

USB-CAN-M/USB-CAN-SI-M USER'S MANUAL

2020 Edition



Titan Electronics Inc.
Web: www.titan.tw

The computer programs provided with the hardware are supplied under a license. The software provided should be used only with the NCOM series hardware designed and manufactured by TITAN Electronics Inc.

Trademarks

TITAN and the logo is a registered trademark of TITAN Electronics Inc. in Taiwan. Microsoft, Windows, Windows XP, Windows Vista, Windows Server, Windows 7, Windows 8, Windows 10 are trademarks of Microsoft Corporation. All other trademarks and brands are property of their respective owners.

Copyright

Copyright © TITAN Electronics Inc. 2016. All right reserved. Reproduction of the manual and software without permission is prohibited.

Disclaimer

TITAN Electronics Inc. provides this document and computer programs “as is” without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. TITAN Electronics Inc. reserves the right to make improvements and changes to this user manual, or to the products, or the computer programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, TITAN Electronics Inc. assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Contents

1 INTRODUCTION	5
2 FEATURES	6
3 DIAGRAM OF USB-CAN-M/USB-CAN-SI-M.....	7
3.1 PCB LAYOUT	7
3.2 BLOCK DIAGRAM.....	8
3.2.1 USB-CAN-M Block Diagram.....	8
3.2.2 USB-CAN-SI-M Block Diagram.....	8
4 SPECIFICATIONS	9
5 PIN-OUT INFORMATION	12
5.1 CAN Bus Pin-out for DB9 connector	12
5.2 Enabling the +5V 100mA power for external devices	12
5.3 Termination Resistors	13
6 HARDWARE INSTALLATION.....	14
7 DRIVER AND SOFTWARE INSTALLATION.....	15
7.1 Driver Installation.....	15
7.2 Verifying the Installation.....	15
7.3 Changing COM Port Properties & COM Port Number	16
8 Aliasing COM Port's Baud Rate to 3Mbits	17
8.1 How to Alias COM Port's Baud Rate to 3Mbits.....	17
9 Linux Driver Installation(SocketCAN)	19
9.1.1 Using SocketCAN(SLCAN) with USB-CAN	19
10 FUNCTION DESCRIPTION.....	20
10.1 LED Indicators	20
10.2 ASCII Command Set.....	21
10.3 Command list	22
10.3.1 Opening the CAN Bus Channel.....	23
10.3.2 Closing the CAN Bus Channel.....	23
10.3.3 Setting CAN Baudrate (Standard)	24
10.3.4 Setting CAN Baudrate (Advanced).....	25
10.3.5 Transmitting a Standard CAN Frame	26
10.3.6 Transmitting a Standard Remote Request CAN Frame	26
10.3.7 Transmitting an Extended CAN Frame.....	27
10.3.8 Transmitting an Extended Remote Request CAN Frame.....	27
10.3.9 Setting Timestamps ON/OFF	28

10.3.10 Setting Acceptance Mask.....	29
10.3.11 Setting Acceptance Code	30
10.3.12 Getting Status Flags	31
10.3.13 Getting Version Information.....	33
10.3.14 Getting Serial Number	33
10.3.15 Resetting the USB CAN adapter.....	33
11 TOOLS.....	34
11.1 Setting 3MBit Baud Rate for CAN Bus.....	34
11.2 Firmware Upgrade	35
11.3 CANHacker	36
11.3.1 Settings procedure for selecting and configuring the USB to CAN adapter	37
11.3.2 Receiving CAN frames	39
11.3.3 Sending CAN frames	39
11.3.4 Assistant features	41
11.4 Titan CAN Test Program.....	43
11.4.1 Settings procedure for selecting and configuring the USB to CAN adapter	44
11.4.2 Receiving CAN frames	46
11.4.3 Sending CAN frames	46
11.4.4 Assistant features	48
11.5 CANopen	52
11.5.1 How to get CanFestival	52
11.5.2 Linux Compilation and installation	52
11.5.4 Windows Compilation.....	53
11.5.5 Compilation with Visual Studio.....	53
11.6 python-can	56
11.7 APPLICATION PROGRAMMING INTERFACE	57
11.7.1 CAN_Open.....	58
11.7.2 CAN_Close.....	60
11.7.3 CAN_Write	61
11.7.4 CAN_Read	62
11.7.5 CAN_Flush.....	63
11.7.6 CAN_Status	64
11.7.7 CAN_Version	66
11.7.8 CAN_MSG Structure.....	67

11.7.9 Example Code for C.....	68
11.7.10 Using the API in C#.....	70
11.7.11 Using the API in Visual Basic .NET.....	71
11.7.12 Using the API in Visual Basic 6.0	73
11.7.13 Using the API in Python	74
11.7.14 Using the API in LabVIEW	75

1 INTRODUCTION

A Controller Area Network (CAN) is a high-integrity asynchronous serial bus system for networking intelligent devices. It is often used in automotive and industrial systems. The USB-CAN-M/USB-CAN-SI-M are designed to make a fast, simple way to communicate with CAN bus devices. Connected to a USB port on your computer or USB hub, the USB-CAN-M/USB-CAN-SI-M instantly adds an industrial CAN bus channel to your host system with easy plug and play (PnP) and hot plug features.

The USB-CAN-M/USB-CAN-SI-M provides a cost-effective solution for customers to enable communication with CAN bus devices. The solution designed by ARM Cortex-M0 32-bit microcontroller and the USB to serial chip makes it very flexible in handling small burst of CAN frames at a high speed.

Upon plugging the USB-CAN-M/USB-CAN-SI-M into the USB port, the adapter is automatically detected and installed. The USB CAN bus adapter provides instant connectivity to CAN bus devices. The USB-CAN-M/USB-CAN-SI-M provides an industrial solution for applications of CAN bus multi-drop communications over short and long distances.

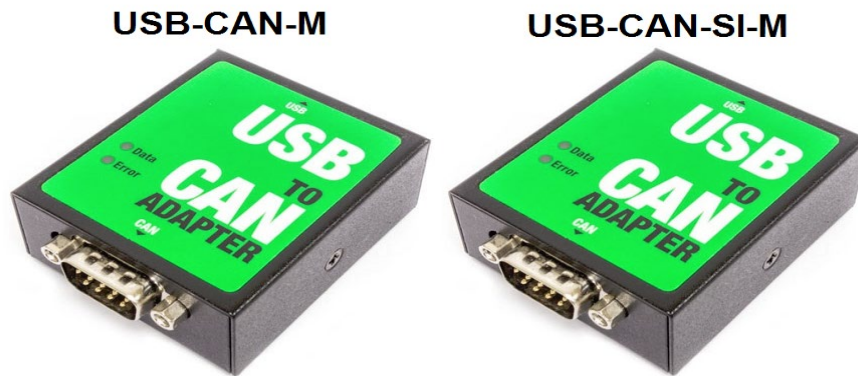
The USB to CAN adapter does not require a power supply. It is powered by a USB port. The USB-CAN-M requires 150mA, while the USB-CAN-SI-M requires 250mA. They must be connected directly to a USB host port or a self-powered USB hub. However, bus-powered USB hubs cannot provide the required current, as they only provide 100mA current.

The USB-CAN-SI-M has 2500V galvanic isolation on its CAN bus. The galvanic isolation protects your computer by preventing spikes and surges from crossing over to your computer.

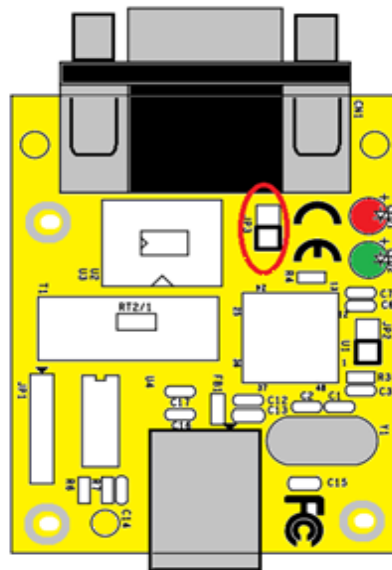
2 FEATURES

- Adds a CAN bus port on your computer by connecting to USB 1.1, 2.0 or 3.0 host and hub ports
- One DB9 male connector
- Includes one USB A/B cable. Cable length: 90cm
- Powered by USB port, no external power adapter required
- LEDs indicate initialization and CAN bus status
- Installs as standard Windows COM port; COM port number can be changed to any COM port number
- CAN bus speed up to 1Mbps
- 512 bytes receive FIFO and 512 bytes transmit FIFO buffer for high speed data throughput
- Supports CAN 2.0A and CAN 2.0B protocols
- Supported CAN modes
 - Standard mode: normal operation on CAN bus
 - Listen mode: passive receiving of CAN frames
 - Echo mode: transmitter also receives sent frames (for testing purposes)
- Easy plug and play installation and CAN bus device connection
- USB CAN adapter can be controlled over serial port using simple ASCII commands
- USB-CAN-SI-M has a 2500V galvanic isolation on its CAN bus
- Wide ambient temperature operation 0°C to 60°C (32°F to 140°F)
- CE, FCC approval
- Designed by ARM Cortex-M0 32-bit microcontroller and the USB to UART chip
- Drivers provided for Windows and Linux OS

3 DIAGRAM OF USB-CAN-M/USB-CAN-SI-M

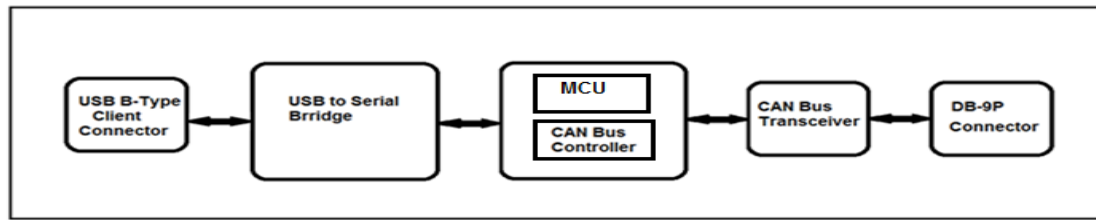


3.1 PCB LAYOUT

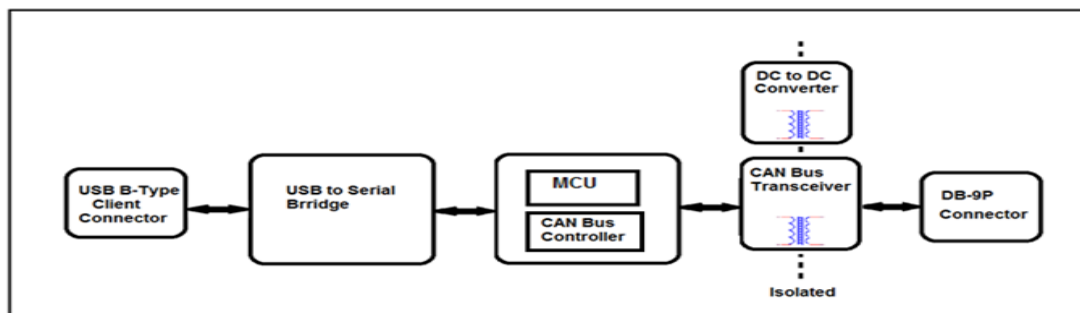


3.2 BLOCK DIAGRAM

3.2.1 USB-CAN-M Block Diagram



3.2.2 USB-CAN-SI-M Block Diagram



4 SPECIFICATIONS

The tables below show the specifications of USB to 1-port CAN bus adapter:

General	
USB Bus	Supports USB 1.1, USB 2.0, USB 3.0
CAN Bus	Supports CAN 2.0A and CAN 2.0B
Chipset	ARM Cortex-M0 32-bit microcontroller
Plug & Play	Supported
IRQ & IO Address	Assigned by system

USB-CAN-M	
Number of Ports	One
Connector	DB9 male connector
CAN Bus Speed	5kbits to 1Mbits for CAN data transmit & receive
Signals	CAN_H, CAN_L, CAN_GND, CAN_V+
CAN Bus Controller	Bosch C_CAN module
LED	CAN bus data activity, CAN bus error
CAN Bus Mode	Standard mode: normal operation on CAN bus Listen mode: passive receiving of CAN frames Echo mode: transmitter also receives sent frames (for testing purposes)
Protection	+/-16 KV ESD protection for CAN signals

USB-CAN-SI-M	
Number of Ports	One
Connector	DB9 male connector
CAN Bus Speed	20kbits to 1Mbits for CAN data transmit & receive
Signals	CAN_H, CAN_L, CAN_GND, CAN_V+
CAN Bus Controller	Bosch C_CAN module
LED	CAN bus data activity, CAN bus error
CAN Bus Mode	Standard mode: normal operation on CAN bus Listen mode: passive receiving of CAN frames Echo mode: transmitter also receives sent frames (for testing purposes)
Protection	+/-16 KV ESD protection for CAN signals 2500V galvanic isolation on CAN bus

Software Features	
API Library	Supports C/C++, C#, VB.NET and LabVIEW
Utility	On-board firmware update utility
OS Driver Support	Windows XP to Windows 10 OS Windows Server 2003 to 2012 R2
Monitoring Tools	Supported by CANHacker, Titan CAN test program

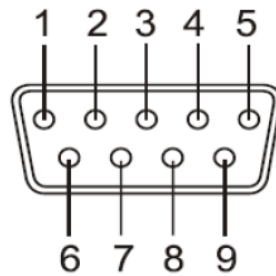
Power Requirement	
Power Input	Power supplied via USB (5V) connector No external power needed
Power Consumption	Max. 150mA@5VDC (USB-CAN-M) Max. 250mA@5VDC (USB-CAN-SI-M)

Mechanical	
Casing	SECC sheet metal (1mm)
Dimensions	61 mm x 50 mm x 23 mm (L x W x H)
Weight	100g

Environment	
Operating Temperature	0°C to 60°C (32°F to 140°F)
Storage Temperature	-20°C to 75°C (-4°F to 167°F)
Humidity	5% to 95% RH
Safety Approvals	CE, FCC

5 PIN-OUT INFORMATION

Following are the pin-out of connector for USB to CAN bus adapter:



DB9 Male connector pin numbers

5.1 CAN Bus Pin-out for DB9 connector

Pin Number	Signals	Description
1	CAN_V+	Provides +DC 5V 100mA power (optional)
2	CAN_L	CAN_L bus line (dominant level is low)
3	CAN_GND	Signal ground
4	-	Reserved
5	-	Reserved
6	CAN_GND	Signal ground
7	CAN_H	CAN_H bus line (dominant level is high)
8	-	Reserved
9	CAN_V+	Provides +DC 5V 100mA power (optional)

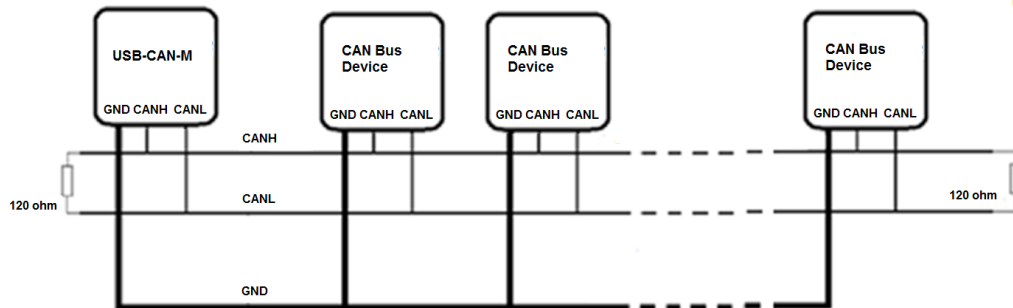
5.2 Enabling the +5V 100mA power for external devices

Inside the unit, there is a 2-pin header block (JP3) which are jumpers for enabling 5V 100mA power for external devices.

