# EMUC-B202

USB to dual isolated CANbus 2.0B/J1939

User Manual

Rev 1.1

## 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. |

# Table of Contents

# 1. Introduction

Innodisk EMUC-B202 CANbus card provides dual isolated CAN ports. It can connect with either mPCIe slot or USB pin header.
EMUC-B202 can save port configurations (baud rate/CAN mode/filter/error setting) into EEPROM automatically and also can export or import configuration by software.

We provide basic CAN 2.0B and J1939 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 API | J1939 API |
|---|---|---|
| EMUC-B202-W1 | Yes | No |
| EMUC-B202-W2 | Yes | Yes |

## Features

- CANbus 2.0B backward compatible with 2.0A
- Support baud rate 100/125/250/500(default)/800/1000K
- Support CAN message acceptance filter
- Keep configuration after hardware reboot
- Up to 6000 CAN messages per second (receive data)
- Support Listen-only mode
- Additional driver to support Linux SocketCAN
- Support SAE J1939 high layer protocol (Optional)
- Termination resistor enabled/disabled by jumper
- Complies with EN61000-4-5 2.5kV Surge protection
- Complies with IEC 60950-1:2005 + A1: 2009 + A2:2013 2.5kV HiPOT protection
- Complies with EN61000-4-2 (ESD) Air-15kV, Contact-8kV
- Supports 3rd mounting hole and USB Pin header for out-of-minicard installation
- 30μ ” golden finger, 3 years warranty
- Supports -40 to +85 degrees
- Industrial design, manufactured in Innodisk Taiwan

## 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 |

## DB9 Pin Define

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| NC | CAN-L | GND | NC | NC | NC | CAN-H | NC | NC |

## CAN Connector Pin Define

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| NC | CAN-H | CAN-L | GND |

## USB Pin Header Pin Define

| 1 | 2 | 3 | | 4 |
|---|---|---|---|---|
| 5V | D- | D+ | | GND |

# 2. Hardware Installation

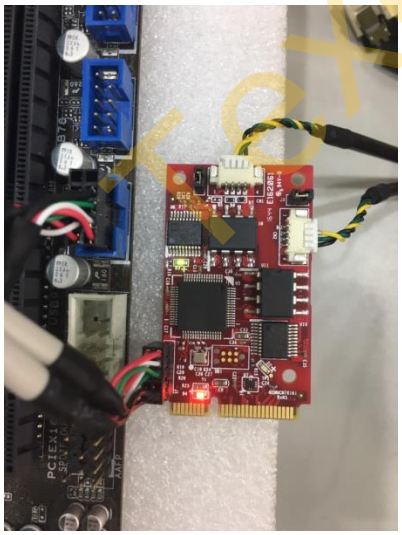
EMUC-B202 CANbus module uses USB 2.0 input interface, there are dual options to install the module.

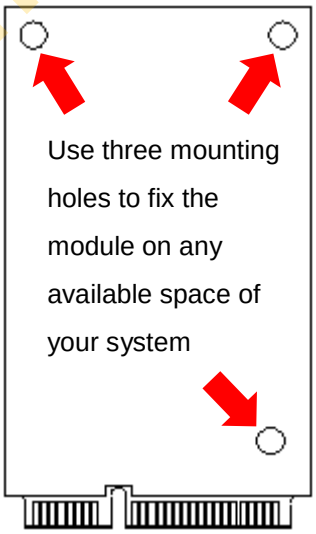
## 2.1. mPCIe

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

## 2.2. 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.





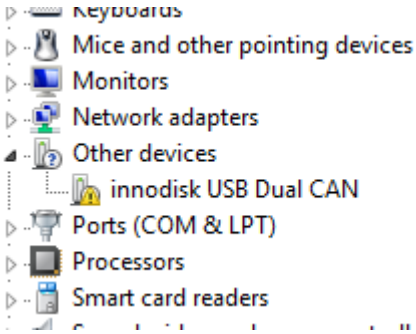

***NOTE:*** *This USB cable in the picture is not included in the package; you need to design your own USB cable.*

# 3. Windows OS

## 3.1. Driver Installation

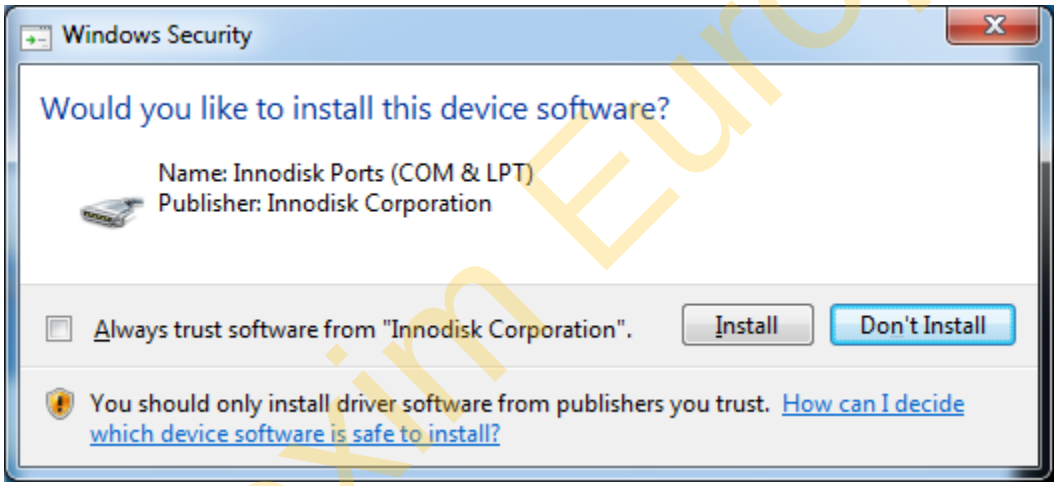
Install EMUC-B202 either into mPCIe slot or with USB pin header. The device named "innodisk USB Dual CAN" can be found in "Device Manager".

Run the driver package as administrator.

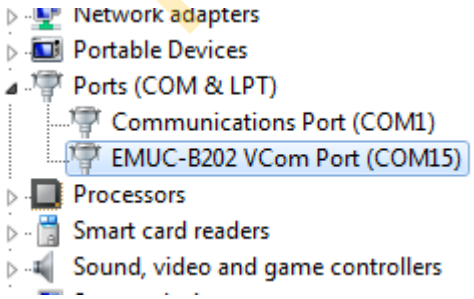




When prompt "Windows Security" dialog, click "Install".

