www.vscom.de

VSCAN Manual

Edition: September 2018

Tel: +49 40 528 401 0
Fax: +49 40 528 401 99
Web: www.visionsystems.de
Support: faq.visionsystems.de

Contents

1 Installation

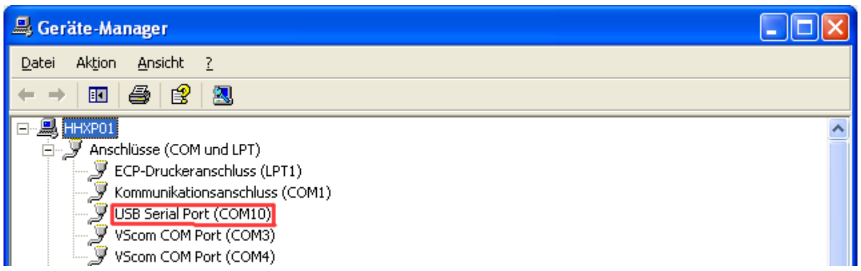
VSCAN devices support both Windows and Linux operating systems. The ASCII protocol is used to exchange data and control information with the devices and hence a serial interface is required to enable the communication.

USB-CAN Plus requires Windows driver installation. This is best done by connecting the device to the PC, and have Windows download the appropriate drivers from Windows Update via Internet. Usually this does not require manual interaction. Modern Linux distributions often include those drivers by default, so no need to install the driver. NetCAN Plus has Windows drivers only, but can be used operating system independent without driver in TCP raw mode. Please read the relevant installation instructions below for your particular device.

For the ease of use ASCII protocol is implemented in vs_can_api library with appropriate API. Linux users can alternatively use SocketCAN driver, that also implements ASCII protocol.

1.1 USB-CAN Plus Device

On a modern Windows system the USB driver probably installs automatically being downloaded from Microsoft on the first connection to the PC. If this is not desired download the driver from our company website[1]. Execute the installer, then plug in the adapter in an USB port of your choice. The device will be registered and you can use the COM port for your further work (see Figure 1).



HHXP01
Anschlüsse (COM und LPT)
ECP-Druckeranschluss (LPT1)
Kommunikationsanschluss (COM1)
USB Serial Port (COM10)
VScom COM Port (COM3)
VScom COM Port (COM4)

Figure 1: Device Manager

To get a better performance and set every standard baudrate as an alias for 3Mbit (**except 115200**), you should use the VS USB-COM Configurator software (Figure 2). This will a) set 'Latency Timer' to 1ms, b) activate 'Event on Surprise Removal', and c) set all baud rates to 3MBit except of 115.2k.

On the left list select the Com Port of your USB-CAN Plus device (Make sure that it is the CAN adapter). Then click 'Optimize for USB-CAN', and finally click 'OK'.

[1] http://www.visionsystems.de/produkte/usb-can-plus-usb-can-plus-iso.html#downloads

On Windows XP as an administrator open a command console window (DOS Box). Navigate to the folder where the script *„regmodify.vbs"* is located and execute it with the desired COM port like this:

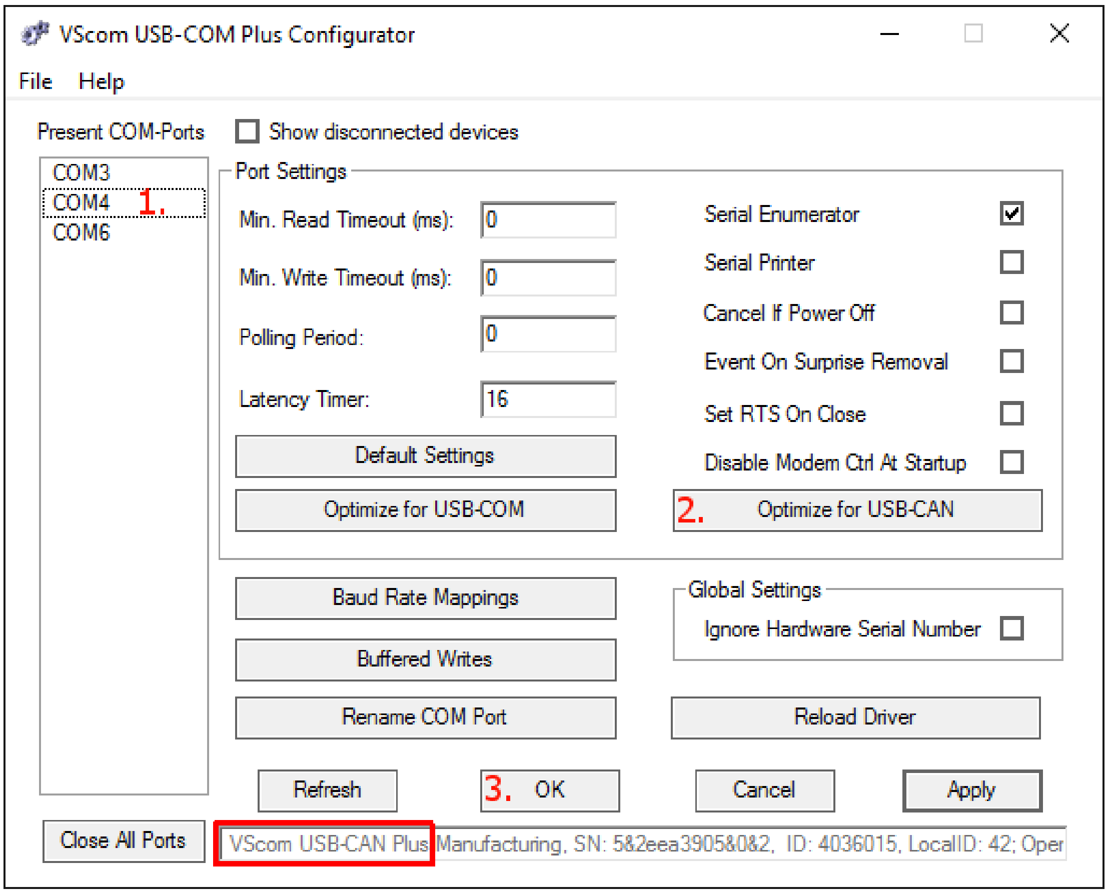
```
cscript regmodify.vbs COM10
```

Please disconnect the device from the system for about 5 seconds after this step and connect again.

Then you can open the COM port of the USB-CAN Plus with any standard baudrate[2] (see Figure 3).

As for the Linux the driver it is already available in modern kernels. The device name will be `/dev/ttyUSBx`. Use port configuration as 3M,8N1 with RTS/CTS flow control.

[2] 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 230400, 460800, 921600



☐ Show disconnected devices

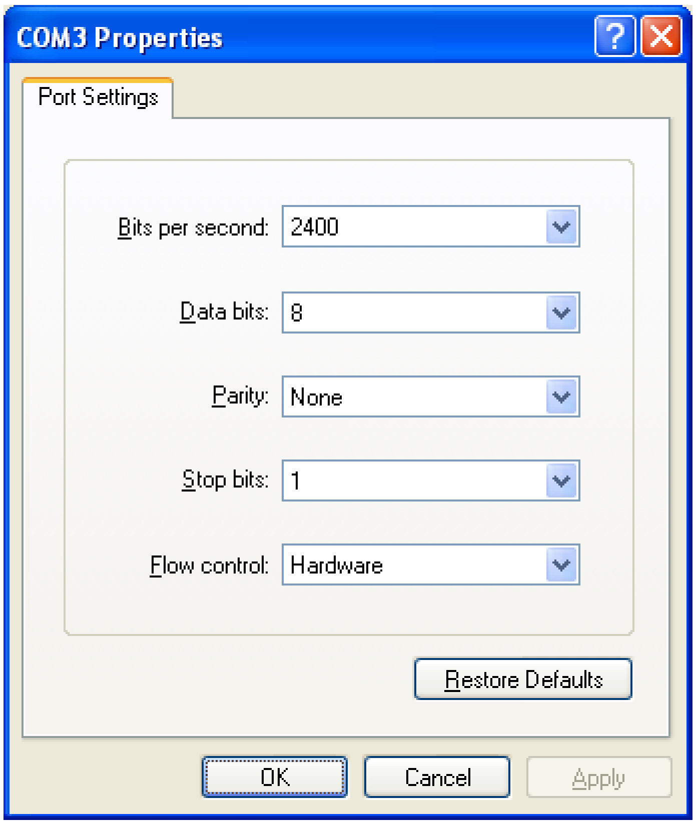
Min. Read Timeout (ms): 0
Min. Write Timeout (ms): 0
Polling Period: 0
Latency Timer: 16

☑ Serial Enumerator
☐ Serial Printer
☐ Cancel If Power Off
☐ Event On Surprise Removal
☐ Set RTS On Close
☐ Disable Modem Ctrl At Startup

☐ Ignore Hardware Serial Number

VScom USB-CAN Plus Manufacturing, SN: 5&2eea3905&0&2, ID: 4036015, LocalID: 42; Oper

Figure 2: USB-COM Configurator


COM3 Properties

Port Settings

Bits per second: 2400

Data bits: 8

Parity: None

Stop bits: 1

Flow control: Hardware

Figure 3: COM Port Properties

1.2 Network CAN Device

NetCAN Plus gateways provide CAN communication over network in versatile ways. For brevity the name NetCAN Plus is sometimes shortened to NetCAN$^+$.

1.2.1 Operational Modes

NetCAN Plus can act either as CAN Server (CAN frames will be exchanged between NetCAN$^+$ and an application connected via COM port or TCP socket) or as CAN over IP Router (two or more NetCAN$^+$ exchange CAN frames between various CAN networks)

CAN Server Mode For CAN Server following approaches are provided:

Driver mode: in this mode the network is transparent for the application. To use this mode the installation of the Windows driver is required (please refer to section 1.4 on page 20 for detailed installation instructions). After driver installation the new virtual COM port will be available to the system, so NetCAN Plus can be used in the same way as USB-CAN Plus. Due to the virtual COM port protocol overhead the performance is lower than by the TCP raw mode. You don't have to set the baudrate and hardware handshake explicitly - it's fixed to 3Mbit and RTS/CTS.

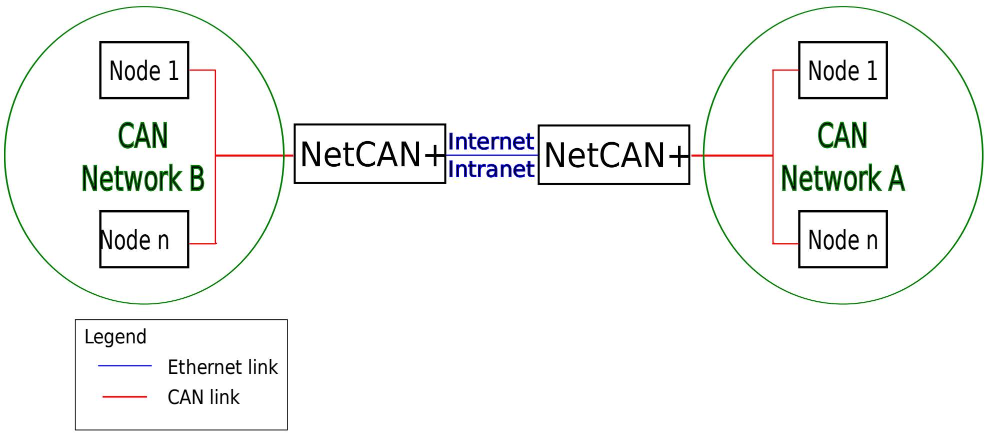
TCP raw mode: the communication will be handled directly via IP address and port number. In this mode no driver installation is required.

NetCAN$^+$ devices can be operated with either ASCII or VSCAN API. For the API there is no difference whether Driver mode or TCP raw mode is used, but due to performance issues TCP raw mode is recommended.

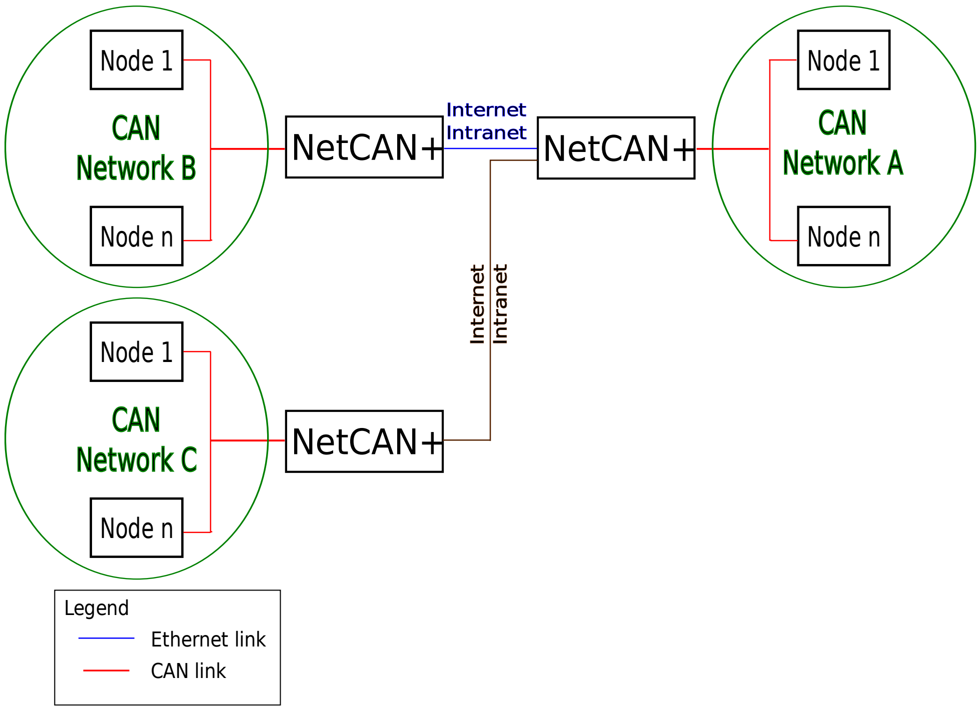
CAN over IP Router In this mode two or more NetCAN Plus devices are interconnected to enable seamless communication between two or more CAN networks. Figure 4a shows the first case, where two NetCAN$^+$ devices act as a tunnel between CAN Network A and B, so all frames sent inside Network A will be transported to Network B and vice versa.

Figure 4b shows extension of the tunnel. In this case additional CAN networks can be attached, so that CAN frames sent inside Network A will be transported to both B and C, but CAN Network B and C communicate only with Network A, so frames from Network B could not be seen by Network C and vice versa. See Section 1.2.4 for configuration instructions.

CAN Acceptance Code and Acceptance Mask can be set to filter the CAN frames, so only dedicated frames are passed via TCP link.



(a) Tunnel



(b) 1-to-n

Figure 4: CAN over IP

1.2.2 Configuration Overview

NetCAN Plus device can be configured in following ways:

- via web interface
- via Telnet
- via NetCom Manager

Before you can use the above mentioned methods you first need to know the IP address of your NetCAN$^+$ device. On the first start the device tries to obtain its IP address using DHCP[3] protocol. If there is no DHCP server on the network a default IP address 192.168.254.254 will be used. In both cases your PC and NetCAN$^+$ must be in the same subnet[4] in order to talk to each other.

Once this requirement is satisfied NetCAN$^+$ device will appear in the Network Places of Windows Explorer (press Windows key + 'e' to open the Windows Explorer and click on the Network symbol). You'll see following icon as shown in Figure 5 on page 11 providing information about device's serial number and IP address. Clicking on this icon would open your default Internet browser showing the main page where NetCAN$^+$ welcomes you with its "Home" screen (see Figure Figure 6 on page 12).