JP3 Jumper	Function
ON	Enable DB9 pins 1 and 9 to provide a 5V 100mA power for external devices
OFF	Disable the 5V 100mA power

5.3 Termination Resistors

The USB CAN adapter does not provide CAN bus termination resistors. A CAN bus network requires 120Ω termination resistors at each end. Generally, this must be done in the cabling. Since this depends on the installation of connections, please check your CAN bus cable specification for proper impedance matching.

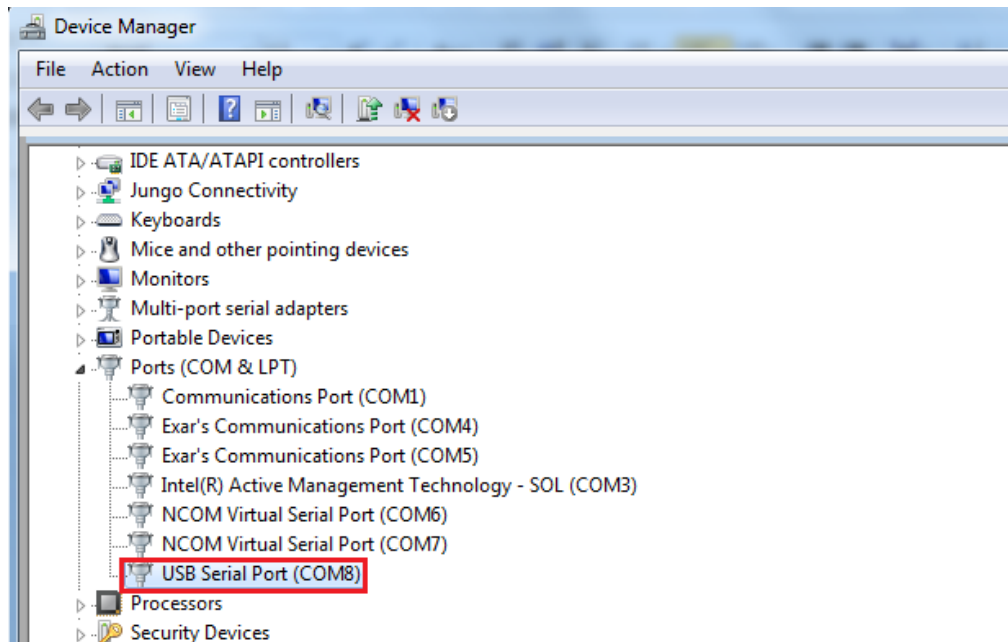


6 HARDWARE INSTALLATION

The USB CAN adapter is a plug and play device. In most cases, the USB CAN adapter's drivers will be installed automatically.

Connect the USB CAN adapter to an unused USB port on your computer.

After the software drivers are loaded, you will find a new "USB Serial Port (COMX)" under "Ports (COM & LPT)" in device manager.



You will need to execute the command "**CAN_BAUDRATE_SET COMX**" to set the USB to CAN adapter to work in high speed (3Mbits). After executing the command, please disconnect the USB to CAN adapter from the system for about 5 seconds and connect it again (refer to page 34).

Note: The set baud rate program (CAN_BAUDRATE_SET.exe) must be executed under command prompt with administrative privileges.

7 DRIVER AND SOFTWARE INSTALLATION

7.1 Driver Installation

In most cases, the driver of USB CAN adapter will be installed driver automatically.

Windows 10, 8.1, 8, 7, Server 2012 R2, Server 2008 R2

Connect your computer to Internet and plug USB CAN adapter to the USB port. The driver will be installed automatically via Internet.

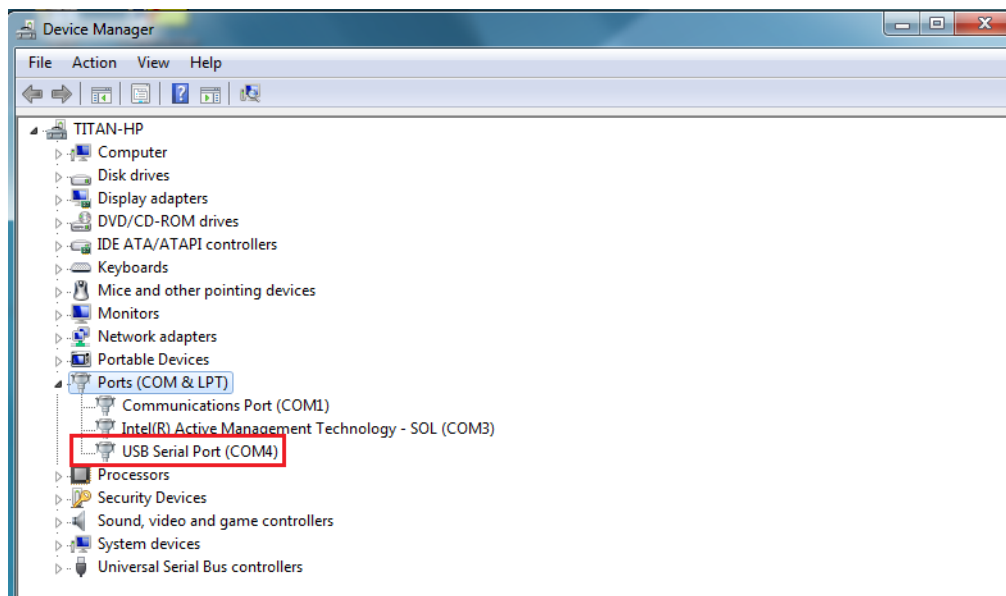
Windows XP, Vista, Server 2003 and 2008

Connect your computer to Internet and plug USB CAN adapter to the USB port, when asked to install the drivers, allow your computer to search the Internet to load and install the drivers automatically.

7.2 Verifying the Installation

You can verify the installation by looking under device manager (Start → Settings → Control Panel → System Properties → Hardware → Device Manager).

The device should have installed as a "USB Serial Port (COMX)" under "Ports (COM & LPT)".



7.3 Changing COM Port Properties & COM Port Number

This feature is particularly useful for programs such as HyperTerminal, which only work with COM1 through COM4. Please ensure that you do not change to a COM port number that is already in use.

To change the virtual COM port properties:

1. Select the "USB Serial Port (COMX)".
2. Click "Properties".
3. Select "Port Setting" and "Advanced".
4. Click the drop down arrow on COM Port Number and scroll to the required COM port.
5. Select "OK".
6. Return to the Device Manager Screen. You will see that the USB Serial Port installation has been changed to the new COM Port Number.

8 Aliasing COM Port's Baud Rate to 3Mbits

After driver installation, the next step you need to do is to aliasing the COM port's baud rate of the USB-CAN Adapter to 3Mbits. It is required to substitute all the standard baud rates of the COM port with 3Mbits. It is very important to set the baud rate to 3Mbits, so that the COM port of the USB-CAN Adapter will transfer CAN data at high speed for better performance.

Note: You must alias the baud rate of the COM port to 3Mbits, by executing the provided baud rate setting program "CAN_BAUDRATE_SET.exe". If the step of Aliasing Baud Rate is missing, the USB-CAN Adapter will not work properly.

8.1 How to Alias COM Port's Baud Rate to 3Mbits

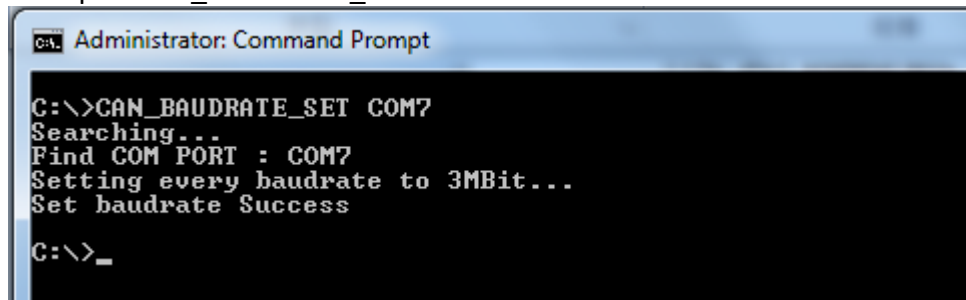
You need to have administrative privileges to run the aliasing baud rate program. Please log onto Windows as "Administrator" to execute the setting baud rate program (CAN_BAUDRATE_SET.exe).

The aliasing baud rate program (CAN_BAUDRATE_SET.exe) must be executed under command prompt with administrative privileges.

Following is aliasing baud rate program command:

`CAN_BAUDRATE_SET <COMx>`

Example: `CAN_BAUDRATE_SET COM7`



```
Administrator: Command Prompt
C:\>CAN_BAUDRATE_SET COM7
Searching...
Find COM PORT : COM7
Setting every baudrate to 3MBit...
Set baudrate Success
C:\>_
```

COM7: The USB to CAN Adapter is installed as "USB Serial Port (COMx)" in the Device Manager. You can check the assigned COM port number of the CAN Adapter in the Device Manager. You need to alias the baud rates of this specific COM port.

Please proceed with the following steps to alias the COM Port's baud rate to 3Mbits:

1. Log onto Windows as "Administrator".
2. Under command prompt, type "`CAN_BAUDRATE_SET COMx`" where "x" is the COM port number of the CAN Adapter.
3. Disconnect the USB-CAN Adapter from the system for about 5 seconds.
4. Connect the CAN Adapter to the USB port of the system again.
5. The following message under Command prompt will be present:
Searching...
Find COM PORT: COM7

Setting every baud rate to 3Mbit...

Set baud rate Success

6. The baud rate aliasing is set successfully.

The installation of the USB-CAN Adapter is now completed.

In place of the standard baud rates, the COM port now will be always sending and receiving data at 3Mbits. Whatever the COM port's baud rate you set in the Device Manager or from an application, the actual baud rate is 3Mbits.

9 Linux Driver Installation(SocketCAN)

SocketCAN is a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel. Formerly known as Low Level CAN Framework (LLCF).

Installing SocketCAN is rather easy. In fact, it just needs to install can-utils packages. To proceed , uses the following command:

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

By default, the SocketCAN device drivers are not automatically loaded by Linux at boot time on all systems: you may need to enable the relevant linux kernel modules. You can manually load these modules with the following command:

```
sudo modprobe can  
sudo modprobe vcan  
sudo modprobe slcan
```

9.1.1 Using SocketCAN(SLCAN) with USB-CAN

To use USB-CAN with SocketCAN, you have to establish a “link” between the drivers and the hardware. This is done with the following command:

```
sudo slcand -o -c -s8 -S3000000 /dev/ttyUSB0 slcan0
```

The -s parameter allow to configure different interface speed as indicated below.

ASCII Command	CAN Bitrate
s0	10 Kbit/s
s1	20 Kbit/s
s2	50 Kbit/s
s3	100 Kbit/s
s4	125 Kbit/s
s5	250 Kbit/s
s6	500 Kbit/s
s7	800 Kbit/s
s8	1000 Kbit/s

If everything worked fine, you should now see the can0 network device. To very it, just type the following command:

```
sudo ip link set slcan0 up
```

10 FUNCTION DESCRIPTION

10.1 LED Indicators

The USB to CAN adapter has two LEDs (green LED & red LED) to indicate firmware initialization and CAN bus status for monitoring CAN bus channel status. The green LED indicates CAN bus data activity while the red LED indicates a CAN bus error. Following are the definition of different LED combinations:

A: Power up (device initialized)

After USB CAN powers up (device initialized), the USB to CAN adapter flashes the green and red LED four times to indicate that the USB CAN adapter has been initialized.

B: CAN bus channel open/close

When CAN bus channel opens, the green LED will turn on to indicate that the CAN bus channel is open; When CAN bus channel closes, the green LED will turn off to indicate that the CAN bus channel is closed.

C: CAN Bus Data Activity

When CAN data frame is sent or received, the green LED flashes continuously to indicate CAN bus data I/O activity.

D: CAN Bus Error

When an error occurs on the CAN bus, the red LED flashes continuously to indicated CAN bus error.

10.2 ASCII Command Set

The USB CAN adapter can be registered as a virtual serial port on the host computer. With simple ASCII commands the USB CAN adapter can be controlled over this serial port. User can send/receive commands from any simple serial terminal program.

Example: Set bitrate to 500Kbps, open CAN channel, send CAN frame (ID = 002h, DLC = 3, Data = 11 22 33), close CAN:

Command	Response	Function
S6[CR]	[CR]	Set bitrate of USB CAN adapter to 500Kbps
O[CR]	[CR]	Open CAN channel
t0023112233[CR]	z[CR]	Send CAN message (ID = 002h, DLC = 3, Data = 11 22 33)
C[CR]	[CR]	Close CAN channel

10.3 Command list

The commands are line based and terminated with newline character CR (0xD). On error the response will be 0x7 (BELL).

The “help” command (**‘H’, ‘h’ or ‘?’**) will list supported commands.