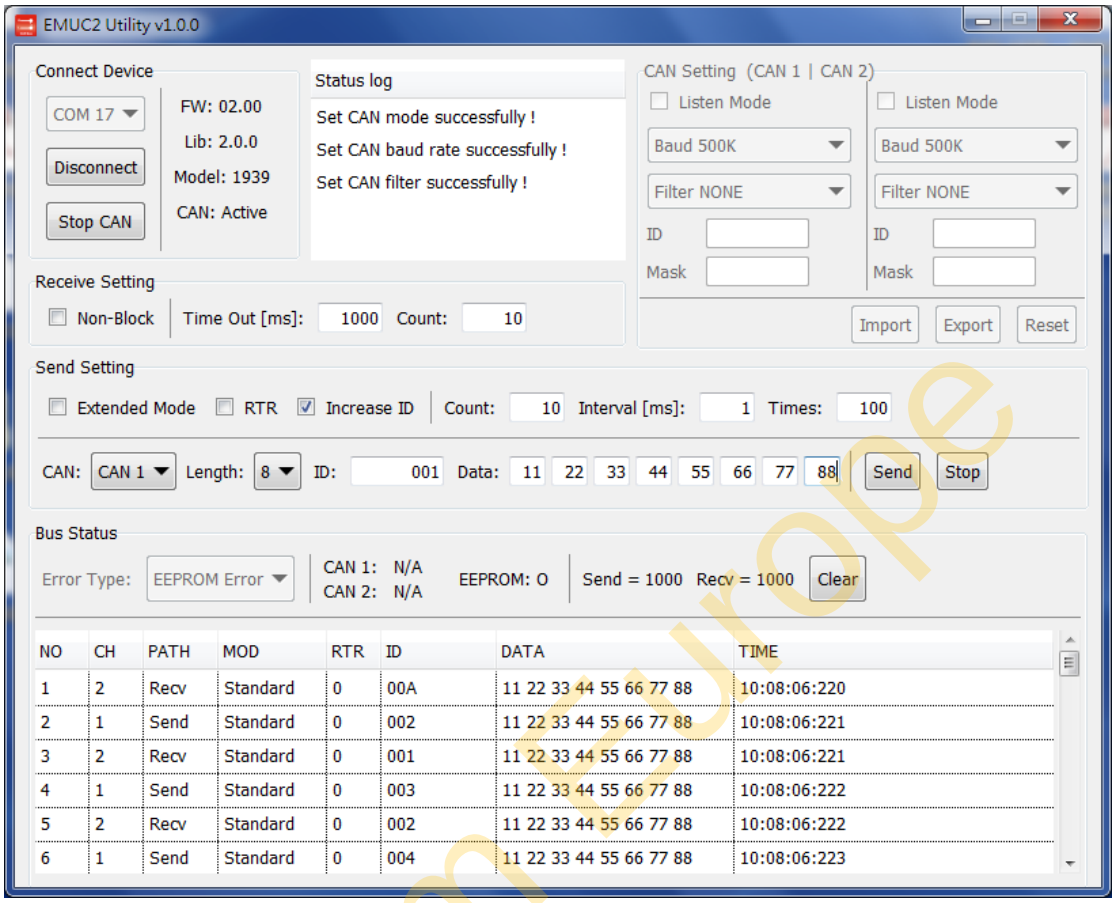



After installing the driver, device can be recognized as a COM port named "EMUC-B202 VCom Port".



## 3.2. Basic CAN 2.0B Test Utility

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



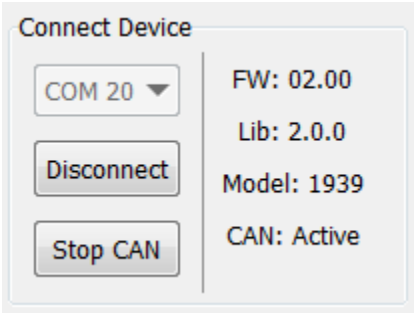

### 3.2.1. Connect Device

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:**

| | |
|---|---|
|  | 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 |

| | |
|---|---|
|  | Firmware version is v2.00 |
| | CAN API version is v2.0.0 |
| | This model can support basic CAN and J1939 API |
| | CAN is active to send/receive CAN frames |

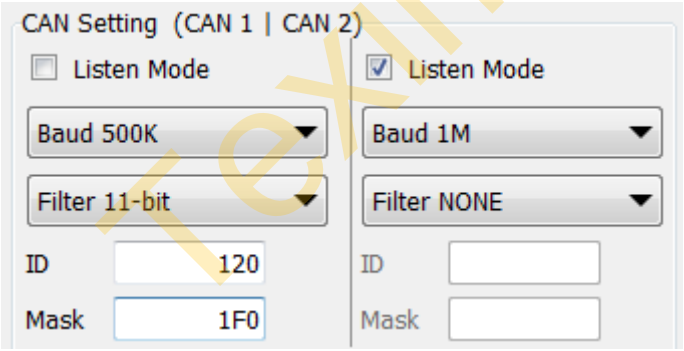
### 3.2.2. CAN Setting

***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:**




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.

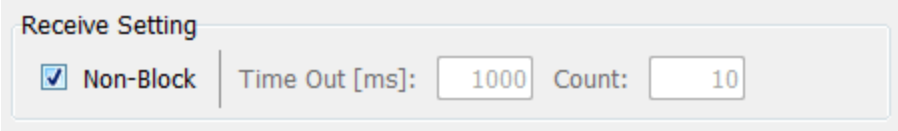
### 3.2.3. Receive Setting

***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:**



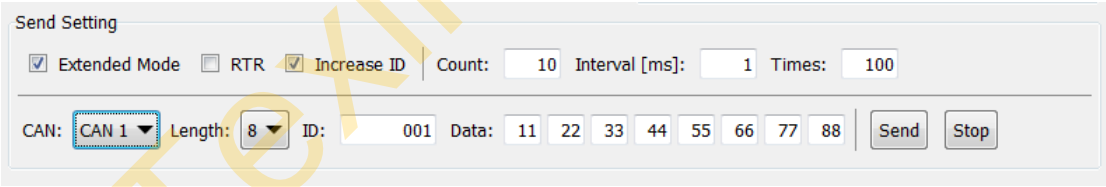

Non-block is enabled. Time Out is 1000ms (1 sec.), data count is 10. It means if receive 10 frames less then 1000ms, it will return 10 frames; if 1000ms time out but only receive 5 frames, it will return 5 frames.

