

**www.vskom.de**

# **VSCAN Manual**

**Edition: March 2008**



Vision Systems GmbH

Tel: +49 40 528 401 0

Fax: +49 40 528 401 99

Web: [www.visionsystems.de](http://www.visionsystems.de)

Support: [service@visionsystems.de](mailto:service@visionsystems.de)

---

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

## **Copyright Notice**

Copyright © 2008 Vision Systems. All rights reserved. Reproduction without permission is prohibited.

## **Trademarks**

Vision Systems is a registered trademark of Vision Systems GmbH. All other trademarks and brands are property of their rightful owners.

## **Disclaimer**

Vision Systems reserves the right to make changes and improvements to its product without providing notice.

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

Information provided in this manual is intended to be accurate and reliable. However, Vision Systems 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

<b>1</b>	<b>Installation</b>	<b>5</b>
1.1	USB CAN Device . . . . .	5
1.2	Serial CAN Device . . . . .	6
1.3	Network CAN Device . . . . .	7
1.3.1	Configuration Overview . . . . .	7
1.3.2	Webbrowser Server Configuration . . . . .	8
1.3.3	Webbrowser Channel Configuration . . . . .	10
1.3.4	Webbrowser Tools . . . . .	11
1.4	General Information . . . . .	14
1.4.1	Baud-rates and Handshake . . . . .	14
1.4.2	Pin-out of the 9 Pin D-Sub Connector . . . . .	14
1.4.3	Termination Resistors . . . . .	14
<b>2</b>	<b>Application Programming Interface</b>	<b>15</b>
2.1	Introduction . . . . .	15
2.2	Functions . . . . .	16
2.2.1	VSCAN_Open . . . . .	16
2.2.2	VSCAN_Close . . . . .	16
2.2.3	VSCAN_Ioctl . . . . .	18
2.2.4	VSCAN_Read . . . . .	21
2.2.5	VSCAN_Write . . . . .	22
2.2.6	VSCAN_Flush . . . . .	23
2.3	Types and Structures . . . . .	24
2.3.1	VSCAN_HANDLE . . . . .	24
2.3.2	VSCAN_STATUS . . . . .	24
2.3.3	VSCAN_HWPARAM . . . . .	24
2.3.4	VSCAN_MSG . . . . .	25
2.3.5	VSCAN_BTR . . . . .	25
2.3.6	VSCAN_CODE_MASK . . . . .	26
<b>3</b>	<b>ASCII Command Set</b>	<b>27</b>
3.1	Introduction . . . . .	27
3.2	Commands . . . . .	28
3.2.1	Open the CAN Channel . . . . .	28
3.2.2	Close the CAN Channel . . . . .	28
3.2.3	Setup the Bus Timing (Standard) . . . . .	28
3.2.4	Setup the Bus Timing (Advanced) . . . . .	29
3.2.5	Transmitting a Standard Frame . . . . .	30
3.2.6	Transmitting a Standard Remote Request Frame . . . . .	30
3.2.7	Transmitting an Extended Frame . . . . .	31
3.2.8	Transmitting an Extended Remote Request Frame . . . . .	31

3.2.9	Set Acceptance Code and Mask Register . . . . .	32
3.2.10	Get Status Flags . . . . .	32
3.2.11	Get Version Information . . . . .	32
3.2.12	Get Serial Number . . . . .	33
3.2.13	Get Extra-Information . . . . .	33
<b>4</b>	<b>Tools</b>	<b>34</b>
4.1	CANHacker . . . . .	34
4.2	ZOC . . . . .	34

## 1 Installation

### 1.1 USB CAN Device

Extract the USB driver from the file „*USB-CAN-cdm20000.zip*” on the product CD or download it from our company website [www.visionsystems.de](http://www.visionsystems.de). If you’ve done this, you could plug in the adapter in an USB port of your choice. When you will be asked for the driver, you must choose the location where you’ve extracted the driver files. The device will be registered and you can use the COM port for your further work (see Figure 1).