Command	Response	Function
H[CR]	[CR]	List all supported commands
h[CR]	[CR]	
?[CR]	[CR]	

Example: H[CR]

Return Code

List of Supported Commands:

- ‘O’ – Open the channel in Normal mode
- ‘L’ – Open the channel in Listen Only mode
- ‘Y’ – Open the channel in Loopback mode
- ‘C’ – Close CAN Channel
- ‘S’ – Set standard CAN bitrate
- ‘s’ – Set non-standard CAN bitrate
- ‘t’ – Transmit a standard frame
- ‘T’ – Transmit an extended frame
- ‘r’ – Transmit a standard remote request frame
- ‘R’ – Transmit an extended remote request frame
- ‘Z’ – Set timestamp on/off
- ‘m’ – Set acceptance mask
- ‘M’ – Set acceptance filter
- ‘F’ – Read status flag
- ‘V’ – Check software version
- ‘N’ – Check serial number
- ‘m’ – Set acceptance mask
- ‘M’ – Set acceptance filter
- ‘RST’ – Reset USB CAN Adapter
- ‘H’, ‘h’ or ‘?’ – List supported commands

10.3.1 Opening the CAN Bus Channel

The CAN bus channel will be opened with the command O[CR], L[CR] or Y[CR]. The command O[CR] will open the CAN bus channel in normal operation mode, the command L[CR] will open the CAN bus channel in listen only mode, in which no bus interaction will be done from the controller. the command Y[CR] will open the CAN bus channel in a loop-back mode, in which the USB to CAN adapter will also receive the frames that it sends. Before you use one of the commands, you should set a bitrate with the commands S or s.

Command	Response	Function
O[CR]	[CR]	Open the channel in Normal mode
L[CR]	[CR]	Open the channel in Listen Only mode
Y[CR]	[CR]	Open the channel in Loopback mode

10.3.2 Closing the CAN Bus Channel

The CAN bus channel will be closed with the command C[CR]. The command can only be used if the CAN bus channel is open.

Command	Response	Function
C[CR]	[CR]	Close the CAN channel if it is opened

10.3.3 Setting CAN Bitrate (Standard)

The CAN bus bitrate can be set with the command SX[CR]. The command can only be used if the CAN bus channel is closed.

Command	Response	Function
S00[CR]	[CR]	Set the CAN bus bitrate to 5K
S0[CR]	[CR]	Set the CAN bus bitrate to 10K
S1[CR]	[CR]	Set the CAN bus bitrate to 20K
S2[CR]	[CR]	Set the CAN bus bitrate to 50K
S3[CR]	[CR]	Set the CAN bus bitrate to 100K
S4[CR]	[CR]	Set the CAN bus bitrate to 125K
S5[CR]	[CR]	Set the CAN bus bitrate to 250K
S6[CR]	[CR]	Set the CAN bus bitrate to 500K
S7[CR]	[CR]	Set the CAN bus bitrate to 800K
S8[CR]	[CR]	Set the CAN bus bitrate to 1M

Example: S6[CR] will be set USB CAN adapter to 500K bps CAN Bitrates.

Note: The USB-CAN-SI-M only supports 20 K bits to 1 M bits.

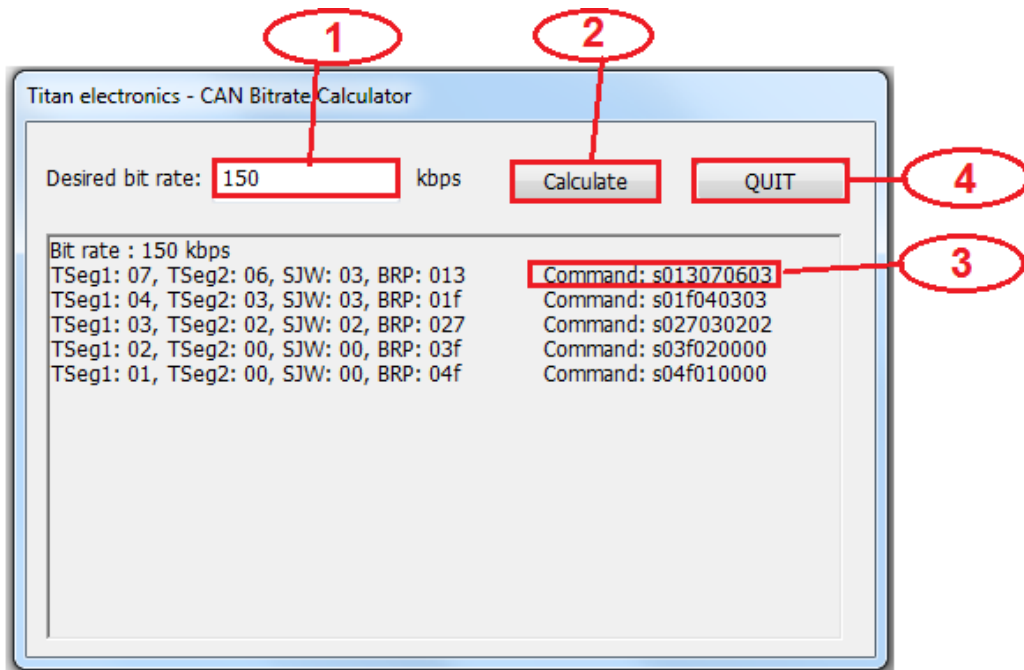
10.3.4 Setting CAN Bitrate (Advanced)

A more user defined bus bitrate can be configured with the command sXXXXXXXX[CR]. As with the standard bus timing command above, you can only use this command when the CAN bus channel is closed.

sXXXXXXXX [CR] sets the bitrate registers of the CAN controller. Users can set **non-standard bitrates** which are not supported by the "SX" command.

The USB to CAN adapter provides a CAN Bitrate Calculator program to calculate the value of CAN bitrate registers for setting **non-standard bitrates**. Follow these steps to calculate and set **non-standard bitrates** for the USB to CAN adapter:

1. Open the CAN Bitrate Calculator program.
2. Enter CAN Bitrate ("150" for 150Kbps CAN Bitrate) in the field "Desired bitrate:".
3. Click "Calculate" to calculate the value of CAN bitrate registers.
4. Remember the topmost value of CAN bitrate registers.
5. **e.g. Command: s013070603** for 150 kbps CAN Bitrate.
6. Click "Quit" to exit the CAN Bitrate Calculator program.



Example: s013070603[CR] will be set the bitrate to 150Kbps.

10.3.5 Transmitting a Standard CAN Frame

Transmitting a standard CAN frame (ID: 11 bit) over a CAN bus can be done with the command `tiiildddd...dd[CR]`. The return value will be `z[CR]` or the normal error byte (BELL). The command is only available when the CAN bus channel is open.

Command	Response	Function
tiiildddd...dd[CR]	<code>z[CR]</code>	Transmits a standard CAN message (11 bit) over the CAN bus

iii: Standard CAN frame (11 bit) identifier in hexadecimal format (000-7FF).

l: CAN data length (0-8) DLC, with the maximum value being 8 (8 bytes).

dd: Data byte value in hexadecimal format (00-FF). The number of bytes must be equal to the data length field.

Example: `t00231199FF[CR]` will send a standard CAN frame with ID = 002h, DLC = 3, Data = 11 99 FF.

10.3.6 Transmitting a Standard Remote Request CAN Frame

Transmitting a standard remote request CAN frame (ID: 11 bit) over a CAN bus can be done with the command `riiil[CR]`. The return value will be `z[CR]` or the normal error byte (BELL). The command is only available when the CAN bus channel is open.

Command	Response	Function
riiil[CR]	<code>z[CR]</code>	Transmits a standard remote request (11 bit) over the CAN bus

iii: Standard remote request CAN frame (11 bit) identifier in hexadecimal format (000-7FF).

l: CAN data length to request (0-8) DLC, with the maximum value being 8 (8 bytes).

Example: `r0023[CR]` will send a standard remote request CAN frame with ID = 002h, DLC = 3 and request 3 data bytes.

10.3.7 Transmitting an Extended CAN Frame

Transmitting an extended CAN frame (ID: 29 bit) over a CAN bus can be done with the command **TiiiiiiiIdddd...dd[CR]**. The return value will be **Z[CR]** or the normal error byte (BELL). The command is only available when the CAN bus channel is open.

Command	Response	Function
TiiiiiiiIdddd...dd[CR]	Z[CR]	Transmits an extended CAN frame (11 bit) over the CAN bus

iiiiiiiI: Extended CAN frame (29 bit) identifier in hexadecimal format (00000000-1FFFFFFF).

I: CAN data length (0-8) DLC, with the maximum value being 8 (8 bytes).

dd: Data byte value in hexadecimal format (00-FF). The number of bytes must be equal to the data length field.

Example: **T1FFFFFFF3112233[CR]** will send an extended CAN frame with ID = 1FFFFFFFh, DLC = 3, data = 11 22 33.

10.3.8 Transmitting an Extended Remote Request CAN Frame

Transmitting an extended remote request CAN frame (ID: 29 bit) over a CAN bus can be done with the command **RiiiiiiiI[CR]**. The return value will be **Z[CR]** or the normal error byte (BELL). The command is only available when the CAN bus channel is open.

Command	Response	Function
RiiiiiiiI[CR]	Z[CR]	Transmits an extended remote request (29 bit) over the CAN bus

iiiiiiiI: Extended remote request CAN frame (29 bit) identifier in hexadecimal format (00000000-1FFFFFFF).

I: CAN data length to request (0-8) DLC, with the DLC maximum value being 8 (8 bytes).

Example: **R100000023[CR]** will send an extended remote request CAN frame with ID = 10000002h, DLC = 3 and request 3 data bytes.

10.3.9 Setting Timestamps ON/OFF

The timestamp command will set the timestamp functionality of received frames ON or OFF. This command is only available when the CAN channel is closed.

Command	Response	Function
Z1[CR]	[CR]	Set the timestamp functionality on received frames ON
Z0[CR]	[CR]	Set the timestamp functionality on received frames OFF

10.3.10 Setting Acceptance Mask

The acceptance mask, in conjunction with the acceptance code (M), defines which CAN message frames (i.e. of a specific ID or range of CAN IDs) will be passed to the serial interface. The acceptance mask value corresponds to bits within a range of valid CAN IDs for either standard or extended CAN frames. This command is only active if the CAN channel is initiated and not opened.

Set Acceptance Mask (m) command should be executed *prior* to Set Acceptance Code (M).

Note: The CAN channel will revert to its prior state after execution. For example, if the channel is open when this command is executed, the channel will update the setting and return to the open state.

Command	Response	Function
miii[CR]	[CR]	Set acceptance mask for standard CAN frame (11 bit) identifier
miiiiiii[CR]	[CR]	Set acceptance mask for extended CAN frame (29 bit) identifier

iii = standard 11-bit CAN mask (0x000 through 0x7FF)

iiiiiii = extended 29-bit CAN mask (0x00000000 through 0x1FFFFFFF)

A value of “0” in a bit location indicates that the bit location ID value is to be *ignored* when filtering messages.

Default is to pass all frames (acceptance mask = 0x000 for standard messages and 0x00000000 for extended messages)

Example: m700[CR] set acceptance mask to check bits 10, 9 and 8 against the filter. Bits 7 through 0 are ignored as “don’t care”. Use the acceptance mask in conjunction with the acceptance code, which is explained next.

10.3.11 Setting Acceptance Code

The acceptance code/filter, in conjunction with the acceptance mask (m), defines which CAN message frames (i.e. of a specific ID or range of CAN IDs) will be passed to the serial interface. The acceptance code value corresponds to a valid CAN IDs for either standard or extended CAN frames. This command is only active if the CAN channel is initiated and not opened.

The Set Acceptance Mask (m) command should be executed *prior* to the Set Acceptance Code (M) command.

Note: The CAN channel will revert to its prior state after execution. For example, if the channel is open when this command is executed, the channel will update the setting and return to the open state.

Command	Response	Function
Miii[CR]	[CR]	Set acceptance code for standard CAN frame (11 bit) identifier
Miiiiiii[CR]	[CR]	Set acceptance code for extended CAN frame (29 bit) identifier

iii = standard 11-bit CAN mask (0x000 through 0x7FF)

iiiiiii = extended 29-bit CAN mask (0x00000000 through 0x1FFFFFFF)

Default is to pass all frames (acceptance code = 0x7FF for standard messages and 0x1FFFFFFF for extended messages)

Example: m1FF[CR] sets acceptance code to receive standard messages with the CAN ID of 0x1FF. If used in conjunction with the acceptance mask example above, frames of the range 0x100 through 0x1FF will be passed, and all other CAN IDs will be blocked.

10.3.12 Getting Status Flags

User can use the command F[CR] to get the status bits when an error occurs. A two-byte BCD number is returned to correspond to the 8-bits of the internal register of the CAN controller.

Command	Response	Function
F[CR]	XX[CR]	Get CAN bus status

Return Codes

XX[CR]

XX = CAN bus status (A bit set to “1” indicates a true condition):

Bits 2, 1, 0: Last Error Code(LEC), The LEC field holds a code, which indicates the type of the last error to occur on the CAN bus.

LEC Bits 2, 1, 0	Meaning
Error Code 0 0, 0, 0	No error.
Error Code 1 0, 0, 1	Stuff error: more than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
Error Code 2 0, 1, 0	Form error: a fixed format part of a received frame has the wrong format.
Error Code 3 0, 1, 1	ACK Error: the message this CAN core transmitted was not acknowledged by another node.
Error Code 4 1, 0, 0	Bit 1 error: during the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value “1”), but the monitored bus value was dominant.
Error Code 5 1, 0, 1	Bit 0 error: Bit 1 error: during the transmission of a message (or acknowledged bit, or active error flag, or overload flag), the device wanted to send a dominant level (bit of logical value “0”), but the monitored bus value was recessive. During the bus-off recovery, this status is set each time a sequence of 11 recessive bits have been monitored. This enables the CPU to monitor the proceedings of the bus-off recovery sequence (indicating the bus is not stuck at dominant or continuously disturbed).
Error Code 6 1, 1, 0	CRC error: the CRC checksum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.
Error Code 7 1, 1, 1	Unused: no CAN bus event was detected since the CPU wrote this value to the LEC.

Bit 3: Transmitted a message successfully

1 = Since this bit was last reset by CPU, a message has been successfully (error-free and acknowledged by at least one other node) transmitted.

0 = Since this bit was last reset by CPU, no message has been transmitted.

Bit 4: Received a message successfully

1 = A message has been successfully received since this bit was last reset by CPU (independent of the result of acceptance filtering).

0 = No message has been successfully received since this bit was last reset by CPU

Bit 5: Error Passive (Read only)

1 = The CAN core is in the error passive state as defined in the CAN specification.

0 = The CAN core is in the error active.

Bit 6: Error Warning Status (Read only)

1 = At least one of the error counters in the EML (Error Management Logic) has reached the error warning limit of 96.

= Both error counters are below the error warning limit of 96.

Bit 7: Bus-off Status (Read only)

1 = The CAN Module is in bus-off state.

0 = The CAN Module is not in bus-off state.

<BELL> = ERROR

Bit 0 ~ Bit 7 returned to correspond to the 8-bits of the internal register of the CAN controller.