### 3.2.4. 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:**



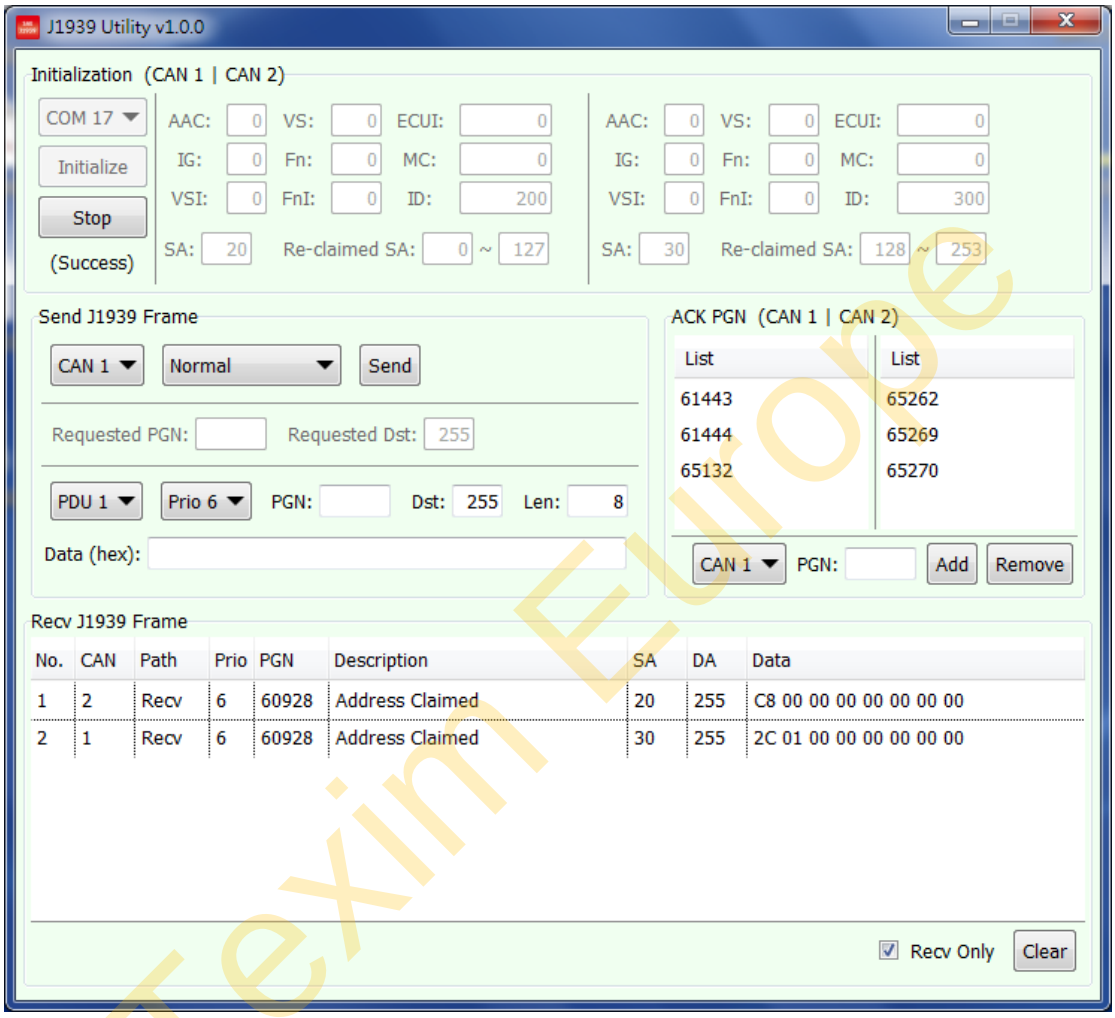

Set 29bit ID without RTR and increased ID when sending next frame.
Send 10 frames with interval 1ms for each frame and repeat 100 times. It will send is 1000 frames totally.

| 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 |

## 3.3. 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”.




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

### 3.3.1. Initialization

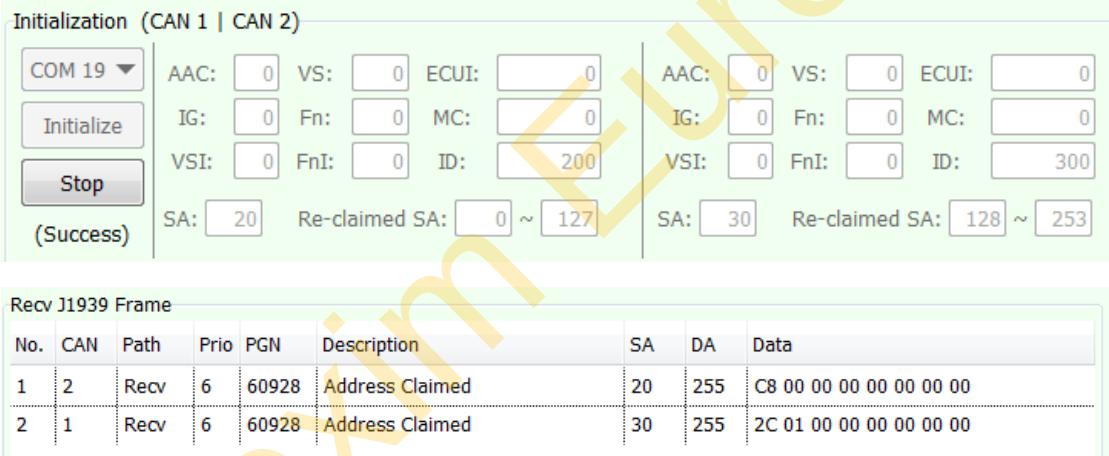
Set NAME and source address of CAN1 and CAN2 before initializing J1939 protocol. All ECUs must claim an address on the network. Initialized procedure set CANbus baud rate to 250 Kbps and sends PGN 60928 with the source address and NAME to claim the address which you want to use.

