

CANbox[®]

Dual CAN to WLAN/LAN Converter

The CANbox®

1. GENERAL INFORMATION	5
1.1. SPECIAL FEATURES.....	5
1.2. INSTALLED MAX MODULES.....	5
1.3. EXTERNAL CONNECTORS.....	6
1.4. WLAN/LAN PROPERTIES.....	6
1.4.1. WLAN.....	6
1.4.2. Transfer rates.....	7
2. CONNECTORS OF THE CANBOX®.....	7
2.1. OVERVIEW.....	7
2.2. ASSIGNMENT OF THE EXTERNAL CONNECTORS.....	7
2.2.1. Connector ST 1, power supply.....	8
2.2.2. Host Interfaces (Connectors ST 2 and ST 3).....	8
2.2.3. D-SUB-9 plug for CAN 1 and CAN 2.....	9
2.2.4. SMA plug for antenna.....	9
3. STARTING INSTRUCTIONS.....	9
3.1. CANBOX® SETUP.....	9
3.2. CANBOX® CONFIG TOOL.....	10
3.2.1. Operation.....	10
3.2.2. CANbox® Connection Dialog.....	12
3.2.3. CANbox® Configuration Dialog.....	13
3.3. CANBOX® AS ROUTER.....	16
3.4. ENHANCED ROUTER (FORWARDER).....	17
4. CANBOX® LIBRARY.....	18
4.1. STRUCTURES AND DEFINITIONS.....	18
4.1.1. CANBOX_VERSION_INFO.....	18
4.1.2. CANBOX_TIME_STAMP.....	18
4.1.3. CANBOX_SCAN_INFO.....	18
4.1.4. CANBOX_DEVICE_INFO.....	18
4.1.5. CANBOX_RESET_EX.....	19
4.1.6. CANBOX_INTERFACE.....	19
4.1.7. CANBOX_INTERFACE_EX.....	20
4.1.8. CANBOX_INTERFACE_STATE.....	20
4.1.9. CANBOX_INTERFACE_READ.....	21
4.1.10. CANBOX_INTERFACE_WRITE.....	21
4.1.11. CANBOX_IDENTIFIER.....	22
4.1.12. CANBOX_ACCEPTANCE_FILTER.....	22
4.1.13. CANBOX_ROUTER.....	23
4.1.14. CANBOX_ROUTER_EX.....	24
4.1.15. CANBOX_LAN_CONFIG.....	25
4.1.16. CANBOX_WLAN_CONFIG.....	25
4.1.17. CANBOX_IDENTIFIER_DATA.....	25
4.1.18. CANBOX_UNIVERSAL_SENDER_DATA.....	26
4.1.19. CANBOX_16.....	26
4.1.20. CANBOX_32.....	26
4.2. ERROR CODES.....	26
4.2.1. General error messages of the CANbox®.....	26
4.2.2. Error messages concerning initialisation.....	27
4.2.3. Error messages concerning release.....	27
4.2.4. Error messages concerning CAN interfaces.....	27

4.2.5. Error messages concerning CAN identifier.....	27
4.2.6. Error messages concerning acceptance filters.....	27
4.2.7. Error messages concerning universal senders.....	28
4.2.8. General error messages of the library.....	28
4.2.9. Error messages of the library concerning the connection to the CANbox®.....	28
4.3. GENERAL FUNCTIONS.....	28
4.3.1. canbox_init_lib.....	28
4.3.2. canbox_get_driver_version.....	28
4.3.3. canbox_get_error_message.....	28
4.3.4. canbox_exit_lib.....	29
4.4. DEVICE FUNCTIONS.....	29
4.4.1. canbox_scan_devices.....	29
4.4.2. canbox_open_device.....	29
4.4.3. canbox_reset_device.....	29
4.4.4. canbox_reset_device_ex.....	29
4.4.5. canbox_get_device_info.....	29
4.4.6. canbox_reset_time_stamp.....	30
4.4.7. canbox_set_time_stamp.....	30
4.4.8. canbox_set_time_stamp_by_ref.....	30
4.4.9. canbox_close_device.....	30
4.5. INTERFACE FUNCTIONS.....	30
4.5.1. canbox_open_interface.....	30
4.5.2. canbox_open_interface_by_ref.....	31
4.5.3. canbox_open_interface_ex.....	31
4.5.4. canbox_open_interface_ex_by_ref.....	31
4.5.5. canbox_get_interface_state.....	31
4.5.6. canbox_get_last_interface_time.....	31
4.5.7. canbox_clear_interface.....	32
4.5.8. canbox_start_interface.....	32
4.5.9. canbox_read_interface.....	32
4.5.10. canbox_read_interface_ex.....	32
4.5.11. canbox_write_interface.....	32
4.5.12. canbox_write_interface_ex.....	33
4.5.13. canbox_stop_interface.....	33
4.5.14. canbox_close_interface.....	33
4.6. IDENTIFIER FUNCTIONS.....	33
4.6.1. canbox_open_identifier.....	33
4.6.2. canbox_open_identifier_by_ref.....	33
4.6.3. canbox_read_identifier.....	34
4.6.4. canbox_write_identifier.....	34
4.6.5. canbox_close_identifier.....	34
4.7. ACCEPTANCE FILTER FUNCTIONS.....	34
4.7.1. canbox_open_acceptance_filter.....	34
4.7.2. canbox_open_acceptance_filter_by_ref.....	35
4.7.3. canbox_read_acceptance_filter.....	35
4.7.4. canbox_close_acceptance_filter.....	35
4.8. UNIVERSAL SENDER FUNCTIONS.....	35
4.8.1. canbox_open_universal_sender.....	35
4.8.2. canbox_write_universal_sender.....	35
4.8.3. canbox_write_universal_sender_ex.....	36
4.8.4. canbox_close_universal_sender.....	36
4.9. ROUTER FUNCTIONS.....	36
4.9.1. canbox_read_router_config.....	36
4.9.2. canbox_read_router_config_ex.....	36
4.9.3. canbox_write_router_config.....	36
4.9.4. canbox_write_router_config_ex.....	37
4.10. SUPPORTING FUNCTIONS FOR VISUAL BASIC.....	37
4.10.1. canbox_convert_to_CANbox16.....	37
4.10.2. canbox_convert_to_CANbox16_by_ref.....	37

