_LEN];
    char api[VER_LEN];
    chat model [VER_LEN];

} VER_INFO;
```

Member:

com_port: [input] The virtual COM port number.

fw: [output] Firmware version, length 16 bytes

api: [output] API version, length 16 bytes

model: [output] Model type, length 16 bytes, show as following

1. **020B:** Only support CAN basic API.
2. **1939:** Support CAN basic API and J1939 API.

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.2. EMUCOpenDevice

Description: Open virtual COM port.

SYSTAX:

`EMUCOpenDevice(int com_port)`

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.3. EMUCCloseDevice

Description: Close virtual COM port.

SYSTAX:

`EMUCCloseDevice(int com_port)`

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.4. EMUCResetCAN

Description: Reset all CAN setting to default value as following.

Baud Rate	500 Kbps
CANbus Mode	Normal mode
Filter Type	None
Filter ID	None
Filter mask	None
Error Setting	EEPROM only

SYSTAX:

```
EMUCResetCAN(int com_port)
```

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.5. EMUCClearFilter

Description: Clear CAN acceptance filter setting of specific CAN port.

SYSTAX:

```
EMUCClearFilter(int com_port, int CAN_port)
```

Member:

com_port: [input] The virtual COM port number.

CAN_port: [input] The CAN port number.

```
enum
{
   EMUC_CAN_1 = 0,
   EMUC_CAN_2 = 1
};
```

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.6. EMUCInitCAN

Description: Set CAN port to active/inactive. Default is inactive.

SYSTAX:

```
EMUCInitCAN(int com_port, int CAN1_sts, int CAN2_sts)
```

Member:

com_port: [input] The virtual COM port number.

CANx_sts: [input] CAN status value. (x=1,2)

```
enum
{
    EMUC_INACTIVE = 0,
    EMUC_ACTIVE = 1
};
```

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.7. EMUCSetBaudRate

Description: Set baud rate of CAN port.

SYSTAX:

```
EMUCSetBaudRate(int com_port, int CAN1_baud, int CAN2_baud)
```

Member:

com_port: [input] The virtual COM port number.

CANx_baud: [input] Baud rate value. (x=1,2)

```
enum
{
    EMUC_BAUDRATE_100K = 4,
    EMUC_BAUDRATE_125K =5,
    EMUC_BAUDRATE_250K =6,
    EMUC_BAUDRATE_500K =7,
    EMUC_BAUDRATE_800K =8,
    EMUC_BAUDRATE_1M =9
};
```

Return Status Code:

Value	Return Value
0	Success

1	Error

6.2.8. EMUCSetMode

Description: Set CAN port to normal mode or listen mode.

1. **Normal mode:** CAN port will send “ACK” package after receiving CAN frames.
2. **Listen mode:** CAN port will not send “ACK” package after receiving CAN frames.

SYSTAX:

```
EMUCSetMode(int com_port, int CAN1_mode, int CAN2_mode)
```

Member:

com_port: [input] The virtual COM port number.

CANx_mode: [input] CAN mode value. (x=1,2)

```
enum
{
    EMUC_NORMAL = 0,
    EMUC_LISTEN = 1
};
```

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.9. EMUCSetFilter

Description: Set CAN acceptance filter.

Please refer to *4.1. Example of CAN acceptance filter.*

SYSTAX:

```
EMUCSetMode(int com_port, FILTER_INFO *filter_info)
```

FILTER_INFO struct:

```
typedef struct
{
    int                    CAN_port;
    int             flt_type;
    unsigned int    flt_id;
    unsigned int    mask;

} FILTER_INFO;
```

Member:

com_port: [input] The virtual COM port number.

CAN_port: [input] The CAN port number.

```
enum
{
    EMUC_CAN_1 = 0,
    EMUC_CAN_2 =1
};
```

flt_type: [input] CAN filter ID type. (SID=11bit, EID=29bit)

```
enum
{
    EMUC_SID = 1,
    EMUC_EID =2
};
```

flt_id: [input]CAN frame filter ID.

mask: [input]CAN frame filter mask.

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.10. EMUCSetErrorType

Description: Set error type to receive CAN error register or EEPROM error message. Default value is EEPROM error only.

1. **EEPROM Error (used to store configuration):** Send event every 5 sec after the module power on.
2. **CANbus Error:** Send register value of CANbus error every 5 sec. Register mapping

is shown as following.

SYSTAX:

```
EMUCSetErrorType(int com_port, int err_type)
```

Member:

com_port: [input] The virtual COM port number.

err_type: [input] Error type value.

```
enum
{
    EMUC_DIS_ALL = 0,
    EMUC_EE_ERR =1,
    EMUC_BUS_ERR =2,
    EMUC_EN_ALL = 255
};
```

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.11. EMUCGetCfg

Description: Set CAN acceptance filter.

SYSTAX:

```
EMUCGetCfg(int com_port, CFG_INFO *cfg_info)
```

CFG_INFO struct:

```
typedef struct
{
    unsigned char    baud[CAN_NUM];
    unsigned char    mode[CAN_NUM];
    unsigned char    flt_type[CAN_NUM];
    unsigned int     flt_id     [CAN_NUM];
    unsigned int     flt_mask[CAN_NUM];
    unsigned char    err_set;
} CFG_INFO;
```

Member:

com_port: [input] The virtual COM port number.

mode: [output] The CAN port number.

```
enum
{
    EMUC_NORMAL = 0,
    EMUC_LISTEN = 1
};
```

flt_type: [output] CAN filter ID type. (SID=11bit, EID=29bit)

```
enum
{
    EMUC_SID = 1,
    EMUC_EID =2
};
```

flt_id: [output] CAN frame filter ID.

mask: [output] CAN frame filter mask.

err_set: [output] Error type value.

```
enum
{
    EMUC_DIS_ALL = 0,
    EMUC_EE_ERR =1,
    EMUC_BUS_ERR =2,
    EMUC_EN_ALL = 255
};
```

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.12. EMUCExpCfg

Description: Export configuration.

SYSTAX:

```
EMUCExpCfg (int com_port, const char *file_name)
```

Member:

com_port: [input] The virtual COM port number

file_name: [input] File name and path

Return Code:

Value	Description
0	Success
1	Error

6.2.13. EMUCImpCfg

Description: Import configuration.

SYSTAX:

```
EMUCImpCfg (int com_port, const char *file_name)
```

Member:

com_port: [input] The virtual COM port number.

file_name: [input] File name and path.

Return Code:

Value	Description
0	Success
1	Error

6.2.14. EMUCSend

Description: Send CAN frames.

SYSTAX:

```
EMUCSend (int com_port, CAN_FRAME_INFO *can_frame_info)
```

CAN_FRAME_INFO struct:

```
typedef struct
{
   int        CAN_port;
```

```
    int         id_type;
    int         rtr;
    int         dlc;
    int         msg_type;

    char        recv_time[TIME_CHAR_NUM];    /* e.g., 15:30:58:789 (h:m:s:ms) */
    unsigned int    id;
    unsigned char data        [DATA_LEN];
    unsigned char data_err[CAN_NUM][DATA_LEN_ERR];

} CAN_FRAME_INFO;
```

Member:

com_port: [input] the virtual COM port number.

CAN_port: [input] The CAN port number.