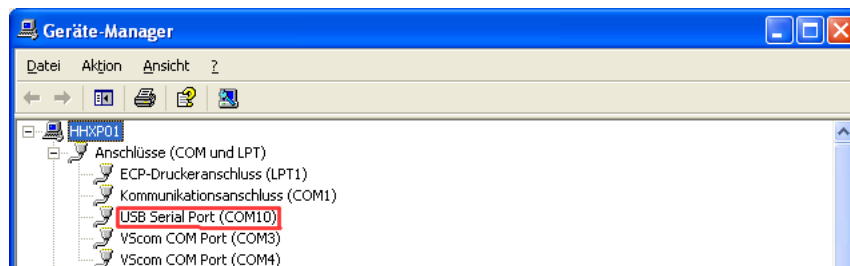


Figure 1: Device Manager

To get a better performance you must configure the latency timer of the COM port to 1 millisecond (see Figure 2).

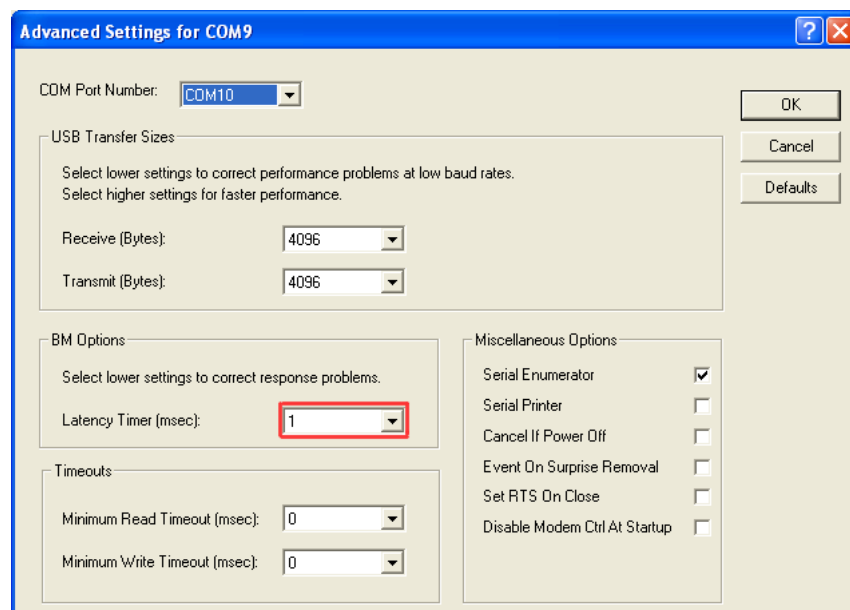


Figure 2: Advanced Settings

In most terminal programs you can’t configure 3 MBit directly. Therefor we’ve changed the driver to use 2400 baud as an alias for 3 MBit. So if you wanna communicate with the

USB-CAN device over HyperTerminal, you will open the port with the settings shown in Figure 3. If you like to change the alias baud-rate, please take a look at the application note „AN232B-05\_BaudRates.pdf” which is included in the drivers zip-package.