4.10.3. <i>canbox_convert_from_CANbox16</i>	37
4.10.4. <i>canbox_convert_to_CANbox32</i>	37
4.10.5. <i>canbox_convert_to_CANbox32_by_ref</i>	38
4.10.6. <i>canbox_convert_from_CANbox32</i>	38
5. SOCKET INTERFACE	38
5.1. STRUCTURES AND DEFINITIONS.....	38
5.1.1. <i>CANBOX_SOCKET_IN</i>	38
5.1.2. <i>CANBOX_SOCKET_OUT</i>	38
5.2. FUNCTIONS.....	39
5.2.1. <i>canbox_init</i> (Index 4).....	39
5.2.2. <i>canbox_reset</i> (Index 5).....	39
5.2.3. <i>canbox_info</i> (Index 6).....	39
5.2.4. <i>canbox_set_time_stamp</i> (Index 7).....	39
5.2.5. <i>canbox_exit</i> (Index 12).....	39
5.2.6. <i>canbox_open_interface</i> (Index 15).....	40
5.2.7. <i>canbox_open_interface_ex</i> (Index 16).....	40
5.2.8. <i>canbox_get_interface_state</i> (Index 17).....	40
5.2.9. <i>canbox_start_interface</i> (Index 18).....	40
5.2.10. <i>canbox_clear_interface</i> (Index 19).....	40
5.2.11. <i>canbox_read_interface</i> (Index 20).....	40
5.2.12. <i>canbox_write_interface</i> (Index 21).....	41
5.2.13. <i>canbox_stop_interface</i> (Index 22).....	41
5.2.14. <i>canbox_close_interface</i> (Index 23).....	41
5.2.15. <i>canbox_open_identifier</i> (Index 25).....	41
5.2.16. <i>canbox_read_identifier_buffer</i> (Index 26).....	41
5.2.17. <i>canbox_read_identifier_actual</i> (Index 27).....	42
5.2.18. <i>canbox_write_identifier</i> (Index 28).....	42
5.2.19. <i>canbox_close_identifier</i> (Index 29).....	42
5.2.20. <i>canbox_open_filter</i> (Index 30).....	42
5.2.21. <i>canbox_read_filter_buffer</i> (Index 31).....	42
5.2.22. <i>canbox_read_filter_actual</i> (Index 27).....	42
5.2.23. <i>canbox_close_filter</i> (Index 32).....	43
5.2.24. <i>canbox_open_sender</i> (Index 13).....	43
5.2.25. <i>canbox_write_sender</i> (Index 24).....	43
5.2.26. <i>canbox_close_sender</i> (Index 14).....	43
5.2.27. <i>canbox_read_router_config</i> (Index 34).....	43
5.2.28. <i>canbox_write_router_config</i> (Index 35).....	44
5.2.29. <i>canbox_read_lan_config</i> (Index 36).....	44
5.2.30. <i>canbox_write_lan_config</i> (Index 37).....	44
5.2.31. <i>canbox_read_wlan_config</i> (Index 38).....	44
5.2.32. <i>canbox_write_wlan_config</i> (Index 39).....	44
6. TECHNICAL DATA	45

1. General Information

The CANbox® is a converter with two high-speed CAN buses (up to 1 MBit/s) that are compliant to CAN specification 2.0A and 2.0B. Fault-tolerant low-speed interfaces are available as an option. The connection to the CANbox® can be established via a serial interface, LAN or WLAN. The software that comes with the CANbox® is described in chapters 3 and 4. Drivers are available for Win32- and Pocket PC 2003-operating systems. Furthermore the socket interface can be used for a direct communication using the TCP/IP socket platform (requires at least firmware version 3.A).

Up to 10 CANboxes can be used simultaneously on a single PC or PDA.

The metal enclosure of the CANbox® measures only 113 x 83 x 33 mm. The CANbox® can be fixed either on DIN-rails or with bolts on any plain surface in case of stationary use. Power supply for the CANbox® is tailored to automotive and industry requirements. The voltage range is 6...60V (DC) which covers all automotive voltages (incl. 42V). The power supply is not galvanically insulated. Galvanic isolation of the CAN buses is achieved with the SORCUS standard I/O module X-CAN-2i.



1.1.Special Features

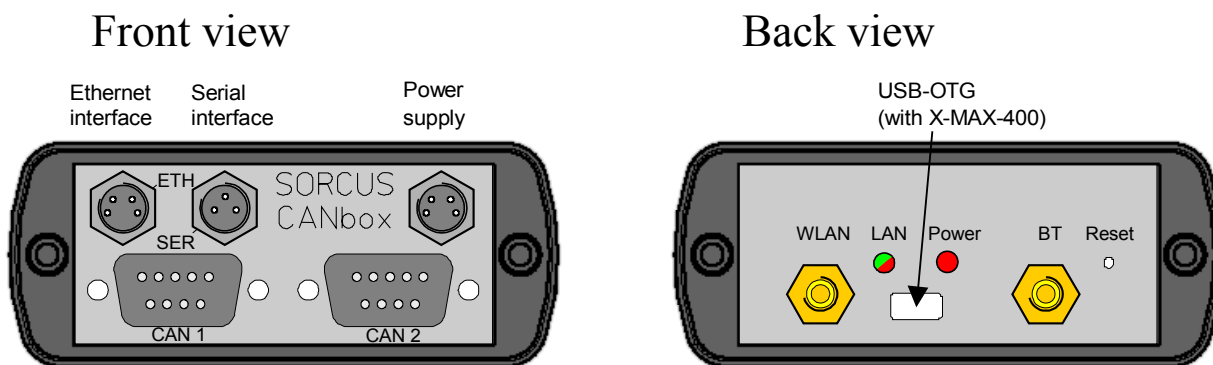
- Intelligent, decentralized, autonomous system
- Interface-converter CAN to WLAN/LAN
- WLAN and/or Bluetooth on-board
- Dimensions 113 x 83 x 33 mm
- Power supply: 6,0V...60V
- Mountable on DIN-rails or plain surfaces (Holder available optionally)

1.2.Installed MAX Modules

The device has two module-slots for MAX modules. The slots are equipped as described below.

Slot-No.	Module	Function	Explanation
1	X-MAX-E/ or X-MAX-400	CPU module with Ethernet 10 CPU module with Ethernet 100 and USB-OTG	This slot is used for a CPU module. Depending on the module type, USB is available in addition to serial and ethernet connections.
2	X-CAN-2i/H	2 CAN-channels with high speed drivers	Optionally, 2 fault-tolerant CAN channels or mixed are possible

1.3.External connectors



1.4.WLAN/LAN properties

1.4.1.WLAN

The WLAN standard 802.11b offers up to 14 channels depending on the region. It has to be considered that not all channels are free from overlap. If more than one WLAN network is used in the same area, different channels should be used with 4 channels free between the two channels.

Channel	Center frequency	Frequency range
1	2412 MHz	2399.5 MHz – 2424.5 MHz
2	2417 MHz	2404.5 MHz – 2429.5 MHz
3	2422 MHz	2409.5 MHz – 2434.5 MHz
4	2427 MHz	2414.5 MHz – 2439.5 MHz
5	2432 MHz	2419.5 MHz – 2444.5 MHz
6	2437 MHz	2424.5 MHz – 2449.5 MHz
7	2442 MHz	2429.5 MHz – 2454.5 MHz
8	2447 MHz	2434.5 MHz – 2459.5 MHz
9	2452 MHz	2439.5 MHz – 2464.5 MHz
10	2457 MHz	2444.5 MHz – 2469.5 MHz
11	2462 MHz	2449.5 MHz – 2474.5 MHz
12	2467 MHz	2454.5 MHz – 2479.5 MHz
13	2472 MHz	2459.5 MHz – 2484.5 MHz
14	2484 MHz	2471.5 MHz – 2496.5 MHz

WLAN channels and their frequencies

The distance range of a WLAN device is typically up to 30m inside (100m maximum) and 100m outside (maximum 300m).