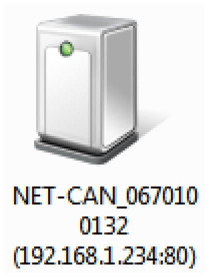
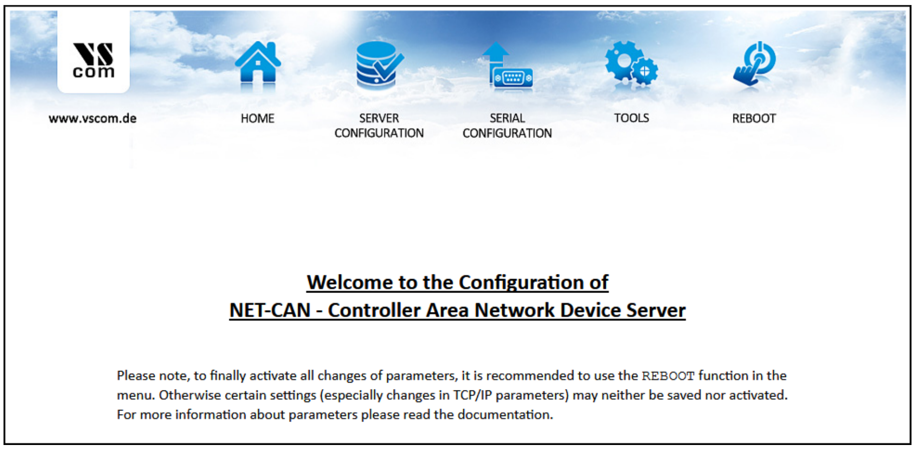


Figure 5: UPnP Device Display

Click on the icon for your desired option. In many menus you'll see a blue question mark. This is a symbol for help. Click it to get a short explanation, informing about the function of this parameter. Some other settings require a reboot to save and activate them. Whenever this situation occurs, the NetCAN$^+$ requests for a Reboot (see Figure 7).

You can instantly reboot or do that later when the configuration is finished.

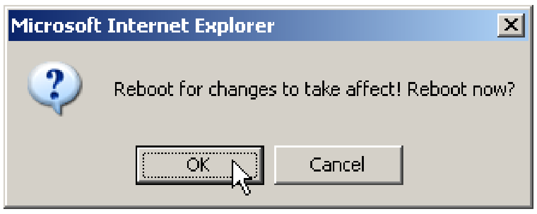
[3] https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
[4] https://support.microsoft.com/en-us/help/164015/understanding-tcp-ip-addressing-and-subnetting-basics



Welcome to the Configuration of
NET-CAN - Controller Area Network Device Server

Please note, to finally activate all changes of parameters, it is recommended to use the REBOOT function in the menu. Otherwise certain settings (especially changes in TCP/IP parameters) may neither be saved nor activated. For more information about parameters please read the documentation.

Figure 6: Web Interface for Configuration



Reboot for changes to take affect! Reboot now?

Figure 7: Web Interface Request to Reboot

1.2.3 Webbrowser Server Configuration

The *Server Configuration* is a very long menu (see Figure 8). There is basic server information (see Figure 8a), the server parameters related to the IP-configuration (see Figure 8b), the section for wireless communication (NetCAN Plus 120 WLAN only) (see Figure 8c), the section for encrypted communication (see Figure 8d), Password settings (see Figure 8e), and finally the configuration for date and time (see Figure 8f).

Information about the selected NetCAN$^+$ is displayed as *Server Info*. Starting with the *Server Type*, this is the model of the NetCAN$^+$, followed by the version of Software and Hardware. This will give a rough overview, which features are implemented, or need an upgrade of the firmware. You can check whether a new firmware version is available via clicking on the „Check for new version" button. The *Serial Nr.* is important to identify the device you are configuring right now. For further information the *UpTime* is listed. *Contact* and *Location* are user-defined information. They may later help to find the device in the installation, and the person responsible for management.

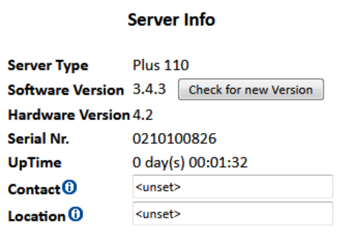
The *Server Parameter* allow configuration of the NetCAN Plus' name and of course all parameters in IP-settings. Generally it is used as information, e.g. in the NetCom Manager program or in SNMP.

Manual changes of IP parameters are only available with *DHCP* set as Disabled. When

DHCP is not used, enter *IP Address* and *Netmask*, as well as the *Broadcast* address. *Gateway* is required, if there are Routers in the network. DNS is used to access other stations by name. The *ConfigPort* is used to access the NetCAN$^+$ for administration via Telnet. It is suggested to use the standard value for Telnet, TCP port number 23. However it may be changed for different purposes. This does not change the function of the Telnet menus.

KeepAlive is an intrinsic function of the TCP/IP protocol. If used it causes network traffic, but connection problems are detected earlier. In a LAN this is usually not a problem. However, if used via DialUp connections this may cause problems. If this function is used, you must define an interval in seconds. NetCAN$^+$ has a better chance to react on network problems, or failed hosts. Even dropping an old connection may be useful in certain environments.

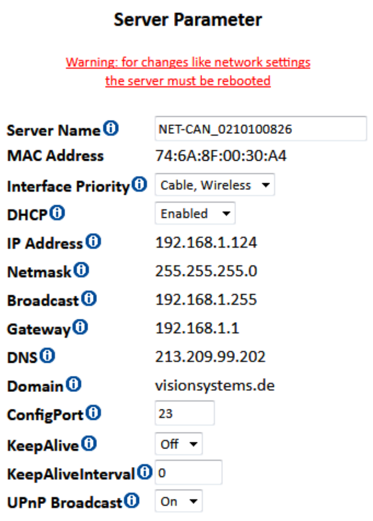
For detailed information about further Server Configuration options please refer to section „Server Configuration" of the NetCom Plus User Manual.



Server Info

Server Type	Plus 110
Software Version	3.4.3
Hardware Version	4.2
Serial Nr.	0210100826
UpTime	0 day(s) 00:01:32
Contact	<unset>
Location	<unset>

(a) Server Info

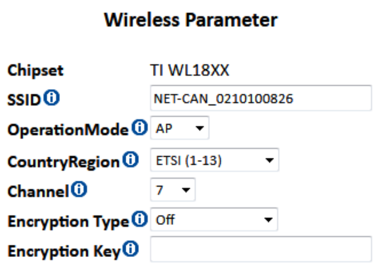


Server Parameter

Warning: for changes like network settings the server must be rebooted

Server Name	NET-CAN_0210100826
MAC Address	74:6A:8F:00:30:A4
Interface Priority	Cable, Wireless
DHCP	Enabled
IP Address	192.168.1.124
Netmask	255.255.255.0
Broadcast	192.168.1.255
Gateway	192.168.1.1
DNS	213.209.99.202
Domain	visionsystems.de
ConfigPort	23
KeepAlive	Off
KeepAliveInterval	0
UPnP Broadcast	On

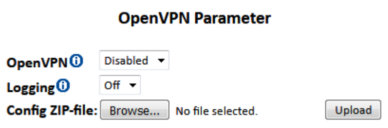
(b) Server Parameter



Wireless Parameter

Chipset	TI WL18XX
SSID	NET-CAN_0210100826
OperationMode	AP
CountryRegion	ETSI (1-13)
Channel	7
Encryption Type	Off
Encryption Key	

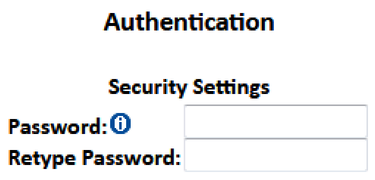
(c) Wireless Parameter



OpenVPN Parameter

OpenVPN	Disabled
Logging	Off
Config ZIP-file:	No file selected.

(d) OpenVPN Parameter

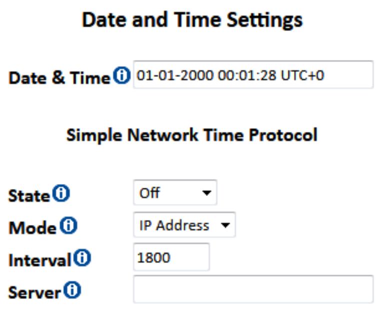


Authentication

Security Settings

Password:	
Retype Password:	

(e) Authentication



Date and Time Settings

Date & Time	01-01-2000 00:01:28 UTC+0

Simple Network Time Protocol

State	Off
Mode	IP Address
Interval	1800
Server	

(f) Date and Time Settings

Figure 8: Server Configuration

1.2.4 Webbrowser Channel Configuration

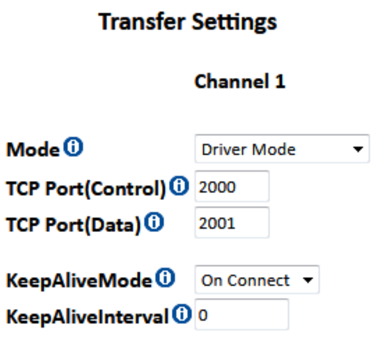
NetCAN Plus can be operated in following modes (see Figure 9):

Driver Mode - Only very few parameters have a function in *Driver Mode* (see Figure 9a). NetCAN$^+$ is operating as a Server. It accepts two connections for the CAN channel. One connection is used to transmit the serial data, this is the TCP Port(Data). And the other is used to transmit control information, TCP Port(Control). This control connection is mostly used to request the status of the virtual serial port. Software may intend to change serial parameters like baudrate or parity, such requests are honored. However they are ignored, because the serial parameters are fixed in the NetCAN$^+$.
The NetCAN$^+$ can check if the connected Client is still alive. This may be done, when a second Client wants to establish a connection (On Connect). It may also be done in regular intervals (Polling). The Driver Mode allows for one Client only.

TCP Raw Server - As TCP Raw Server NetCAN$^+$ operates very simple (see Figure 9b). It only waits for incoming data connections in Raw IP mode. As with the Driver Mode only the data connection is defined, there is no connection for control.
You can connect multiple times to the NetCAN$^+$ also from different machines.

CAN Bridge Server - Configures server side of the CAN over IP Router functionality. In this mode NetCAN$^+$ waits for incoming connections. Max. Clients value defines how many CAN Bridge Clients can connect to the server. CAN Speed sets the appropriate baudrate of the attached CAN network. Acceptance Code and Mask allow CAN frame filtering so only the dedicated frames pass over the TCP link (See Figure 9c).

CAN Bridge Client - Configures client side of the CAN over IP Router functionality. In this mode NetCAN$^+$ connects to the other NetCAN$^+$ in the CAN Bridge Server Mode. This is specified as destination by IP Address and the TCP Port number (e.g. 192.168.1.97:2001). CAN Speed sets the appropriate baudrate of the attached CAN network. Acceptance Code and Mask allow CAN frame filtering so only the dedicated frames pass over the TCP link (See Figure 9d).

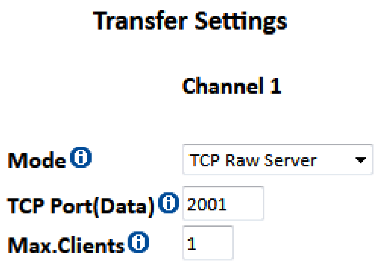


Transfer Settings

Channel 1

Mode	Driver Mode
TCP Port(Control)	2000
TCP Port(Data)	2001
KeepAliveMode	On Connect
KeepAliveInterval	0

(a) Driver Mode

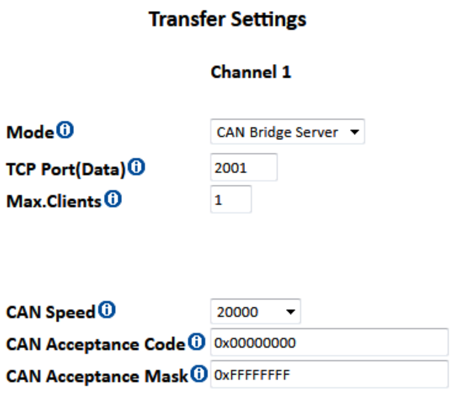


Transfer Settings

Channel 1

Mode	TCP Raw Server
TCP Port(Data)	2001
Max.Clients	1

(b) TCP Raw Server

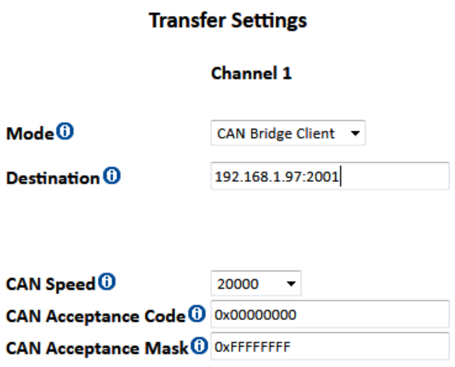


Transfer Settings

Channel 1

Mode	CAN Bridge Server
TCP Port(Data)	2001
Max.Clients	1
CAN Speed	20000
CAN Acceptance Code	0x00000000
CAN Acceptance Mask	0xFFFFFFFF

(c) CAN Bridge Server



Transfer Settings

Channel 1

Mode	CAN Bridge Client
Destination	192.168.1.97:2001
CAN Speed	20000
CAN Acceptance Code	0x00000000
CAN Acceptance Mask	0xFFFFFFFF

(d) CAN Bridge Client

Figure 9: Channel Configuration

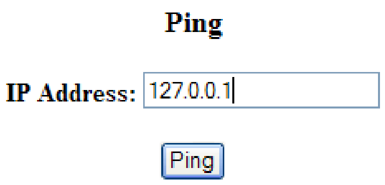
1.2.5 Webbrowser Tools

The available tools are (see Figure 10):

- *The Ping* utility will be used to check if a station is available (see Figure 10a). Enter the IP-Address or the name of a station in the field, and click the *Ping* button. The network connection is checked by sending certain ICMP data packages. If the target responds, the network between the NetCAN$^+$ and the target is operational. The time required for an echo depends on the speed of the network. In a typical Ethernet this is only very few Milliseconds, while it can be several seconds throughout the Internet.
- *The Netstat* utility will be used to monitor TCP connections (see Figure 10b). Use Netstat to see the actual status of NetCAN$^+$ IP Ports. This is a standard tool for network debugging. A *Foreign Address* of 0.0.0.0 is listed when NetCAN$^+$ is waiting for an incoming connection (`LISTEN`). If the value is not 0.0.0.0, the connection is either active (`ESTABLISHED`) or closed (`TIMEWAIT`).
- *The Firmware Update* option is used to update the firmware (see Figure 10c). To upload a new version of the firmware, put the name of the file in the field. Your

Webbrowser may allow to search for the file. Click on the “Update” button. While loading the file is checked. If it is valid, it is stored in the Flash Memory. When the upload is finished, NetCAN$^+$ will Reboot.

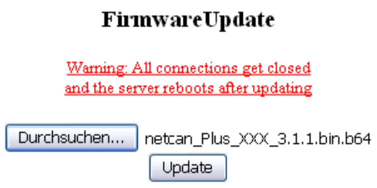
- *The Saving of Configuration to / Loading from a file* option will be used to manage NetCAN$^+$ configuration (see Figure 10d). It is possible to save the current configuration to a text file. Of course it is also possible to load the saved configuration into a NetCAN Plus.
- *The Syslog* option will be used to send logging information to the syslog facility (see Figure 10e). Syslogging requires a server the information is sent to. Facility allows to select the data sent to that server.
- *The DebugLog* option will be used to show logging information via TCP connection (see Figure 10e). For this kind of logging the NetCAN$^+$ behaves as the server. Open a TCP connection to the configured port, and receive all information generated.


Ping

IP Address: 127.0.0.1

(a) Ping

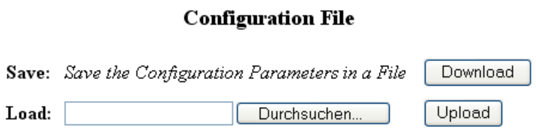
(b) Netstat


FirmwareUpdate

Warning: All connections get closed and the server reboots after updating

netcan_Plus_XXX_3.1.1.bin.b64

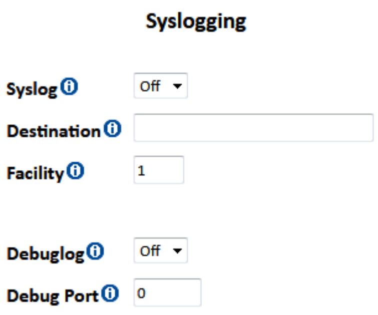
(c) Firmware Update


Configuration File

Save: *Save the Configuration Parameters in a File*

Load:

(d) Configuration File


Syslogging

Syslog: Off

Destination:

Facility: 1

Debuglog: Off

Debug Port: 0

(e) Syslog

Figure 10: Tools

1.2.6 Factory Settings

NetCAN Plus provides DIP-switches to set the configuration back to its defaults. Set the DIP-switches to the „Factory Settings" position (refer to the Table 1) and reboot the device (press reset button or perform a power off/on cycle). As soon as the green LED is on again, the DIP-switches must be returned to the „CAN Operation" position.