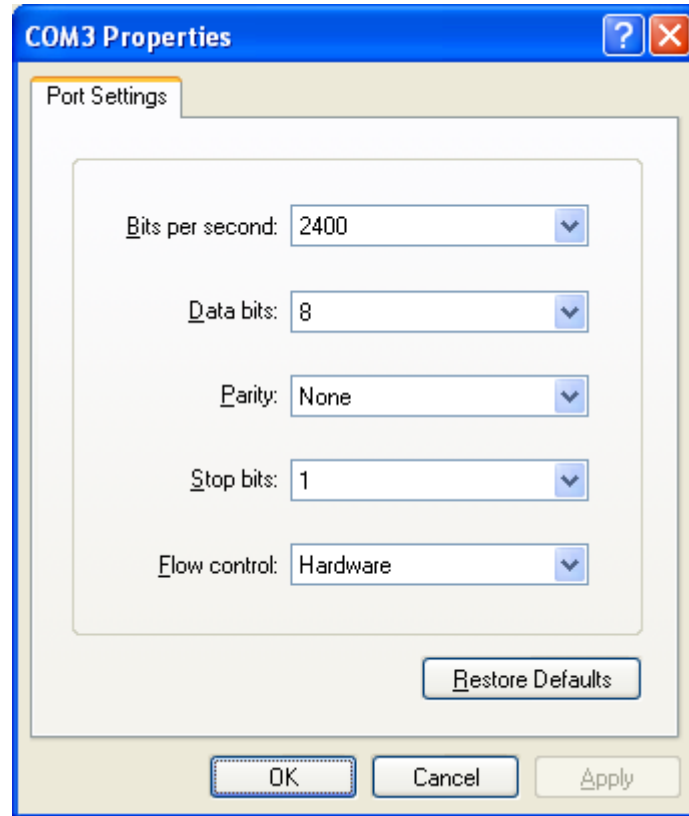


Figure 3: COM Port Properties

### 1.2 Serial CAN Device

To use the serial CAN device you mustn't do anything except for finding a free serial port on you host computer. To get a better performance you should change the port settings to the following parameters (see Figure 4).

Reboot to ensure that they were set.

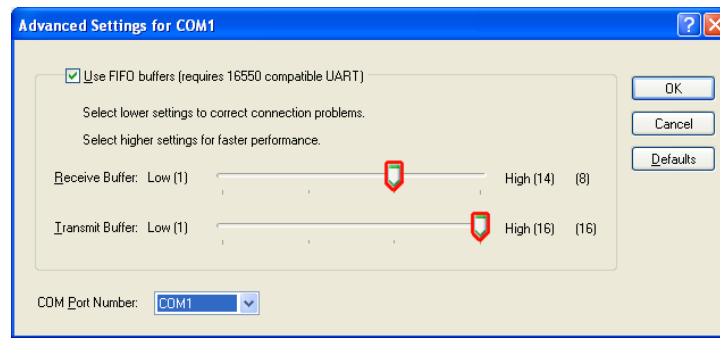


Figure 4: Advanced Settings for COM Port

### 1.3 Network CAN Device

NET-CAN provides CAN communication over network. There are two ways to communicate with the NET-CAN device:

- Driver mode: in this mode the network is transparent for the application. To use this mode installation of the Windows driver is required (please refer to the ["NetCom Wireless Serial Device Server User Manual"](#) for installation instructions). After driver installation the new virtual COM port will be available to the system, so NET-CAN can be used in the same way as SER-CAN or USB-CAN. Due to the virtual COM port protocol overhead the performance is lower than by the TCP raw mode.
- TCP raw mode: the communication will be handled directly via IP address and port number. In this mode no driver installation is required.

For the API there is no difference what operational mode is used, but due to performance issues TCP raw mode is preferable.

#### 1.3.1 Configuration Overview

NET-CAN device can be configured in following ways:

- via web interface
- via Telnet
- via NetCom Manager (not the whole amount of settings)

For Telnet and NetCom Manager configuration refer to the ["NetCom Wireless Serial Device Server User Manual"](#).

Open your Webbrowser for the web interface configuration. Type the address of the NET-CAN Server in the address line . In the example from above type `http://192.168.254.254` as the target. You may do this on any operating system you prefer.