1.4.2. Transfer rates

The following table shows the achieved transfer rates with the CANbox[®] in dependence of the transfer types. The CANbox[®] was **running idle**, the distance between the PC and the CANbox[®] was about 1m. This specification is meant to be as a guideline only.

Type of transfer	Transfer rate
Connection with WLAN-PC using AdHoc	140 kB/s
Connection with WLAN-PC using an AccessPoint	90 kB/s
Connection with LAN-PC using an AccessPoint	195 kB/s
Connection with LAN-PC via LAN	250 kB/s

2. Connectors of the CANbox[®]

2.1. Overview

The following external connectors are present:

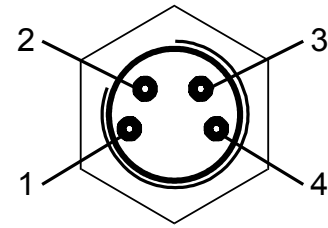
Connector	Type	Function
ST1	M8-Connector 4 pins	Power supply (6..60V)
ST2	M8-Connector 4 pins	Ethernet interface
ST3	M8-Connector 3 pins	Serial interface
CAN 1	D-SUB-9 male	CAN bus 1
CAN 2	D-SUB-9 male	CAN bus 2
WLAN	SMA female	WLAN antenna
BT	SMA female	Bluetooth antenna (with CPU module X-MAX-400 only)
USB-OTG	Mini USB-B	USB with OTG functionality (with CPU module X-MAX-400 only)

2.2. Assignment of the external connectors

The power supply and the host interfaces are connected by connectors of type M8. The cable with 3- or 4-pin plug in 2m and 5m length are available from many distributors. The cable with 2m length are also available from SORCUS. In addition, SORCUS offers cable equipped with D-SUB-9 socket for serial connection and with RJ45 plug for direct connection of Ethernet on a company network or a PC (Cross-Over).

The cable uses the following color assignment:

Color	3 Pin	4 Pin
Brown	1	1
White	-	2
Black	2	3
Blue	3	4

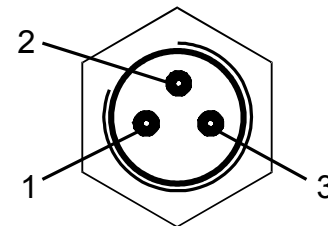


View of
M8 plug
from outside

The cable of the Hirschmann company have the following type designation:

- Cable M8 3 pins: ELKA-KV 3308
- Cable M8 4 pins: ELKA-KV 4408

Cables of other manufacturers (Binder, Phoenix etc.) use different declarations.



2.2.1. Connector ST 1, power supply

At M8 plug ST1, the power supply of the CANbox[®] has to be applied. Contacts 3 and 4 are to be used for connecting GND, contacts 1 and 2 supports power supplies with voltages from 6...60V. If the power supply is connected the wrong way a fuse will brake inside of the CANbox[®] which may be replaced by the SORCUS service only.

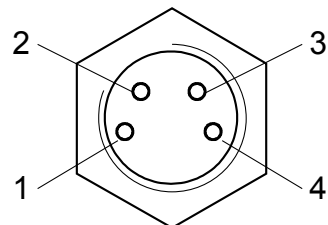
ST1	Signal	Cable Color
1,2	+Vbatt	Brown, White
3,4	-Vbatt (=XGND)	Black, Blue

Because of the integrated DC/DC converter, the current consumption is higher when using small input voltages than high input voltages.

2.2.2. Host Interfaces (Connectors ST 2 and ST 3)

2.2.2.1. Assignment Connector ST 2, M8 plug with 4 pins (Ethernet):

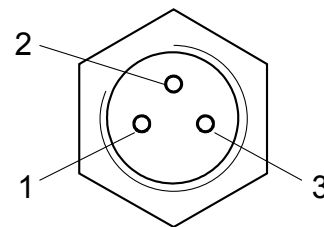
ST 2	Signal of CPU modul	Signal	RJ45 for Hub PC	
1	Connector A Pin 1	TX+	1	3
2	Connector A Pin 2	TX-	2	6
3	Connector A Pin 3	RX+	3	1
4	Connector A Pin 4	RX-	6	2



The assignment of the RJ45 plug is given for a connection to a hub and for direct connection to a PC.

2.2.2.2. Assignment Connector ST 3, M8 plug with 3 pins (serial host interface):

ST 3	Signal of CPU modul	Signal	Pin at D-SUB-9
1	Connector A Pin 8	TXD	2
2	Connector A Pin 6	RXD	3
3	ST5 Pin 10	XGND	5



The assignment for the D-SUB-9 socket is given for direct connection to a PC.

2.2.3. D-SUB-9 plug for CAN 1 and CAN 2

The two CAN interfaces are galvanically isolated from each other and the rest of the CANbox[®]. The options of the CAN interfaces are described in the software part of this manual.

D-SUB-9 plug for CAN channel 1	
Signal	Pin D-SUB
CAN1-GND	3
CAN1-L	2
CAN1-H	7

D-SUB-9 plug for CAN channel 2	
Signal	Pin D-SUB
CAN2-GND	3
CAN2-L	2
CAN2-H	7

2.2.4. SMA plug for antenna

At the back side of the CANbox[®], there is a SMA plug connector for a WLAN antenna. Here, various types of antennas can be connected. The following antennas are available among others:

- Short antenna (93mm) 90° angle with 2,1dBi gain
- Table antenna (190mm height) with 1,8m cable, 5dBi gain

3. Starting Instructions

3.1. CANbox[®] Setup

The CANbox[®] setup named CANbox.exe is to be used for the installation of the different components which are available for CANbox[®] operation. After the installation is started, a dialog for selection of the components appears. The following components are available:

- Win32 driver: Driver and configuration tool for CANbox[®] administration with Win32 operating systems
- Win32 library: Library for programming CANbox[®] access with Win32 operating systems (C, C++, VB6.0, VB.NET)
- Win32 samples: Examples for programming the CANbox[®] with Win32 operating systems
- Win32 tools: Tools for using the CANbox[®] with Win32 operating systems
- POCKET PC 2003 driver: Driver and configuration tool for CANbox[®] administration with POCKET PC 2003 operating systems
- POCKET PC 2003 SE driver: Driver and configuration tool for CANbox[®] administration with POCKET PC 2003 SE operating systems
- POCKET PC 2003 library: Library for programming CANbox[®] access with POCKET PC 2003 (SE) operating systems (C, C++, VB6.0, VB.NET)
- POCKET PC 2003 samples: Examples for programming the CANbox[®] with POCKET PC 2003 (SE) operating systems
- POCKET PC 2003 tools: Tools for using the CANbox[®] with POCKET PC 2003 (SE) operating systems
- Documentation: Documentation of the CANbox[®]

After the selection of the components, a path dialog for destination path selection appears. The selected components will be installed in sub directories of the selected path. The driver and the configuration tool will be installed in designated windows directories. After path selection, the installation of the components takes place.

All components are available in English language only, except this manual, which is also available in German language.