```
enum
{
    EMUC_CAN_1 = 0,
    EMUC_CAN_2 = 1
};
```

id_type: [input] CAN ID type. (SID=11bit, EID=29bit)

```
enum
{
    EMUC_SID = 1,
    EMUC_EID =2
};
```

rtr: [input] Remote transmit request

```
enum
{
    EMUC_DIS_RTR = 0,
    EMUC_EN_RTR =1
};
```

dlc: [input] Data length.

id: [input] CAN frame ID.

data: [input] CAN frame data.

msg_type: Don't care in sending data.
recv_time: Don't care in sending data.
data_err: Don't care in sending data.

Return Code:

Value	**Description**
0	Success
1	Error
	Queue is full (When enable send queue)

6.2.15. EMUCEnableSendQueue

Description: Allocate a queue size (10-10000) for sending data.

SYSTAX:

```
int EMUCEnableSendQueue (int com_port, bool is_enable, unsigned int queue_size)
```

Member:
com_port: [input] The virtual COM port number.
is_enable: [input] 0=false, 1=true
queue_size: [input] CAN bus frame amount.

Return Status Code:

Value	**Return Value**
0	Success
1	Error

6.2.16. EMUCReceive

Description: Receive one data.

There three types of received data define in msg_type.

1. **EMUC_DATA_TYPE:** Normal CAN frame.
2. **EMUC_EEERR_TYPE:** EEPROM error message.
3. **EMUC_BUSERR_TYPE:** Register of CANbus error status.

SYSTAX:

```
int EMUCReceive (int com_port, CAN_FRAME_INFO *can_frame_info);
```

CAN_FRAME_INFO struct:

```
typedef struct
{
    int          CAN_port;
    int          id_type;
    int          rtr;
    int          dlc;
    int          msg_type;

    char         recv_time[TIME_CHAR_NUM];    /* e.g., 15:30:58:789 (h:m:s:ms) */
    unsigned int    id;
    unsigned char data        [DATA_LEN];
    unsigned char data_err[CAN_NUM][DATA_LEN_ERR];

} CAN_FRAME_INFO;
```

Member:

com_port: [input] The virtual COM port number.

msg_type: [output] Message type of received data.

```
enum
{
    EMUC_DATA_TYPE = 0,
    EMUC_EEERR_TYPE =1,
    EMUC_BUSERR_TYPE =2
};
```

- **If msg_type=0**

CAN_port: [output] Get CAN port number

```
enum
{
    EMUC_CAN_1 = 0,
    EMUC_CAN_2 = 1
};
```

id_type: [output] Get CAN ID type (SID=11bit, EID=29bit)

```
enum
{
    EMUC_SID = 1,
    EMUC_EID =2
};
```

rtr: [output] Get remote transmit request value.

```
enum
{
    EMUC_DIS_RTR = 0,
    EMUC_EN_RTR =1
};
```

dlc: [output] Get Data length.
id: [output] Get CAN frame ID
data: [output] Get CAN frame data.
recv_time: [output] Timestamp of received data.

- **If msg_type=1**

No data need to get.

- **If msg_type=2**

data_err: [output] Get register of CAN bus error status. Please refer to *8.2.Register mapping table of CAN error status.*

Return Status Code:

Value	Return Value
0	No data
1	Get one data

6.2.17. EMUCReceiveNonblock

Description: Receive multiple data.

SYSTAX:

```
int EMUCReceiveNonblock (int com_port, NON_BLOCK_INFO *non_block_info)
```

NON_BLOCK_INFO struct:

```
typedef struct
{
    unsigned int        cnt;
    unsigned int          interval;    /* [ms] */

    CAN_FRAME_INFO     *can_frame_info;

} NON_BLOCK_INFO;
```

Member:
com_port: [input] The virtual COM port number.
cnt: [input]: Count of CAN_FRAME_INFO structure.
interval: [input] interval (ms) of receiving multiple data.
CAN_FRAME_INFO: Received data structure.

Return Status Code:

Value	**Return Value**
>0	The amount of received CAN frames
0	No data

6.2.18. EMUCReceiveNonblockCS (Used for C#)

Description: Receive multiple data in C#.

SYSTAX:

```
int EMUCReceiveNonblock (int com_port, unsigned int cnt, unsigned int interval,
CAN_FRAME_INFO *can_frame_info)
```

Member:
Please refer to the sections of *EMUCReceive* and *EMUCReceiveNonblock*.

6.2.19. EMUCSetRecvBlock (Linux only)

Description: Set block mode for EMUCReceive to receive data. Enable block mode can reduce CPU loading.
***NOTE:** EMUCReceiveNonblock cannot be used when enable receive block mode.*

The following table describes the difference between enable and disable.

Enable	EMUCReceive will not return 0 and keep block if no data.
Disable	EMUCReceive will return 0 if no data.

SYSTAX:

```
int EMUCSetRecvBlock (int com_port, bool is_enable)
```

Member:

com_port: [input] The virtual COM port number.

is_enable: [input] 0=false, 1=true

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.20. EMUCOpenSocketCAN (Linux only)

Description: Use for SocketCAN driver to Open virtual COM port.

SYSTAX:

```
EMUCOpenSocketCAN (int com_port)
```

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.21. EMUCGetBusError

Description: Need firmware v02.10. Return the register of CANbus error status immediately. This function still uses EMUCReceive to receive the returned value (msg_type=2).

SYSTAX:

```
int EMUCGetBusError (int com_port)
```

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	Return Value
0	Success
1	Error

6.2.22. EMUCOpenDeviceSCT (Windows only)

Description: Open virtual COM port with SetCommTimeouts.

SYSTAX:

```
EMUCOpenDeviceSCT(int com_port)
```

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	Return Value
0	Success
1	Error

6.3. J1939 Function Description

This chapter describes J1939 API functions and parameters.

Header file (lib_J1939.h) includes declaration and data structure requested for programming.

We can support J1939 transport protocol to send or receive CAN frames data more than 8 byte for up to 1785 byte by using “Connection Management” (PGN 60416) and “Data Transfer” (PGN 60160)

6.3.1. EMUCJ1939Init

Description: Initialize J1939 protocol with CAN baud rate 250K, specific ECU source address and ECU NAME, and then send the claim address frame (PGN 60928).

SYSTAX:

```
EMUCJ1939Init(J1939_INIT_INFO init)
```

J1939_INIT_INFO struct:

```
typedef struct
{
    int                     com_port;
    uint8_t                 sa      [CAN_PORT_NUM];    /* [0]: CAN_1, [1]: CAN_2 */
    J1939_NAME_INFO         name [CAN_PORT_NUM];

} J1939_INIT_INFO;
```

J1939_NAME_INFO struct:

```
typedef struct
{
    uint8_t    aac;
    uint8_t    ind_grp;
    uint8_t    veh_sys_inst;
    uint8_t    veh_sys;
    uint8_t    func;
    uint8_t    func_inst;
    uint8_t    ecu_inst;
    uint16_t mfg_code;
    uint32_t identy_num;

} J1939_NAME_INFO;
```

Member:

com_port: [input] The virtual COM port number

sa: [input] J1939 source address

name: [input] J1939 NAME

aac: [intput] 1-bit Arbitrary Address Capable

ind_grp: [input] 3-bit Industry Group

veh_sys_inst: [input] 4-bit Vehicle System Instance

veh_sys: [input] 7-bit Vehicle System

func: [input] 8-bit Function

func_inst: [input] 5-bit Function Instance

ecu_inst: [input] 3-bit ECU Instance

mfg_code: [input] 11-bit Manufacturer Code

identy_num: [input] 21-bit Identity Number

Return Status Code:

Value	**Return Value**
0	Success
1	Load basic CAN library failed (Windows only)
2	Open COM port failed
3	Get version failed
4	Not support J1939 protocol
5	Set baud rate failed
6	Active CAN failed
7	Create thread failed

6.3.2. EMUCJ1939Stop

Description: Stop J1939 thread

SYSTAX:

```
EMUCJ1939Stop(int com_port)
```

Member:

com_port: [input] The virtual COM port number.

Return Status Code:

Value	Return Value
0	Success
1	Error

6.3.3. EMUCJ1939Send

Description: Send J1939 frame.

SYSTAX:

```
EMUCJ1939Send(J1939_FRAME_INFO init)
```

J1939_FRAME_INFO struct:

```
typedef struct
{
    uint32_t    pgn;
    uint8_t    *buf;
    uint16_t    buf_len;
    uint8_t     dst;
    uint8_t     src;
    uint8_t     pri;
    uint8_t     port;