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:**




### 3.3.2. Normal J1939 Frame

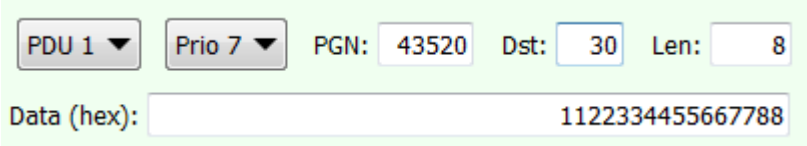
You can select CAN1 or CAN2 to send normal J1939 frame.

**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**

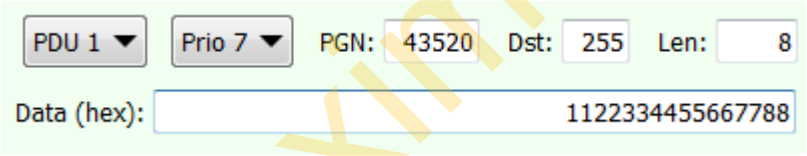
CAN1 (SA=20) sends normal J1939 frame of PDU1 to CAN2 (SA=30), priority is 7, PGN is 43520 (0xAA00), destination is 30, data length is 8, data is 0x1122334455667788.


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.

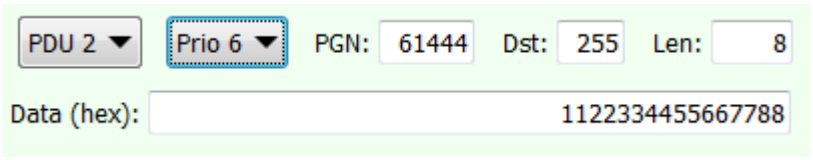



Recv J1939 Frame

| 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.

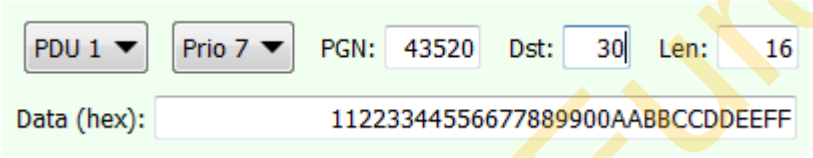



Recv J1939 Frame

| 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**

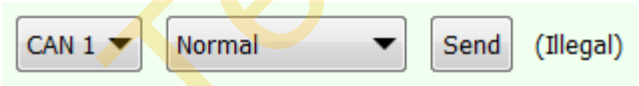
CAN1 (SA=20) sends normal J1939 frame of PDU1 data > 8 to CAN2 (SA=30), priority is 7, PGN is 43520 (0xAA00), destination is 30, data length is 8, data is 0x1122334455667788990AABBCCDDEEFF.


| 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.

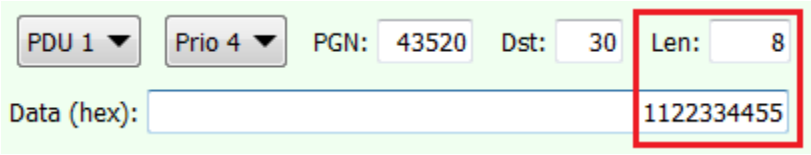



PDU format of PDU1 < 240, PGN must equal to or lower than 61184 (0xEF00, $PF=EF_{16}=239_{10}$), and the second bytes of PGN must be 0x00 such as 61184 (0xEF00), 60928 (0xEE00), 60672 (0xED00)…

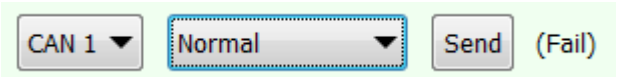
PGN 43210 is 0xA8CA, PF=0xA8=168. It is PDU1; the second bytes of PGN cannot have value, so it is illegal. Correct the value from 43210 to 43008 (0xA800).

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

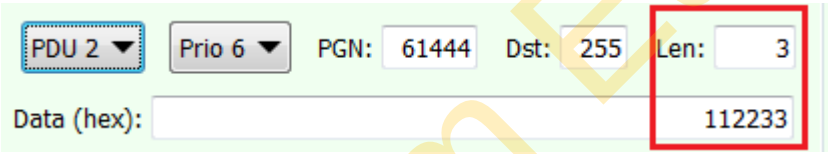

Data length is 8, but there are only 5 bytes data, so it is illegal. Fill the data to 8 bytes.




**Example 5: Fail**



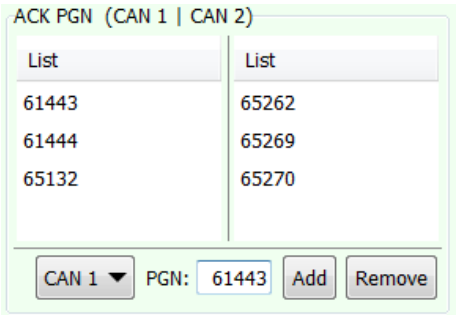

Only PGN 59904 can have 3 bytes data, others PGN must have 8 bytes of more than 8 bytes data. Correct the value of data length from 3 to 8 and fill the data to 8 bytes.




### 3.3.3. Request (PGN 59904)

You can select CAN1 or CAN2 to send request PGN.

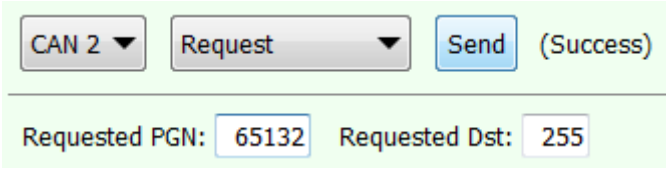
**Requested PGN (Dec):** The PGN which you want to request.
**Requested Dst (Dec):** The destination address you want to send this request, it can be specific of global address (255).
**ACK PGN (Dec):** The PGNs of CAN1 and CAN2 which will send “Positive ACK” if receive PGN 50094 and requested PGN is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



**Example 1: Send Request**

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


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




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

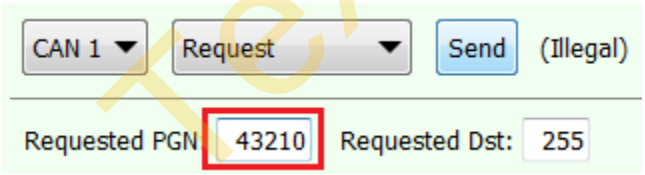
CAN1 receives the request then returns PGN 59392 with Positive ACK to CAN2.