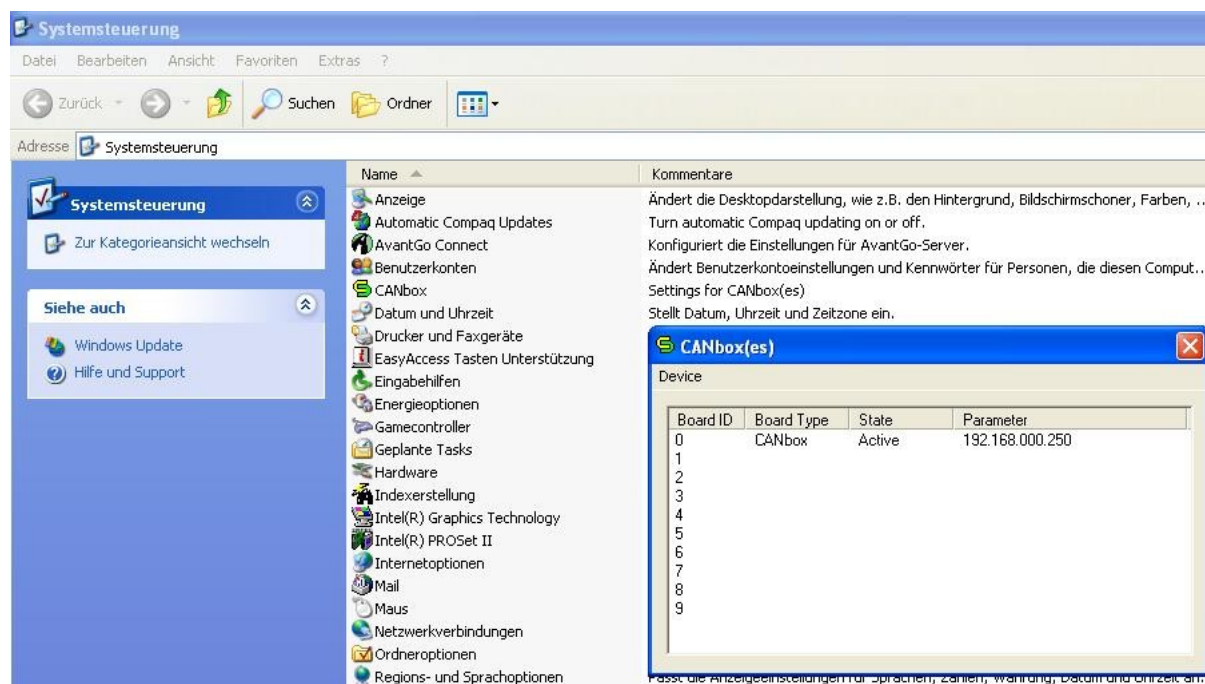
3.2. CANbox[®] Config Tool

3.2.1. Operation

The CANbox[®] Config Tool serves for administration and configuration of the CANbox(es). It is installed in the control panel with the name **CANbox[®]** and can be started from this place.

It contains a list of the installed CANboxes where each CANbox[®] is assigned an unequivocal ID. This ID is needed for the configuration of the router mode for example.

3.2.1.1. Win32 operating systems



CANbox[®] configuration at Win32 operating systems

After choosing an available **Board ID**, a CANbox[®] can be installed via the **Device** menu of the main menu respectively the context menu of the CANbox[®]; a double click on a CANbox[®] makes its configuration possible.

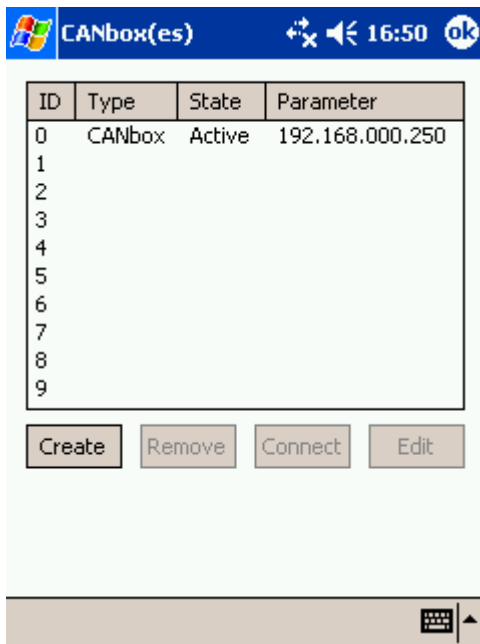
Moreover, a connection to a CANbox[®] is possible, if the communication parameter correspond with the configuration. After a successful connection, the communication parameters and some other settings can be changed. The changes take effect after restarting the CANbox[®].

The configuration of the CANbox[®] can also take place by using the serial interface. Here, a cross over connection between COM1 of the PC and the serial interface of the CANbox[®] is required. In this case, **Connect by COM-1** of the **Device** menu respectively the context menu can be used for building up a connection to the CANbox[®] and to inspect and configure its connection parameters. A firmware update is also possible if a new firmware version is available.

Furthermore, an existing router configuration can be reseted by **Reset Router Configuration** respectively **Reset Router Configuration by COM-1**. After this reset, the CANbox[®] has to be restarted.

Removal of CANboxes also takes place via the **Device** menu respectively the context menu.

3.2.1.2. POCKET PC 2003



CANbox® Configuration using POCKET PC 2003

After selecting an available **Board ID** the button **Create** respectively the context menu can be used for CANbox® installation; a double click on a CANbox or pressing the button **Edit** enables the configuration of the CANbox® connection.

Moreover, a connection to a CANbox® is possible, if the communication parameters correspond with the configuration. After a successful connection, the communication parameters and some other settings can be changed. POCKET PC 2003 does not offer the configuration of all available parameters.

The removal of CANboxes can also take place by using the button **Remove** or via the context menu.

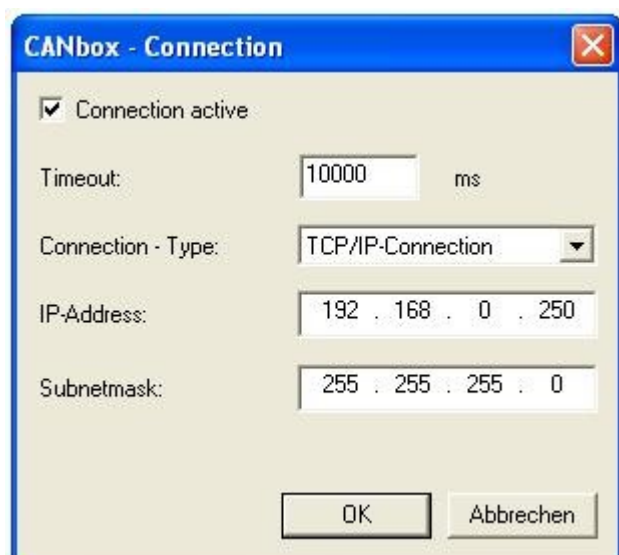
3.2.2. CANbox® Connection Dialog

Selecting the **Properties** menu of the **Device** menu respectively a double click on a CANbox® opens the CANbox® Connection dialog, where the connection parameters can be set.