} J1939_FRAME_INFO;
```

Member:

pgn: [input] Parameter group number

***buf:** [input] Pointer to data

buf_len: [input] Size of data

dst: [intput] Destination address

src: [input] Source address

pri: [input] Priority

port: [input] CAN port number

```
enum
{
    CAN_1 = 0,
    CAN_2,
};
```

Return Status Code:

Value	Return Value
0	Success
1	Error

6.3.4. EMUCJ1939RegCbFunc (call back function)

Description: Register this call back function to receive J1939 events.

The following describes the cases of J1939 events:

1. **Normal PGN:** ECU receives J1939 frames with normal PGN. You can parse the data by referring J1939 PGN definition in your application code. Please refer to *8.3.Example of J1939 PGN definition.*

2. **Request PGN:** ECU receives the J1939 frame of request PGN (PGN 59904), ECU needs to return "Positive ACK (ACK_P)", "Negative ACK (ACK_N)", "Access Denied (ACK_AD)" or "Cannot Respond (ACK_CR)" base on which PGN the ECU have in your application code.

3. **Change source address:** Re-claim the source address if ECU receives the frame of claiming address (PGN 60928) that has the same source address but lower value NAME field. You must set what source address you attempt to re-claim in your application code.

   NOTE: *If another ECU claims the same address, the ECU with the lower value NAME field wins. NAME field is 64 bits long and is placed in the data field of the address claimed message.*

4. **Commanded address:** ECU receives the J1939 frame of commanded address (PGN 65240), and the NAME in the data field is the same as ECU owns, the 9th byte of data is the source address which is used to set the ECU to this specific address. This can be done by a diagnostic tool or an interconnecting ECU (bridge, gateway).

SYSTAX:

```
EMUCJ1939RegCbFunc(J1939_CB_INFO *cb_info)
```

J1939_CB_INFO struct:

```
typedef struct
```

```
{
    int                     msg_type;
    int                     ack_type;

    uint8_t                 sa;
    uint8_t                 sa_req_port;
    uint32_t                req_pgn;

    J1939_FRAME_INFO    frame;
    J1939_CB_FUNC           cb_func;

} J1939_CB_INFO;
```

Member:

cb_func: [input] register a call back function below. The function name could be modified.

```
void j1939_cb_handler (void *ptr);
J1939_CB_INFO           cb_info;
cb_info.cb_func = j1939_cb_handler;
EMUCJ1939RegCbFunc(&cb_info);
```

msg_type: [output] Identify the PGN cases

```
enum
{
    NORMAL_PGN = 0,
    REQUEST_PGN =1,
    CHANGE_SA = 2,
    CMD_SA=3
};
```

- **If msg_type=0 (NORMAL_PGN)**

Receive J1939 frames directly then parse them in the application code.

frame: [output] J1939 frame information

J1939_FRAME_INFO struct:

```
typedef struct
{
    uint32_t    pgn;
    uint8_t    *buf;
    uint16_t    buf_len;
    uint8_t     dst;
    uint8_t     src;
    uint8_t     pri;
    uint8_t     port;

} J1939_FRAME_INFO;
```

- **If msg_type=1 (REQUEST_PGN)**

frame: [output] J1939 frame information
req_pgn: [output] PGN which is being requested. (Data field of PGN 59904)
sa_req_port: [output] The CAN port of the source address.
ack_type: [input] Return "Positive ACK (ACK_P)", "Negative ACK (ACK_N)", "Access Denied (ACK_AD)" or "Cannot Respond (ACK_CR)".

```
enum
{
    ACK_P = 0,
    ACK_N=1,
    ACK_AD=2,
    ACK_CR=3
};
```

- **If msg_type=2 (CHANGE_SA)**

frame: [output] J1939 frame information
sa: [input] The source address which ECU uses to re-claims.
sa_req_port: [output] The CAN port of the source address.

- **If msg_type=3 (CMD_SA)**

frame: [output] J1939 frame information
sa: [output] The source address which ECU is commanded to change.
sa_req_port: [output] The CAN port of the source address.

7. Sample Code

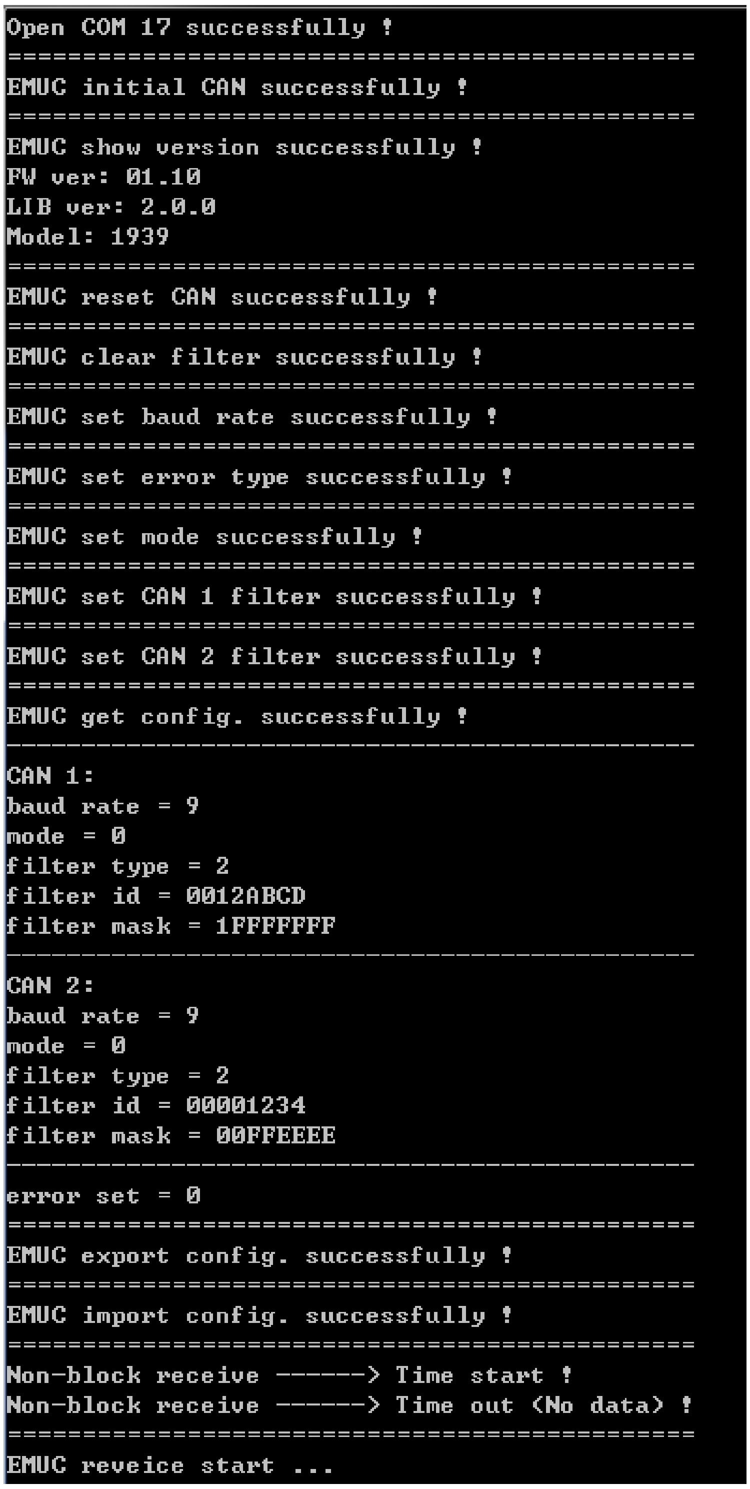
We provide Windows and Linux sample code of APIs for reference

7.1. Basic CAN 2.0B Sample Code

This sample code can be configured by editing "setup.ini"

7.1.1. Running Result

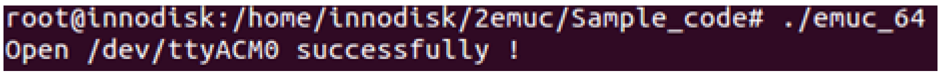
Windows sample code running result.