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).

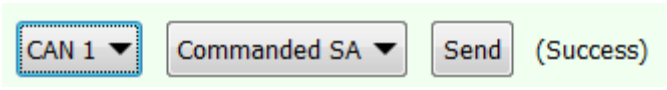


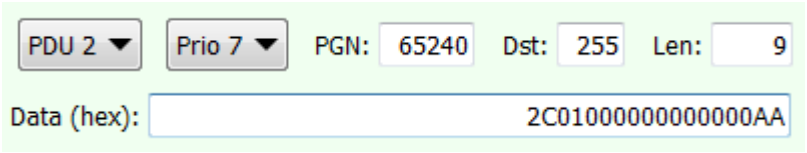

### 3.3.4. Commanded Source Address (PGN 65240)

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).




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

Recv J1939 Frame

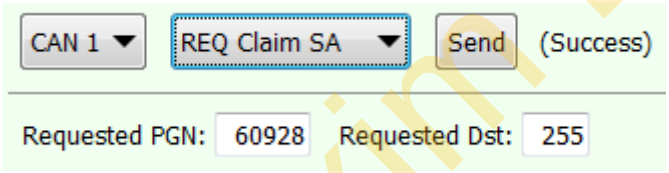
| No. | CAN | Path | Prio | PGN | Description | SA | DA | Data |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Send | 7 | 65240 | Commanded Address | 20 | 255 | 2C 01 00 00 00 00 00 00 AA |
| 2 | 2 | Recv | 7 | 65240 | Commanded Address | 20 | 255 | 2C 01 00 00 00 00 00 00 AA |
| 3 | 1 | Recv | 6 |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

### 3.3.5. 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.




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

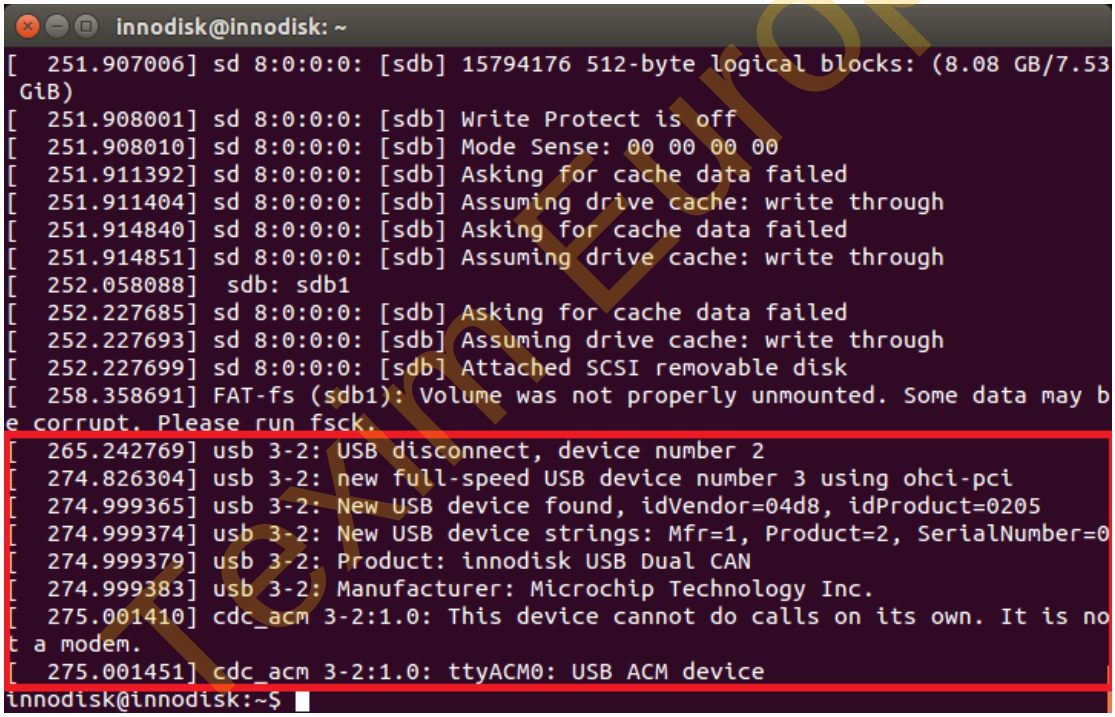
The following sections use Ubuntu 14.04.

## 4.1. Driver Installation

Install EMUC-B202 either into mPCIe slot or with USB pin header. 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.