With the help of the check box **Connection active** a connection can be set active respectively inactive. With the command **Connect** the connection to the CANbox® is established (*see next chapter*). An inactive CANbox® cannot be addressed.

Timeout describes the maximum duration of waiting for an answer from the CANbox®.

With **Connection Type** the type of connection is set. For TCP/IP connections the IP address together with the subnetmask for the CANbox® has to be specified.



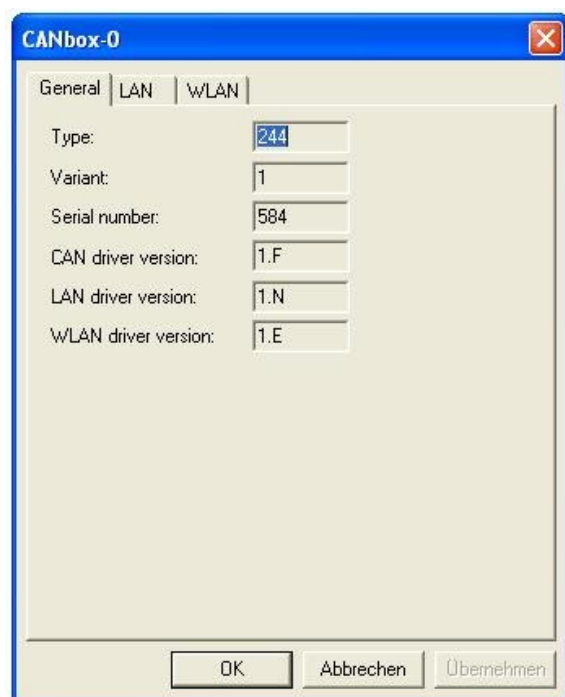
CANbox[®] Connection dialog at Win32 operating systems

3.2.3. CANbox[®] Configuration Dialog

The menu **Connect** respectively **Connect by COM-1** offers the possibility of accessing the CANbox[®] for testing and configuration purposes. Note: If parameters are changed, these will take effect after a restart of the CANbox[®].

3.2.3.1. General

This property page shows general information about the CANbox[®] and versions of the installed drivers. If the firmware of the CANbox[®] is older than the configuration tool, a firmware update can be done by activating the check box **Update**. After pressing the OK button, the firmware is transferred into the CANbox[®] and will be activated after a restart of the CANbox[®].



CANbox[®] Connection dialog at Win32 operating systems

3.2.3.2.LAN

The property page LAN offers all necessary parameters for a conventional LAN connection:

Element	Meaning	Default
IP-Address	IP address of the CANbox [®]	192.168.000.240
Subnetmask	Subnetmask of the CANbox [®]	255.255.255.000

Changes of these parameters will be activated by restarting the CANbox[®].

3.2.3.3.WLAN

The property page WLAN contains all needed parameters for a wireless LAN connection:

Element	Meaning	Default
IP-Address	IP address of the CANbox [®]	192.168.000.250
Subnetmask	Subnetmask of the CANbox [®]	255.255.255.000
SSID	Networkname with up to 34 characters	CANbox
Mode	AdHoc connection or connection via Access Point	AdHoc
Channel	WLAN channel from 1 to 14	10
Authentication	Networkauthentication <i>Open System</i> or <i>Shared Key</i>	Open System
WEP-Encryption	Data encryption <i>Inactive</i> or <i>WEP</i>	Inactive
Key0	64 bits or 128 bits network key	
Key1	64 bits or 128 bits network key	
Key2	64 bits or 128 bits network key	
Key3	64 bits or 128 bits network key	
KeyId	Index of used key	0

The default parameters should be changed directly after the installation process in order to prevent unallowed access to the CANbox[®] by third persons.

Input and presentation of the network keys takes place in the form of a password. For guaranteeing correct input those values have to be set twice.

Changes of these parameters will be activated by restarting the CANbox[®].

3.2.3.4.Configuration example for Windows XP

The following example describes the configuration for a WLAN ad hoc connection between a PC and a CANbox[®] with default settings. This configuration takes place without using third party software and assumes a correctly installed and active WLAN adapter.

- Open the network connections by selecting **Show all connections** in the menu **Connect with** of the **Start** menu
- Select the **Wireless network connection** which has to be used with the CANbox[®] and open its property dialog (conext menu - **Properties**)

- Activate **Internet protocol (TCP/IP)** on the page **General** in the list **This connection uses the following elements** and push the **Properties** button
- Choose the radio button **Use the following IP address**, enter a fitting **IP address** (e.g. 192.168.0.200) with **Subnetmask** (255.255.255.0) and Leave the dialog with the **OK** button.

If the network **CANbox[®]** at the available networks (**Wireless networks – Available networks – Show wireless networks**) is not visible, you have to create it by executing the following steps otherwise you only have to select it.

- Switch to the page **Wireless networks** and push the button **Add**.
- Enter **CANbox[®]** the Edit **Networkname (SSID)** of the page **Assignment** and deactivate the **data encryption**. Activate the check box **This is an computer to computer network (Ad hoc)**.
- Leave the dialog with the **OK** button.

Please pay attention that no other network uses an IP address of type 192.168.0.xxx.

3.2.3.5. Configuration example for POCKET PC 2003

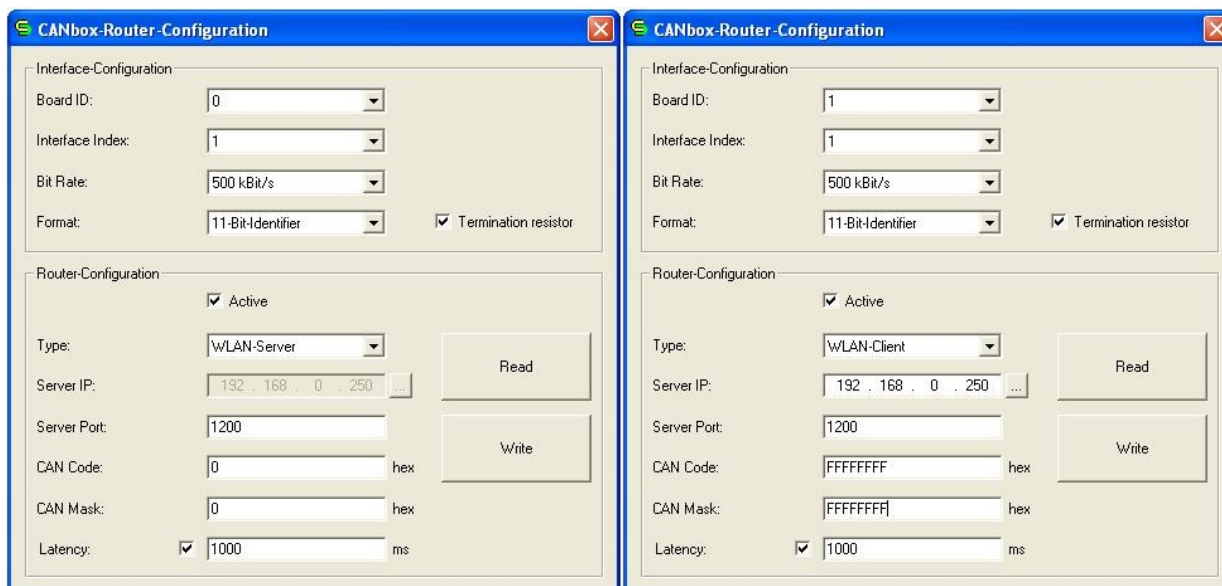
The following example describes the configuration for a WLAN ad hoc connection between a Toshiba e800 with integrated WLAN and a CANbox[®] with default settings. This configuration may be similar for other PDAs but it could be different in some kind.