```
Open COM 17 successfully !
==============================================
EMUC initial CAN successfully !
==============================================
EMUC show version successfully !
FW ver: 01.10
LIB ver: 2.0.0
Model: 1939
==============================================
EMUC reset CAN successfully !
==============================================
EMUC clear filter successfully !
==============================================
EMUC set baud rate successfully !
==============================================
EMUC set error type successfully !
==============================================
EMUC set mode successfully !
==============================================
EMUC set CAN 1 filter successfully !
==============================================
EMUC set CAN 2 filter successfully !
==============================================
EMUC get config. successfully !
----------------------------------------------
CAN 1:
baud rate = 9
mode = 0
filter type = 2
filter id = 0012ABCD
filter mask = 1FFFFFFF
----------------------------------------------
CAN 2:
baud rate = 9
mode = 0
filter type = 2
filter id = 00001234
filter mask = 00FFEEEE
----------------------------------------------
error set = 0
==============================================
EMUC export config. successfully !
==============================================
EMUC import config. successfully !
==============================================
Non-block receive ------> Time start !
Non-block receive ------> Time out (No data) !
==============================================
EMUC reveice start ...
```

Linux sample code running result is the same as Windows. Only the COM port is different.

NOTE: *Please run the command "make clean" then "make" to build the executed file.*



```
root@innodisk:/home/innodisk/2emuc/Sample_code# ./emuc_64
Open /dev/ttyACM0 successfully !
```

7.2. J1939 Sample Code

This sample code will do the following function.

1. Auto-detect COM port and Initialize J1939 protocol. (All the values are Decimal)

CAN Port	CAN1	CAN2
Baud Rate	250 Kbps	250 Kbps
Source Address	20	30
Arbitrary Address Capable	0	0
Industry Group	0	0
Vehicle System Instance	0	0
Vehicle System	0	0
Function	0	0
Function Instance	0	0
ECU Instance	0	0
Manufacturer Code	0	0
Identity Number	200	201

2. If there is another ECU claims the same address and CAN1 lose, CAN1 will reclaims address by using 253, 252, 251…3, 2, 1, 0, if all addresses are used up, the address will be set to 254 (Cannot claim source address).

3. If there is another ECU claims the same address and CAN2 lose, CAN2 will reclaims address by using 0, 1, 2, 3…251, 252, 253, if all addresses are used up, the address will be set to 254 (Cannot claim source address).

4. CAN1 send the following J1939 frame.

PGN 256 (0x0100)	Undefined
Data Length	8
PDU Format	1
PDU Specification	Destination Address (global or specific)
Priority	6
Source Address	20
Designation Address	30
Data (hex)	0x1122334455667788

PGN 61444 (0xF004)	Electronic Engine Controller 1
Data Length	8
PDU Format	240
PDU Specification	4

Priority	6
Source Address	20
Designation Address	255
Data (hex)	0x1122334455667788

PGN 256 (0x0100)	Undefined
Data Length	16 (transport protocol)
PDU Format	1
PDU Specification	Destination Address (global or specific)
Priority	7
Source Address	20
Designation Address	255
Data (hex)	0x11223344556677889900AABBCCDDEEFF

PGN 59904 (0xEA00)	Request PGN
Data Length	3
PDU Format	234
PDU Specification	Destination Address (global or specific)
Priority	6
Source Address	20
Designation Address	255
Data (hex)	0x04F000 (PGN 61444)

PGN 59904 (0xEA00)	Request PGN
Data Length	3
PDU Format	234
PDU Specification	Destination Address (global or specific)
Priority	6
Source Address	20
Designation Address	255
Data (hex)	0x03F000 (PGN 61443)

5. CAN1 sends PGN 59392 automatically with “Positive ACK” when receiving PGN 59904 and requested PGN is 61443. Receiving all the other requested PGNs will return “Negative ACK”.

6. CAN2 sends PGN 59392 automatically with “Positive ACK” when receiving PGN

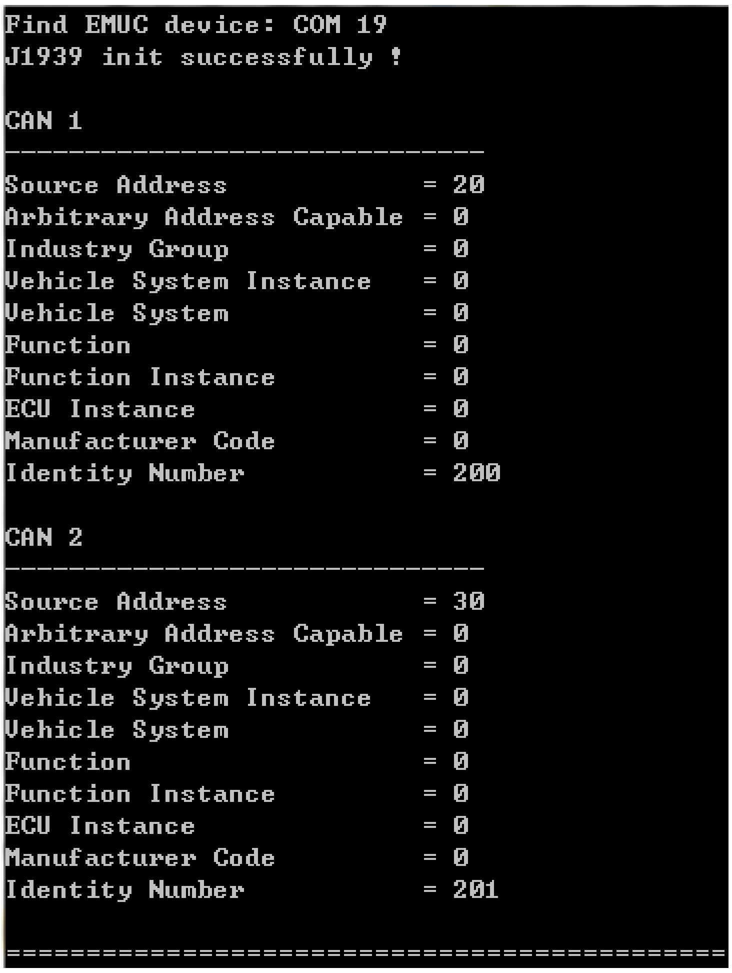
59904 and requested PGN is 61444. Receiving all the other requested PGNs will return "Negative ACK".

7. CAN1 sends PGN 65240 (Commanded address) to ask CAN2 change its source address to 170.

PGN 65240 (0xFED8)	Commanded address
Data Length	9
PDU Format	254
PDU Specification	216
Priority	6
Source Address	20
Designation Address	255
Data (hex)	0xC900000000000000AA

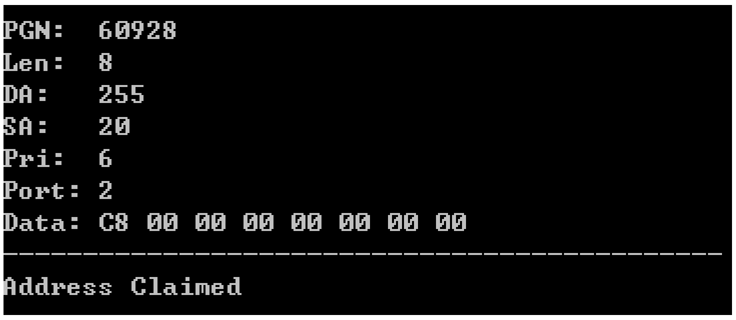
7.2.1. Running Result

Windows J1939 sample code running result by connecting CAN1 and CAN2 with each other.



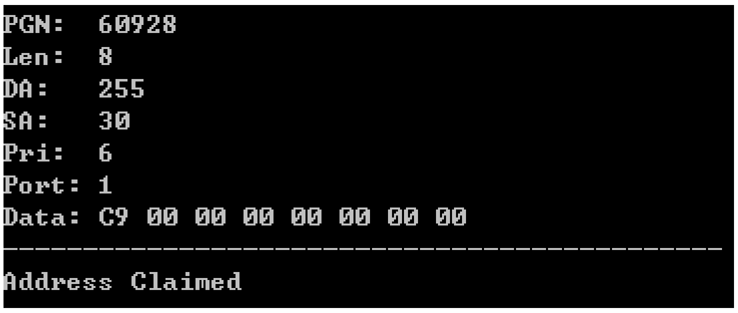
```
Find EMUC device: COM 19
J1939 init successfully !