Operation Mode	**S1**	**S2**	**S3**	**S4**
CAN Operation	OFF	OFF	OFF	OFF
Factory Settings	OFF	OFF	OFF	ON

Table 1: Switches

1.3 Linux Installation (SocketCAN)

SocketCAN[5] is a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel. As of Linux kernel 2.6.38 ASCII protocol (slcan) will be also supported. To use it you'll first need to install can-utils either from your Linux distribution repository[6] or from project's git[7] repository[8].

For a USB-CAN Plus device, that is attached to `/dev/ttyUSB0` device, execute following commands:

```
slcand -o -s8 -t hw -S 3000000 /dev/ttyUSB0
ip link set up slcan0
```

Parameter `-s` in `slcand` specifies CAN birate as described in section Setup the Bus Timing (Standard). After execution you'll get your USB-CAN Plus operated at 1MBit/s, with `candump` and `cansend` you can make first send/receive tests.

For Network CAN devices (NetCAN Plus) you'll first need to create a symbolic link to it via `socat`. Assume your NetCAN$^+$ has following IP address: 192.168.254.254. So the needed commands are:

```
socat pty,link=/dev/netcan0,raw tcp:192.168.254.254:2001&
slcand -o -s8 /dev/netcan0
ip link set up slcan0
```

Please note, that NetCAN$^+$ must be either in „Driver Mode" or in „TCP Raw Server" mode in order to properly work with slcand.

[5] http://en.wikipedia.org/wiki/Socketcan

[6] `can-utils` package is available in Debian since vesion 8. Debian derivates like Ubuntu should also include this package.

[7] Git

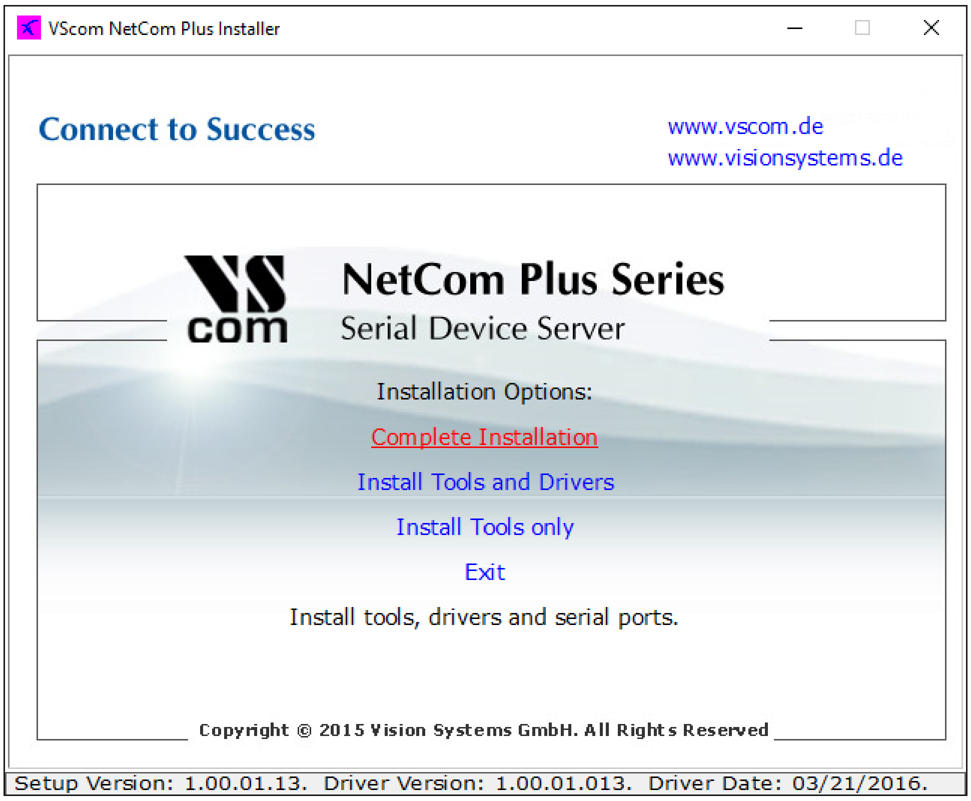
[8] SocketCAN git repository

1.4 Windows Driver Installation

The use of the Driver Mode for NetCAN Plus CAN Bus Gateway is not necessary in most installations. A more easy way of operation is the use of the VSCAN API (section 2 on page 25), which supports TCP Raw Mode. This way of installation is recommended, especially because it will support any future device in the same way.

The only reason for using the Virtual Com Port Driver are applications specifically designed to operate via ASCII Protocol (section 3 on page 42) on Windows Com Ports.

Since the Virtual Com Port Driver is rarely used, it is not covered in detail in this manual. Instead please use the NetCom Plus User Manual to get more information.



Install tools, drivers and serial ports.

Copyright © 2015 Vision Systems GmbH. All Rights Reserved

Setup Version: 1.00.01.13. Driver Version: 1.00.01.013. Driver Date: 03/21/2016.

Figure 11: NetCom Plus Driver Installation

Start the installation program, select „Complete Installation" and follow the instructions. When finished your Windows system has a new Com Port, e.g. named as COM5.

1.5 General Information

1.5.1 LED Status

The red error LED lights up, when a bus error occurs. And the green data LED is on, when a frame was sent to or received from the bus.

1.5.2 Baud-rates and Handshake

Ensure you've set the handshake to RTS/CTS (hardware) when you open the port!

Device	Baud-rate	Handshake
USB-CAN Plus	3 Mbit	RTS/CTS
NetCAN Plus	3 Mbit	RTS/CTS

1.5.3 Pin-out of the 9 Pin D-Sub Connector

Pin	Signal	Description
1	-	
2	CAN_L	CAN_L bus line (dominant level is low)
3	CAN_GND	CAN ground
4	-	reserved
5	-	reserved
6	-	
7	CAN_H	CAN_H bus line (dominant level is high)
8	-	reserved
9	-	

1.5.4 Pin-out of the 4 Pin Connector (USB-CAN Plus mPCIe)

Pin	Signal	Description
1	CAN_H	CAN_H bus line (dominant level is high)
2	CAN_GND	CAN ground
3	CAN_L	CAN_L bus line (dominant level is low)
4	CAN_GND	case ground (connected to Pin 2)

1.5.5 CAN Topology, Wiring and Termination

CAN bus requires three signals (CAN_H, CAN_L and CAN_GND) to be connected between all CAN bus participants. In addition both ends of the bus are to be terminated. Figure 12 on page 22 shows standard termination scheme and Figure 13 on page 22 shows split termination scheme. For more information about these two approaches please refer to this blog article[9].

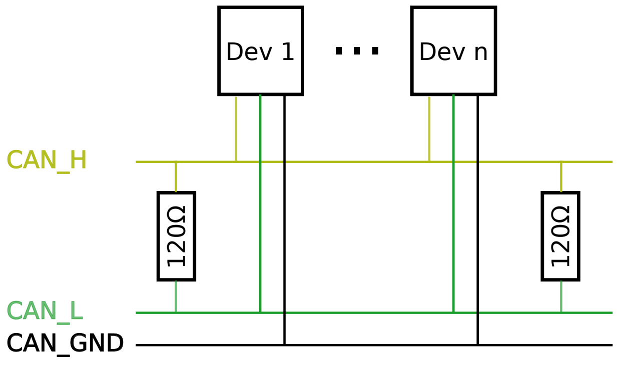


Figure 12: CAN Topology: Standard Termination

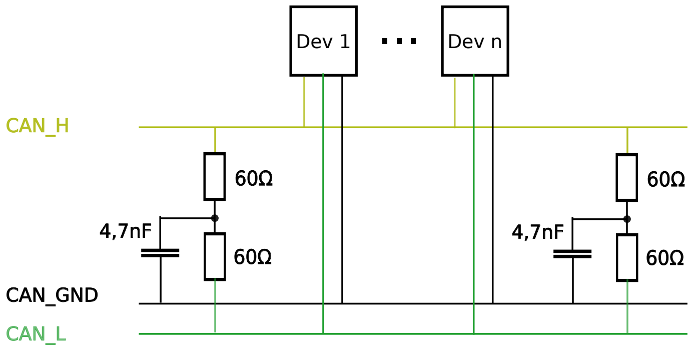


Figure 13: CAN Topology: Split Termination

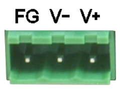
1.5.6 Termination Resistors (USB-CAN Plus Devices)

The USB-CAN Plus has an internal termination resistors inside (120Ω), activated by a jumper J1, but external termination shown in Section CAN Topology, Wiring and Termination is recommended. It's up to you to choose the correct combination and values for your topology.

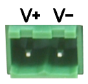
1.5.7 Terminal Block Power

NetCAN Plus Devices The Terminal Block power connector receives positive voltage on the right (V+) pin. The center (V-) pin connector is negative, which is connected to GND and the case. GND is the same as Field GND (FG), so the standard adapter does not connect to this pin.

[9] https://e2e.ti.com/blogs_/b/industrial_strength/archive/2016/07/14/the-importance-of-termination-networks-in-can-transceivers



NetCAN Plus Mini Devices The Terminal Block power connector for the Mini model receives positive voltage on the left (V+) pin. The right (V-) pin receives ground voltage. Connect the case to Protective Earth Rail.



1.6 Products

Figure 14 shows USB-CAN Plus and NetCAN Plus 120 WLAN devices. Also available are USB-CAN Plus mPCIe and the isolated USB-CAN Plus ISO as also NetCAN Plus 110 without Wireless LAN and NetCAN Plus 110 Mini.

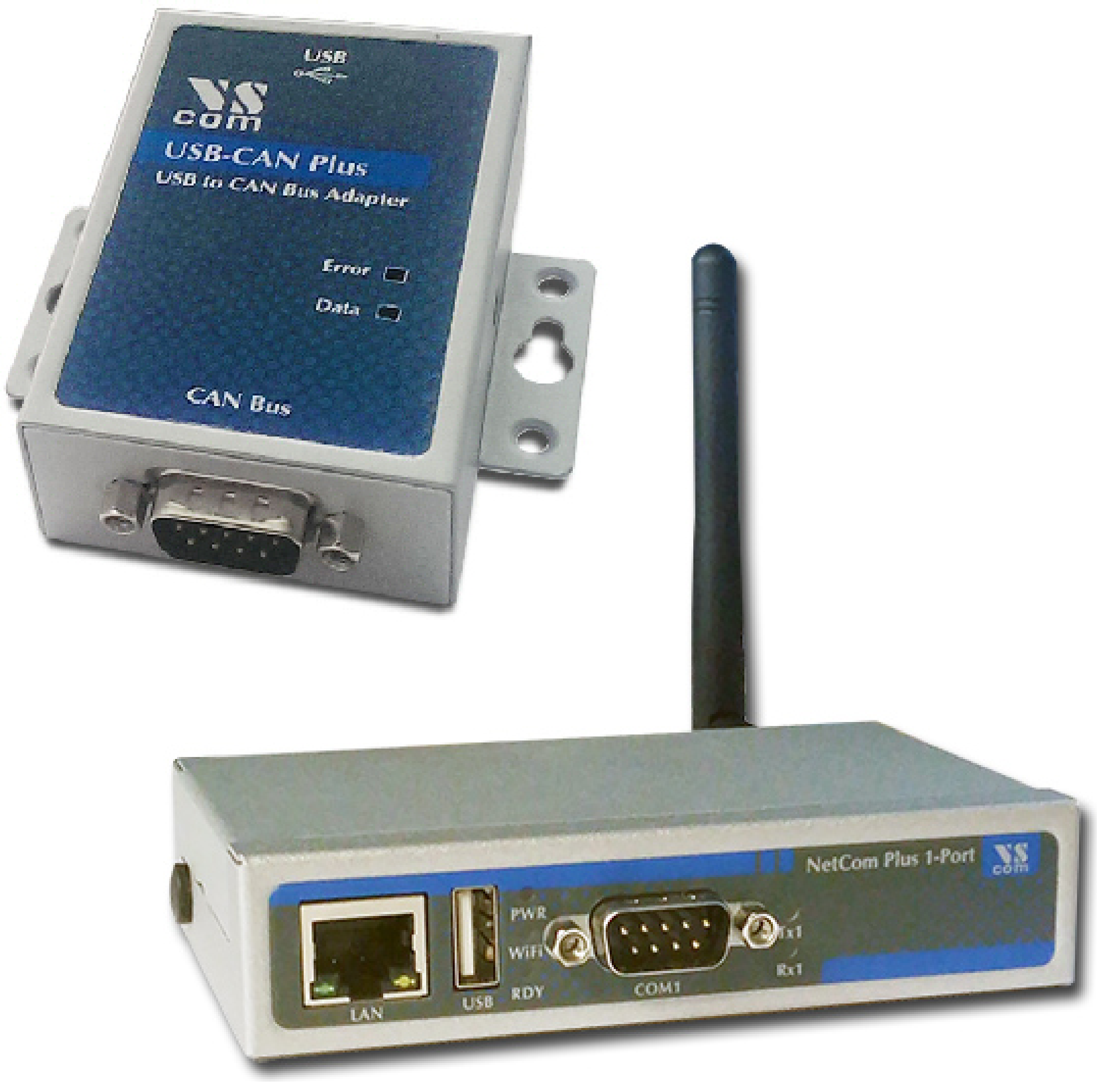


Figure 14: CAN Plus Models

2 Application Programming Interface

2.1 Introduction

The Application Programming Interface (API) gives you the right tools to use all of the functions that the VSCAN devices provide. It will make your life much easier to build your own CAN controlling software with these functions, than to implement your application directly on top of the ASCII protocol. All functions and data structures are explained in the next sub-sections. You'll find programming examples in our repository on GitHub[10].

Windows For Windows, the only thing you must do, is to copy `vs_can_api.dll` (the dynamic link library), `vs_can_api.lib` (the linker input file) and `vs_can_api.h` (the header file) into your project directory. Include the header in your source code and add the `vs_can_api.lib` to your project configuration.

Linux Linux users are advised to use SocketCAN framework described in Section 1.3. If you do want to use the VSCAN API then copy the library (`libvs_can_api.so`) to your global libraries path and add it to your compilation parameters. You must also include the header file (`vs_can_api.h`) in your source file.

Other Operating Systems For other operating systems like MacOS you'll have to implement ASCII protocol yourself. Alternatively you can use Python and `python-can` module. See the next paragraph.

Python `python-can`[11] module provides native support for the ASCII protocol, so VS-CAN devices can be used without vs_can_api library. You'll find programming examples in our repository on GitHub[12].

Programming Language Bindings You can also use our API library with programming languages other than C/C++. To do so you'll have to learn how to import the functions from our library to your application. To ease this task a collection of bindings for following programming languages will be provided together with `VSCAN_API_x_y_z.zip` under Wrapper\Languages:

- C#
- VisualBasic.NET
- Delphi
- more to come

[10] https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/c
[11] `https://python-can.readthedocs.io/en/2.2.0/`
[12] https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/python

2.2 Functions

2.2.1 VSCAN_Open

The `VSCAN_Open` function opens the CAN channel.

```
VSCAN_HANDLE VSCAN_Open(CHAR *SerialNrORComPortORNet, DWORD Mode);
```

Parameters:

`SerialNrORComPortORNet`

[in] A char pointer with one of the following values.

- VSCAN_FIRST_FOUND - the first device found will be opened
- Serial number of the specific device
- COM port or CAN name where the device is located
- IP address and port number of the device

`Mode`

[in] The mode in which the CAN channel shall be opened.