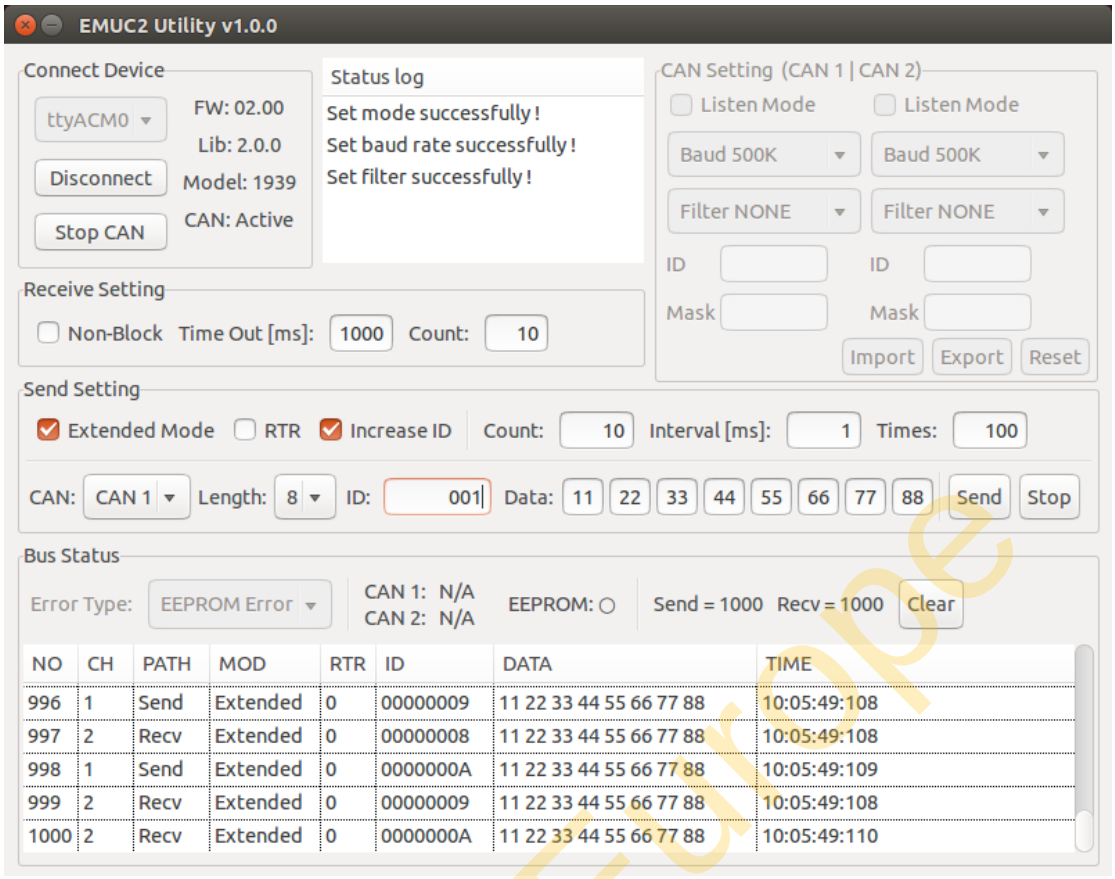



## 4.2. Basic CAN 2.0B Test Utility

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
```




## 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:

- src/emuccan.ko : Kernel driver of EMUC SocketCAN
- src/emucd :User-space tool for enabling EMUC SocketCAN

```
root@innodisk:/home/innodisk/2emuc/SocketCAN# make
make -C /lib/modules/3.13.11.8-custom/build M=/home/innodisk/2emuc/SocketCAN/sr
 EMUCDIR=/home/innodisk/2emuc/SocketCAN modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.11.8-custom'
  CC [M]  /home/innodisk/2emuc/SocketCAN/src/emuc.o
  CC [M]  /home/innodisk/2emuc/SocketCAN/src/lib_emuccan.o
  LD [M]  /home/innodisk/2emuc/SocketCAN/src/emuccan.o
  HOSTCC  /home/innodisk/2emuc/SocketCAN/src/emucd
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/innodisk/2emuc/SocketCAN/src/emuccan.mod.o
  LD [M]  /home/innodisk/2emuc/SocketCAN/src/emuccan.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.13.11.8-custom'
root@innodisk:/home/innodisk/2emuc/SocketCAN#
```

### 4.3.2. Usage and Example

After installing driver by “insmod” command, you can set CAN speed for two channels by executing “emucd” daemonYou can type “emucd -h” for help.

```
root@innodisk:/home/innodisk/2emuc/SocketCAN/src# ./emucd -h

Usage: ./emucd [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 KPS
                9: 1 MBPS
         -F          (stay in foreground; no daemonize)
         -h          (show this help page)
         -v          (show version info)

Examples:
emucd -s7 ttyACM0
emucd -s79 /dev/ttyACM0 can0 can1

root@innodisk:/home/innodisk/2emuc/SocketCAN/src#
```



./emucd –s7 /dev/ttyACM0 (500 KBPS on both channel)
./emucd –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”*

The following picture is an example to set EMUC to network interface.
You can see the CAN interface name by “ifconfig” command.

```
root@innodisk:/home/innodisk/2emuc/SocketCAN/src# insmod emuccan.ko
root@innodisk:/home/innodisk/2emuc/SocketCAN/src# ./emucd -s7 ttyACM0 can0 can1
root@innodisk:/home/innodisk/2emuc/SocketCAN/src# ip link set can0 up
root@innodisk:/home/innodisk/2emuc/SocketCAN/src# ip link set can1 up
root@innodisk:/home/innodisk/2emuc/SocketCAN/src# 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

eth2      Link encap:Ethernet  HWaddr 08:60:6e:71:39:f1
          inet addr:172.16.50.154  Bcast:172.16.50.255  Mask:255.255.255.0
          inet6 addr: fe80::a60:6eff:fe71:39f1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25843 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3155 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9383436 (9.3 MB)  TX bytes:550657 (550.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:626 errors:0 dropped:0 overruns:0 frame:0
          TX packets:626 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:52901 (52.9 KB)  TX bytes:52901 (52.9 KB)