CAN 1
-------------------------------
Source Address              = 20
Arbitrary Address Capable  = 0
Industry Group              = 0
Vehicle System Instance     = 0
Vehicle System              = 0
Function                    = 0
Function Instance           = 0
ECU Instance                = 0
Manufacturer Code           = 0
Identity Number             = 200

CAN 2
-------------------------------
Source Address              = 30
Arbitrary Address Capable  = 0
Industry Group              = 0
Vehicle System Instance     = 0
Vehicle System              = 0
Function                    = 0
Function Instance           = 0
ECU Instance                = 0
Manufacturer Code           = 0
Identity Number             = 201


==============================================
```

CAN2 receives address claimed from CAN1.



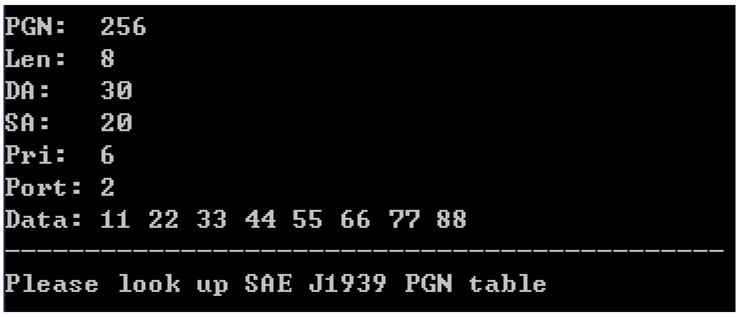
```
PGN:  60928
Len:  8
DA:   255
SA:   20
Pri:  6
Port: 2
Data: C8 00 00 00 00 00 00 00
---------------------------------------------
Address Claimed
```

CAN1 receives address claimed from CAN2.

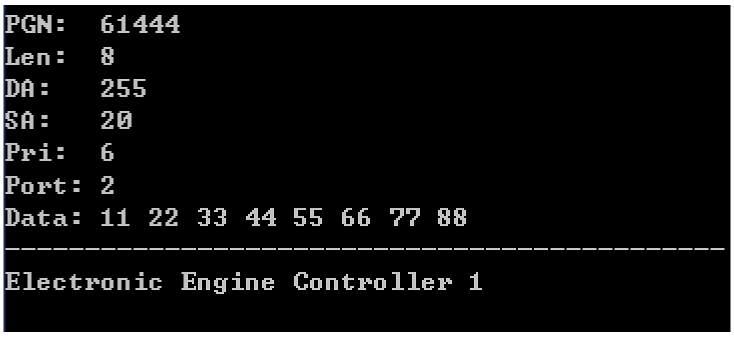


```
PGN:  60928
Len:  8
DA:   255
SA:   30
Pri:  6
Port: 1
Data: C9 00 00 00 00 00 00 00
---------------------------------------------
Address Claimed
```

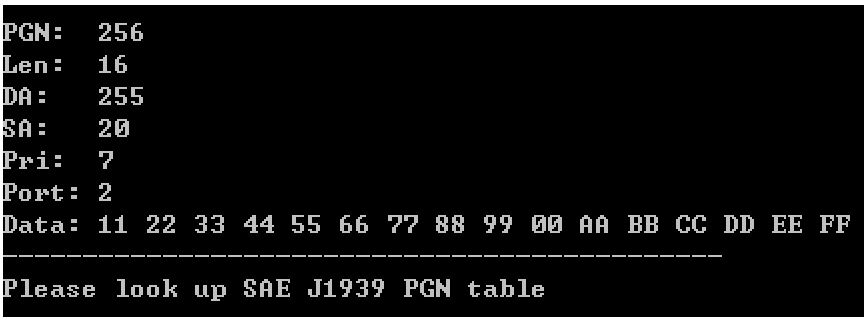
CAN2 receives J1939 frames from CAN1.



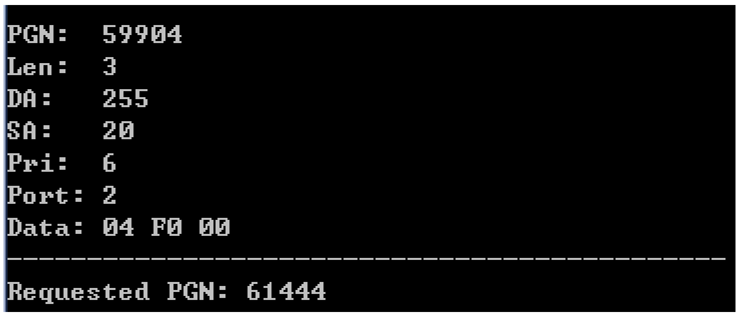
```
PGN:  256
Len:  8
DA:   30
SA:   20
Pri:  6
Port: 2
Data: 11 22 33 44 55 66 77 88
---------------------------------------------
Please look up SAE J1939 PGN table
```



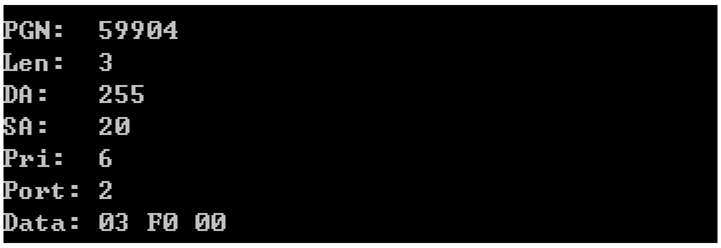
```
PGN:  61444
Len:  8
DA:   255
SA:   20
Pri:  6
Port: 2
Data: 11 22 33 44 55 66 77 88
---------------------------------------------
Electronic Engine Controller 1
```



```
PGN:  256
Len:  16
DA:   255
SA:   20
Pri:  7
Port: 2
Data: 11 22 33 44 55 66 77 88 99 00 AA BB CC DD EE FF
--------------------------------------------------------
Please look up SAE J1939 PGN table
```

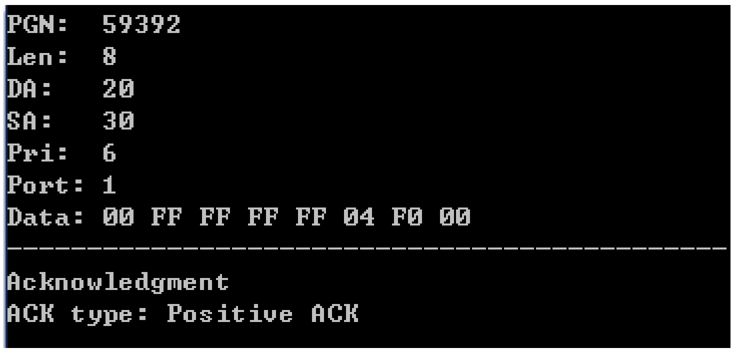