- VSCAN_MODE_NORMAL - the normal operation mode
- VSCAN_MODE_LISTEN_ONLY - the listen only mode, in which no CAN interaction will be done from the controller
- VSCAN_MODE_SELF_RECEPTION - the self reception mode, in which the device receives also the frames that it sends. *The firmware version must be 1.4 or greater and the DLL version 1.6 or greater.*

Examples:

```
// Windows, Linux
handle = VSCAN_Open(VSCAN_FIRST_FOUND, VSCAN_MODE_NORMAL);

// Windows, Linux
handle = VSCAN_Open("123456", VSCAN_MODE_LISTEN_ONLY);

// Windows, WinCE
handle = VSCAN_Open("COM3", VSCAN_MODE_NORMAL);

// Linux
handle = VSCAN_Open("/dev/ttyUSB0", VSCAN_MODE_NORMAL);

// Linux on Alena (CAN on local bus)
handle = VSCAN_Open("/dev/can0", VSCAN_MODE_LISTEN_ONLY);

// Windows, WinCE, Linux
handle = VSCAN_Open("192.168.254.254:2001", VSCAN_MODE_SELF_RECEPTION);
```

2.2.2 VSCAN_Close

The `VSCAN_Close` function will close the CAN channel.

```
VSCAN_STATUS VSCAN_Close(VSCAN_HANDLE Handle);
```

Parameters:

`Handle`

[in] The handle of the CAN device, which shall be closed.

Example:

```
status = VSCAN_Close(handle);
```

2.2.3 VSCAN_Ioctl

You can get and set special information and commands of the CAN device with the `VSCAN_Ioctl` function.

```
VSCAN_STATUS VSCAN_Ioctl(VSCAN_HANDLE Handle, DWORD Ioctl, VOID *Param);
```

Parameters:

`Handle`

[in] The handle of the CAN device, which should be used.

`Ioctl`

[in] Tells the function which of the following ioctl should be called.

`Param`

[in, out] A pointer to the data for the ioctls which are listed below.

VSCAN_IOCTL_SET_DEBUG

You can set the debug verbosity with this ioctl. The higher the debug level the more debug information you get. The `VSCAN_HANDLE` can be `NULL`.

Possible debug levels are:

- VSCAN_DEBUG_NONE *(no debug information)*
- VSCAN_DEBUG_LOW *(errors from the VSCAN API)*
- VSCAN_DEBUG_MID *(information from the VSCAN API)*
- VSCAN_DEBUG_HIGH *(errors from system functions)*

Example:

```
status = VSCAN_Ioctl(NULL, VSCAN_IOCTL_SET_DEBUG, VSCAN_DEBUG_HIGH);
```

VSCAN_IOCTL_SET_DEBUG_MODE

You can set the debug mode with this ioctl. It is possible to log the error to the standard error console output (default value) or to save it in a log file. The log file will be saved in the directory from which your application is running and will be named *„vs_can_api.log"*. The `VSCAN_HANDLE` can be `NULL`.

The debug mode defines are:

- VSCAN_DEBUG_MODE_CONSOLE
- VSCAN_DEBUG_MODE_FILE

Example:

```
status = VSCAN_Ioctl(NULL, VSCAN_IOCTL_SET_DEBUG_MODE, VSCAN_DEBUG_MODE_FILE);
```

VSCAN_IOCTL_GET_API_VERSION

You can request the API version number with this ioctl. Therefore you must commit a pointer of the type VSCAN_API_VERSION to the function. *The DLL version must be 1.6 or greater.*

Example:

```
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_GET_API_VERSION, &version);
```

VSCAN_IOCTL_GET_HWPARAM

This ioctl gives you the possibility to get the hardware parameters (serial number, hardware and software version) of the device. Therefore you must commit a pointer of the type VSCAN_HWPARAM to the function.

Example:

```
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_GET_HWPARAM, &hwparam);
```

VSCAN_IOCTL_SET_SPEED

With this ioctl you can set the speed of your CAN device. The following constant speed values are supported:

- VSCAN_SPEED_1M
- VSCAN_SPEED_800K
- VSCAN_SPEED_500K
- VSCAN_SPEED_250K
- VSCAN_SPEED_125K
- VSCAN_SPEED_100K
- VSCAN_SPEED_50K
- VSCAN_SPEED_20K

With the „NetCAN Plus” you have also the choice to use your desired baudrate directly as the parameter in bit/s.

Example:

```
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_SPEED, VSCAN_SPEED_1M);

// set NET-CAN Plus bitrate to 25kbit/s:
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_SPEED, (void*)25000);
```

VSCAN_IOCTL_SET_BTR

This ioctl gives you the possibility to configure the speed registers manually (bus timing registers). Therefore you must commit a structure from the type VSCAN_BTR. For more information on this registers, please take a look at the SJA1000 datasheet from NXP Semiconductors or you can use an online bittiming calculator[13] .

Example:

```
VSCAN_BTR btr = { .Btr0 = 0x00, .Btr1 = 0x14 }; // for 1Mb/s

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_BTR, &btr);
```

VSCAN_IOCTL_SET_FILTER_MODE

This ioctl let you set the desired filter mode for the acceptance code and mask. You can switch between single and dual filter mode. For more informations, please take a look at the SJA1000 datasheet from NXP Semiconductors. *The firmware version must be 1.4 or greater and the DLL version 1.6 or greater.*

Example:

```
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_FILTER_MODE, VSCAN_FILTER_MODE_DUAL);
```

VSCAN_IOCTL_SET_ACC_CODE_MASK

You can set the acceptance code and acceptance mask register with this ioctl. This gives you the possibility to filter for special frame types you want to receive. Therefore you must commit a structure from the type VSCAN_CODE_MASK. For more information on this specific registers, please take a look at the SJA1000 datasheet from NXP Semiconductors.

Example:

```
VSCAN_CODE_MASK codeMask;

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_FILTER_MODE, VSCAN_FILTER_MODE_DUAL);

// will receive the ids between 0x300 and 0x3ff (dual filter mode set filter 2)
codeMask.Code = 0x6000;
codeMask.Mask = 0x1ff0;

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_ACC_CODE_MASK, &codeMask);

// receive all frames on the CAN bus (default)
codeMask.Code = VSCAN_IOCTL_ACC_CODE_ALL;
codeMask.Mask = VSCAN_IOCTL_ACC_MASK_ALL;

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_ACC_CODE_MASK, &codeMask);
```

[13]On http://www.bittiming.can-wiki.info/ select „NXP SJA1000(Philips) or Intel” from the drop-down list and specify the desired bitrate, click on „Request Table” button and you'll get calculated BTR settings.

VSCAN_IOCTL_SET_FILTER

This ioctl let set you up to 16 advanced filters for „NetCAN Plus". The logic is `<received_can_id> & Mask == Id & Mask`. *The DLL version must be 1.8 or greater.*

Example:

```
// set two filters
VSCAN_FILTER filter[2];
filter[0].Size = 2;
filter[0].Id = 0x543;
filter[0].Mask = 0xfff;
filter[0].Extended = 1;
filter[1].Id = 0x231;
filter[1].Mask = 0x0ff;
filter[1].Extended = 0;

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_FILTER, filter);

// clear all filters
VSCAN_FILTER filter;
filter.Size = 0;

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_FILTER, &filter);
```

VSCAN_IOCTL_GET_FLAGS

To get extended status and error flags use this ioctl. Commit a DWORD(32bit) pointer as the `Param` argument. The bit flags and their equivalent macro names are:

- Bit 0: VSCAN_IOCTL_FLAG_RX_FIFO_FULL
- Bit 1: VSCAN_IOCTL_FLAG_TX_FIFO_FULL
- Bit 2: VSCAN_IOCTL_FLAG_ERR_WARNING
- Bit 3: VSCAN_IOCTL_FLAG_DATA_OVERRUN
- Bit 4: VSCAN_IOCTL_FLAG_UNUSED
- Bit 5: VSCAN_IOCTL_FLAG_ERR_PASSIVE
- Bit 6: VSCAN_IOCTL_FLAG_ARBIT_LOST
- Bit 7: VSCAN_IOCTL_FLAG_BUS_ERROR

Take a look at the SJA1000 datasheet from NXP Semiconductors, if you want more information on what's behind bit 2 to 7.

Example:

```
DWORD flags;

status = VSCAN_Ioctl(handle, VSCAN_IOCTL_GET_FLAGS, &flags);
```

VSCAN_IOCTL_SET_TIMESTAMP

You can set on and off the time-stamp functionality with this ioctl. If you switch it on, every received frame will have a valid time-stamp value in the VSCAN_MSG structure. The time base is in milliseconds and will be overrun after 60 seconds (timestamps between 0-60000ms).

Example:

```
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_TIMESTAMP, VSCAN_TIMESTAMP_ON);
```

VSCAN_IOCTL_SET_BLOCKING_READ

This ioctl will set theVSCAN_Read function to blocking mode (default is unblock).

Example:

```
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_BLOCKING_READ, VSCAN_IOCTL_ON);
```

2.2.4 VSCAN_Read

To read one or more CAN frames from the CAN bus, you must use the `VSCAN_Read` function. The read mode of this function is set to non-blocking mode per default. This means that `VSCAN_Read` will return immediately - even when there are no frames at the moment. Use the ioctl VSCAN_IOCTL_SET_BLOCKING_READ to make make the `VSCAN_Read` blocking - then it will return only when frames were received.

```
VSCAN_STATUS VSCAN_Read(VSCAN_HANDLE Handle, VSCAN_MSG *Buf, DWORD Size, DWORD *Read);
```

Parameters:

`Handle`

[in] The handle of the CAN device, which should be used.

`Buf`

[in] A pointer to one element or an array of the structure VSCAN_MSG.

`Size`

[in] The number of the array elements in `Buf`.

`*Read`

[out] A pointer to a DWORD that will receive the real number of the frames read.

Example:

```
VSCAN_MSG msgs[10];
DWORD read;

status = VSCAN_Read(handle, msgs, 10, &read);
```

2.2.5 VSCAN_SetRcvEvent

With the `VSCAN_SetRcvEvent` function you can set an event which will be set when a frame arrives. There are different versions for Windows and Linux. *The DLL version must be 1.6 or greater.*

```
// Windows Prototype:
VSCAN_STATUS VSCAN_SetRcvEvent(VSCAN_HANDLE Handle, HANDLE Event);
// Linux Prototype:
VSCAN_STATUS VSCAN_SetRcvEvent(VSCAN_HANDLE Handle, sem_t *Event);
```

Parameters:

`Handle`

[in] The handle of the CAN device, which should be used.

`Event`

[in] In Windows an event handle of the type `HANDLE` and in Linux a pointer to the `sem_t` union.

Windows Example:

```
// don't forget your own error handling for the API and system functions
// for further informations on these functions take a look at the MSDN

HANDLE hEvent;
DWORD dwRetCode;

hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

VSCAN_SetRcvEvent(handle, hEvent);

dwRetCode = WaitForSingleObject(hEvent, INFINITE);

switch(dwRetCode)
{
    case WAIT_OBJECT_O :
        // a CAN frame arrived
        break;
    default:
        // probe for error
}

CloseHandle(hEvent);
```

Linux Example:

```
// don't forget your own error handling for the API and system functions
// take also a look at sem_trywait, sem_timedwait and the rest of the sem_* functions

sem_t sem;
int retCode;

retCode = sem_init(&sem, 0, 0);

VSCAN_SetRcvEvent(handle, &sem);

retCode = sem_wait(&sem);

// a CAN frame arrived

retCode = sem_destroy(&sem);
```

2.2.6 VSCAN_Write

With the `VSCAN_Write` function you can write one or more frames to the CAN bus. The frames will be buffered and send out after some time - this time can grow up to one time slice of the scheduler (Windows = ˜16ms and Linux = ˜10ms). If you want to send the frames immediately, you must call the functionVSCAN_Flush.

```
VSCAN_STATUS VSCAN_Write(VSCAN_HANDLE Handle,
                         VSCAN_MSG *Buf, DWORD Size, DWORD *Written);
```

Parameters:

`Handle`

[in] The handle of the CAN device, which should be used.

`Buf`

[in] A pointer to one element or an array of the structure VSCAN_MSG.

`Size`

[in] The number of the array elements in `Buf.`

`*Written`

[out] A pointer to a DWORD that will receive the number of frames written.

Example:

```
VSCAN_MSG msgs[10];
DWORD written;

msgs[0].Flags = VSCAN_FLAGS_EXTENDED;
msgs[0].Id = 100;
msgs[0].Size = 1;
msgs[0].Data[0] = 0x1B;

// we will send ten frames with the same data
// to the ids 100-109
for (i = 1; i < 10; i++)
{
    memcpy(msgs + i, &msgs[0], sizeof(msgs[0]));
    msgs[i].Id++;
}

status = VSCAN_Write(handle, msgs, 10, &written);
```

2.2.7 VSCAN_Flush

The `VSCAN_Flush` function will send all frames immediately out to the CAN bus.

```
VSCAN_STATUS VSCAN_Flush(VSCAN_HANDLE Handle);
```

Parameters:

`Handle`

[in] The handle of the CAN device, whose data should be flushed.

Example:

```
status = VSCAN_Flush(handle);
```

2.2.8 VSCAN_GetErrorString

The `VSCAN_GetErrorString` function retrieves the associated human readable error string.

```
VOID VSCAN_GetErrorString(VSCAN_STATUS Status, CHAR *String, DWORD MaxLen);
```

Parameters:

`Status`

[in] The status for which the error string should be retrieved.

`String`

[out] A pointer of a string array which will receive the error string.

`MaxLen`

[in] The maximum possible length of the error string (without the terminating zero).

Example:

```
VSCAN_STATUS status = VSCAN_ERR_NO_DEVICE_FOUND;
char string[33];

VSCAN_GetErrorString(status, string, 32);

printf(string);
```

2.3 Types and Structures

2.3.1 VSCAN_HANDLE

```
typedef int VSCAN_HANDLE;
```

This type definition holds the handle of an opened CAN channel. In this case the value is greater than zero. Otherwise the value is equal to one of the type definition VSCAN_STATUS.

2.3.2 VSCAN_STATUS

```
typedef int VSCAN_STATUS;
```

The type definition `VSCAN_STATUS` can have one of the following status value.

- **VSCAN_ERR_OK** - indicates that everything is okay
- **VSCAN_ERR_ERR** - indicates a general error
- **VSCAN_ERR_NO_DEVICE_FOUND** - indicates that no CAN device was found with the specific functions
- **VSCAN_ERR_SUBAPI** - indicates that an error occurred in a subordinated library
- **VSCAN_ERR_NOT_ENOUGH_MEMORY** - indicates that there is not enough memory to complete the function
- **VSCAN_ERR_NO_ELEMENT_FOUND** - indicates that there is no requested element available (e.g. from an input buffer)
- **VSCAN_ERR_INVALID_HANDLE** - indicates that the handle which is used is not valid (e.g. CAN channel closed)
- **VSCAN_ERR_IOCTL** - indicates that an ioctl request failed; ensure that you've used the right parameter values
- **VSCAN_ERR_MUTEX** - indicates that there was a problem with a used mutex in the VSCAN API (e.g. timeout)
- **VSCAN_ERR_CMD** - indicates that there was a problem to complete a given command on the CAN device

2.3.3 VSCAN_API_VERSION

This structure holds the version information of the API.

```
typedef struct
{
    UINT8  Major;
    UINT8  Minor;
    UINT8  SubMinor;
} VSCAN_API_VERSION;
```

2.3.4 VSCAN_HWPARAM

This structure holds the values of the hardware parameters.

```
typedef struct
{
    UINT32 SerialNr;
    UINT8  HwVersion;
    UINT8  SwVersion;
    UINT8  HwType;
} VSCAN_HWPARAM;
```

The `SerialNr` element comprised the unique serial number reserved for this device. The `HwVersion` holds the revision of the CAN hardware and in the opposite `SwVersion` the actual software version of the firmware. The upper four bits of these variables hold the major and the lower four the minor number. And `HwType` retrieves the type of CAN hardware (e.g. Serial, USB, Net).