- Set the PDA switch for wireless communication to **ON**.
- The **Wireless LED** of the PDA should light orange.
- Select the menu **Settings** in the PDA **Start Menu** and switch to the page **Connection**.
- Click the symbol **Connection** and change to the page **Extended** in the following dialog.
- Push the button **Network adapter**.
- On the page **Wireless** click on **New Settings** in the box **Wireless Networks**.
- A dialog with the pages **General** and **Authentication** will appear.
- On the page **General** you type **CANbox** for the **Network Name**, choose **Connection with Company** and activate the check box **This is an Ad Hoc Connection**.
- On the page **Authentication** deactivate the check box **Encrytion (WEP activated)**.
- Leave the dialog by using the **OK** button.
- Switch to the page **Network Adapters** and select **IEEE 802.11b WLAN Adapter** at the **Adapters** using the pen.
- Activate the radio button **Specific IP Address** and put in a fitting **IP Address** (for example 192.168.0.200) with **Subnetmask** (255.255.255.0).
- Confirm your inputs with the **OK** button and move back to the **Start Page**.
- Switch on the **CANbox[®]** and build up a connection using the **CANbox[®]** symbol in the control panel.
- If the connection does not work restart the PDA with a soft reset.

3.3.CANbox[®] as Router

Firmware version 3.A and later offers the possibility to use the CANbox[®] as router. In this case, two CANboxes can be connected to each other via LAN/WLAN. Both CANboxes are configured for reception of all identifiers or certain identifiers or exclusively for sending by using a CAN acceptance filter. The optimal parameters for the acceptance filter can be determined by using the tool *AcceptanceFilterCalculator* which is installed with the CANbox[®] tools. All CAN messages that pass the acceptance filter are sent to the other CANbox[®] via LAN/WLAN where they are passed to the CAN bus. The settings of the router mode are stored in the EEPROM of the CANbox[®]. These settings are read when the CANbox[®] is switched on and the router is activated if necessary. Each CAN interface offers a separate router functionality.

The router mode can be configured by using library functions and/or with the tool „Router-Configuration“ which is included in the CANbox[®] software. Therefore, the CANboxes have to be installed in the control panel and a connection via TCP/IP is possible. In the configuration one CANbox[®] has to be set to TCP/IP server and one CANbox[®] to TCP/IP client. The server port has to be identical in both CANboxes and can have a value between 1200 and 65535. At the TCP/IP client the IP address of the TCP/IP server CANbox[®] has to be set for building up the connection. If a CANbox[®] as Router is not wanted to receive data from the local CAN bus, the acceptance filter has to be set to the value 0xFFFFFFFF. The router configuration of an interface can be read by selecting the desired Board ID (corresponding to the configuration in the control panel) and interface index by using the **Read** button. The interface parameters have to be set to the actually used ones, if the interfaces router is activated. By pressing the button **Write**, the actual dialog settings are transferred to the CANbox[®]. A message box informs about the successful transmission.



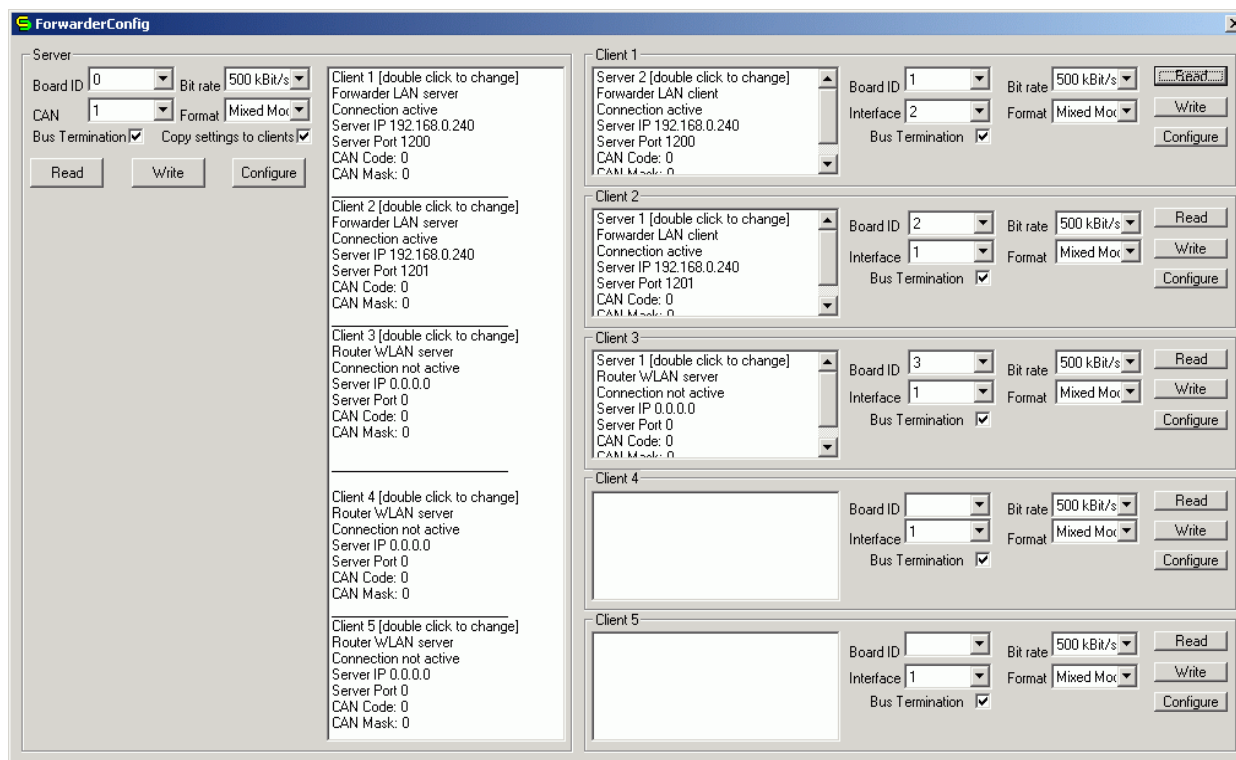
Configuration tool RouterConfiguration

With firmware version 3.H or higher the router mode offers the possibility of setting a maximum latency. All messages arriving at the CAN sender, which possess a time stamp older than the configured latency allows, are rejected. Both CANbox[®] es have to be configured to the same mode (latency off/on).