10.3.13 Getting Version Information

The command V[CR] to retrieve the current firmware version of the USB CAN adapter.

Command	Response	Function
V[CR]	VXXXX[CR]	Get the current firmware version of the USB CAN adapter

This command is always available and will return the version information formatted like this: VXXXX[CR].

10.3.14 Getting Serial Number

The command N[CR] will retrieve the serial number of the USB CAN adapter.

Command	Response	Function
N[CR]	TXXXXXXXX[CR]	Get the serial number of the USB CAN adapter

This command is always available and will return the decimal serial number like this: TXXXXXXXX[CR].

10.3.15 Resetting the USB CAN adapter

The command RST[CR] will reset the USB CAN adapter.

Command	Response	Function
RST[CR]	-	Reset the USB CAN adapter

This command is always available.

11 TOOLS

11.1 Setting 3MBit Baud Rate for CAN Bus

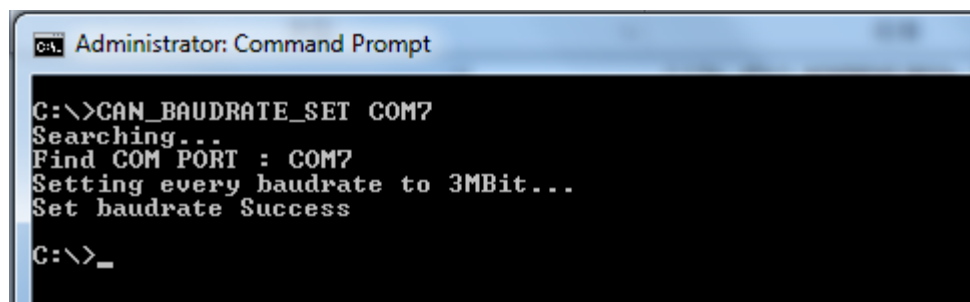
To get a better performance, set the USB to CAN adapter to work in high speed (3Mbits). **After executing the command, please disconnect the USB to CAN adapter from the system for about 5 seconds and connect it again.**

The set baud rate program (CAN_BAUDRATE_SET.exe) must be executed under command prompt with administrative privileges.

Following is set baud rate program command:

`CAN_BAUDRATE_SET <COM-PORT>`

Example: `CAN_BAUDRATE_SET COM7`



```
Administrator: Command Prompt
C:\>CAN_BAUDRATE_SET COM7
Searching...
Find COM PORT : COM7
Setting every baudrate to 3Mbit...
Set baudrate Success
C:\>_
```

COM7: the USB to CAN adapter installed as "USB Serial Port (COMX)".

You can check the port number under Device Manager.

After setting the baud rate successfully, you will find following message under "Command Prompt":

```
Searching...
Find COM PORT: COM7
Setting every baud rate to 3Mbit...
Set baud rate Success
```

11.2 Firmware Upgrade

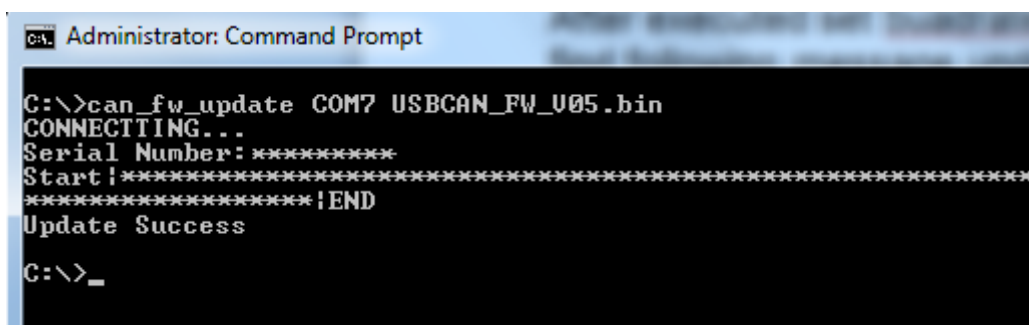
The USB to CAN adapter firmware can be updated for bug fixes and enhanced features. You can use our tool program to upgrade the firmware contents via serial port.

The firmware upgrade program (can_fw_update.exe) must be executed under "Command Prompt" with administrative privileges.

Following is the firmware upgrade command:

```
can_fw_update <COM-PORT> <FIRMWARE FILE>"
```

Example: can_fw_update COM7 USBCAN_FW_V05.bin



```
Administrator: Command Prompt
C:\>can_fw_update COM7 USBCAN_FW_V05.bin
CONNECTING...
Serial Number: *****
Start!*****
*****!END
Update Success
C:\>_
```

COM7: the USB to CAN adapter installed as "USB Serial Port (COMX)". You can check the port number under Device Manager.

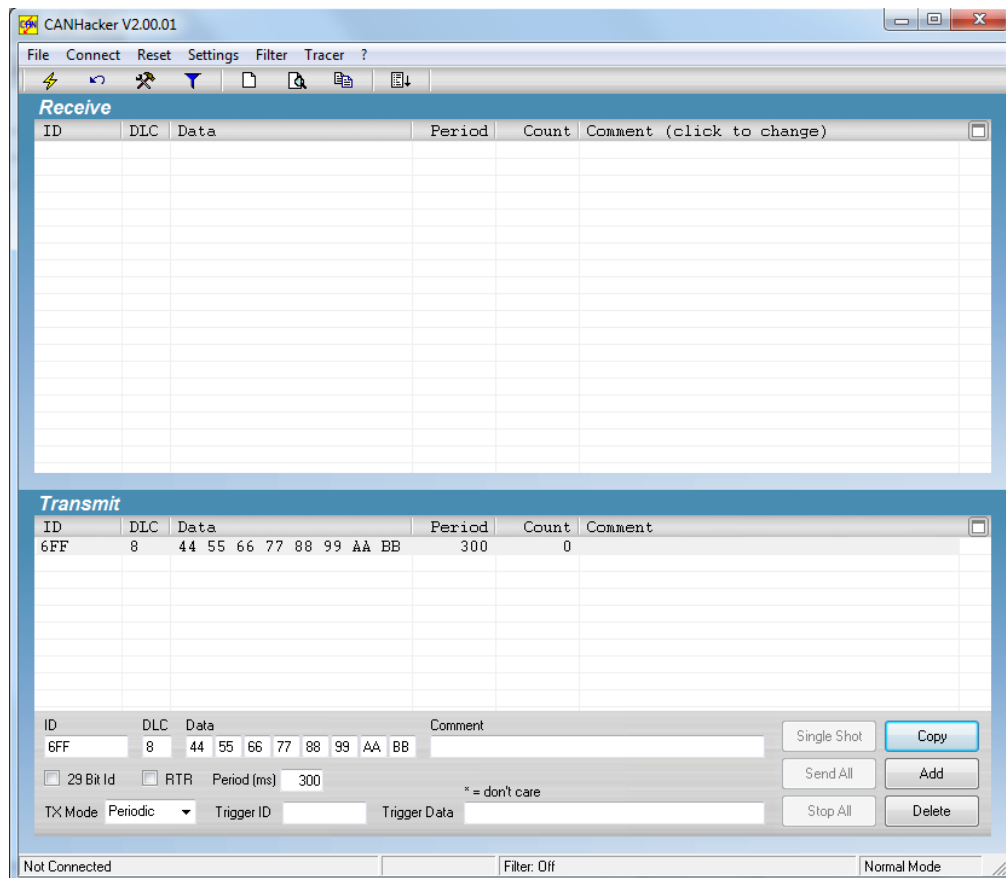
USBCAN_FW_V5.bin: new firmware file (binary file) of USB to CAN adapter.

After executing the firmware upgrade successfully, you will find following message under "Command Prompt":

```
CONNECTING...
Serial Number: T12345678
Start:
*****
*****: END
Update Success
```

11.3 CANHacker

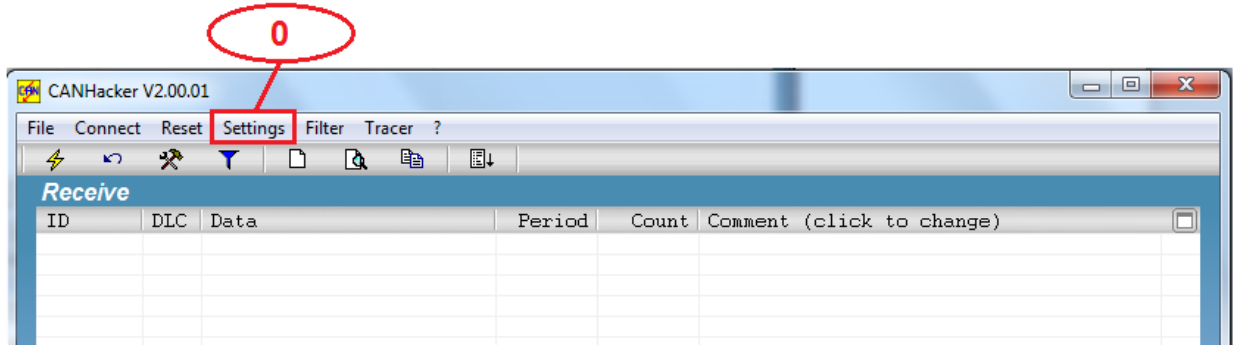
CANHacker is a Windows application software for analyzing and transmitting/receiving CAN frames. The CANHacker software has a friendly interface and is easy to use. Through the software user can easily test and analyze the CAN frames. Following shows its main panel:



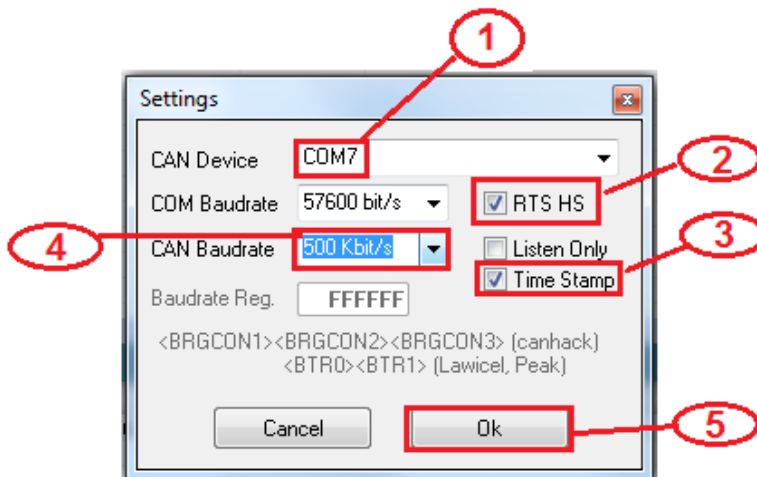
The following sections will briefly introduce the necessary steps on how to use the software.

11.3.1 Settings procedure for selecting and configuring the USB to CAN adapter

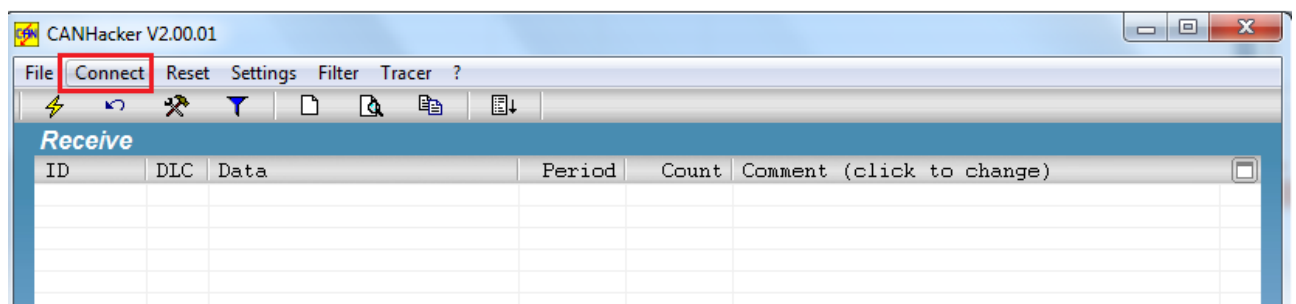
1. Open CANHacker and click “Settings” under the menu.



2. Select COM port of the USB to CAN adapter.
3. Check “RTS HS” to enable RTS handshake function.
4. Check “Time Stamp” to enable timestamp function.
5. Select CAN Baudrate for the CAN bus operating speed.
6. Finally, click “OK” to finish the settings and return to the main panel.



You may connect the USB to CAN adapter after configuration. Click “Connect”, as shown in the figure, to start the CANHacker software operation.



When USB to CAN adapter successfully connects, you will find the message

“Connected to XXX kbits/s”, firmware version VXXXX and operation mode at the bottom of the main panel.

Interface showing configuration parameters and status:

- ID: 12345688
- DLC: 8
- Data: AA CC DD EE FF 66 77 88
- Comment:
- ☒ 29 Bit Id
- ☐ RTR
- Period (ms): 100
- TX Mode: Periodic
- Trigger ID:
- Trigger Data:
- * = don't care
- Buttons: Single Shot, Copy, Send All, Add, Stop All, Delete
- Status Bar (highlighted):
 - Connected to 100 kbit/s
 - Firmware: V0.73
 - Filter: Off
 - Normal Mode

11.3.2 Receiving CAN frames

When CANHacker receives CAN frames from another CAN node, it will show all CAN frame messages in the middle of main panel. The CAN frame messages includes ID, DLC, Data, Period, Count.

[illegible]

11.3.3 Sending CAN frames

CANHacker provides many parameters for sending CAN frames to another CAN node, you can set the following parameters on the bottom of the main panel for CAN data transmission:

ID: 12345688 DLC: 8 Data: AA CC DD EE FF 66 77 88 Comment:
☒ 29 Bit Id ☐ RTR Period (ms): 100 * = don't care
 TX Mode: Periodic Trigger ID: Trigger Data:
 Single Shot Copy Send All Add Stop All Delete

Connected to 100 kbit/s Firmware: V0.73 Filter: Off Normal Mode

Select transmit an extended CAN Frame (29 bits ID) or a standard CAN frame (11 bits ID).

Check "29 Bit Id"  29 Bit Id to transmit an extended CAN Frame (29 bits ID) and uncheck "29 Bit Id"  29 Bit Id to transmit a standard CAN frame (11 bits ID).

Select remote request frame mode or transmit CAN frame mode.

Check “RTR”  for a remote request frame mode or uncheck “RTR”  for transmit CAN frame mode.

Enter CAN frame messages in the respective fields, including ID, DLC, Data.

ID	DLC	Data							
12345688	8	AA	CC	DD	EE	FF	66	77	88

In “TX Mode” dialog box, you can select “off”, “Periodic”, “RTR”, “Trigger” modes.