2.3.5 VSCAN_MSG

The structure is used for the information of each CAN frame which will be received or transmitted.

```
typedef struct
{
    UINT32 Id;
    UINT8  Size;
    UINT8  Data[8];
    UINT8  Flags;
    UINT16 Timestamp;
} VSCAN_MSG;
```

The element `Id` holds the identifier of the standard or extended CAN frame. The width of the data bytes is saved in the `Size` element and the maximum eight data bytes itself in `Data`. The member `Flags` is a bit-mask to retrieve or set some of these special flags: `VSCAN_FLAGS_STANDARD` - is set when this message is a standard frame, `VSCAN_FLAGS_EXTENDED` - this bit is set in the case of an extended frame and the `VSCAN_FLAGS_REMOTE` bit could be set, when it was or should be a remote request frame. The `Timestamp` element holds the time-stamp of the received frame, when this special function is activated over the ioctl VSCAN_IOCTL_SET_TIMESTAMP. If a frame was received with a time-stamp, also the flag `VSCAN_FLAGS_TIMESTAMP` is set in the member `Flags`.

2.3.6 VSCAN_BTR

This structure is used for the setting of the bus timing register.

```
typedef struct
{
    UINT8 Btr0;
    UINT8 Btr1;
} VSCAN_BTR;
```

The elements `Btr0` and `Btr1` implements the values for the bus timing register one and two. For more information read the chapter 2.2.3 or take a look at the SJA1000 datasheet from NXP Semiconductors.

2.3.7 VSCAN_CODE_MASK

The structure stores the acceptance filter code and filter mask.

```
typedef struct
{
    UINT32 Code;
    UINT32 Mask;
} VSCAN_CODE_MASK;
```

The structure member Code stores the acceptance code and Mask the acceptance mask. For more information see chapter 2.2.3 or take a look at the SJA1000 datasheet from NXP Semiconductors.

3 ASCII Command Set

3.1 Introduction

The ASCII command set gives you the possibility to use the VSCAN device even with a simple terminal program. This makes it very easy for you, to send some frames by hand or to sniff the frames on the CAN bus in a simple human readable view. It will also be possible to use such a simple semantic in a scripting system (e.g. Linux bash-script).

Every binary data will be sent and received by their ASCII hexadecimal equivalents. The return values of all functions will be `CR (ASCII 13)` if the function succeeds or `BELL (ASCII 7)` if the function fails. Some functions have extended return values, but this will be described per function in the command description.

The received frames will be send directly to your ASCII communication channel - e.g. serial port or network connection.

3.2 Commands

3.2.1 Open the CAN Channel

The CAN channel will be opened with the command `O[CR]`, `L[CR]` or `Y[CR]`. The difference between these three types is, that the second command will open the channel in a listen only mode, in which no bus interaction will be done from the controller. The last command will open the channel in a self reception mode, in which the device will also receive the frames that it sends *(only available in firmware version 1.4 or greater)*. Before you will use one of the commands, you should setup a bus timing with the command `S` or `s`. *Anyway, the last configured bit rate is stored in the device and used as the standard bus timing at power up.*

Examples:

Open the channel in normal operation mode.

```
O[CR]
```

Open the channel in the listen only mode.

```
L[CR]
```

Open the channel in the self reception mode.

```
Y[CR]
```

3.2.2 Close the CAN Channel

The CAN channel will be closed with the command `C[CR]`. The command is only active if the CAN channel is open.

Example:

```
C[CR]
```

3.2.3 Setup the Bus Timing (Standard)

The bus timing will be setup-ed with the command `Sn[CR]`. With the „NetCAN Plus" it is also possible to define the bitrate directly (e.g. `S125000[CR]`). You can only use this command if the CAN channel is closed.

Parameters:

`n`

Could be one of the following values:

- 1 - 20 KBit
- 2 - 50 KBit
- 3 - 100 KBit
- 4 - 125 KBit
- 5 - 250 KBit
- 6 - 500 KBit
- 7 - 800 KBit
- 8 - 1 MBit

Example:

Configure a bus timing of 1 MBit.

```
S8[CR]
```

3.2.4 Setup the Bus Timing (Advanced)

A more user defined bus timing could be configured with the command `sxxyy[CR]`. This command is not available for the „NetCAN Plus" series. As with the standard bus timing command above, you can only use it when the CAN channel is closed.

Parameters:

`xx`

This is the hex value of the bit timing register 0.
For more information please take a look at the SJA1000 datasheet from NXP Semiconductors.

`yy`

This is the hex value of the bit timing register 1.

Example:

Configure a bus timing of 100 KBit.

```
s041C[CR]
```

3.2.5 Transmitting a Standard Frame

Transmitting a standard frame (11bit) over the CAN bus will be done with command `tiiildd[0-8]`. The return value will be `z[CR]` or the normal error byte (`BELL`). As you can imagine, this command is only available when the CAN channel is open.

Parameters:

`iii`

Standard frame (11bit) identifier.

`l`

Data length (0-8)

`dd[0-8]`

Data bytes in hex. The number of the bytes must be equal with the data length field.

Example:

Sending a frame with id 0x111 and three data bytes 0x10, 0x20, 0x30.

```
t1113102030[CR]
```

3.2.6 Transmitting a Standard Remote Request Frame

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

Parameters:

`iii`

Standard frame (11bit) identifier.

`l`

Data length (0-8)

Example:

Sending a remote request frame with id 0x111 and request 3 data bytes.

```
r1113[CR]
```

3.2.7 Transmitting an Extended Frame

Transmitting an extended frame (29bit) over the CAN bus will be done with command `Tiiiiiiiildd[0-8]`. The return value will be `Z[CR]` or the normal error byte (`BELL`). The command is only available when the CAN channel is open.

Parameters:

`iiiiiiii`

Extended frame (29bit) identifier.

`l`

Data length (0-8)

`dd[0-8]`

Data bytes in hex. The number of the bytes must be equal with the data length field.

Example:

Sending an extended frame with id 0x111 and three data bytes 0x10, 0x20, 0x30.

```
T000001113102030[CR]
```

3.2.8 Transmitting an Extended Remote Request Frame

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

Parameters:

`iiiiiiii`

Extended frame (29bit) identifier.

`l`

Data length (0-8)

Example:

Sending an extended remote request frame with id 0x111 and a request for 3 data bytes.

```
R000001113[CR]
```

3.2.9 Set Time-Stamps

The time-stamp command will set the time-stamp functionality on received frames on or off. This command will only function, when the CAN channel is closed.

Example:

Will set time-stamps on or off.

```
Z1[CR]
Z0[CR]
```

3.2.10 Set Filter Mode

The command `D1[CR]` switch on the dual filter mode and with `D0[CR]` you switch over to the single filter mode. For more information please take a look at chapter 2.2.3. This command is only available if the CAN channel is closed. *The firmware version must be 1.4 or greater.*

Example:

Will set dual or single filter mode.

```
D1[CR]
D0[CR]
```

3.2.11 Set Acceptance Code and Mask Register

With the acceptance code command `Mxxxxxxxx[CR]` and mask register command `mxxxxxxxx[CR]`, you have the choice to filter for specific CAN messages directly on the CAN controller side. For example the acceptance code addresses in SJA1000 format are `MC0C1C2C3`, same for the mask addresses. If a mask bit is set, then this element from the code bit will be ignored. For more information please take a look at chapter 2.2.3. For the „NetCAN Plus" you could only set the bits for the id part and not from the data. This command is only available if the CAN channel is closed.

Example:

We will filter for all standard frames between 0x300 and 0x3ff (dual filter mode set filter 2).

```
M00006000[CR]
m00001ff0[CR]
```

3.2.12 Set Advanced Filter

The advanced filter command is only available for „NetCAN Plus". Set the filter with `fxxxxxxxx,yyyyyyyy,z[CR]`, where `x` is the id, `y` the mask and `z` for optional extended ('e') or standard frames ('s'). This equals to <received_can_id> & mask == id & mask. You can set up to 16 individual filters when calling the command multiple times, or deleting the filter set with id and mask set to zero.

First we set up filter for the id 123 and 234 and then deleting all filters.

```
f123,fff[CR]
f234,fff,e[CR]
f0,0[CR]
```

3.2.13 Get Status Flags

To get the status bits when an error occurred, you must use the command `F[CR]`. For more information on the bit-mask please take a look at chapter 2.2.3. The command is only available if the CAN channel is open.

Example:

Retrieve the status bits as a hexadecimal value. The return value will be formatted like this: `Fxx[CR]`

```
F[CR]
```

3.2.14 Get Version Information

To retrieve the current hard- and software version of the device, you must use the command `V[CR]`. The command is always available and will return the versions formatted like this: `Vxxyy[CR]`. The hardware version is coded in the `xx` (major and minor version) and the software version in the `yy` (also coded as major and minor).

Example:

Retrieving the versions.

```
V[CR]
```

3.2.15 Get Serial Number

With the command `N[CR]` you will retrieve the serial number of the device. This command is always active and will return the decimal serial number like this: `N12345678[CR]`.

Example:

Retrieving the serial number.

```
N[CR]
```

3.2.16 Get Extra-Information

You can retrieve extra information with the command `I[CR]`. The command is always available and will return the values of the bus timing registers, the acceptance code and mask register values, the counter of the arbitration lost interrupt, the arbitration lost capture register, the status register and the value of the error code capture register of the CAN chip. For more information please take a look at the SJA1000 datasheet from NXP Semiconductors.

Example:

Retrieving the extra-information.

```
I[CR]
```

4 Tools

4.1 Firmware-Update

For USB-CAN Plus devices you can use our tool vs_fw_update.exe. It will determine the type of device and will use the correct baudrate. If it did not, you can set the baudrate explicitly.

```
vs_fw_update.exe COMx vs_can_1_7.bin 3000000
```

For the NetCAN Plus you should use the base64 web-frontend update file (e.g. netcan_Plus_XXX_3.1.1.bin.b64), to update CAN bus and network operation firmware in the device.

4.2 Busmaster

BUSMASTER[14] is an Open Source Software utility to test, simulate and analyze data bus systems like CAN. Following usage example was tested on BUSMASTER 3.2.2 (use the download link from our product page).

We will setup USB-CAN Plus device assigned to COM143 and set CAN bitrate to 50kbit/s.

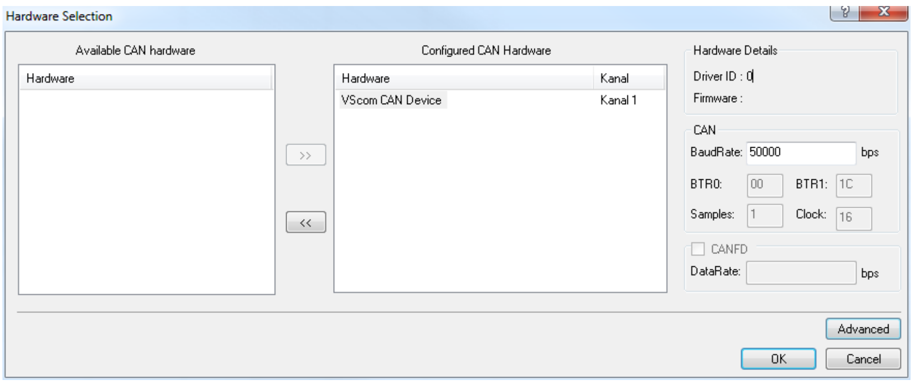
1. Start BUSMASTER
2. Select VScom CAN API: CAN->Driver Selection->VScom CAN-API
3. In the „Hardware Selection” dialog move „VScom CAN Device” from „Available CAN Hardware” to „Configured CAN Hardware” (see Figure 15)
4. Click „Advanced” button to configure USB-CAN Plus device (see Figure 16). Setup COM143 as Serial Port, set Baudrate to 50kbit and Acceptance Mask to 0xFFFFFFFF
5. Once in the main window click on the green „Connect/Disconnect” button. If the connection was successfully established, this button would change to red with a caption „Disconnect”
6. If there are CAN frames on your CAN bus, you'll see them in the „Message Window” (see Figure 17)

To send a CAN message perform the above actions on order to establish a connection and select CAN->Transmit Window (see Figure 18). To create a new CAN frame double-click on „Add Message” in the „Tx Frame List” and enter message ID in hex (in this example it is 0x102) and press „Enter”. New message is created. Perform single-click on the „Add Message” and then select previously created message and „Send Message”

[14] http://www.vscom.de/download/multiio/winXP/tools/BUSMASTER_Installer_Ver_3.2.2_VScom_1.0.exe

button will be activated. Clicking on „Send Message” button would send the selected frame.

You can now change frame payload in the „Data Byte View (HEX)” window below. DLC, Message Type etc. can be modified in the „Tx Frame List”.



Hardware Selection

Available CAN hardware

Hardware

Configured CAN Hardware

Hardware	Kanal
VScom CAN Device	Kanal 1

Hardware Details

Driver ID : 0

Firmware :

CAN

BaudRate: 50000 bps

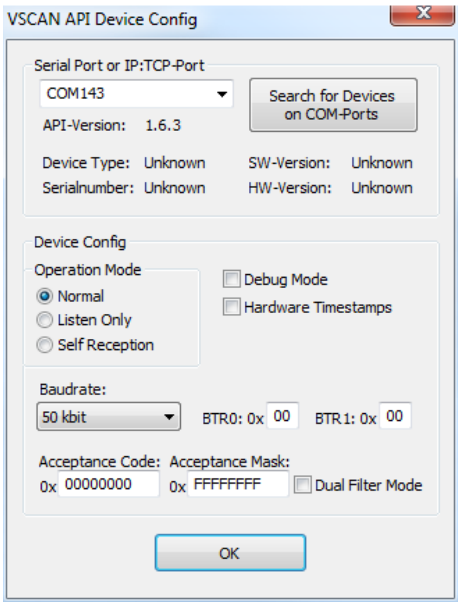
BTR0: 00 BTR1: 1C

Samples: 1 Clock: 16

CANFD

DataRate: bps

Figure 15: BUSMASTER: Select Hardware Interface



VSCAN API Device Config

Serial Port or IP:TCP-Port

COM143

API-Version: 1.6.3

Device Type: Unknown SW-Version: Unknown

Serialnumber: Unknown HW-Version: Unknown

Device Config

Operation Mode

[✓ Normal]

Listen Only

Self Reception

Debug Mode

Hardware Timestamps

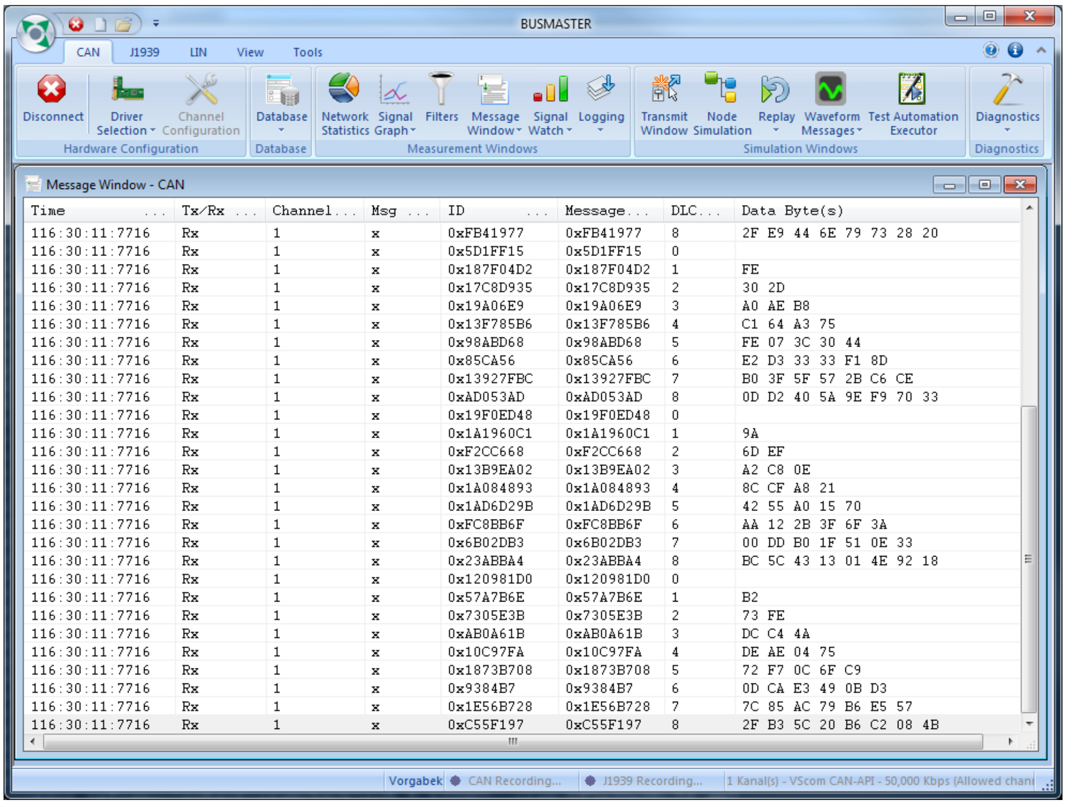
Baudrate: 50 kbit BTR0: 0x 00 BTR1: 0x 00