```
PGN:  59904
Len:  3
DA:   255
SA:   20
Pri:  6
Port: 2
Data: 04 F0 00
------------------------------------------------------------
Requested PGN: 61444
```

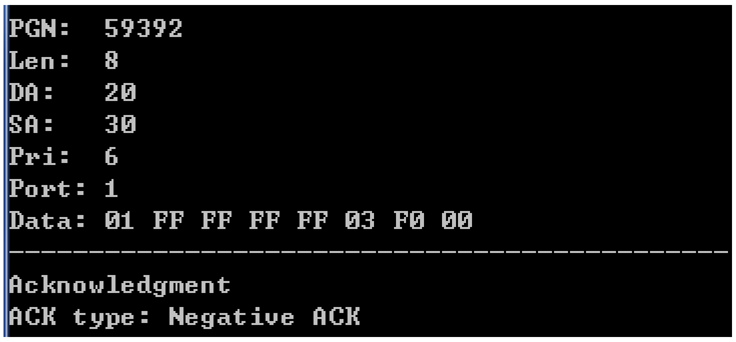


```
PGN:  59904
Len:  3
DA:   255
SA:   20
Pri:  6
Port: 2
Data: 03 F0 00
```

CAN1 receives acknowledges of requested PGN 61443 and 61444 from CAN2.



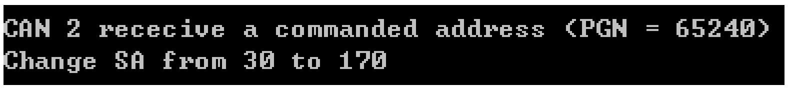
```
PGN:  59392
Len:  8
DA:   20
SA:   30
Pri:  6
Port: 1
Data: 00 FF FF FF FF 04 F0 00
------------------------------------------------------------
Acknowledgment
ACK type: Positive ACK
```



```
PGN:  59392
Len:  8
DA:   20
SA:   30
Pri:  6
Port: 1
Data: 01 FF FF FF FF 03 F0 00
------------------------------------------------------------
Acknowledgment
ACK type: Negative ACK
```

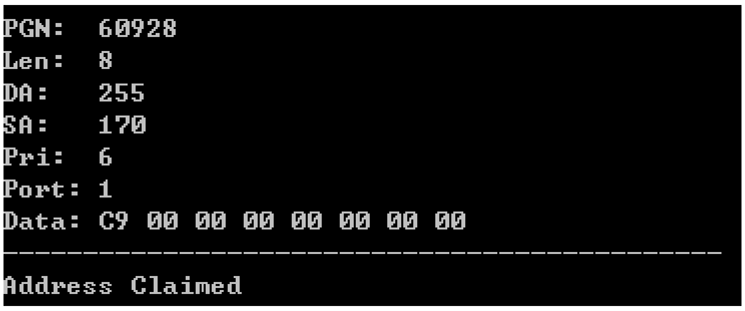
CAN1 send a commanded address to CAN2.

After CAN2 receive the command, it changes its source address from 30 to 170 and claims address again.



```
CAN 2 rececive a commanded address (PGN = 65240)
Change SA from 30 to 170
```

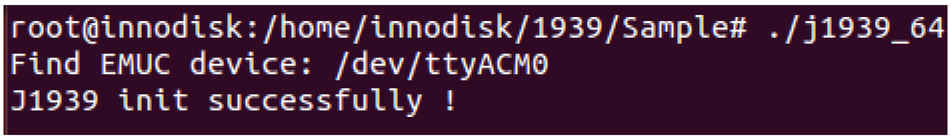
CAN1 receives new address claimed from CAN2.



```
PGN:   60928
Len:   8
DA:    255
SA:    170
Pri:   6
Port:  1
Data:  C9  00 00 00 00 00 00 00
-------------------------------------------
Address Claimed
```

Linux J1939 sample code running result is the same as Windows. Only the COM port is different.

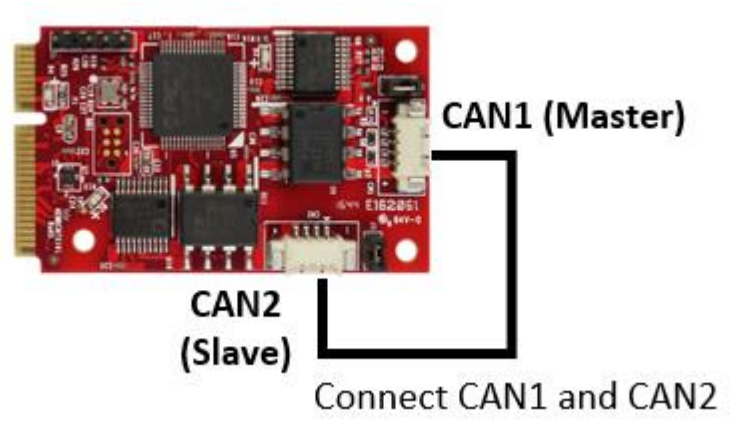
NOTE: *Please run the command "make clean" then "make" to build the executed file.*



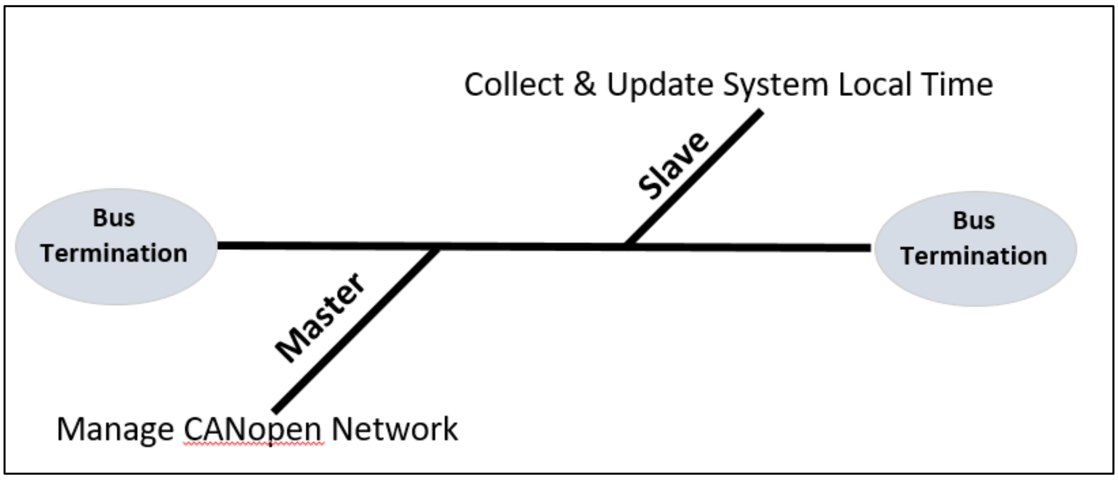
```
root@innodisk:/home/innodisk/1939/Sample# ./j1939_64
Find EMUC device: /dev/ttyACM0
J1939 init successfully !
```

7.3. CANopen Sample Code

Sample code uses CAN1 be the CANopen Master and CAN2 be the CANopen Slave. Please connect CAN1 and CAN2 with each other before running the sample code.

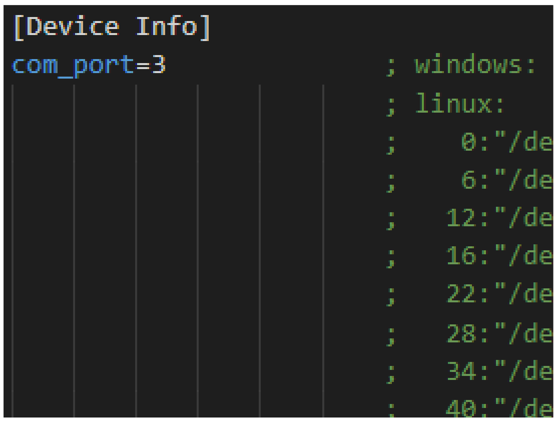


The following is the simple CANopen network performed this sample code. Master manages the CANopen network. Slave collects local time information and updates by EMUCWriteOD().



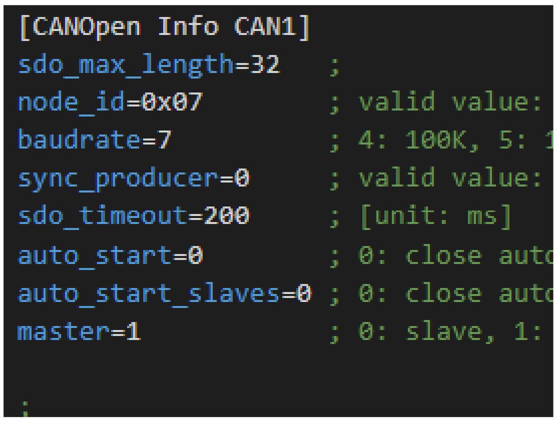
7.3.1. Sample code setting description (setup.ini)

Real COM port number-1 would be the "int" value for API. (windows: 0=COM1, 1=COM2…255=COM256; linux: 24=ttyACM0, 25=ttyACM1)



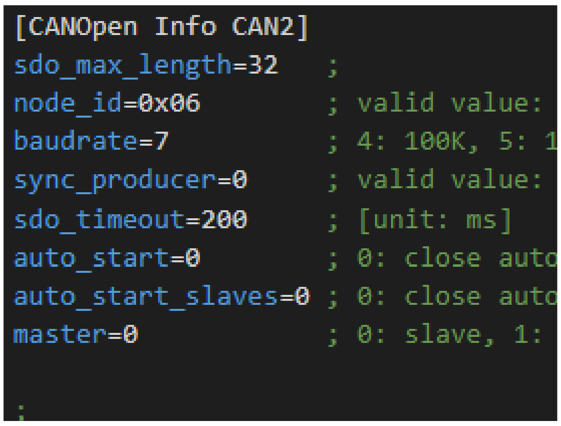
```
[Device Info]
com_port=3                  ; windows:
                            ; linux:
                            ;    0:"/de
                            ;    6:"/de
                            ;   12:"/de
                            ;   16:"/de
                            ;   22:"/de
                            ;   28:"/de
                            ;   34:"/de
                            ;   40:"/de
```

Initial CANopen information of CAN1 be the CANopen master (master=1).



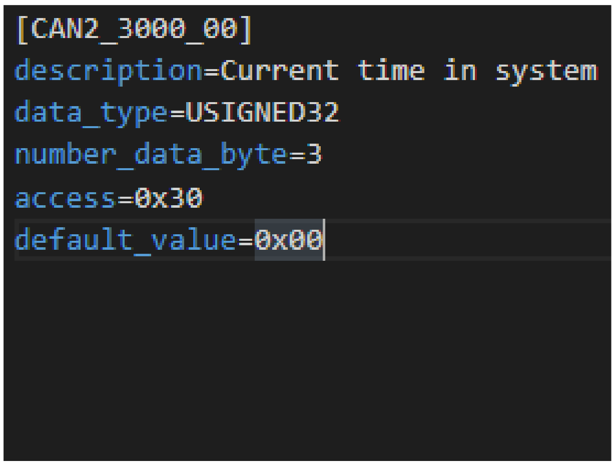
```
[CANOpen Info CAN1]
sdo_max_length=32   ;
node_id=0x07        ; valid value:
baudrate=7          ; 4: 100K, 5: 1
sync_producer=0     ; valid value:
sdo_timeout=200     ; [unit: ms]
auto_start=0        ; 0: close auto
auto_start_slaves=0 ; 0: close auto
master=1            ; 0: slave, 1:

;
```

Initial CANopen information of CAN2 be the CANopen slave (master=0).



```
[CANOpen Info CAN2]
sdo_max_length=32   ;
node_id=0x06        ; valid value:
baudrate=7          ; 4: 100K, 5: 1
sync_producer=0     ; valid value:
sdo_timeout=200     ; [unit: ms]
auto_start=0        ; 0: close auto
auto_start_slaves=0 ; 0: close auto
master=0            ; 0: slave, 1:

;
```

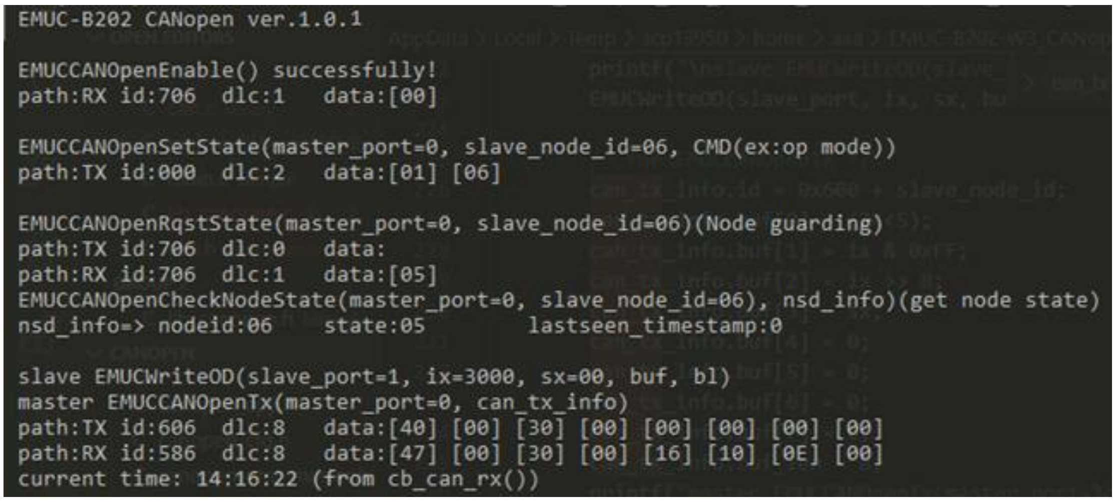
Initial CANopen object dictionary(OD) of CAN2. In the sample code, CAN2 writes current system time per second. (index: 0x3000, sub-index: 0x00, number data byte: 3, access: r/w)
Setup.ini can create by ini_generator application.



```
[CAN2_3000_00]
description=Current time in system
data_type=USIGNED32
number_data_byte=3
access=0x30
default_value=0x00
```

7.3.2. Running Result

First, use EMUCCANOpenEnable() to enable a CANopen network with the parameter information of a selected INI file and function pointers of callback handlers.
Second, master transmits a CANOpen NMT communication object for setting slave to become op-mode by EMUCCANOpenSetState(). Then use EMUCCANOpenRqstState() and EMUCCANOpenCheckNodeState() to check node state.
Finally, slave use EMUCWriteOD() to update current system time per second and master use EMUCCANOpenTx to get it (from cb_can_rx()).
In this case, data[7]=0x0E (current hour=14), data[6]=0x10 (current minute=16), data[5]=0x16 (current second=22).



```
EMUC-B202 CANopen ver.1.0.1

EMUCCANOpenEnable() successfully!
path:RX id:706  dlc:1   data:[00]

EMUCCANOpenSetState(master_port=0, slave_node_id=06, CMD(ex:op mode))
path:TX id:000  dlc:2   data:[01] [06]