TX Mode	Periodic
	off
	Periodic
nnected to	RTR
	Trigger

When “Periodic” mode is selected, you can enter “Period(ms)” to send CAN frames message repeatedly (enter “500” to send CAN messages every 500ms).

Period (ms)	500
-------------	-----

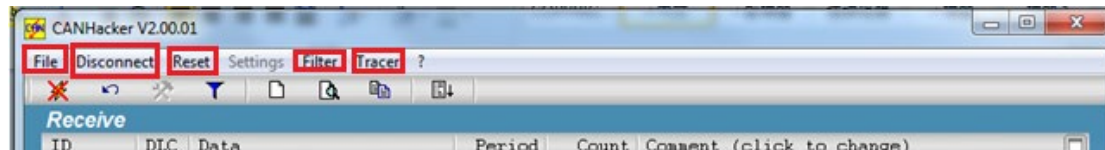
To send a single CAN frame message, click “Single Shot”. Click “Send All” to send CAN frames message repeatedly.

To stop sending CAN frame messages, click “Stop All”.

Single Shot
Send All
Stop All

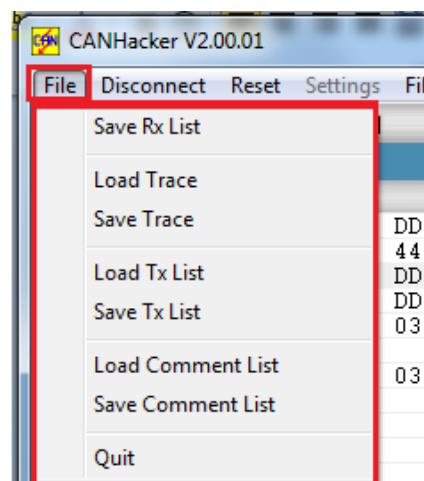
11.3.4 Assistant features

There are many assistant features included in CANHacker, as shown in the figure below:

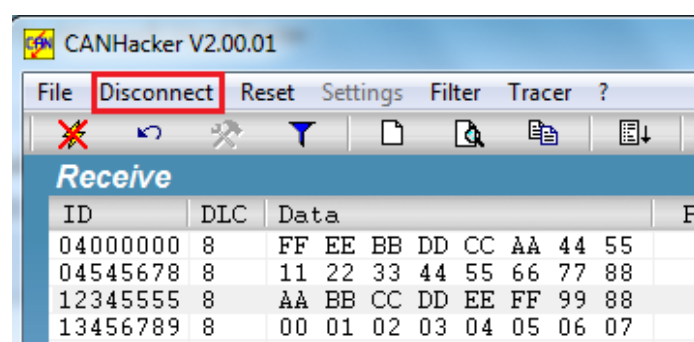


Saving data to file or loading data from file:

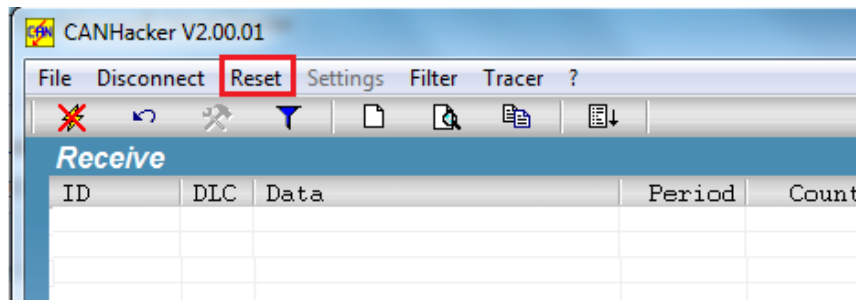
Select “File” option to save Rx List, Trace, Tx List, Command List and Load Trace, Tx List, Command List.



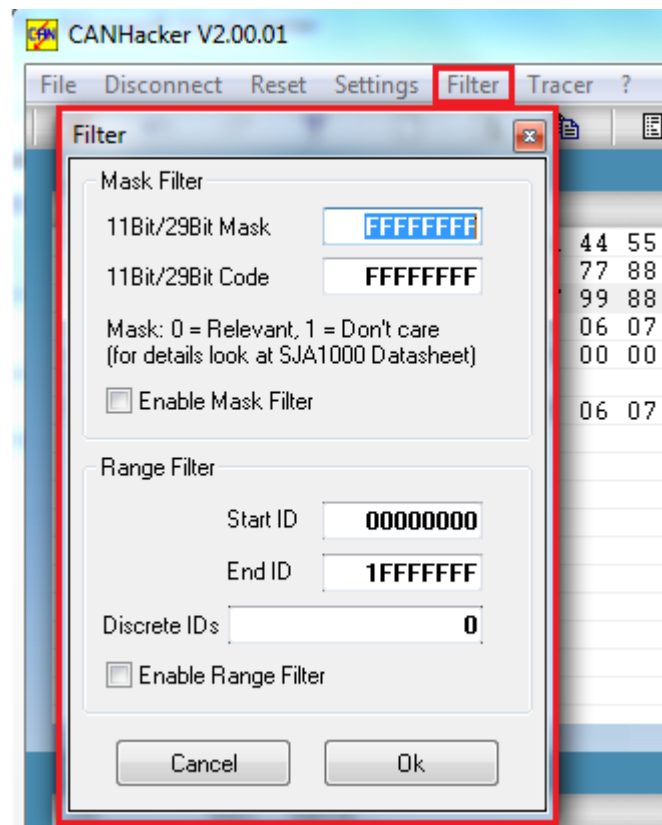
Click “Disconnect” to stop CANHacker.



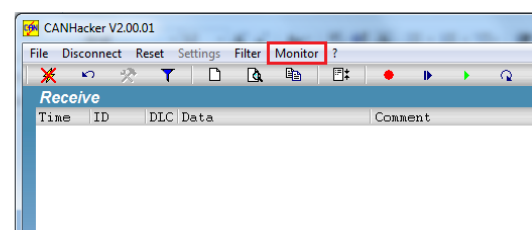
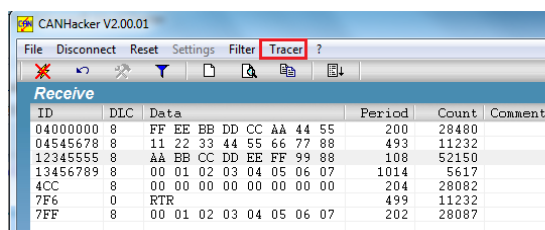
Click “Reset” to renew the received CAN frame messages and reset the transmission (received) count.



Select "Filter" to set mask filter and range filter.



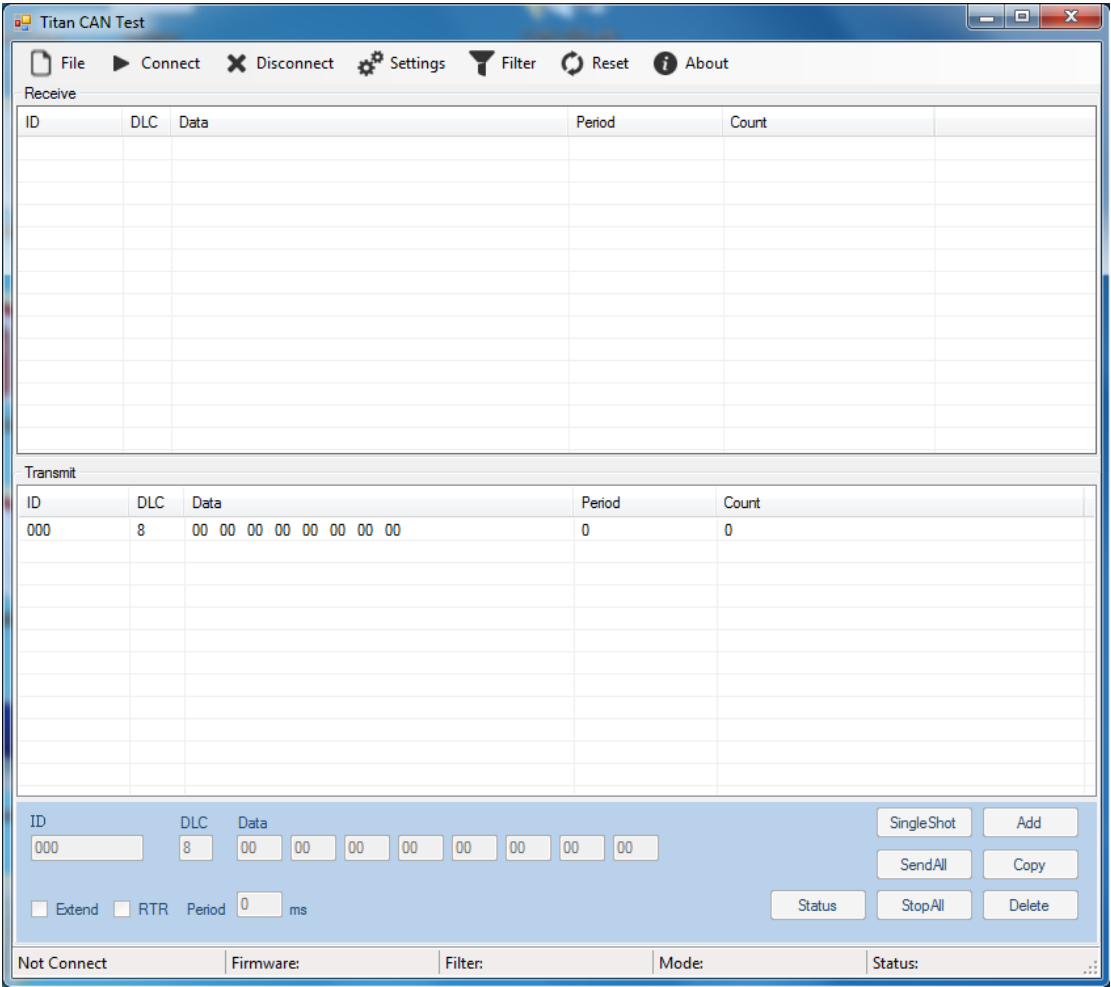
Select "Tracer" or "Monitor" to trace or monitor the CAN frame messages.



11.4 Titan CAN Test Program

Titan CAN test program is a Windows application software for testing and transmitting/receiving CAN frames. The Titan CAN test program is an easy to use software. Through the software users can easily test and analyze the CAN frames.

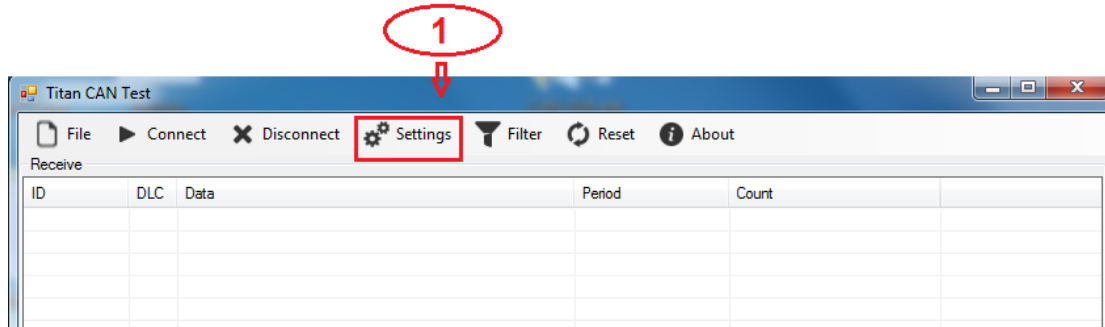
Following shows its main panel:



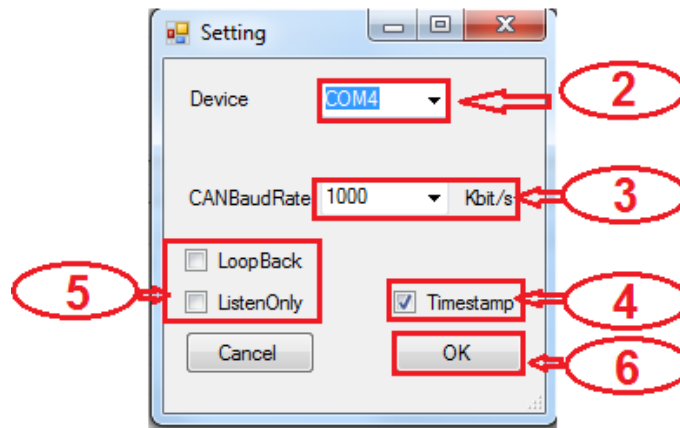
The following section will briefly introduce the necessary steps on how to use the Titan CAN test program.

11.4.1 Settings procedure for selecting and configuring the USB to CAN adapter

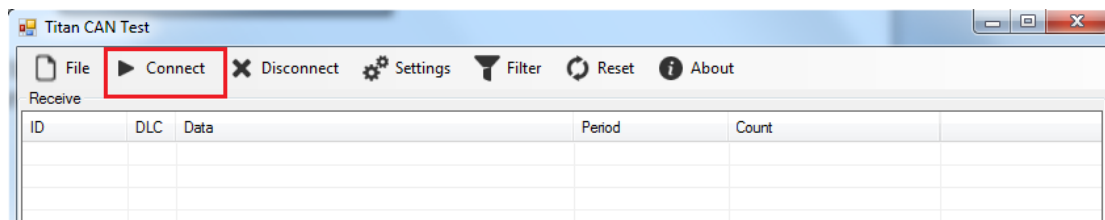
1. Open Titan CAN test program and click **"Settings"** under the menu.



2. Select COM port of the USB to CAN adapter.
3. Select CAN Baudrate for the CAN bus operating speed.
4. Check "Time Stamp" to enable timestamp function.
5. Check "LoopBack" or "ListenOnly" to open the CAN bus adapter in loopback or listen only operation mode, otherwise the CAN bus adapter will open in normal operation mode.
6. Finally, click "OK" to finish the settings and return to the main panel.



You may connect the USB to CAN adapter after configuration. Click **"Connect"**, as shown in the figure, to start the Titan CAN test program operation.



When USB to CAN adapter successfully connects, you will find the message "Connected to XXX kbits/s", firmware version VXXXX and operation mode at the bottom of the main panel.

ID	DLC	Data			
000	8	00 00 00 00 00 00 00 00			
☐ Extend	☐ RTR	Period 0 ms			
			SingleShot Add		
			SendAll Copy		
			Status StopAll Delete		
Connected to 1000Kbit/s			Firmware:V1.00	Filter:Off	Mode:Normal
			Status: ...		