Acceptance Code: 0x 00000000

Acceptance Mask: 0x FFFFFFFF

Dual Filter Mode

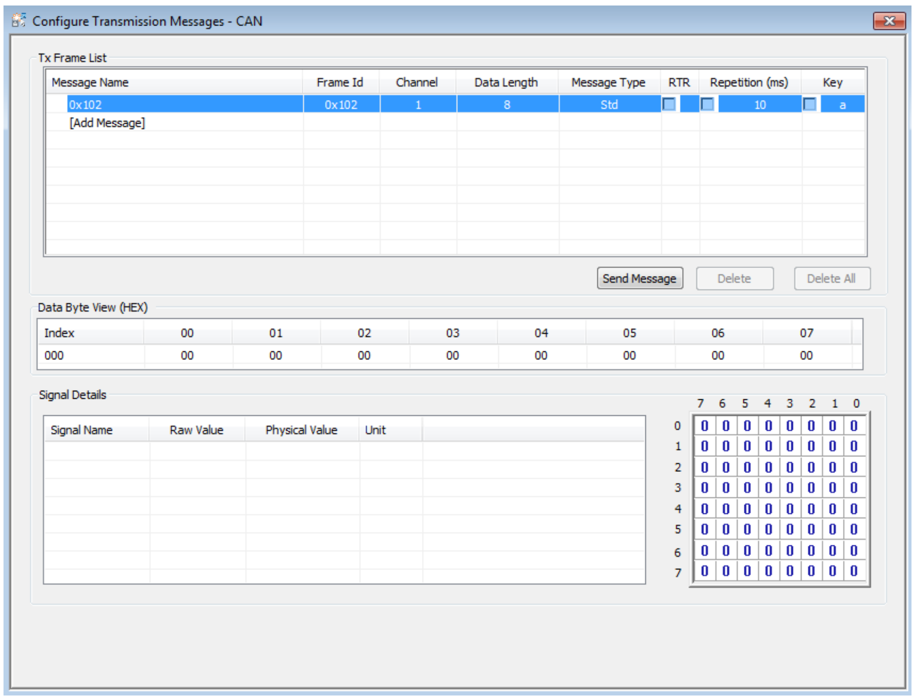
Figure 16: BUSMASTER: Configure Device



Time	Tx/Rx	Channel	Msg	ID	Message	DLC	Data Byte(s)
116:30:11:7716	Rx	1	x	0xFB41977	0xFB41977	8	2F E9 44 6E 79 73 28 20
116:30:11:7716	Rx	1	x	0x5D1FF15	0x5D1FF15	0	
116:30:11:7716	Rx	1	x	0x187F04D2	0x187F04D2	1	FE
116:30:11:7716	Rx	1	x	0x17C8D935	0x17C8D935	2	30 2D
116:30:11:7716	Rx	1	x	0x19A06E9	0x19A06E9	3	A0 AE B8
116:30:11:7716	Rx	1	x	0x13F785B6	0x13F785B6	4	C1 64 A3 75
116:30:11:7716	Rx	1	x	0x98ABD68	0x98ABD68	5	FE 07 3C 30 44
116:30:11:7716	Rx	1	x	0x85CA56	0x85CA56	6	E2 D3 33 33 F1 8D
116:30:11:7716	Rx	1	x	0x13927FBC	0x13927FBC	7	B0 3F 5F 57 2B C6 CE
116:30:11:7716	Rx	1	x	0xAD053AD	0xAD053AD	8	0D D2 40 5A 9E F9 70 33
116:30:11:7716	Rx	1	x	0x19F0ED48	0x19F0ED48	0	
116:30:11:7716	Rx	1	x	0x1A1960C1	0x1A1960C1	1	9A
116:30:11:7716	Rx	1	x	0xF2CC668	0xF2CC668	2	6D EF
116:30:11:7716	Rx	1	x	0x13B9EA02	0x13B9EA02	3	A2 C8 0E
116:30:11:7716	Rx	1	x	0x1A084893	0x1A084893	4	8C CF A8 21
116:30:11:7716	Rx	1	x	0x1AD6D29B	0x1AD6D29B	5	42 55 A0 15 70
116:30:11:7716	Rx	1	x	0xFC8BB6F	0xFC8BB6F	6	AA 12 2B 3F 6F 3A
116:30:11:7716	Rx	1	x	0x6B02DB3	0x6B02DB3	7	00 DD B0 1F 51 0E 33
116:30:11:7716	Rx	1	x	0x23ABBA4	0x23ABBA4	8	BC 5C 43 13 01 4E 92 18
116:30:11:7716	Rx	1	x	0x120981D0	0x120981D0	0	
116:30:11:7716	Rx	1	x	0x57A7B6E	0x57A7B6E	1	B2
116:30:11:7716	Rx	1	x	0x7305E3B	0x7305E3B	2	73 FE
116:30:11:7716	Rx	1	x	0xAB0A61B	0xAB0A61B	3	DC C4 4A
116:30:11:7716	Rx	1	x	0x10C97FA	0x10C97FA	4	DE AE 04 75
116:30:11:7716	Rx	1	x	0x1873B708	0x1873B708	5	72 F7 0C 6F C9
116:30:11:7716	Rx	1	x	0x9384B7	0x9384B7	6	0D CA E3 49 0B D3
116:30:11:7716	Rx	1	x	0x1E56B728	0x1E56B728	7	7C 85 AC 79 B6 E5 57
116:30:11:7716	Rx	1	x	0xC55F197	0xC55F197	8	2F B3 5C 20 B6 C2 08 4B

Vorgabek | CAN Recording... | J1939 Recording... | 1 Kanal(s) - VScom CAN-API - 50,000 Kbps (Allowed chan

Figure 17: BUSMASTER: Received Frames



Tx Frame List

Message Name	Frame Id	Channel	Data Length	Message Type	RTR	Repetition (ms)	Key
0x102	0x102	1	8	Std	☐	☐ 10	☐ a
[Add Message]							

Data Byte View (HEX)

Index	00	01	02	03	04	05	06	07
000	00	00	00	00	00	00	00	00

Signal Details

Signal Name	Raw Value	Physical Value	Unit

	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Figure 18: BUSMASTER: Transmit Window

4.3 vscandump and vscansend

`vs_can_tools.zip`[15] package provides following tools for sending/receiving CAN frames on the command prompt:

- `vscandump.exe` - connects to either USB-CAN Plus or NetCAN Plus and shows all received frames
- `vscansend.exe` - connects to either USB-CAN Plus or NetCAN Plus and sends specified CAN frame

Please note theat you can use only one program with one CAN adapter at a time. Provided you have two or more CAN adapters, you can then use `vscandump.exe` on one of them and `vscansend.exe` on another.

vscandump.exe has following parameters:

- COM port or IP address and TCP port number
- CAN bitrate in ASCII notation, i.e. S1, S2 etc. (see Section Setup the Bus Timing (Standard))

In this example `vscandump.exe` will connect to a NetCAN Plus with IP address 192.168.254.254 and default TCP port 2001 at 50Kbit/s. You can see two different frames. The first one with ID 0x111 is a standard frame and the extended one with ID 0x00000110. Both have a payload of 4 bytes.

```
c:\>vscandump.exe 192.168.254.254:2001 S2
VSCAN-API Version 1.8.0
111    [4]    00 11 22 33
00000110    [4]    00 11 22 33
```

vscansend.exe has following parameters:

- COM port or IP address and TCP port number
- CAN bitrate in ASCII notation, i.e. S1, S2 etc. (see Section Setup the Bus Timing (Standard))
- CAN frame to send

CAN frame syntax looks like this:

- CAN ID in hex: either 3 digits for the standard ID or 8 digits for the extended ID
- CAN ID delimiter '#'
- CAN payload in hex. You can specify between 0 and 8 bytes

[15] http://www.vscom.de/download/multiio/winXP/tools/vs_can_tools.zip

The first example sends a standard frame via USB-CAN Plus device, the second example sends an extended frame:

```
c:\>vscansend.exe COM143 S2 100#00aadd
```

```
c:\>vscansend.exe COM143 S2 00000102#00aadd44
```

4.4 vs_can_test_simple.exe

In addition to `vscansend.exe` and `vscandump.exe` `vs_can_tools.zip` also provides a self-diagnostic tool `vs_can_test_simple.exe`. You can use it either with two VScom CAN devices or with only one device. In both cases you'll need a properly terminated CAN cable.

Once invoked this utility sends a CAN frame on the first given CAN channel and receives it on the other channel. After this CAN channels will be swapped and a send/receive cycle will be repeated.

`vs_can_test_simple.exe` has following parameters:

- first CAN channel: COM port or IP address and TCP port number
- second CAN channel: COM port or IP address and TCP port number
- loop flag (`-l`) is an optional parameter, that repeats the test endlessly

The first example shows a test with two USB-CAN Plus devices:

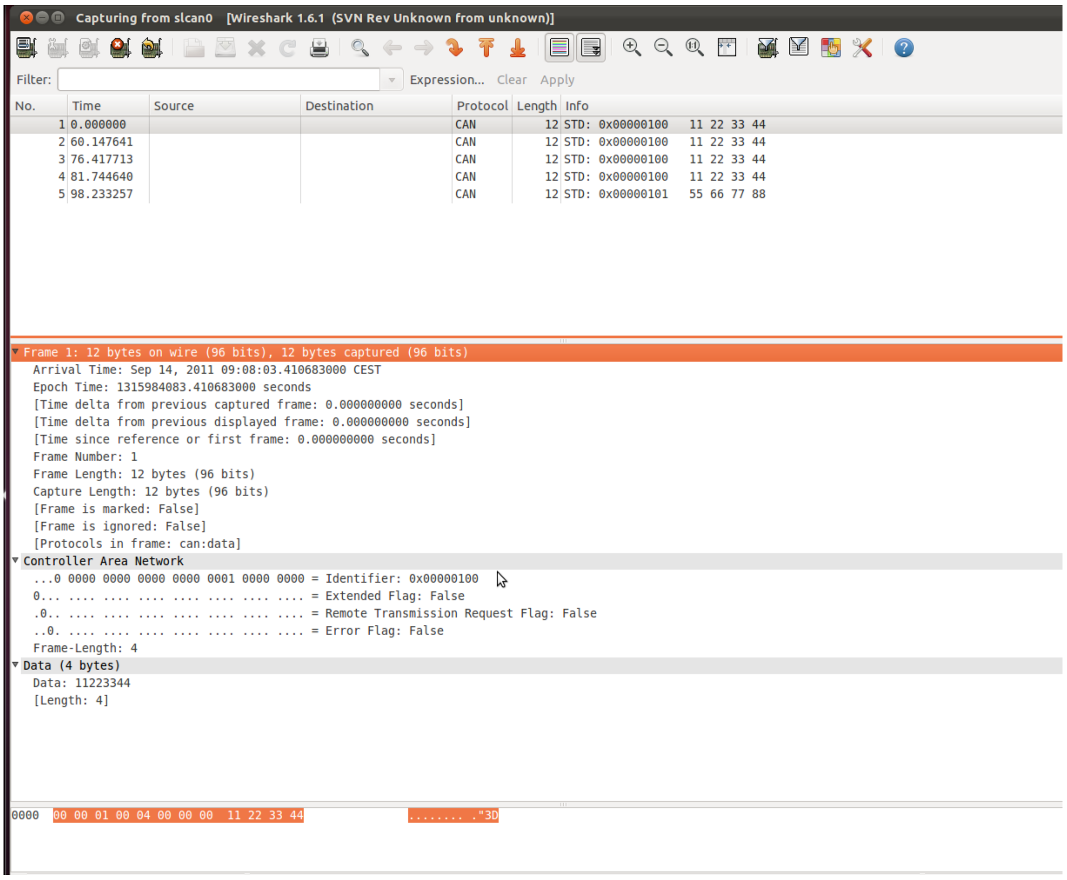
```
c:\>vs_can_test_simple.exe COM168 COM32
vs_can_test_simple.exe v1.6

DLL version: 1.8.0

SUCCESS: Sending frame from COM168 to COM32 succeeded.
SUCCESS: Sending frame from COM32 to COM168 succeeded.
```

The second example shows a test with only one USB-CAN Plus device. In this case the same COM port is given twice:

```
c:\>vs_can_test_simple.exe COM168 COM168
vs_can_test_simple.exe v1.6

DLL version: 1.8.0

SUCCESS: Sending frame from COM168 to COM168 succeeded.
SUCCESS: Sending frame from COM168 to COM168 succeeded.
```

4.5 Wireshark

Wireshark[16] is a well-known network packet sniffer. It is now possible to sniff via Socket-CAN interface (Linux only) and parse CAN frames (Linux/Windows) . Figure 19 shows frames captures via USB-CAN Plus device (`/dev/ttyUSB0` attached as `slcan0`).


Capturing from slcan0 [Wireshark 1.6.1 (SVN Rev Unknown from unknown)]
Filter:
Expression... Clear Apply

```
No.  Time       Source  Destination  Protocol Length Info
  1  0.000000                        CAN          12 STD: 0x00000100   11 22 33 44
  2  60.147641                       CAN          12 STD: 0x00000100   11 22 33 44
  3  76.417713                       CAN          12 STD: 0x00000100   11 22 33 44
  4  81.744640                       CAN          12 STD: 0x00000100   11 22 33 44
  5  98.233257                       CAN          12 STD: 0x00000101   55 66 77 88

▼ Frame 1: 12 bytes on wire (96 bits), 12 bytes captured (96 bits)
    Arrival Time: Sep 14, 2011 09:08:03.410683000 CEST
    Epoch Time: 1315984083.410683000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 12 bytes (96 bits)
    Capture Length: 12 bytes (96 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: can:data]
▼ Controller Area Network
    ...0 0000 0000 0000 0000 0001 0000 0000 = Identifier: 0x00000100
    0... .... .... .... .... .... .... .... = Extended Flag: False
    .0.. .... .... .... .... .... .... .... = Remote Transmission Request Flag: False
    ..0. .... .... .... .... .... .... .... = Error Flag: False
    Frame-Length: 4
▼ Data (4 bytes)
    Data: 11223344
    [Length: 4]

0000  00 00 01 00 04 00 00 00  11 22 33 44             ........ ."3D
```

Figure 19: Wireshark

16 www.wireshark.org

4.6 CANopen

4.6.1 Introduction

CANopen is a CAN-based higher layer protocol. It was developed as a standardized embedded network with highly flexible configuration capabilities. CANopen was designed for motion-oriented machine control networks, such as handling systems. By now it is used in various application fields, such as medical equipment, off-road vehicles, maritime electronics, railway applications or building automation.

CANopen unburdens the developer from dealing with CAN-specific details such as bit-timing and implementation-specific functions. It provides standardized communication objects for real-time data, configuration data as well as network management data.[17]

One of the protocol implementations is the CanFestival Project (www.canfestival.org) . It is an Open Source (LGPL and GPL) CANopen framework and is part of the Beremiz Project (www.beremiz.org), an Open Source framework for automation. 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 Micro-controllers. VScom devices will be already supported in the latest CanFestival version.

Wireshark can be used to capture and analyze CANopen traffic since version 1.7.1 (see Section 4.5).