3.4.Enhanced Router (Forwarder)

The enhanced router allows it to define a CANbox®-Server that can handle up to 10 Clients. This mode is available from FW version 3.M. The configuration is done using the tool „ForwarderConfig“ shipped with the CANbox®. Necessary parameters are identical to those used with the normal router described above. The server keeps 10 sets of parameters (5 per CAN interface) one for each connected client. A client keeps 2 sets of parameters (1 per CAN interface).

To read or write a configuration the used CAN interface has to set up first (bit rate, address format, bus termination). This settings are later used for the router operation. If the server and the clients will use the same bus parameters, the settings can be automatically copied from the server setup to the client setup by checking „Copy settings to clients“. The current configuration stored in the EEPROM of a CANbox® can be read by clicking the button „Read“. The configuration parameters are the displayed in a list. By clicking „Configure“ or double-clicking a line marked with „[double click to change]“ opens a dialog which allows to change the parameters. Clicking „Write“ writes the configuration back into the EEPROM. After writing the configuration the CANbox®es need to be restarted in order to get the routing activated. It can take approx. 60s until the clients start to establish connections to the server



Configuration tool *ForwarderConfig.exe*

4. CANbox[®] library

The library offers all functions which are necessary for a CAN communication. All functions give an error code as return value. A function returns CB_ERROR_OK, if it has been executed successfully. The CANbox[®] library allows hardware configuration and initialization. Furthermore, the library provides functions for reception and transmission of CAN identifiers. The library is available for Win32 as well as for POCKET PC 2003 operating systems.

4.1. Structures and Definitions

4.1.1. CANBOX_VERSION_INFO

Structure for version information.

4.1.1.1. Parameter

USHORT	usBuild:	Build number
UCHAR	ucMinor:	Minor version number
UCHAR	ucMajor:	Major version number
UCHAR	ucDay:	Day of build
UCHAR	ucMonth:	Month of build
USHORT	usYear:	Year of build

4.1.2. CANBOX_TIME_STAMP

Structure with 64 bits time stamp.

4.1.2.1. Parameter

ULONG	ulTimeLo:	Low-Dword of the time stamp
ULONG	ulTimeHi:	High-Dword of the time stamp

4.1.3. CANBOX_SCAN_INFO

Structure for device information of an installed CANbox.

4.1.3.1. Elemente

USHORT	usDeviceId:	ID of the CANbox device
USHORT	usConnection:	Type of connection
ULONG	ulTimeout:	Timeout value

4.1.4. CANBOX_DEVICE_INFO

Structure for device information of a connected CANbox.

4.1.4.1. Parameter

USHORT	usType:	Type of the CANbox
--------	---------	--------------------

USHORT	usVariant:	Variant of the CANbox
ULONG	ulSerial:	Serial number of the CANbox
CANBOX_VERSION_INFO	sDriverCan:	Version of the CAN driver
CANBOX_VERSION_INFO	sDriverLan:	Version of the LAN driver
CANBOX_VERSION_INFO	sDriverWlan:	Version of the WLAN driver

4.1.5.CANBOX_RESET_EX

Structure to be used for the extended reset function.

4.1.5.1.Parameter

USHORT	usRouterRestart:	Restart of the router after reset
--------	------------------	-----------------------------------

4.1.6.CANBOX_INTERFACE

Structure for interface configuration with declaration of the bit rate.

4.1.6.1.Parameter

ULONG	ulBufferSize- Max:	Maximum count of messages stored by the CANbox [®] (0-150.000; 0: available maximum).
USHORT	usIndex:	Index of the interface (1, 2 when using the library or 0, 1 when using the socket interface)
USHORT	usBitRate:	Bit rate
USHORT	usShutOff:	Switching on / off of the CAN bus termination resistance
USHORT	usFormat:	CAN specification (11 bit / 29 bit identifier)

4.1.6.2.Bit rates

CB_BITRATE_0010_KBIT:	10 kBit/s
CB_BITRATE_0020_KBIT:	20 kBit/s
CB_BITRATE_0050_KBIT:	50 kBit/s
CB_BITRATE_0080_KBIT:	80 kBit/s
CB_BITRATE_0083_3_KBIT:	83.3 kBit/s
CB_BITRATE_0100_KBIT:	100 kBit/s
CB_BITRATE_0125_KBIT:	125 kBit/s
CB_BITRATE_0250_KBIT:	250 kBit/s
CB_BITRATE_0500_KBIT:	500 kBit/s
CB_BITRATE_0666_KBIT:	666 kBit/s
CB_BITRATE_0800_KBIT:	800 kBit/s
CB_BITRATE_1000_KBIT:	1 MBit/s

4.1.6.3.CAN bus termination resistor

CB_NO_SHUT_OFF:	Off
CB_SHUT_OFF:	On

4.1.6.4.Format

CB_FORMAT_MIXED:	11 bit and 29 bit identifier
CB_FORMAT_11_BIT:	11 bit identifier
CB_FORMAT_29_BIT:	29 bit identifier
CB_FORMAT_STD:	Standard mode
CB_FORMAT_BUFFER:	Buffer mode: Each identifier respectively acceptance filter has a configurable buffer size to be set by usBufferSize.

4.1.7.CANBOX_INTERFACE_EX

Extended structure for interface configuration with direct declaration of the bit timing parameters.

4.1.7.1.Parameter

ULONG	ulBufferSizeMax:	Maximum count of messages stored by the CANbox [®] (0-150.000; 0: available maximum).
USHORT	usIndex:	Index of the interface (1, 2 when using the library or 0, 1 when using the socket interface)
USHORT	usTq:	Time quantum in nano seconds
USHORT	usTseg1:	Time before the nominal scan time
USHORT	usTseg2:	Time after the nominal scan time
USHORT	usTsjw:	Synchronisation jump width
USHORT	usSpB:	Number of samples per bit
USHORT	usShutOff:	Switching on / off of theCAN bus termination resistance
USHORT	usFormat:	CAN specification (11 bit / 29 bit identifier)

4.1.7.2.CAN bus termination resistance

CB_NO_SHUT_OFF:	Off
CB_SHUT_OFF:	On

4.1.7.3.Format

CB_FORMAT_MIXED:	11 bit and 29 bit identifier
CB_FORMAT_11_BIT:	11 bit identifier
CB_FORMAT_29_BIT:	29 bit identifier
CB_FORMAT_STD:	Standard mode
CB_FORMAT_BUFFER:	Buffer mode: Each identifier respectively acceptance filter has a configurable buffer size to be set by usBufferSize.

4.1.8.CANBOX_INTERFACE_STATE

Structure for state information of an interface.

4.1.8.1. Parameter

CANBOX_TIME_STAMP	sTime:	Time stamp of the last change in bus state
USHORT	usBusState:	Bus state
USHORT	usBusInfo:	Information concerning the bus state
USHORT	usUserState:	User state

4.1.8.2. Bus state

CB_BUS_STATE_ON:	CAN interface is taking part in bus communication.
CB_BUS_STATE_HEAVY:	Bus is heavily disturbed.
CB_BUS_STATE_OFF:	CAN controller does not take part in bus communication any longer because of too many bus errors.
CB_BUS_STATE_LINE:	Line error signaled from transceiver (only for fault tolerant interfaces).