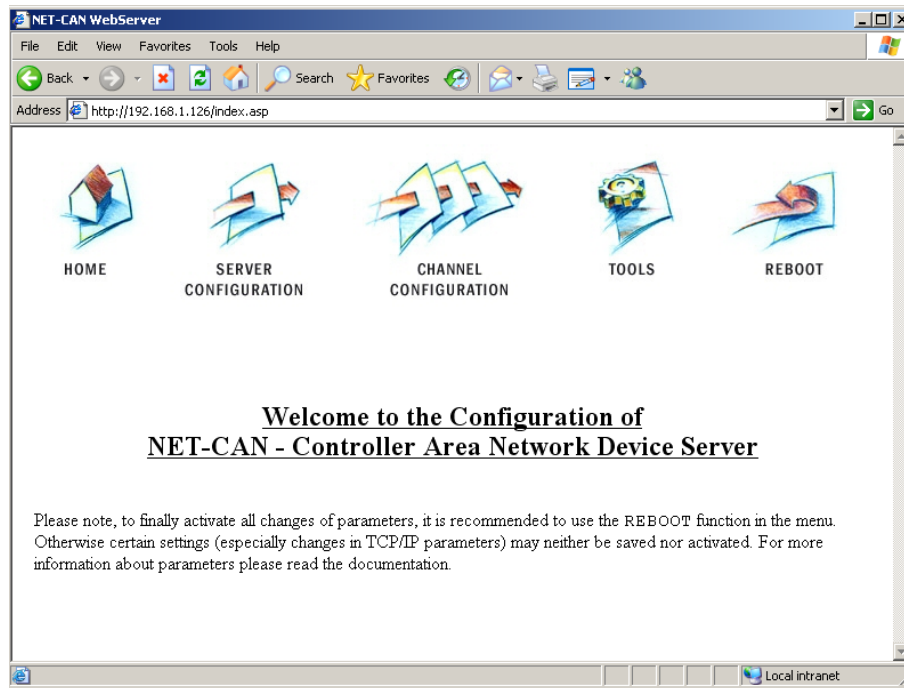


Figure 5: Web Interface for Configuration

The NET-CAN welcomes you with its “Home” screen (see Figure 5). 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 NET-CAN requests for a Reboot (see Figure 6).

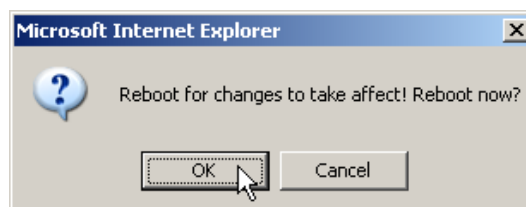


Figure 6: Web Interface Request to Reboot

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

### 1.3.2 Webbrowser Server Configuration

The *Server Configuration* is a very long menu (see Figure 7). There is basic server information (see Figure 7(a)), the server parameters related to the IP-configuration (see



Figure 7(b)), the parameters for Wireless communication as far as applicable (see Figure 7(c)), the section for encrypted communication (see Figure 7(d)), Password settings (see Figure 7(e)), and finally the configuration for date and time (see Figure 7(f)).

**Server Info**

Server Type: 110  
 Software Version: 1.0.0  
 Hardware Version: 1.0  
 Serial Nr.: 0000120120  
 UpTime: 0 day(s) 01:15:57  
 Contact: <unset>  
 Location: <unset>

**Server Parameter**

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

Server Name: NET-CAN\_0000120120  
 MAC Address: 00:04:D9:80:69:DE  
 DHCP: Enabled  
 IP Address: 192.168.1.126  
 Netmask: 255.255.255.0  
 Broadcast: 192.168.1.255  
 Gateway: 192.168.1.1  
 DNS: 192.168.1.3  
 Domain: visionsystems.de  
 ConfigPort: 23  
 KeepAlive: Off  
 KeepAliveInterval: 0

**Wireless Parameter**

SSID: NetCom\_0000123456  
 OperationMode: Ad-hoc  
 WirelessMode: 11 b+g  
 CountryRegion: ETSI (1-13)  
 Channel: 7  
 Encryption Type: Off  
 Encryption Key:  
 RTSThreshold: 2312  
 FragmentationThreshold: 2312

(a) Server Info
(b) Server Parameter
(c) Wireless Parameter

**OpenVPN Parameter**

OpenVPN: Disabled  
 TCP Port: 1194  
 IP Address: 192.168.127.254  
 Netmask: 255.255.255.0  
 Broadcast: 192.168.127.255  
 Max. Clients: 8  
 TCP Port (Destination): 1194  
 IP Address (Destination): 0.0.0.0  
 Encryption: AES-256-CBC  
 Logging: Off

[Configuration-Settings of the Encryption-Key](#)

**Date and Time Settings**

Date & Time: 01-01-1970 01:22:19 UTC+0

**Simple Network Time Protocol**

State: Off  
 Mode: DHCP  
 Interval: 1800  
 Server:

**Authentication**

Security Settings  
 Password:  
 Retype Password:

**OpenVPN Parameter**

**Authentication**

**Date and Time Settings**

Figure 7: Server Configuration

Information about the selected NET-CAN is displayed as *Server Info*. Starting with the *Server Type*, this is the model of the NET-CAN, 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. 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 NET-CANs name and of course all parameters in IP-settings. The Server Name is used as the ESSID of the Wireless Ad-hoc mode. Generally it is used as information, e.g. in the NetCom Manager program

or in SNMP. You may choose the network interface as Cable, Wireless or both (with priority).