ID	DLC	Data							
12345655	8	DD	AA	33	44	45	55	66	77

When “Periodic” mode is selected, you can enter “Period(ms)” to send CAN frames message repeatedly (enter “100” to send CAN messages every 100ms).

Period	100	ms
--------	-----	----

To send a single CAN frame message, click “Single Shot”. Click “Send All” to send CAN frames message repeatedly.

To stop sending CAN frame messages, click “Stop All”.

SingleShot
SendAll
StopAll

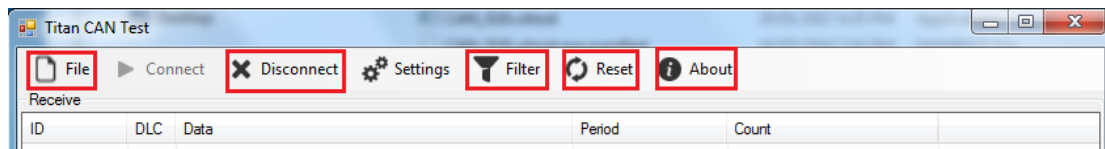
To add a new send CAN frame message, click “Add” to add new send CAN frame message and click “Copy” to copy a send CAN frame message repeatedly.

To delete a send CAN frame message, click “Delete” to delete send CAN frame message.

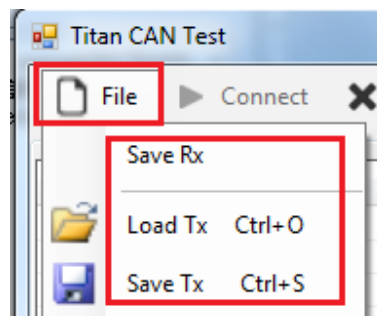
Add
Copy
Delete

11.4.4 Assistant features

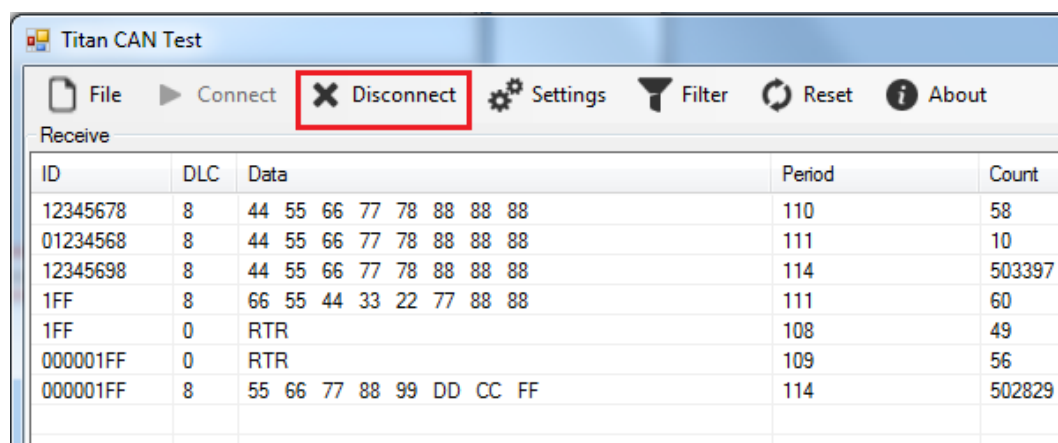
There are many assistant features included in Titan CAN test program, as shown in the figure below:



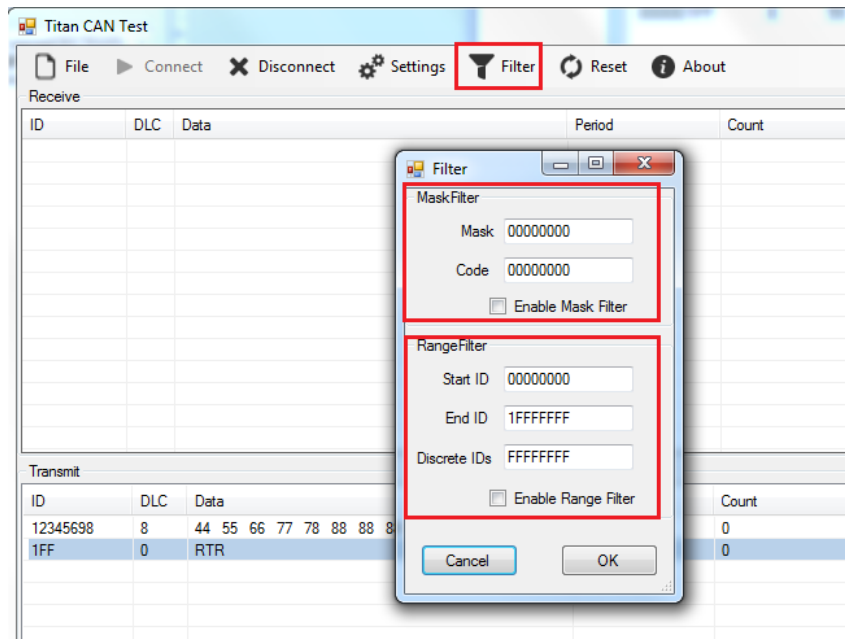
Select “File” option to save Rx List, Tx List and Load Tx List.



Click “Disconnect” to stop Titan CAN test program.

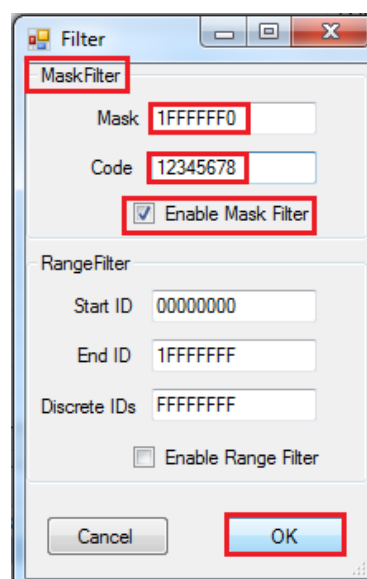


Select “Filter” to set mask filter and range filter.



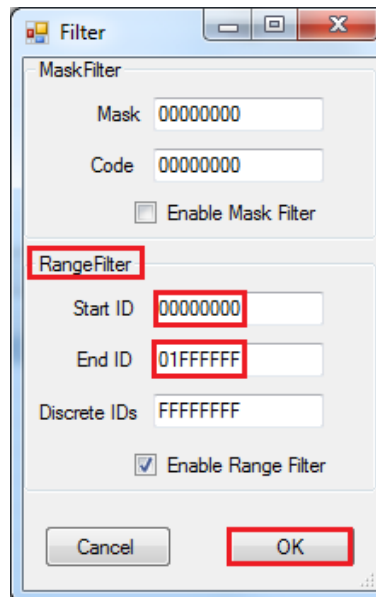
Mask Filter: Set “Acceptance Code Register” and “Acceptance Mask Register” for CAN bus controller to specify the CAN IDs that are passed or blocked; after setting “Mask” and “Code”, check “Enable Mask Filter” then click “OK” to finish the Mask Filter settings and return to the main panel.

Note: Before you set the “Mask Filter” function, you need to disconnect the USB adapter. After setting the value of “Mask” + “Code”, connect the USB adapter again to enable the “Mask Filter” function, because the “Mask Filter” function is only available if the CAN adapter is initiated and not opened.



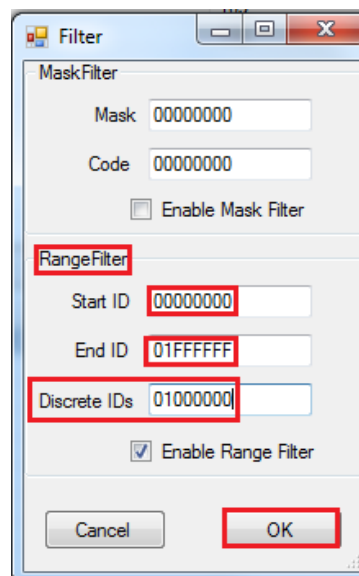
Mask Filter example: After setting “Mask” to 1FFFFFF0 and “Code” to 12345678, CAN message frames of the range 0x12345670 through 0x1234567F will be passed and all other CAN IDs will be blocked.

Range Filter: Set “Start ID” and “Stop ID” for USB CAN adapter to specify a range of CAN IDs that are to be passed; after setting “Start ID” and “Stop ID”, check “Enable Range Filter” then click “OK” to finish the Ranger Filter settings and return to the main panel.



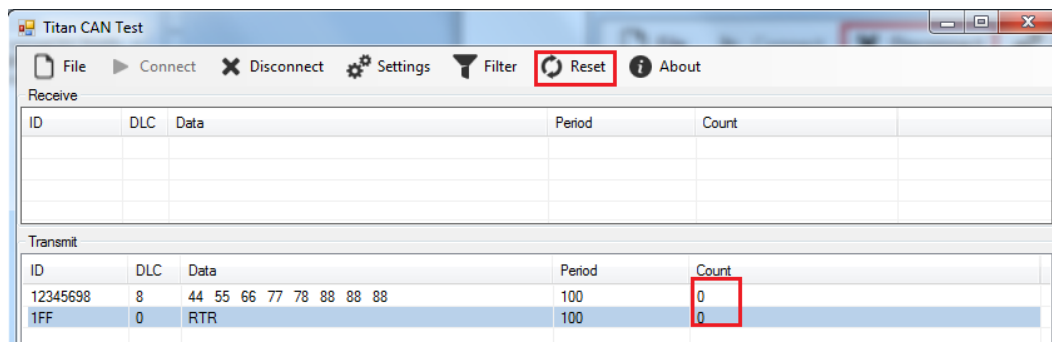
Range Filter example: After setting “Start ID” to 00000000 and “End ID” to 01FFFFFF, The CAN message frames of the range 0x00000000 through 0x01FFFFFF will be passed and all other CAN IDs will be blocked.

The Range Filter can also set “Discrete IDs” to block a unique CAN ID.

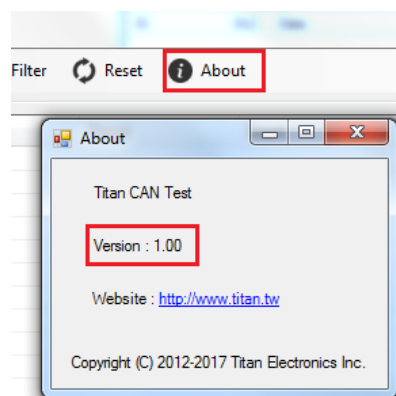


Discrete IDs Filter example: After setting “Start ID” to 00000000, “End ID” to 01FFFFFF and setting “Discrete IDs” to 01000000; The CAN ID range 0x00000000 through 0x01FFFFFF will be passed but only CAN ID 0x01000000 will be blocked.

Click “**Reset**” option to renew the received CAN frame message and reset the transmitted (received) count.



Click “**About**” option to show the version information of Titan CAN test program.



11.5 CANopen

CANopen is a CAN-based communication system. It comprises higher-layer protocols and profile specifications. CANopen has been developed as a standardized embedded network with highly flexible configuration capabilities. It was designed originally for motion-oriented machine control systems, such as handling systems. Today it is used in various application fields, such as medical equipment, off-road vehicles, maritime electronics, railway applications, or building automation.

CanFestival project is an open source CANopen multi-platform framework. (<http://www.canfestival.org/>) CanFestival focuses on providing an ANSI-C platform independent CANopen stack that can be implemented as master or slave nodes on PCs, Real-time IPCs, and Microcontrollers.

For detailed information about using CanFestival in your project see the "[The CanFestival CANopen stack manual](#)".

11.5.1 How to get CanFestival

You can get the CanFestival source code from [repository](#). Then get [TITAN CAN driver](#) for CanFestival. Or you can download the code with TITAN driver from [TITAN web site](#).

11.5.2 Linux Compilation and installation

Linux target is default configure target.

Call `./configure -help` to see all available compile time options.

After invoking `./configure` with your platform specific switches, just type `make`.

```
./configure --can=titan  
make  
make install
```

11.5.4 Windows Compilation

CanFestival can be compiled and run on Windows platform. It is possible to use both Cygwin and win32 native runtime environment.

Minimal Cygwin installation is required at configuration time in order to create specific header files (config.h and cancfg.h). Once these files created, Cygwin is not necessary anymore. Project and solution files have been created and tested with Visual Studio Express 2005. Be sure to have installed Microsoft Platform SDK, as recommended at the end of Visual Studio installation.

Cygwin must be installed with those packages:

1. gcc
2. unzip
3. wget
4. make

Extract CanFestival source code into your Cygwin home. Then configure CanFestival.

```
cd CanFestival
./configure --can=titan
make
```

11.5.5 Compilation with Visual Studio

You can either load independent “*.vcproj” project’ files along your own projects in your own solution or load the provided “CanFestival-3.vc8.sln” solution files directly. Build CanFestival-3 project first.

The “examples” directory contains some test program you can use as example you’re your own developments.

You'll find an example on the supplied CD showing the communication between master and slave nodes. Following baudrates are supported: 20K, 50K, 100K, 125K, 250K, 500K and 1M.

- CanFestival_example_win_x86.zip For Windows 32 bit
- CanFestival_example_win_x64.zip For Windows 64 bit
- CanFestival_example_linux_x86.tar.gz For Linux 32 bit
- CanFestival_example_linux_x64.tar.gz For Linux 64 bit

Under Linux connect two CAN devices, installed as /dev/ttyUSB0 and /dev/ttyUSB1. Open two terminal windows and change to the directory where examples were extracted to and execute

```
export LD_LIBRARY_PATH=.
./TestMasterSlave -s "/dev/ttyUSB0" -S 125K -M none -l ./libcanfestival_can_titan.so
```

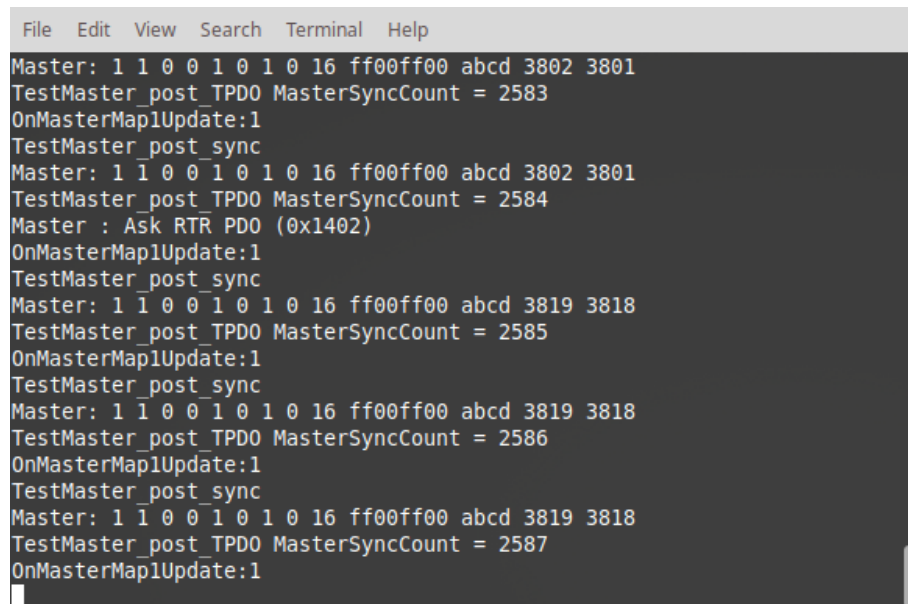
in the first terminal window and

```
export LD_LIBRARY_PATH=.
./TestMasterSlave -m "/dev/ttyUSB1" -M 125K -S none- ./libcanfestival_can_titan.so
```

in the second.