4.6.2 Running Example

You'll find a small CanFestival example in the CAN examples archive[18] showing the communication between master and slave nodes. Following baudrates are supported: 20K, 50K, 100K, 125K, 250K, 500K and 1M. To execute this example decompress one of the following archives:

- CANopen_example_win32.zip for Windows
- CANopen_example_linux.tar.bz2 for Linux

Under Windows connect two VScom CAN devices - for example two USB-CAN Plus devices installed as `COM10` and `COM11`. Open two command windows and change to the directory where the CANopen examples were extracted to and execute

```
TestMasterSlave -s COM10 -S 125K -M none -l libcanfestival_can_vscom.dll
```

in the first window and

```
TestMasterSlave -m COM11 -M 125K -S none -l libcanfestival_can_vscom.dll
```

in the second. Figure 20 shows the output messages of both nodes.

[17] For more information see the website of CAN in Automation organization www.can-cia.org
[18] http://www.vscom.de/download/multiio/winXP/tools/CAN_Examples.ZIP.exe

Under Linux connect two VScom CAN devices - for example two USB-CAN Plus devices installed as `/dev/ttyUSB0` and `/dev/ttyUSB1`. Open two terminal windows and change to the directory where CANopen examples were extracted to and execute

```
export LD_LIBRARY_PATH=.
./TestMasterSlave -s „/dev/ttyUSB0” -S 125K -M none -l ./libcanfestival_can_vscom.so
```

in the first window and

```
export LD_LIBRARY_PATH=.
./TestMasterSlave -m „/dev/ttyUSB1” -M 125K -S none -l ./libcanfestival_can_vscom.so
```

in the second. Figure 21 shows the output messages of both nodes.

4.6.3 Compilation Instructions

CanFestival is stored in a Mercurial[19] repository. To get CanFestival source code execute:

```
hg clone http://dev.automforge.net/CanFestival-3/
cd CanFestival-3
```

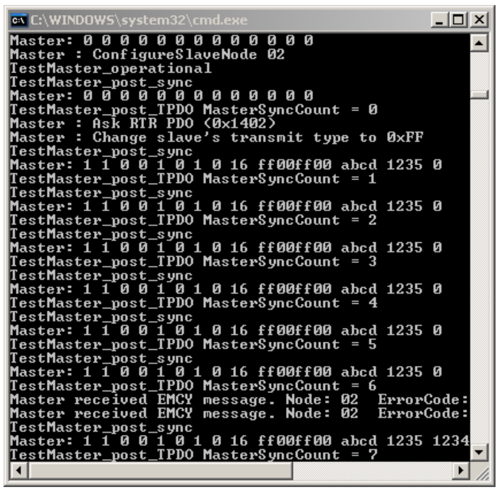
Configure and compile the library and examples:

```
./configure --can=vscom
make
```

The TestMasterSlave is located under `examples/TestMasterSlave/` and the vscom library is located under `drivers/can_vscom/`.

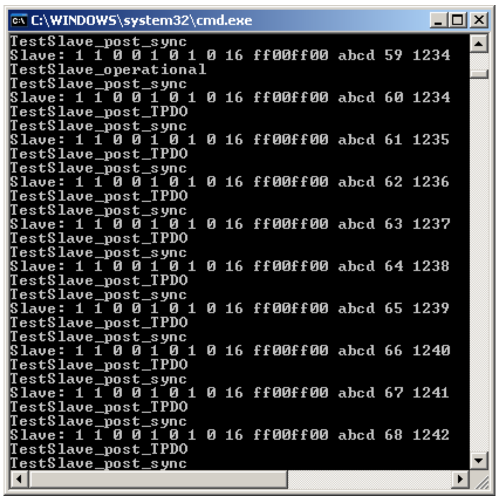
For detailed information about these examples and about using CanFestival in your project refer to the `doc/` folder.

[19] Mercurial



```
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
Master : ConfigureSlaveNode 02
TestMaster_operational
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 0
Master : Ask RTR PDO (0x1402)
Master : Change slave's transmit type to 0xFF
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 0
TestMaster_post_TPDO MasterSyncCount = 1
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 0
TestMaster_post_TPDO MasterSyncCount = 2
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 0
TestMaster_post_TPDO MasterSyncCount = 3
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 0
TestMaster_post_TPDO MasterSyncCount = 4
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 0
TestMaster_post_TPDO MasterSyncCount = 5
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 0
TestMaster_post_TPDO MasterSyncCount = 6
Master received EMCY message. Node: 02  ErrorCode:
Master received EMCY message. Node: 02  ErrorCode:
TestMaster_post_sync
Master: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 1235 1234
TestMaster_post_TPDO MasterSyncCount = 7
```

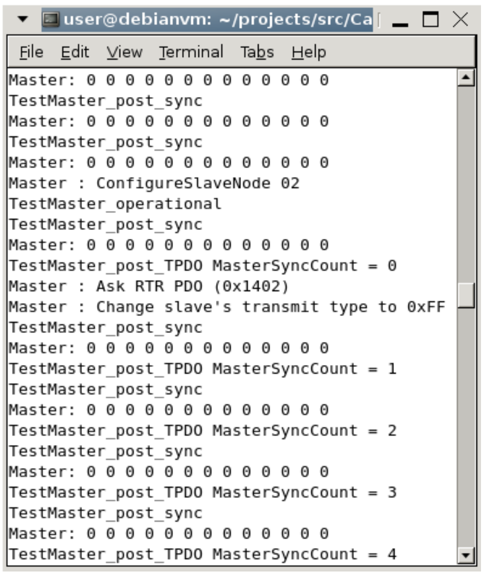
(a) Master



```
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 59 1234
TestSlave_operational
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 60 1234
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 61 1235
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 62 1236
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 63 1237
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 64 1238
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 65 1239
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 66 1240
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 67 1241
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 68 1242
TestSlave_post_TPDO
TestSlave_post_sync
```

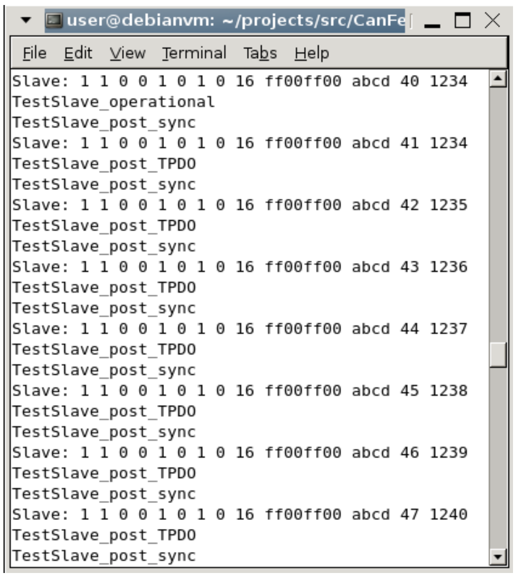
(b) Slave

Figure 20: TestMasterSlave under Windows



```
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
Master : ConfigureSlaveNode 02
TestMaster_operational
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 0
Master : Ask RTR PDO (0x1402)
Master : Change slave's transmit type to 0xFF
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 1
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 2
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 3
TestMaster_post_sync
Master: 0 0 0 0 0 0 0 0 0 0 0 0 0
TestMaster_post_TPDO MasterSyncCount = 4
```

(a) Master



```
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 40 1234
TestSlave_operational
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 41 1234
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 42 1235
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 43 1236
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 44 1237
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 45 1238
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 46 1239
TestSlave_post_TPDO
TestSlave_post_sync
Slave: 1 1 0 0 1 0 1 0 16 ff00ff00 abcd 47 1240
TestSlave_post_TPDO
TestSlave_post_sync
```

(b) Slave

Figure 21: TestMasterSlave under Linux

4.7 Wrapper DLL System

In Linux operating system any application software may use CAN bus adapters from different manufacturers, without modifying the program. An official CAN API named SocketCAN exists for the Linux Kernel.
In Windows systems the situation is different. There is neither an API from Microsoft[20], nor a widely accepted de-facto standard used by manufacturers. All products come with a proprietary driver to access the CAN bus adapter. On top is a set of libraries for application programmers, encapsulating the hardware in higher layer function sets. Application software tailored to use one set of libraries binds the application to certain CAN bus hardware. Those libraries come in the form of DLLs.

VScom provides a system of wrapper DLLs. They provide the possibility to use any VSCAN product as a replacement of products from other manufacturers, when those are given a set of DLLs. The wrapper DLLs use the same name as the original set. They provide the same functions used by the application software, so the existing software shall not notice that exchange.

Installation: Copy the desired wrapper DLL over the original API DLL in your program directory, thus replacing the version from the other manufacturer. It may be useful to create a backup copy before doing so. You must also copy the latest VSCAN-API (vs_can_api.dll) into the same directory.

Configuration: If you want to specify a special configuration for the mapping of a VS-CAN product, you can do this over a configuration file (`vscan.ini`). When there is no configuration file available, the wrapper API uses the first VSCAN product which will be found in the PC. You can also get extra debug information when you configure the debug option for the CAN channel in the configuration file. The debug output will be saved in a file called `vs_can_api.log` in the program directory.

Configuration File Example:

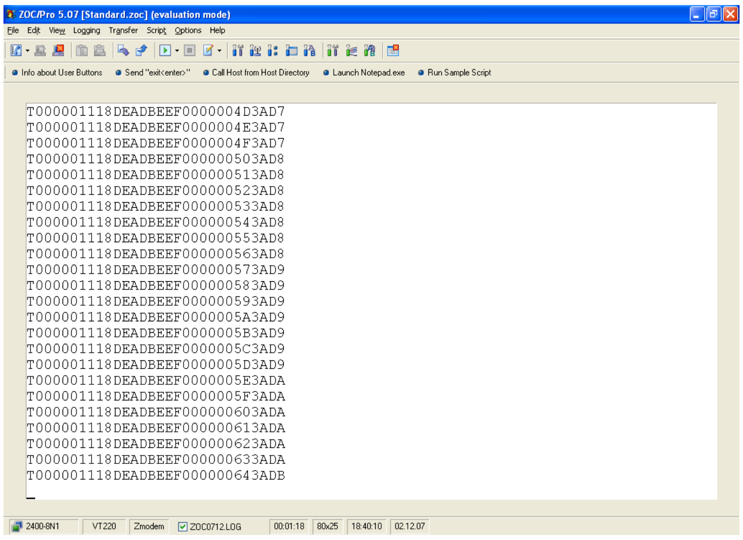
```
[CAN_1.dll]
Port = "COM5"                   ; VSCAN device over a (virtual) COM port
debug = 0

[CAN_2.dll]
Port = "192.168.254.254:2001"   ; mapping to NetCAN+ over IP and TCP port (raw mode)
debug = 1                       ; debug output is switched on (vs_can_api.log)
```

Note: In this example CAN_1.dll and CAN_2.dll are just sample text. Replace the text in the brackets with the name of the DLL used by your application. So if in reality the names are XCAN_USB.dll and yCAN_PCI.dll, use those names for the section titles.
The wrapper DLL finds the desired configuration by searching for a section, which is titled by the DLLs name. The set of wrapper DLLs usually provides a sample configuration file.

[20]See serial ports, manufacturers use the Windows API, products are interchangeable.

4.8 ZOC

ZOC (see Figure 22) is a powerful terminal program which has good logging functionality and will also let you make connections over the network (telnet client).



```
T000001118DEADBEEF0000004D3AD7
T000001118DEADBEEF0000004E3AD7
T000001118DEADBEEF0000004F3AD7
T000001118DEADBEEF000000503AD8
T000001118DEADBEEF000000513AD8
T000001118DEADBEEF000000523AD8
T000001118DEADBEEF000000533AD8
T000001118DEADBEEF000000543AD8
T000001118DEADBEEF000000553AD8
T000001118DEADBEEF000000563AD8
T000001118DEADBEEF000000573AD9
T000001118DEADBEEF000000583AD9
T000001118DEADBEEF000000593AD9
T000001118DEADBEEF0000005A3AD9
T000001118DEADBEEF0000005B3AD9
T000001118DEADBEEF0000005C3AD9
T000001118DEADBEEF0000005D3AD9
T000001118DEADBEEF0000005E3ADA
T000001118DEADBEEF0000005F3ADA
T000001118DEADBEEF000000603ADA
T000001118DEADBEEF000000613ADA
T000001118DEADBEEF000000623ADA
T000001118DEADBEEF000000633ADA
T000001118DEADBEEF000000643ADB
_
```

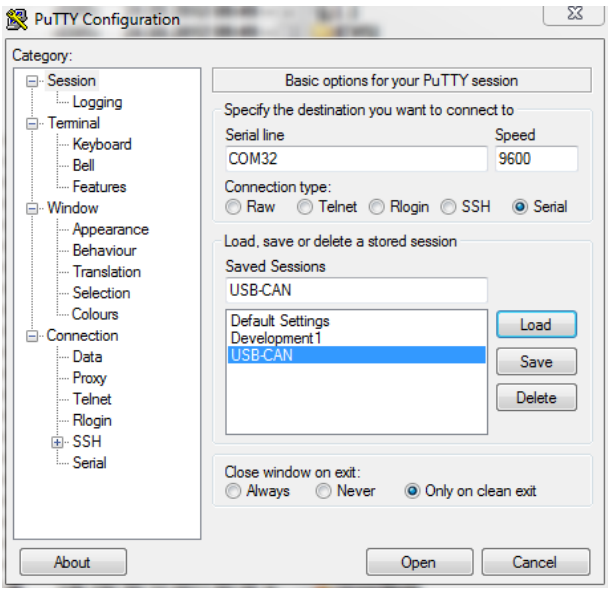
Figure 22: ZOC

4.9 putty

With putty[21] you can both talk to the network based devices NetCAN Plus (see Figure 23b) and via the serial interface to USB-CAN Plus and NetCAN Plus with the driver for virtual Com Ports (see Figure 23a).

It is important to enable hardware handshake for the USB-CAN Plus devices (see Figure 24a). Also make sure LF is added to every CR and local echo is on for more convenience (see Figure 24b).