EMUCCANOpenRqstState(master_port=0, slave_node_id=06)(Node guarding)
path:TX id:706  dlc:0   data:
path:RX id:706  dlc:1   data:[05]
EMUCCANOpenCheckNodeState(master_port=0, slave_node_id=06), nsd_info)(get node state)
nsd_info-> nodeid:06    state:05        lastseen_timestamp:0

slave EMUCWriteOD(slave_port=1, ix=3000, sx=00, buf, bl)
master EMUCCANOpenTx(master_port=0, can_tx_info)
path:TX id:606  dlc:8   data:[40] [00] [30] [00] [00] [00] [00] [00]
path:RX id:586  dlc:8   data:[47] [00] [30] [00] [16] [10] [0E] [00]
current time: 14:16:22 (from cb_can_rx())
```

8. Appendix

8.1. Example of CAN acceptance filter

The filter mask is used to determine which bits in the identifier of the received f are compared with the filter

- If a mask bit is set to a zero, the corresponding ID bit will automatically be accepted, regardless of the value of the filter bit.

- If a mask bit is set to a one, the corresponding ID bit will be compare with the value of the filter bit; if they match it is accepted otherwise the frames is rejected.

Example 1:

We wish to accept only frames with ID of 00001567 (hexadecimal values)

- set filter to 00001567
- set mask to 1FFFFFFF

When a frame arrives its ID is compared with the filter and all bits must match; any frame that does not match ID 00001567 is rejected

Example 2:

We wish to accept only frames with IDs of 00001560 through to 0000156F

- set filter to 00001560
- set mask to 1FFFFFF0

When a frame arrives its ID is compared with the filter and all bits except bits 0 to 3 must match; any other frame is rejected

Example 3:

We wish to accept only frames with IDs of 00001560 through to 00001567

- set filter to 00001560
- set mask to 1FFFFFF8

When a frame arrives its ID is compared with the filter and all bits except bits 0 to 2 must match; any other frame is rejected

Example 4:

We wish to accept any frame

- set filter to 0
- set mask to 0

All frames are accepted

8.2. Register mapping table of CAN error status

bit 21 TXBO: Transmitter in Error State Bus OFF (TERRCNT >= 256)

bit 20 TXBP: Transmitter in Error State Bus Passive (TERRCNT >= 128)

bit 19 RXBP: Receiver in Error State Bus Passive (RERRCNT >= 128)

bit 18 TXWARN: Transmitter in Error State Warning (128 > TERRCNT >= 96)

bit 17 RXWARN: Receiver in Error State Warning (128 > RERRCNT >= 96)

bit 16 EWARN: Transmitter or Receiver is in Error State Warning

bit 15-8 TERRCNT<7:0>: Transmit Error Counter

bit 7-0 RERRCNT<7:0>: Receive Error Counter

8.3. Example of J1939 PGN definition

PGN 60928 (0xEE00)		Address Claimed
Transmission Repetition		As required
Data Length		8 bytes
PDU Format		238
PDU Specification		255 (global address)
Default Priority		6
Source Address		0 to 253 (254 for cannot claim)
Data Position	**Length**	**Parameter Name**
1-3.5	21 bits	Identity Number
3.6-4.8	11 bits	Manufacturer Code
5.1-5.3	3 bits	ECU Instance
5.4-5.8	5 bits	Function Instance
6.1-6.8	8 bits	Function
7.2-7.8	7 bits	Vehicle System
8.1-8.4	4 bits	Vehicle System Instance
8.5-8.7	3 bits	Industry Group
8.8	1 bit	Arbitrary Address Capable

PGN 65240 (0xFED8)		Commanded Address
Transmission Repetition		As required
Data Length		9 bytes
PDU Format		254
PDU Specification		216
Default Priority		6
Data Position	**Length**	**Parameter Name**
1-3.5	21 bits	Identity Number
3.6-4.8	11 bits	Manufacturer Code
5.1-5.3	3 bits	ECU Instance
5.4-5.8	5 bits	Function Instance
6.1-6.8	8 bits	Function
7.2-7.8	7 bits	Vehicle System
8.1-8.4	4 bits	Vehicle System Instance
8.5-8.7	3 bits	Industry Group
8.8	1 bit	Arbitrary Address Capable
9.1-9.8	8 bits	New Source Address (Data range: 0-253)

PGN 61444 (0xF004)		**Electronic Engine Controller 1**	
Transmission Repetition		100ms	
Data Length		8 bytes	
PDU Format		240	
PDU Specification		4	
Default Priority		3	
Data Position	**Length**	**Parameter Name**	**SPN**
1.1-1.4	4 bits	Engine Torque Mode	899
2.1-2.8	1 byte	Driver's Demand Engine - Percent Torque	512
3.1-3.8	1 byte	Actual Engine - Percent Torque	513
4.1-5.8	2 bytes	Engine Speed	190
6.1-6.8	1 byte	Source Address of Controlling device	1483
7.1-7.4	4 bits	Engine Starter Mode	1675
8.1-8.8	1 byte	Engine Demand – Percent Torque	2432

PGN 61443 (0xF003)		**Electronic Engine Controller 2**	
Transmission Repetition		50ms	
Data Length		8 bytes	
PDU Format		240	
PDU Specification		3	
Default Priority		3	
Data Position	**Length**	**Parameter Name**	**SPN**
1.1-1.2	2 bits	Accelerator Pedal 1 Low Idle Switch	558
1.3-1.4	2 bits	Accelerator Pedal Kickdown Switch	559
1.5-1.6	2 bits	Road Speed Limit Status	1437
1.7-1.8	2 bits	Accelerator Pedal 2 Low Idle Switch	2970
2.1-2.8	1 byte	Accelerator Pedal Position 1	91
3.1-3.8	1 byte	Engine Percent Load At Current Speed	92
4.1-4.8	1 byte	Remote Accelerator Pedal Position	974
5.1-5.8	1 byte	Accelerator Pedal Position 2	29
6.1-6.2	2 bits	Vehicle Acceleration Rate Limit Status	2979
7.1-7.8	1 byte	Actual Maximum Available - Percent Torque	3357

PGN 65262 (0xFEEE)		**Engine Temperature 1**	
Transmission Repetition		1s	
Data Length		8 bytes	
PDU Format		254	
PDU Specification		238	
Default Priority		6	
Data Position	**Length**	**Parameter Name**	**SPN**
1.1-1.8	1 byte	Engine Coolant Temperature	110
2.1-2.8	1 byte	Engine Fuel Temperature 1	174
3.1-4.8	2 bytes	Engine Oil Temperature 1	175
5.1-6.8	2 bytes	Engine Turbocharger Oil Temperature	176
7.1-7.8	1 byte	Engine Intercooler Temperature	52
8.1-8.8	1 byte	Engine Intercooler Thermostat Opening	1134

PGN 65269 (0xFEF5)		**Ambient Conditions**	
Transmission Repetition		1s	
Data Length		8 bytes	
PDU Format		254	
PDU Specification		245	
Default Priority		6	
Data Position	**Length**	**Parameter Name**	**SPN**
1.1-1.8	1 byte	Barometric Pressure	108
2.1-3.8	2 byte	Cab Interior Temperature	170
4.1-5.8	2 bytes	Ambient Air Temperature	171
6.1-6.8	1 bytes	Engine Air Inlet Temperature	172
7.1-8.8	2 byte	Road Surface Temperature	79

PGN 59904 (0xEA00)	**Request PGN**
Data Length	3 bytes
PDU Format	234
PDU Specification	Destination Address (global or specific)
Default Priority	6
Byte: 1,2,3 Parameter Group Number being requested	

PGN 59392 (0xE800)		**Acknowledgement**
Transmission Repetition		As required
Data Length		8 bytes
PDU Format		232
PDU Specification		Destination Address (global or specific)
Default Priority		6
Data Position	**Length**	**Parameter Name**
1.1-1.8	8 bits	• Positive Acknowledgment: Control byte = 0 • Negative Acknowledgment: Control byte = 1 • Access Denied (PGN supported but security denied access) Control byte = 2 • Cannot Respond (PGN supported but ECU is busy and cannot respond now. Re-request the data at a later time.) Control byte = 3
2.1-2.8	8 bits	Group Function Value (If applicable)
3.1-5.8	24 bits	Reserved for assignment by SAE, these bytes should be filled with 0xFF
6.1-8.8	24 bits	PGN of the requested message

Contact us

Headquarters (Taiwan)

5F., No. 237, Sec. 1, Datong Rd., Xizhi Dist., New Taipei City 221, Taiwan
Tel: +886-2-77033000
Email: sales@innodisk.com

Branch Offices:

USA

usasales@innodisk.com
+1-510-770-9421

Europe

eusales@innodisk.com
+31-40-3045-400

Japan

jpsales@innodisk.com
+81-3-6667-0161

China

sales_cn@innodisk.com
+86-755-21673689

www.innodisk.com