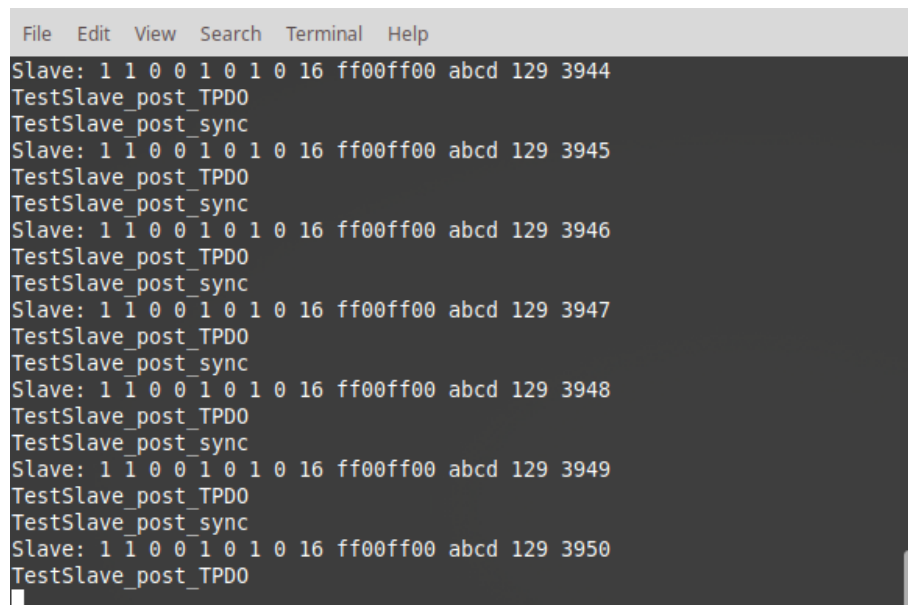
Following pictures shows the output messages of both nodes.

Master:



```
File Edit View Search Terminal Help
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 3802 3801
TestMaster_post_TPDO MasterSyncCount = 2583
OnMasterMap1Update:1
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 3802 3801
TestMaster_post_TPDO MasterSyncCount = 2584
Master : Ask RTR PDO (0x1402)
OnMasterMap1Update:1
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 3819 3818
TestMaster_post_TPDO MasterSyncCount = 2585
OnMasterMap1Update:1
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 3819 3818
TestMaster_post_TPDO MasterSyncCount = 2586
OnMasterMap1Update:1
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 3819 3818
TestMaster_post_TPDO MasterSyncCount = 2587
OnMasterMap1Update:1
```

Slave:



```
File Edit View Search Terminal Help
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3944
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3945
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3946
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3947
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3948
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3949
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 129 3950
TestSlave_post_TPDO
```

11.6 python-can

The `python-can` library provides Controller Area Network support for Python, providing common abstractions to different hardware devices, and a suite of utilities for sending and receiving messages on a CAN bus.

More information you can find in:

<https://python-can.readthedocs.io/en/master/>

USB-CAN-M is compatible to `slcan`-interfaces (slcan ASCII protocol)

<https://python-can.readthedocs.io/en/master/interfaces/slcan.html>

11.7 APPLICATION PROGRAMMING INTERFACE

The Application Programming Interface (API) gives the user tools to use all of the functions that the CAN adapter provides. It will make it much easier for users to build their own CAN controlling software with these functions, than to implement their application command by command on the ASCII protocol.

Users can use Windows-based API for use with high-level languages. Please refer to the following website for our GUI, sample codes and updates:

<https://www.titan.tw/drivers/can-api.html>

11.7.1 CAN_Open

CAN_Open(ComPort, szBitrate, acceptance_code, acceptance_mask, flags, Mode)

Function:

Opens a channel to the device.

Parameters:

- ComPort
 - Type: String
 - The COM port to be opened.
 - Format: "COMXXX"
 - Example: "COM1", "COM57", "COM118"
- szBitrate
 - Type: String
 - The bitrate to operate at. Can be one of the standard bitrates or a user-defined non-standard bitrate.
 - Format:
 - 10 = 10Kbps
 - 20 = 20Kbps
 - 50 = 50Kbps
 - 100 = 100Kbps
 - 125 = 125Kbps
 - 250 = 250Kbps
 - 500 = 500Kbps
 - 800 = 800Kbps
 - 1000 = 1000Kbps
 - XXXXXXXXX, non-standard bitrate
 - Example: "50", "1000", "000000150"
- acceptance_code
 - Type: String
 - Used in conjunction with the acceptance mask to filter CAN messages. Set to "00000000" for NULL to allow all messages. Also referred to as acceptance filter in other parts of the manual.
 - Format: "XXXXXXXX"
 - Example: "00000700"
- acceptance_mask
 - Type: String
 - Used in conjunction with the acceptance code to filter CAN messages. Set to "00000000" for NULL to allow all messages.
 - Format: "XXXXXXXX"
 - Example: "000001FF"

- flags
 - Type: IntPtr
 - Determines whether or not the timestamp function should be enabled.
 - Format:
 - 1 = Timestamp will be enabled
 - 0 = Timestamp will be disabled
 - Example: 1

- Mode
 - Type: Integer
 - Determines the mode the USB CAN should operate at.
 - Format:
 - 0 = Normal, the device will operate under normal circumstances
 - 1 = Listen only, the device will passively receive CAN messages
 - 2 = Loopback, the device will also receive messages it transmits
 - Example: 2

- Return value:
 - Type: Integer
 - Handle to the device.
 - Result:
 - > 0, CAN_Open is successful
 - -1, error communicating with COM port
 - -2, error in opening channel, COM port may be already in use
 - -3, error in parameter settings
 - Example: 2508

Sample Command:

`CAN_Open("COM3", "50", "00000000", "00000000", 1, 2)`

Opens a channel to COM3 at 50kbps, with all messages allowed, timestamp enabled and operating in loopback mode.

11.7.2 CAN_Close

CAN_Close(Handle)

Function:

Closes the channel with the specified handle.

Parameters:

- Handle
 - Type: Integer
 - The handle of the CAN channel to be closed.
 - Format: A numeric value provided by the return value of CAN_Open
 - Example: 2508
- Return value:
 - Type: Integer
 - Code indicating result of CAN_Close.
 - Result:
 - 1, CAN_Close is successful
 - -1, error communicating with COM port
 - -4, error: CAN channel is not open

Sample Command:

CAN_Close(2508)

Closes device connected to channel with the handle 2508.

11.7.3 CAN_Write

CAN_Write(Handle, Buf)

Function:

Writes a message to the channel with the specified handle.

Parameters:

- Handle
 - Type: Integer
 - The handle of the CAN channel to write to.
 - Format: A numeric value provided by the return value of CAN_Open
 - Example: 2508
- Buf¹
 - Type: CAN_MSG structure
 - The standard structure of CAN frame messages.
 - Format: Name of an instance of the CAN_MSG structure
 - Example: myCANMsg
- Return value:
 - Type: Integer
 - Code indicating result of CAN_Write.
 - Result:
 - 1, CAN_Write is successful
 - -1, error communicating with COM port
 - -4, error: CAN channel is not open

Sample Command:

CAN_Write(2508, myCANMsg)

Writes the message contained in myCANMsg to device connected to channel with the handle 2508.

¹ Refer to the “CAN_MSG Structure” section for more information

11.7.4 CAN_Read

CAN_Read(Handle, Buf)

Function:

Reads a message from the channel with the specified handle.

Parameters:

- Handle
 - Type: Integer
 - The handle of the CAN channel to read from.
 - Format: A numeric value provided by the return value of CAN_Open
 - Example: 2508
- Buf²
 - Type: CAN_MSG structure
 - The standard structure of CAN frame messages.
 - Format: Name of an instance of the CAN_MSG structure
 - Example: myCANMsg
- Return value:
 - Type: Integer
 - Code indicating result of CAN_Read.
 - Result:
 - 1, CAN_Read is successful
 - -1, error communicating with COM port
 - -4, error: CAN channel is not open
 - -5, error: there are no messages

Sample Command:

CAN_Read(2508, myCANMsg)

Reads the message from device connected to channel with the handle 2508 and stores it into myCANMsg.

² Refer to the “CAN_MSG Structure” section for more information

11.7.5 CAN_Flush

CAN_Flush(Handle)

Function:

Clears the buffers of the channel with the specified handle.

Parameters:

- Handle
 - Type: Integer
 - The handle of the CAN channel whose buffers are to be cleared.
 - Format: A numeric value provided by the return value of CAN_Open
 - Example: 2508
- Return value:
 - Type: Integer
 - Code indicating result of CAN_Flush.
 - Result:
 - 1, CAN_Flush is successful
 - -1, error communicating with COM port
 - -4, error: CAN channel is not open

Sample Command:

CAN_Flush(2508)

Clears the buffers of device connected to channel with the handle 2508.

11.7.6 CAN_Status

CAN_Status(Handle)

Function:

Checks the status bits for more specific details when an error occurs.

Parameters:

- Handle
 - Type: Integer
 - The handle of the CAN channel whose status bits are to be inquired.
 - Format: A numeric value provided by the return value of CAN_Open
 - Example: 2508
- Return value:
 - Type: Integer
 - Code indicating result of CAN_Status.
 - Result:
 - Bit [2, 1, 0]
 - 0, 0, 0: no error
 - 0, 0, 1: stuff error
 - 0, 1, 0: form error
 - 0, 1, 1: ACK error
 - 1, 0, 0: Bit1Error
 - 1, 0, 1: Bit0Error
 - 1, 1, 0: CRCError
 - 1, 1, 1: unused
 - Bit [3]
 - 1: message successfully transmitted
 - 0: no message has been transmitted
 - Bit [4]
 - 1: message successfully received
 - 0: no message has been received
 - Bit [5]
 - 1: CAN core is in error passive state
 - 0: CAN core is in error active state
 - Bit [6]
 - 1: at least one error counter in EML has reached the warning limit of 96
 - 0: both error counters are below the warning limit of 96
 - Bit [7]
 - 1: CAN module is in bus-off state
 - 0: CAN module is not in bus-off state
 - <BELL> = ERROR

Sample Command:

CAN_Status(2508)

Checks the status bits of device connected to channel with the handle 2508.

11.7.7 CAN_Version

CAN_Version(Handle, buf)

Function:

Retrieves the firmware version of the device connected to channel with the specified handle.

Parameters:

- Handle
 - Type: Integer
 - The handle of the CAN channel whose version information is to be inquired.
 - Format: A numeric value provided by the return value of CAN_Open
 - Example: 2508
- buf
 - Type: Character array/string
 - Information about the firmware version will be stored into this array.
 - Format: Name of a character array
 - Example: myVersion
- Return value:
 - Type: Integer
 - Code indicating result of CAN_Version.
 - Result:
 - 1, CAN_Version is successful
 - -1, error communicating with COM port
 - -4, error: CAN channel is not open

Sample Command:

CAN_Version(2508)

Retrieves the firmware version of device connected to channel with the handle 2508.

11.7.8 CAN_MSG Structure

- Members:
 - Id
 - Type: Unsigned Integer
 - Message ID.
 - Format: XXX (standard), XXXXXXXX (extended)
 - Example: 1FF
 - Size
 - Type: Byte
 - Message size.
 - Format: A numeric value from 0~8
 - Example: 8
 - Data
 - Type: Byte array with 8 elements
 - Content of the data to be sent/received.
 - Format: XX
 - Example: 11
 - Flags
 - Type: Byte
 - Determines the message ID type and timestamp settings.
 - Format:
 - 1, timestamp off, standard
 - 2, timestamp off, extended
 - 9, timestamp on, standard
 - 10, timestamp on, extended
 - Example: 9
 - Timestamp
 - Type: Unsigned Short
 - Value of the timestamp.
 - Format: No input from the user is required
 - Example: 0
- Sample Message:
 - With a CAN_MSG structure instance declared as myCANMSG:
 - myCANMsg.ID = 1FF
 - myCANMsg.Size = 3
 - myCANMsg.Data(0) = 11
 - myCANMsg.Data(1) = 22
 - myCANMsg.Data(2) = 33
 - myCANMsg.Flags = 10

11.7.9 Example Code for C

```
#include <stdio.h>
#include <stdlib.h>
#include "CAN_API.h"

int main() {
    TCAN_HANDLE Handle;
    TCAN_STATUS Status;
    CHAR *ComPort = "COM23";
    CHAR *szBtrRate = "800";
    CHAR *acceptance_code = "1FFFFFFF";
    CHAR *acceptance_mask = "00000000";
    VOID *flags = CAN_TIMESTAMP_OFF;
    DWORD Mode = LoopBack;

    char version[10];
    CAN_MSG SendMSG;
    CAN_MSG RecvMSG;
    Handle = -1;
    Status = 0;

    SendMSG.Flags = CAN_FLAGS_EXTENDED;
    SendMSG.Id = 0x12345678;
    SendMSG.Size = 8;
    SendMSG.Data[0] = 0x11;
    SendMSG.Data[1] = 0x22;
    SendMSG.Data[2] = 0x33;
    SendMSG.Data[3] = 0x44;
    SendMSG.Data[4] = 0x55;
    SendMSG.Data[5] = 0x66;
    SendMSG.Data[6] = 0x77;
    SendMSG.Data[7] = 0x88;

    Handle = CAN_Open ( ComPort, szBtrRate, acceptance_code, acceptance_mask, flags,
Mode );

    printf ( "handle= %d\n", Handle );
    if ( Handle < 0 ) {
        return 0;
    }

    memset ( version, 0, sizeof ( char ) * 10 );
    Status = CAN_Flush ( Handle );
    Status = CAN_Version ( Handle, version );

    if ( Status == CAN_ERR_OK ) {
        printf ( "Version : %s\n", version );
    }

    Status = CAN_Write ( Handle, &SendMSG );
    if ( Status == CAN_ERR_OK ) {
        printf ( "Write Success\n" );
    }

    while ( 1 ) {
        Status = CAN_Read ( Handle, &RecvMSG );
        if ( Status == CAN_ERR_OK ) {
            printf ( "Read ID=0x%X, Type=%s, DLC=%d, FrameType=%s, Data=",
RecvMSG.Id, ( RecvMSG.Flags & CAN_FLAGS_STANDARD ) ? "STD" : "EXT",
RecvMSG.Size, ( RecvMSG.Flags & CAN_FLAGS_REMOTE ) ? "REMOTE" : "DATA" );

            for ( int i = 0; i < RecvMSG.Size; i++ ) {
                printf ( "%X,", RecvMSG.Data[i] );
            }
        }
    }
}
```

```
        break;
    }
}

Status = CAN_Close ( Handle );
printf ( "Test finish\n" );
return 0;
}
```

11.7.10 Using the API in C#

1. Ensure that the DLL file is placed in the same folder as your application executable.
2. Import the functions you need from the DLL into your source code with the Declare statement:

```
[DllImport("can_api.dll", EntryPoint = "CAN_Open",  
CallingConvention = CallingConvention.Cdecl)]  
static extern Int32 CAN_Open(string SerialNrORComPortORNet,  
string szBtrrate, string acceptance_code, string acceptance_mask,  
Int32 flags, UInt32 Mode);
```

3. Create a definition of the CAN_MSG structure for the CAN_Write and CAN_Read functions, if needed.

```
public struct CAN_MSG  
{  
    public UInt32 Id;  
    public byte Size;  
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]  
    public byte[] Data;  
    public byte Flags;  
    public UInt16 TimeStamp;  
}
```

The keyword MarshalAs is used for all structure members to ensure that the structure size corresponds to what the DLL expects.