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 NET-CAN 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 functions is used, you must define an interval in seconds. NET-CAN 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 the ["NetCom Wireless Serial Device Server User Manual"](#).

### 1.3.3 Webbrowser Channel Configuration

NET-CAN can be operated in two modes (see Figure 8):

- Driver Mode - Only very few parameters have a function in *Driver Mode* (see Figure 8(a)). NET-CAN is operating as a Server. It accepts two connections per 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 includes the configuration of the serial port, as well as signals for changed Modem Status lines. This mode is required when the CAN channel will be used with the Virtual Com Driver, it is enabled by default. The NET-CAN 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).
- TCP Raw Server - As TCP Raw Server NET-CAN operates very simple (see Figure 8(b)). It only waits for incoming data connections in Raw IP mode. As with the Driver Mode only the data connection is defined. You can connect multiple times to the NET-CAN also from different machines. When a password is configured, the NET-CAN sends the question "Password: " to the client. The user (his application) must first send the password, followed by a <CR> character. The password is not echoed to allow usage with Telnet on a Monitor.

Transfer Settings		Transfer Settings	
Channel 1		Channel 1	
Mode ?	<input type="text" value="Driver Mode"/>	Mode ?	<input type="text" value="TCP Raw Server"/>
TCP Port(Control) ?	<input type="text" value="2000"/>	TCP Port(Data) ?	<input type="text" value="2001"/>
TCP Port(Data) ?	<input type="text" value="2001"/>	Max.Clients ?	<input type="text" value="1"/>
KeepAliveMode ?	<input type="text" value="On Connect"/>	Password ?	<input type="text"/>
KeepAliveInterval ?	<input type="text" value="0"/>	Retype Password:	<input type="text"/>

(a) Driver Mode

(b) TCP Raw Server

Figure 8: Channel Configuration

### 1.3.4 Webbrowser Tools

The available tools are (see Figure 9):

- *The Ping* utility will be used to check if a station is available (see Figure 9(a)). 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 NET-CAN 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 9(b)). Use Netstat to see the actual status of NET-CAN IP Ports. This is a standard tool for network debugging. A *Foreign Address* of 0.0.0.0 is listed when NET-CAN 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 Wireless* option will be used to detect WLAN devices in the proximity (see Figure 9(c)). When it comes to Wireless communications, it is useful to see a list of possible partner stations on the WLAN. This function is available in many drivers, and also in the NET-CAN WLAN Serial Device Servers. This function is often referred to as *Range Scan*. On the NET-CAN it will open a separate browser window with the results. An example of this is shown below. This example lists two other NET-CAN configured for Ad-Hoc communication on channel 7. Both use no encryption. There is also an Access Point (listed as Managed), of course in Infrastructure-mode. To connect to this AP the NET-CAN must use encryption. Since the NET-CAN itself is in Ad-Hoc mode, the communication is limited to the 802.11b, which results in 11Mb/s as raw transmission speed.

- *The Firmware Update* option is used to update the firmware (see Figure 9(d)). 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, NET-CAN will Reboot.
- *The Saving of Configuration to / Loading from a file* option will be used to manage NET-CAN configuration (see Figure 9(e)). It is possible to save the actual configuration to a text file. Of course it is also possible to load the saved configuration into a NET-CAN.
- *The Syslog* option will be used to send logging information to the syslog facility (see Figure 9(f)). 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 9(f)). For this kind of logging the NET-CAN behaves as the server. Open a TCP connection to the configured port, and receive all information generated.