[21] http://www.putty.org



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM32 Speed: 9600

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin ( ) SSH [✓ Serial]

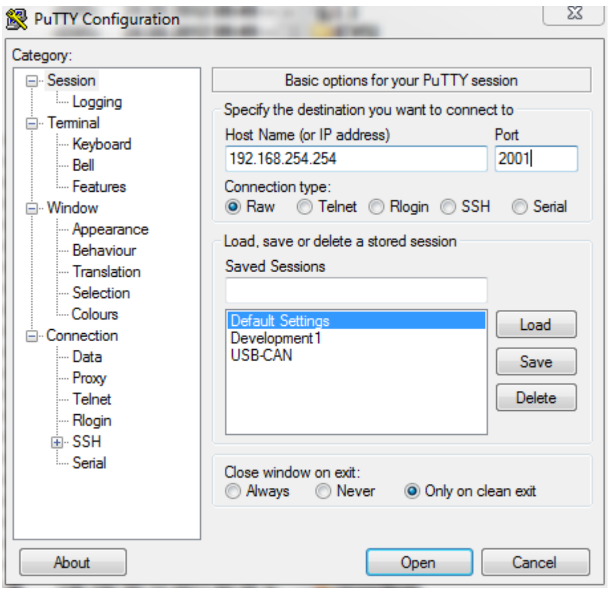
Load, save or delete a stored session

Saved Sessions: USB-CAN

Default Settings
Development1
USB-CAN

Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

(a) Serial Connection



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.168.254.254 Port: 2001

Connection type: [✓ Raw] ( ) Telnet ( ) Rlogin ( ) SSH ( ) Serial

Load, save or delete a stored session

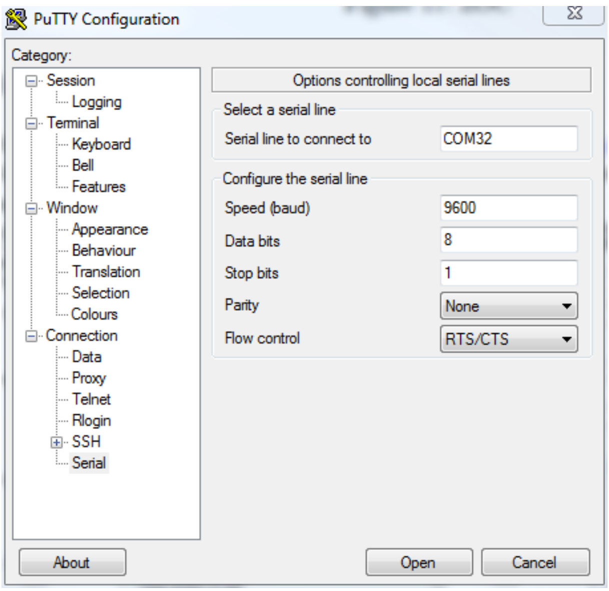
Saved Sessions

Default Settings
Development1
USB-CAN

Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

(b) TCP Raw Connection

Figure 23: Putty



Options controlling local serial lines

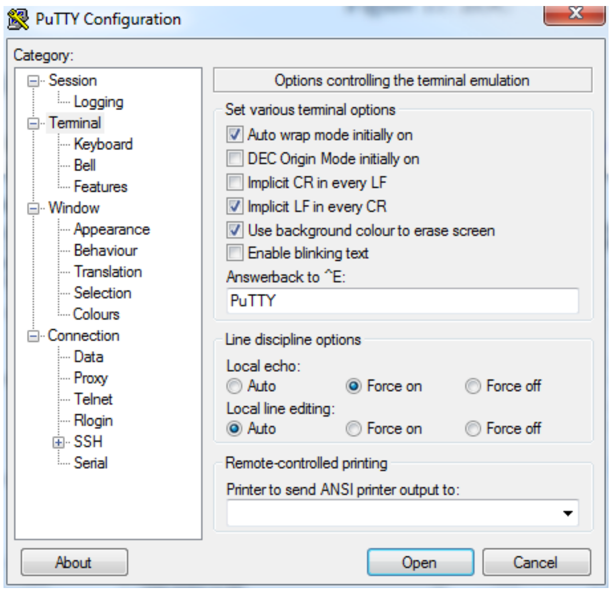
Select a serial line

Serial line to connect to: COM32

Configure the serial line

Speed (baud): 9600
Data bits: 8
Stop bits: 1
Parity: None
Flow control: RTS/CTS

(a) Hardware Handshake



Options controlling the terminal emulation

Set various terminal options

[✓] Auto wrap mode initially on
[ ] DEC Origin Mode initially on
[ ] Implicit CR in every LF
[✓] Implicit LF in every CR
[✓] Use background colour to erase screen
[ ] Enable blinking text

Answerback to ^E: PuTTY

Line discipline options

Local echo: ( ) Auto [✓ Force on] ( ) Force off

Local line editing: [✓ Auto] ( ) Force on ( ) Force off

Remote-controlled printing

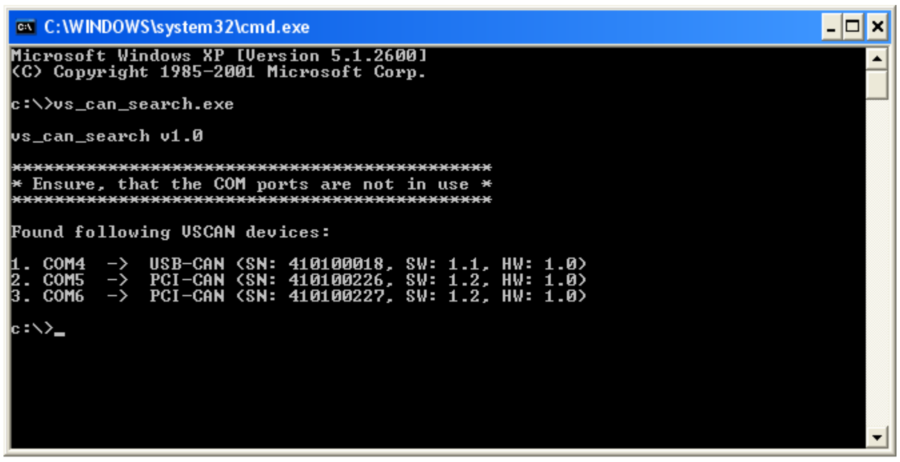
Printer to send ANSI printer output to:

(b) Terminal Settings

Figure 24: Putty: Additional Connection Settings

4.10 vs_can_search

This tool search for VSCAN devices on every COM port. You can also get extra debug information with the parameter„-d[1-3]” - eg. *„vs_can_search.exe -d3”*.



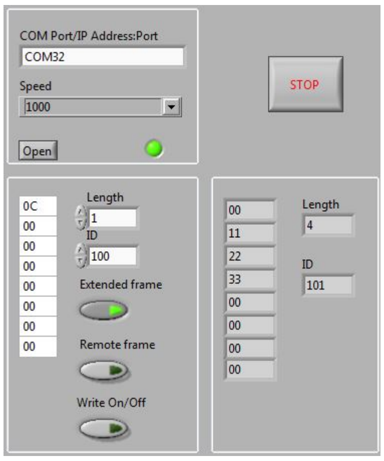
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\>vs_can_search.exe

vs_can_search v1.0

*********************************************
* Ensure, that the COM ports are not in use *
*********************************************

Found following VSCAN devices:

1. COM4  ->  USB-CAN (SN: 410100018, SW: 1.1, HW: 1.0)
2. COM5  ->  PCI-CAN (SN: 410100226, SW: 1.2, HW: 1.0)
3. COM6  ->  PCI-CAN (SN: 410100227, SW: 1.2, HW: 1.0)

c:\>_
```

4.11 LabVIEW

LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is a platform and development environment for a visual programming language from National Instruments. This section shows one possible way using `vs_can_api.dll` with LabVIEW. Specially prepared example (see Figure 25) can be found in our GitHub repository[22]. It lets the user send and read CAN frames after opening the channel with appropriate bitrate. The LabVIEW example was compiled and tested on LabVIEW 8.6.1.

Before trying the LabVIEW example please make sure you can send/receive CAN frames using vscandump and vscansend or Busmaster.

You can send CAN frames using the left panel. After configuring your CAN frame just press „Write On/Off" Button and frames will be sent each second. The incoming frames will be shown on the right panel one per second if any available.



COM Port/IP Address:Port: COM32
Speed: 1000

Left panel: 0C, 00, 00, 00, 00, 00, 00, 00
Length: 1
ID: 100

Right panel: 00, 11, 22, 33, 00, 00, 00, 00
Length: 4
ID: 101

Figure 25: Example Application

4.11.1 Open CAN Channel

The OpenChannel.vi has following input parameters:

- COM Port or IP address and port string, i.e. COM32 or 192.168.1.100:2001 (as in Section 2.2.1)
- speed as a number between 1 and 8 (as in Section 3.2.3).

Output parameters:

- return error code of VSCAN_Open() or VSCAN_Ioctl() routines
- CAN channel handle.

[22] https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/LabVIEW

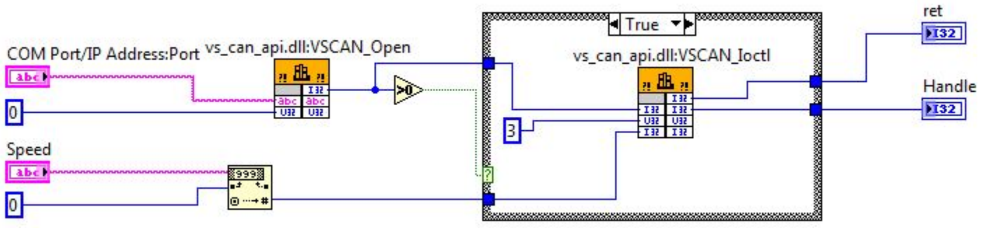


Figure 26: Open CAN Channel

4.11.2 Read CAN Frame

The CanRead.vi has following input parameter:

- handle

Output parameters:

- return code value
- VSCAN_MSG structure
- number of read bytes

CanRead.vi is designed to read one CAN frame per call. To read more bytes at once, the buffer must be increased.

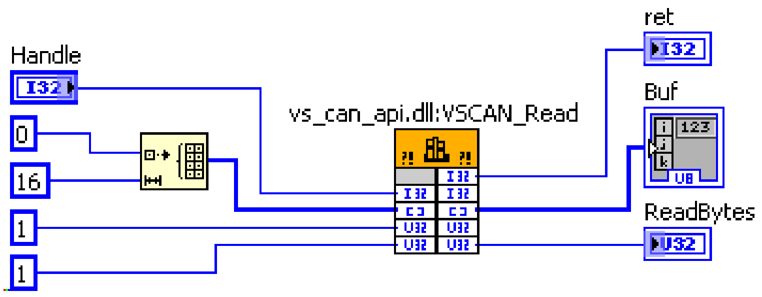


Figure 27: Read CAN Frame

4.11.3 Write CAN Frame

The CanWrite.vi has following input parameters:

- handle
- parts of the VSCAN_MSG structure

Output parameters:

- return code value
- number of written bytes

The VSCAN_Write call is followed directly by VSCAN_Flush call, so the CAN frame will be sent immediately

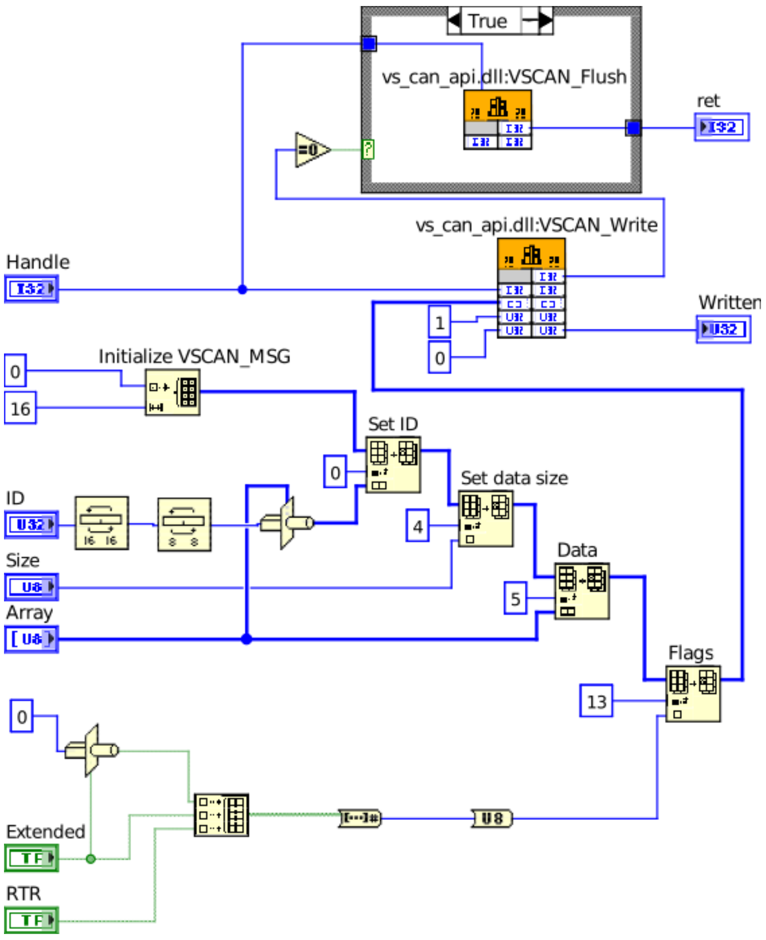
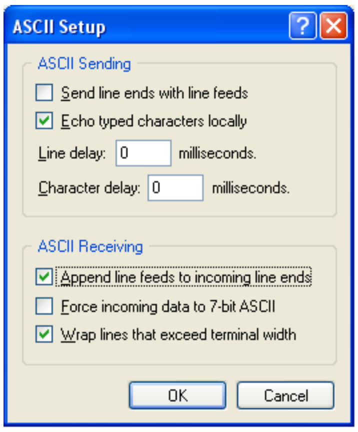


Figure 28: Write CAN Frame

5 Frequently Asked Questions

5.1 All output from the CAN adapter will be written in one line in HyperTerminal?

You must configure the correct settings and switch on "**Append line feeds to incoming line ends**":



ASCII Setup

ASCII Sending

- [ ] Send line ends with line feeds
- [✓] Echo typed characters locally

Line delay: 0 milliseconds.

Character delay: 0 milliseconds.

ASCII Receiving

- [✓] Append line feeds to incoming line ends
- [ ] Force incoming data to 7-bit ASCII
- [✓] Wrap lines that exceed terminal width

5.2 I've updated the driver of my USB-CAN Plus, but the alias baudrate 9600 is not functioning anymore?

There is a VBScript in the driver package, which must be called for each installed virtual USB-CAN Plus COM port:

cscript regmodify.vbs COM<x>

After executing the script the USB-CAN Plus must be disconnected for about 5 seconds and then connected again in order to use the baudrate aliases. Then you can open the port with any standard baudrate - **except 115kbps!**

5.3 The Error LED is permanently on

This LED state indicates bus error. The most typical causes are:

- wrong wiring and missing termination (refer to Section CAN Topology, Wiring and Termination)
- wrong CAN bitrate (refer to Section Setup the Bus Timing (Standard))

5.4 SocketCAN Troubleshooting

FTDI driver After inserting the USB-CAN Plus device your `dmesg` should produce following output:

```
ftdi_sio 1-1.1:1.0: FTDI USB Serial Device converter detected
usb 1-1.1: Detected FT-X
usb 1-1.1: FTDI USB Serial Device converter now attached to ttyUSB0
```

Most desktop Linux distributions provide `ftdi_sio` driver, but the situation can be different on an embedded Linux machine. So if you don't see such a message and no `/dev/ttyUSBx` device appeared, make sure `CONFIG_USB_SERIAL_FTDI_SIO` is activated for your kernel.

slcan driver If you get following message after starting `slcand`, it means the symbol `CONFIG_CAN_SLCAN` wasn't activated for your kernel and you'll have to recompile the kernel with this symbol enabled.

```
modprobe: FATAL: Module slcan not found in directory /lib/modules/*
```

slcan0 interface As soon as `slcan` driver was loaded `slcan0` device should appear. So invoking `ip addr show slcan0` should produce following output:

```
10: slcan0: <NOARP> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
```

If `slcan0` device is missing make sure you've selected the proper serial device when invoking `slcand`.

CAN Error LED If after trying to send/receive packets the CAN Error LED is permanently on, you should first check the wiring as shown in Section CAN Topology, Wiring and Termination and if the wiring is correct then check whether `-sx` parameter corresponds to the required CAN bitrate as shown in Section Setup the Bus Timing (Standard).

5.5 USB-CAN Plus Troubleshooting in Windows

If you encounter any issues with USB-CAN Plus like port cannot be opened, no frames sent or received, you can perform following actions in order to find the issue or at least to gather as much info for the support request as possible. Please always provide output from the tools mentioned below and also include a photo/scheme of your cabling.

Powerup LED Blinking Sequence When you plug the USB-CAN Plus device to USB both Error and Data LEDs will blink twice and remain off afterwards. If you don't see this blinking sequence either device or firmware is damaged.

Checking Driver Installation First of all make sure you've performed all steps described in Section USB-CAN Plus Device. This is important for a correct function.

Perform CAN device search using vs_can_search utility described in Section vs_can_search. You should see your device with the corresponding serial number.

Check CAN Wiring Make sure your CAN wiring corresponds to the wiring described in Section CAN Topology, Wiring and Termination and Error LED is not permanent on when sending/receiving CAN frames.

Perform Self-Diagnostic Tests Use `vs_can_test_simple.exe` to perform such a test either against another VScom device or with your current device itself as described in vs_can_test_simple.exe.

Check CAN Filter Settings If you can send but not receive CAN frames it is most likely, that Acceptance Code and Mask registers are setup wrong. These settings are stored in internal non-volatile memory so that they are persistent across power cycles. Try default settings i.e. Acceptance Code 0x00000000 and Acceptance Mask 0xFFFFFFFF. `vscandump.exe` or `vs_can_test_simple.exe` will make these settings on every invocation.

Perform Basic CAN Transmission Tests If your CAN devices send frames autonomously, you can use `vscandump.exe` to receive the traffic. You can also inject CAN frames using `vscansend.exe`. Both tools are described in Section vscandump and vscansend. Alternatively you can use BUSMASTER software described in Section Busmaster.