4. In order to communicate with the channel with other functions after opening it with CAN_Open, you need to create a variable to store the handle value.

```
Int myHandle;  
myHandle = CAN_Open(("COM3", "50", "00000000", "00000000", 1, 2);
```

5. This concludes the basic setup process of using the DLL in C#. Imported functions can then be easily called from the DLL with the parameters created above.

11.7.11 Using the API in Visual Basic .NET

6. Ensure that the DLL file is placed in the same folder as your application executable.
7. Import the functions you need from the DLL into your source code with the Declare statement:

```
Private Declare Function CAN_Open Lib "CANDLL_STDCALL.dll" (ByVal  
ComPort As String, ByVal szBtrate As String, ByVal acceptance_code  
As String, ByVal acceptance_mask As String, ByRef Flags As IntPtr,  
ByVal Mode As Integer) As Integer
```

8. Create a definition of the CAN_MSG structure for the CAN_Write and CAN_Read functions, if needed.

```
Imports System.Runtime.InteropServices  
  
Public Structure CAN_MSG  
    <MarshalAs (UnmanagedType.U4)>  
    Public Id As UInteger  
    <MarshalAs (UnmanagedType.U1)>  
    Public Size As Byte  
    <MarshalAs (UnmanagedType.ByValArray, SizeConst:=8,  
ArraySubType:=UnmanagedType.U1)>  
    Public Data As Byte()  
    <MarshalAs (UnmanagedType.U1)>  
    Public Flags As Byte  
    <MarshalAs (UnmanagedType.U2)>  
    Public Timestamp As UShort  
End Structure
```

The keyword `MarshalAs` is used for all structure members to ensure that the structure size corresponds to what the DLL expects.

To use the `CAN_MSG` structure, you will need to create an instance of the structure you just defined.

```
Private myCANMSG As CAN_MSG
```

Before accessing this instance you just created for the first time, set the size for the `Data` member to avoid “array out of bounds” error. This can be done in your program’s constructor.

```
ReDim myCANMSG (7)
```

9. In order to communicate with the channel with other functions after opening it with CAN_Open, you need to create a variable to store the handle value.

```
Private myHandle As Integer  
myHandle = CAN_Open(("COM3", "50", "00000000", "00000000", 1, 2)
```

10. This concludes the basic setup process of using the DLL in Visual Basic .NET. Imported functions can then be easily called from the DLL with the parameters created above.

11.7.12 Using the API in Visual Basic 6.0

1. Ensure that the DLL file is placed in the same folder as your application executable.
2. Import the functions you need from the DLL into your source code with the Declare statement:

```
Private Declare Function CAN_Open Lib "CANDLL_STDCALL.dll" (ByVal  
ComPort As String, ByVal szBtrrate As String, ByVal acceptance_code  
As String, ByVal acceptance_mask As String, ByRef Flags As Long,  
ByVal Mode As Long) As Long
```

3. Create a definition of the CAN_MSG structure for the CAN_Write and CAN_Read functions, if needed.

```
Private Type CAN_MSG  
    Id As Long  
    Size As Byte  
    Data(0 To 7) As Byte  
    Flags As Byte  
    Timestamp As Integer  
End Type
```

To use the CAN_MSG structure, you will need to create an instance of the structure you just defined.

```
Private myCANMSG As CAN_MSG
```

4. In order to communicate with the channel with other functions after opening it with CAN_Open, you need to create a variable to store the handle value.

```
Private myHandle As Long  
myHandle = CAN_Open(("COM3", "50", "00000000", "00000000", 1, 2)
```

5. This concludes the basic setup process of using the DLL in Visual Basic 6.0. Imported functions can then be easily called from the DLL with the parameters created above.

11.7.13 Using the API in Python

6. Ensure that the DLL file is placed in the same folder as your application executable.

7. Import the DLL using the ctypes library function LoadLibrary.

```
from ctypes import windll

DLL = windll.LoadLibrary("CANDLL_STDCALL.dll")
```

8. Create a definition of the CAN_MSG structure for the CAN_Write and CAN_Read functions, if needed.

```
from ctypes import Structure, c_uint, c_ubyte, c_ushort

class CAN_MSG(Structure):
    _fields_ = [("Id", c_uint),
                ("Size", c_ubyte),
                ("Data", c_ubyte * 8),
                ("Flags", c_ubyte),
                ("Timestamp", c_ushort)]
```

To use the CAN_MSG structure, you will need to create an instance of the structure you just defined.

```
myCANMSG = CAN_MSG()
```

9. In order to communicate with the channel with other functions after opening it with CAN_Open, you need to create a variable to store the handle value.

```
myHandle = DLL.CAN_Open(b"COM3", b"50", b"00000000", b"00000000",
1, 2)
```

10. This concludes the basic setup process of using the DLL in Python. Imported functions can then be easily called from the DLL with the parameters created above.

11.7.14 Using the API in LabVIEW

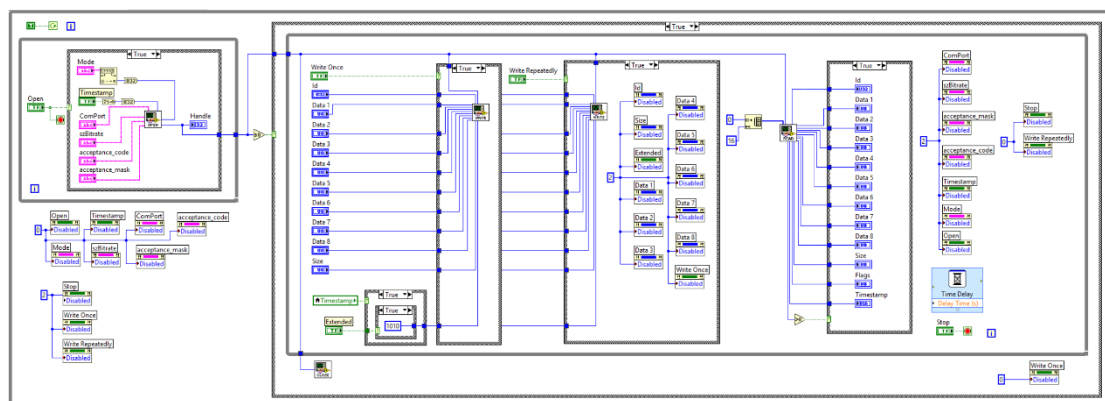
- CAN_Main.vi

The main panel is a simple, easy-to-use example program which contains most of the important functions available for use in the CAN API. Different functions can be tested by changing the settings on the leftmost side, which are restricted to legal parameters to prevent an error in operation. For example, the user can choose from Normal, Listen Only, or Loopback mode to suit their purposes.

ComPort
COM3
BaudRate: 50
acceptance_mask: NULL
acceptance_code: NULL
Timestamp Mode: ☐
Mode: Normal
Handle: 0
Id: 1FFFFFFF
Size: 8
Extended: ☐
Write Once
Write Repeatedly
Id: 0
Size: 0
Flags: 0
Timestamp: 0
Extended: 0, 1, 2, 3, 4, 5, 6, 7

Once the channel is opened, the user can use either “Write Once” or the “Write Repeatedly” button to send messages as configured in the fields. Messages received will appear on the fields on the rightmost side, if they are available.

All subVI icons have been customized, with the terminals wired to be user-friendly, increase readability and allow for cleanliness in larger projects, as seen in the block diagram for the main panel.



- CAN_Open.vi

Description

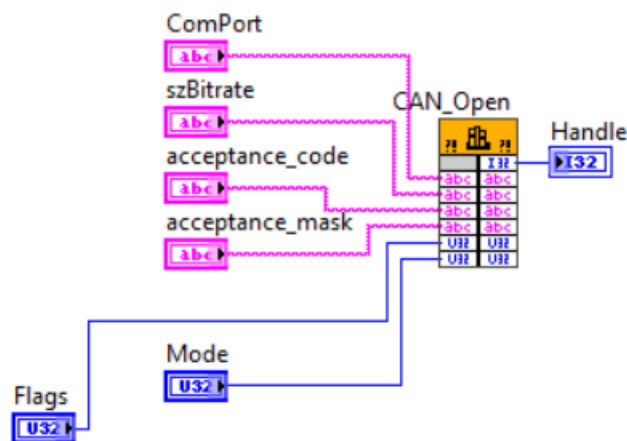
Opens a channel to the device.

Input

- ComPort: The COM port to establish a connection with.
- szBitrate: The speed at which the connection is to be made, with preset values of 10, 20, 50, 100, 125, 250, 500, 800, 100
- acceptance_code: Used for filtering CAN messages. To be used with the acceptance mask.
- acceptance_mask: Used for filtering CAN messages. To be used with the acceptance code.
- Flags: Whether or not the timestamp function should be enabled.
- Mode: The mode at which the device should operate at, with choices being Normal, Listen Only and Loopback

Output

- Return: Handle to the device. A positive value indicates success in opening the channel, while -2 represents error when opening channel and -3 represents error in input parameters



- CAN_Close.vi

Description

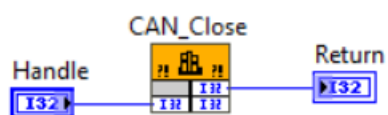
Closes the CAN channel with the specified handle.

Input

- Handle: The handle of the CAN channel which is to be closed

Output

- Return: A positive value indicates success in closing the channel, whereas a negative value indicates an error in closing the channel.



- CAN_Write.vi

Description

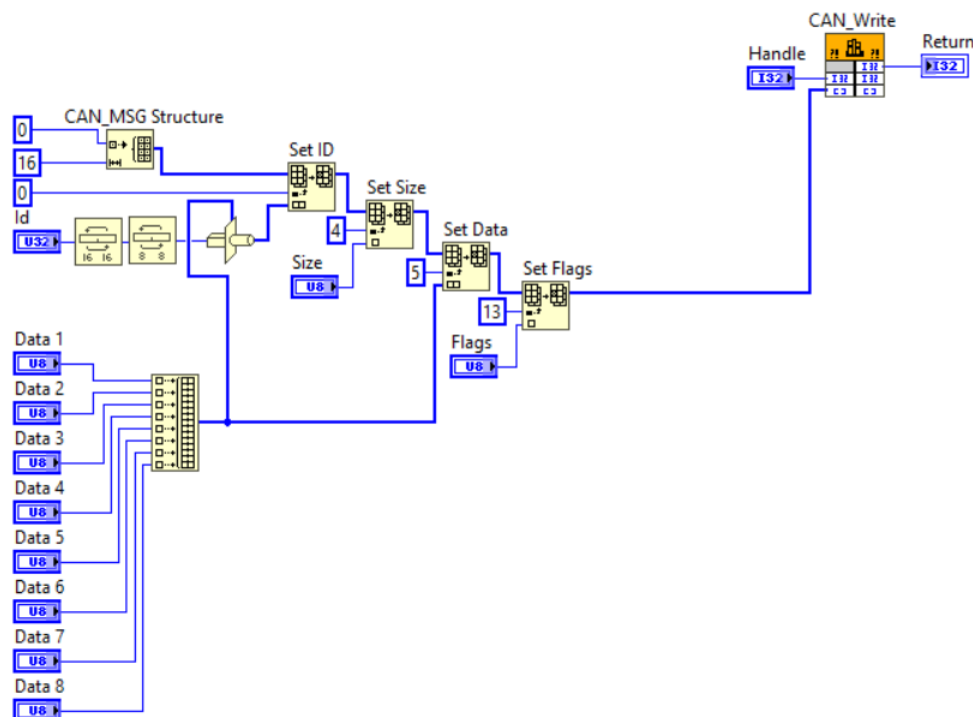
Writes a message to the CAN channel with the specified handle.

Input

- Handle: The handle of the CAN channel which the message is to be sent to.
- Id: Message ID.
- Size: Frame size (0~8).
- Data[8]: Data bytes 0~7.
- Flags: 1 (standard), 2 (extended), 9 (standard + timestamp), 10 (extended + timestamp)
- Timestamp: Timestamp (ms)

Output

- Return: A positive value indicates success in sending the message, whereas a negative value indicates an error in sending the message, with -4 representing that the channel is not open.



- CAN_Read.vi

Description

Read a message from the CAN channel with the specified handle.

Input

- Handle: The handle of the CAN channel which the message is to be read from.

Output

- Return: A positive value indicates success in reading the message, whereas a negative value indicates an error in reading the message, with -4 representing that the channel is not open and -5 representing that there is no message to be read.
- Id: Message ID.
- Size: Frame size (0~8).
- Data[8]: Data bytes 0~7.
- Flags: 1 (standard), 2 (extended), 9 (standard + timestamp), 10 (extended + timestamp)
- Timestamp: Timestamp (ms).