root@innodisk:/home/innodisk/2emuc/SocketCAN/src#
```

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).

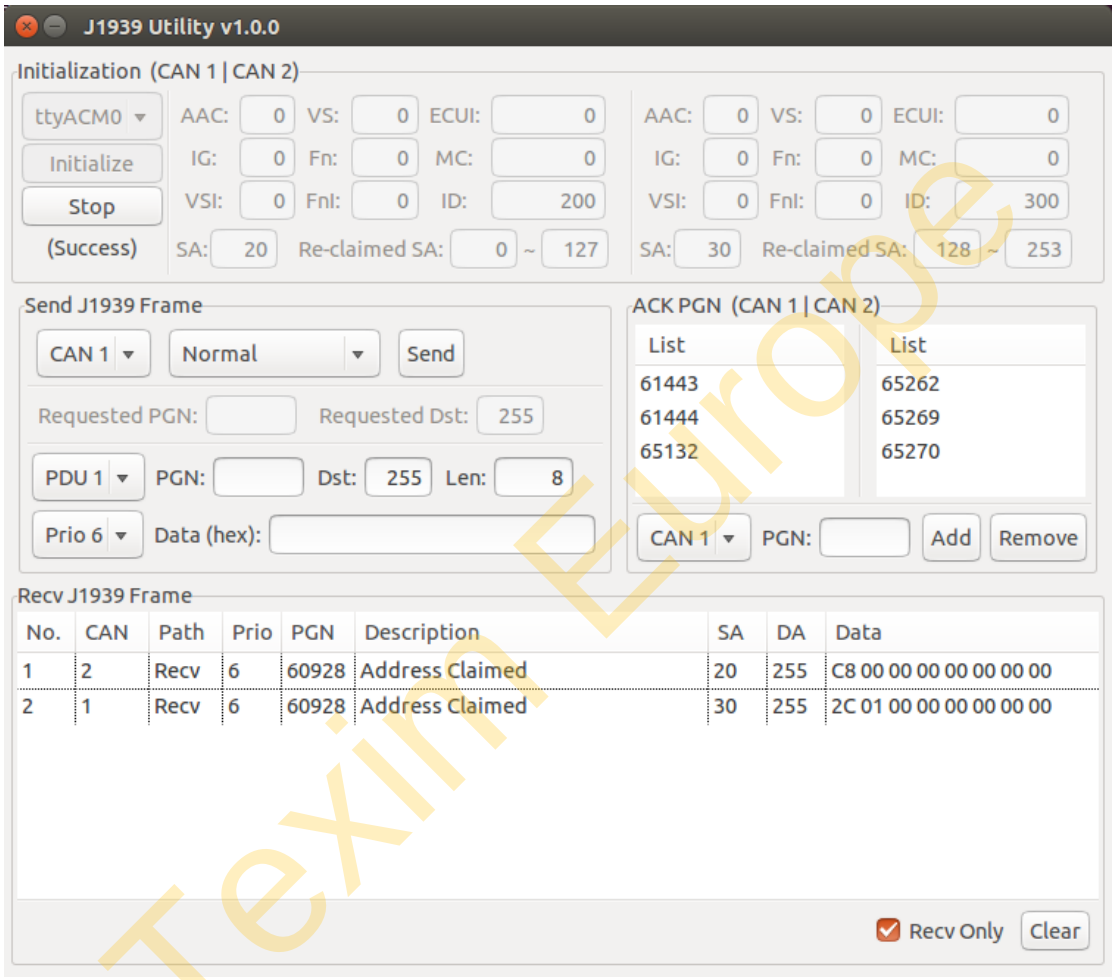
```
root@innodisk:/# cansend can0 7FF#1122334455667788
root@innodisk:/# cansend can0 1FFFFFFF#1122334455667788
root@innodisk:/# cansend can0 123#R
root@innodisk:/# candump can0
  can0  001   [8]  11 22 33 44 55 66 77 88
  can0  002   [8]  11 22 33 44 55 66 77 88
  can0  003   [8]  11 22 33 44 55 66 77 88
  can0  004   [8]  11 22 33 44 55 66 77 88
  can0  005   [8]  11 22 33 44 55 66 77 88
  can0  006   [8]  11 22 33 44 55 66 77 88
  can0  007   [8]  11 22 33 44 55 66 77 88
  can0  008   [8]  11 22 33 44 55 66 77 88
  can0  009   [8]  11 22 33 44 55 66 77 88
  can0  00001111   [8]  11 22 33 44 55 66 77 88
  can0       333   [0]  remote request
```

## 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
```



## 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.

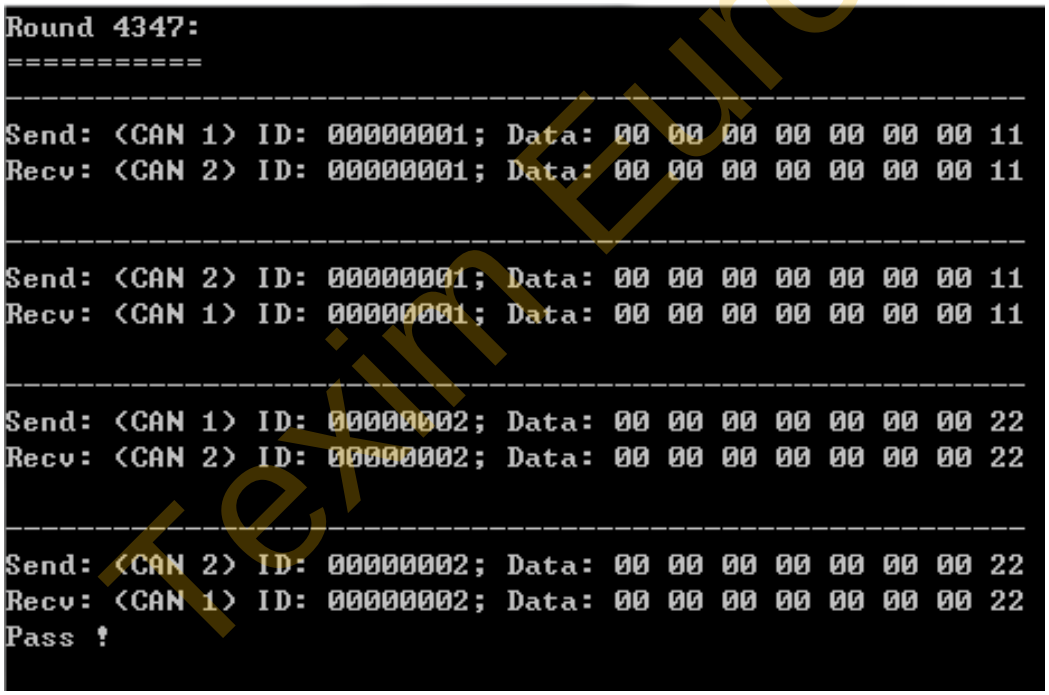
Before running the program, you can modify the "setup.ini" to set your test conditions.