### Ping

IP Address:

### Netstat

[View connections](#)

(a) Ping

(b) Netstat

### FirmwareUpdate

*Warning: the server reboots after updating*

### Wireless

[View Wireless-Devices in Range](#)

(c) Wireless

(d) Firmware Update

### Syslogging

Syslog?

Destination?

Facility?

### Configuration File

Save: *Save the Configuration Parameters in a File*

Load:

(e) Configuration File

Debuglog?

Debug Port?

(f) Syslog

Figure 9: Tools

### 1.4 General Information

#### 1.4.1 Baud-rates and Handshake

Device	Baud-rate	Handshake
USB CAN	3 MBit	RTS/CTS
Network CAN	3 MBit	RTS/CTS
Serial CAN	115200 KBit	RTS/CTS

#### 1.4.2 Pin-out of the 9 Pin D-Sub Connector

Pin	Signal	Description
1	CAN_V+	like pin 9, but it is intermitted through a solder break inside the device
2	CAN_L	CAN_L bus line (dominant level is low)
3	CAN_GND	CAN ground
4	-	reserved
5	CAN_SHLD	CAN line shielding (optional)
6	GND	ground (optional)
7	CAN_H	CAN_H bus line (dominant level is high)
8	-	reserved
9	CAN_V+	+5V to drive the transceiver or optocouplers (optional)

#### 1.4.3 Termination Resistors

The USB CAN, serial CAN and network CAN devices have no termination resistors inside. It's up to you to choose the right combination and values for your topology.

## 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 have. 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.

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

For Linux, you must 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.

All functions and data structures are explained in the next sub-sections.

### 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

Mode

[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 where the device is located
- IP address and port number of the device

[in] The mode in which the CAN channel should 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

**Examples:**

```
handle = VSCAN_Open (VSCAN_FIRST_FOUND, VSCAN_MODE_NORMAL);  
handle = VSCAN_Open ("123456", VSCAN_MODE_LISTEN_ONLY);  
handle = VSCAN_Open ("COM3", VSCAN_MODE_NORMAL);  
handle = VSCAN_Open ("192.168.254.254:2001", VSCAN_MODE_NORMAL);
```

#### 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 should 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. *This will only function if your target is a console application.*

Possible debug levels are:

- VSCAN\_DEBUG\_NONE
- VSCAN\_DEBUG\_LOW
- VSCAN\_DEBUG\_MID
- VSCAN\_DEBUG\_HIGH

#### Example:

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

#### VSCAN\_IOCTL\_GET\_HWPARAM

This ioctl gives you the possibility to get the hardware parameters (serial number, hardware and software version) of the device. Therefor 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 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

#### Example:

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

### VSCAN\_IOCTL\_SET\_BTR

This ioctl gives you the possibility to configure the speed registers manually (bus timing registers). Therefor 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 Philips or at the following website: [www.port.de](http://www.port.de)

#### Example:

```
VSCAN_BTR btr = { .Btr0 = 0x00, .Btr1 = 0x14 };  
  
status = VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_BTR, &btr);
```

### 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. Therefor 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 Philips.

#### Example:

```
VSCAN_CODE_MASK codeMask;  
  
// will receive the ids between 0x300 and 0x3ff  
codeMask.Code = 0x6000;  
codeMask.Mask = 0x1fff;  
  
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);
```

### VSCAN\_IOCTL\_GET\_FLAGS

To get extended status and error flags use this ioctl. When a special status occurred, it will also be indicated by the flashing red led on the device. 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 Philips, 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);
```

### 2.2.4 VSCAN\_Read

To read one or more CAN frames from the CAN bus, you must use the `VSCAN_Read` function.

```
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 size 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\_Write

With the VSCAN\_Write function you can write one or more frames to the CAN bus.

```
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 size 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);
```

### 2.2.6 VSCAN\_Flush

The VSCAN\_Flush function will send all data immediately out to the CAN bus. *Normally the frames are buffered and in the case of the USB device we get a much higher performance, because the USB packets are completely filled.*

```
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);
```

### 2.3 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 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 an 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\_HWPARAM

This structure holds the values of the hardware parameters.