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

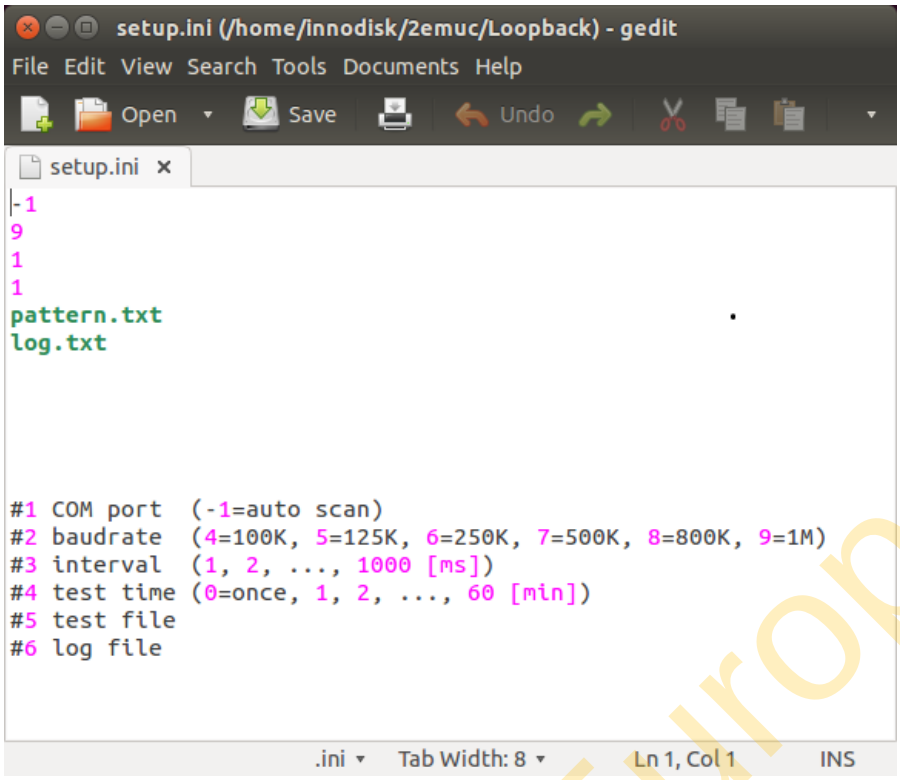
**Example:**

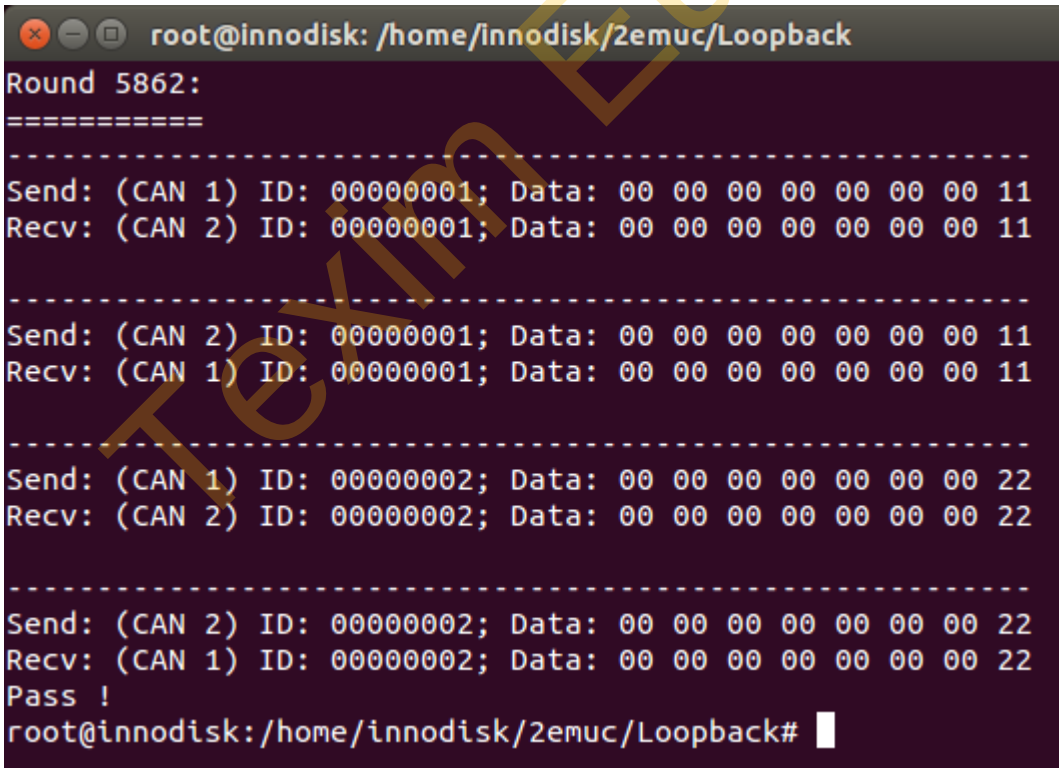
Use baud rate 1M to keep testing 1 min in Windows.

```
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
```



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



# 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 | ttyS0 | 1 | ttyS1 | 2 | ttyS2 |
| 3 | ttyS3 | 4 | ttyS4 | 5 | ttyS5 |
| 6 | ttyS6 | 7 | ttyS7 | 8 | ttyS8 |
| 9 | ttyS9 | 10 | ttyS10 | 11 | ttyS11 |
| 12 | ttyS12 | 13 | ttyS13 | 14 | ttyS14 |
| 15 | ttyS15 | 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 | No |
| EMUCCloseDevice | Yes | No |
| 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 |

### 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 |

### 6.2.15. 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 *4.2.Register mapping table of CAN error status.*

**Return Status Code:**

| Value | Return Value |
|---|---|
| 0 | No data |
| 1 | Get one data |

### 6.2.16. 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 |

### 6.2.17. 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*.

## 6.3. 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 9$^{th}$ 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 will do the following function.

1. Auto-scan device COM port and connect.
2. Initialize CAN status to be inactive.
3. Show version information.
4. Reset CAN configuration to default.
5. Clear all filter setting.
6. Set baud rate to 1000 Kbps.
7. Set error type to disable all error messages.
8. Set CAN mode to normal mode.
9. Set CAN1 filter with EID 0x0012ABCD, mask 1FFFFFFF.
10. Set CAN2 filter with EID 0x00001234, mask 00FFEEEE.
11. Get all CAN configurations.
12. Export CAN configurations to a file named as “emuc_config”
13. Import “emuc_config” CAN configurations.
14. Clear CAN filter setting and initialize CAN status to be active
15. Send 2000 CAN frame.
16. Use EMUCReceiveNonblock to receive CAN frames for 10 sencond.
17. Create a thread with EMUCReceive to receive CAN frames.

### 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 !
```

# 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<br>• Negative Acknowledgment: Control byte = 1<br>• Access Denied (PGN supported but security denied access) Control byte = 2<br>• 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**



September 19, 2017