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

The `SerialNr` element comprised the linear serial number especially for this device. The `HwVersion` holds the revision of the CAN hardware and in the opposite `SwVersion` the actual software version of the firmware.

### 2.3.4 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`.

### 2.3.5 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 Philips.

### 2.3.6 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 Philips.

## 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).

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.

### 3.2 Commands

#### 3.2.1 Open the CAN Channel

The CAN channel will be opened with the command `O[CR]` or `L[CR]`. The difference between these two types is, that the last command will open the channel in a listen only mode, in which no bus interaction will be done from the controller. 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]
```

#### 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 closed.

##### Example:

```
C[CR]
```

#### 3.2.3 Setup the Bus Timing (Standard)

The bus timing will be setup-ed with the command `Sn[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]`. 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 Philips or at the following website: [www.port.de](http://www.port.de)

`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 `tiiiiidd[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 `Tiiiiiiiidd[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:**

`iiiiiii`

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.

<code>T000001113102030[CR]</code>
-----------------------------------

### 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:**

`iiiiiii`

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.

<code>R000001113[CR]</code>
-----------------------------

### 3.2.9 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 more information please take a look at chapter 2.2.3. This command is only available if the CAN channel is closed.

#### Example:

We will filter for all standard frames between 0x300 and 0x3ff.

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

### 3.2.10 Get Status Flags

To get the status bits when an error occurred, you must use the command `F[CR]`. The red led on the device will show you, when there is a special status available. 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]
```

### 3.2.11 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]
```



### 3.2.12 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]
```

### 3.2.13 Get Extra-Information

You can retrieve extra informations 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 Philips.

#### Example:

Retrieving the extra-information.

```
I[CR]
```

## 4 Tools

### 4.1 CANHacker

CANHacker is a tool for analyzing and transmitting frames on the CAN bus. The Figure 10 will show you, how the settings must be configured for the VSCAN devices. *Note: The baud-rate 2400 is an alias for the desired speed of the USB and network CAN devices (3MBit).*

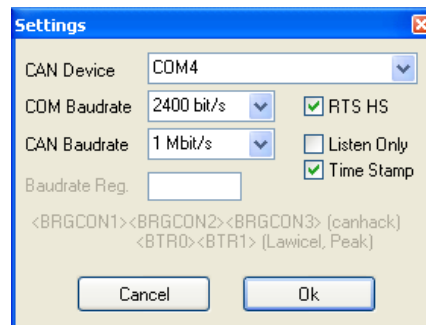


Figure 10: CANHacker Settings

### 4.2 ZOC

ZOC (see Figure 11) is a powerful terminal program which has good logging functionalities and will also let you make connections over the network (telnet client).

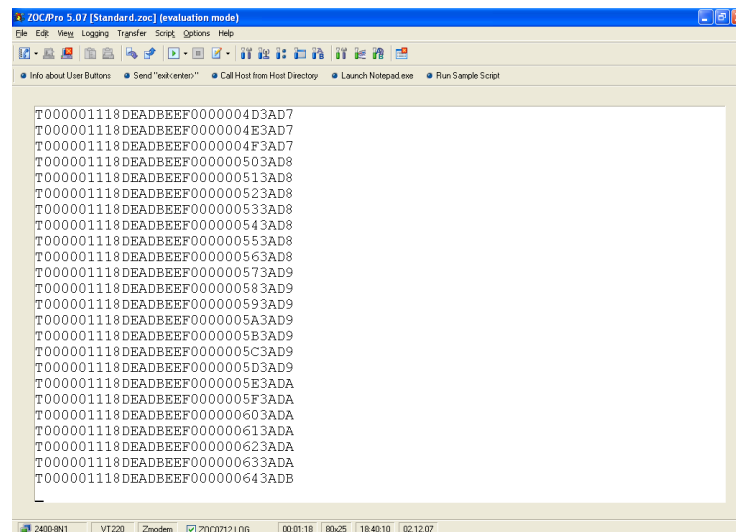


Figure 11: ZOC