4.1.8.3. Bus info

CB_BUS_INFO_STUFF:	Stuff error (more than 5 equal bits in a message in a row).
CB_BUS_INFO_FORM:	Form error (format error in the message).
CB_BUS_INFO_ACK:	Ack error (sent message has not been acknowledged).
CB_BUS_INFO_BIT_LO:	Bit error 0 (after sending a message, 0 has been read back instead of 1).
CB_BUS_INFO_BIT_HI:	Bit error 1 (after sending a message, 1 has been read back instead of 0).
CB_BUS_INFO_CRC:	CRC error (bad check sum).

4.1.8.4. User state

CB_USER_STATE_OFF:	Interface has not been configured yet.
CB_USER_STATE_STARTED:	Interface has been configured and started.
CB_USER_STATE_STOPPED:	Interface has been configured and stopped.

4.1.9. CANBOX_INTERFACE_READ

Structure for reception data of an interface.

4.1.9.1. Parameter

CANBOX_TIME_STAMP	sTime:	Time stamp
ULONG	uId:	CAN Id of the identifier
ULONG	ulDataSize:	Number of valid data bytes
UCHAR	ucData[8] (ucData0 to ucData7 at VB):	Data bytes of the identifier

4.1.10. CANBOX_INTERFACE_WRITE

Structure for sending data of an interface.

4.1.10.1. Parameter

CANBOX_TIME_STAMP	sTime:	Time stamp
CANBOX_HANDLE	hIdentifier:	Handle of the identifier to be sent

ULONG	ulDataSize:	Number of valid data bytes for sending
UCHAR	ucData[8] (ucData0 to ucData7 at VB):	Data bytes of the identifier

4.1.11.CANBOX_IDENTIFIER

Structure for identifier configuration.

4.1.11.1.Parameter

ULONG	ulId:	Id of the Identifier. In mixed mode, the most significant bit can be set for marking 29 bit identifiers. This bit is then set in the received data.
USHORT	usType:	Type of the identifier.
USHORT	usFlags:	Options for the identifier.
USHORT	usTimeout:	Maximum time in μ s of waiting for the answer for active receivers.
USHORT	usReserved:	Reserved parameter which has to be filled with the zero based index of the interface when using the socket interface.
USHORT	usBufferSize:	Buffer size. This parameter has no effect when using the socket interface.

4.1.11.2.Types

CB_DEVICE_RECEIVER:	Standard receiver (passive)
CB_DEVICE_RECEIVER_ACTIVE:	Active receiver
CB_DEVICE_TRANSMITTER:	Standard sender (active)
CB_DEVICE_TRANSMITTER_PASSIVE:	Passive sender (not available in mixed mode)

4.1.11.3.Options

CB_FLAG_ACKNOWLEDGE:	Creation of acknowledgements for senders
CB_FLAG_EXCLUSIVE:	Exclusive buffer reservation on the CAN controller (not available in mixed mode)
CB_FLAG_FORMAT_29_BIT:	Acceptance filter for 29 bit identifiers in mixed mode

4.1.12.CANBOX_ACCEPTANCE_FILTER

Structure for configuration of the acceptance filter. An acceptance filter allows the reception of a group of CAN identifiers. All identifiers are received which fulfill the criterion $(ID \& Filter.ulMask) == (Filter.ulId \& Filter.ulMask)$. If the identifiers 0xA (10dec) and 0xB (11dec) are to be received, the acceptance filter has to be set to $ulId=0x0000000A$ and $ulMask=0x0000000E$.

4.1.12.1.Parameter

ULONG	ulId:	Identifier of the filter. In mixed mode, the most significant bit can be set for marking 29 bit identifiers. This bit is then set in the received data.
ULONG	ulMask:	Bit mask of the filter.
USHORT	usType:	Type of the filter.
USHORT	usFlags:	Options for the filter.
USHORT	usReserved:	Reserved parameter which has to be filled with the zero based index of the interface when using the socket interface.

USHORT usBufferSize: Buffer size. This parameter has no effect when using the socket interface.

4.1.12.2. Types

CB_FILTER_DATA: Acceptance filter for data frames.
 CB_FILTER_REMOTE: Acceptance filter for remote frames (not available in mixed mode).

4.1.12.3. Options

CB_FLAG_FORMAT_29_BIT: Acceptance filter for 29 bit identifiers in mixed mode.

4.1.13. CANBOX_ROUTER

Structure for router configuration. A router allows a connection between two CANboxes via LAN or WLAN respectively. A group of identifiers (see acceptance filter) can be received and sent over LAN/WLAN to the other CANbox. This CANbox sends the data which has been received via LAN/WLAN to the CAN bus.

4.1.13.1. Parameter

USHORT	usIndex:	Reserved parameter which has to be set to the value 1.
USHORT	usType:	Type of the router.
USHORT	usMode:	Mode of the router.
USHORT	usFlags:	Options for the router.
ULONG	ulServerIp:	IP address of the server CANbox.
USHORT	usServerPort:	Port address of the server CANbox.
USHORT	usReserved:	Reserved parameter which has to be filled with the zero based index of the interface when using the socket interface.
ULONG	ulCanId:	Identifier of the CAN acceptance filter (0xFFFFFFFF: no data will be received on the local CAN bus).
ULONG	ulCanMask:	Bit mask of the CAN acceptance filter.

4.1.13.2. Types

CB_ROUTER_LAN_SERVER: Connection via LAN as TCP/IP server.
 CB_ROUTER_LAN_CLIENT: Connection via LAN as TCP/IP client.
 CB_ROUTER_WLAN_SERVER: Connection via WLAN as TCP/IP server.
 CB_ROUTER_WLAN_CLIENT: Connection via WLAN as TCP/IP client.

4.1.13.3. Mode

CB_ROUTER_INACTIVE: Router is not activated.
 CB_ROUTER_ACTIVE: Router is activated.

4.1.13.4. Options

CB_FLAG_FORMAT_29_BIT: Acceptance filter for 29 bit identifiers in mixed mode.

4.1.14. CANBOX_ROUTER_EX

Structure for router configuration with latency consideration. A router allows a connection between two CANboxes via LAN or WLAN respectively. A group of identifiers (see acceptance filter) can be received and sent over LAN/WLAN to the other CANbox. This CANbox sends the data which has been received via LAN/WLAN to the CAN bus.

4.1.14.1. Parameter

USHORT	usIndex:	normal mode: Reserved parameter which has to be set to the value 1 enhanced mode: Zero based index of the requested data set (0..9)
USHORT	usType:	Type of the router.
USHORT	usMode:	Mode of the router.
USHORT	usFlags:	Options for the router.
ULONG	ulServerIp:	IP address of the server CANbox.
USHORT	usServerPort:	Port address of the server CANbox.
USHORT	usReserved:	Reserved parameter which has to be filled with the zero based index of the interface when using the socket interface.
ULONG	ulCanId:	Identifier of the CAN acceptance filter (0xFFFFFFFF: no data will be received on the local CAN bus).
ULONG	ulCanMask:	Bit mask of the CAN acceptance filter.
ULONG	ulLatency:	Maximum latency time for sending the message to the CAN bus. (5- 10.000 ms)
ULONG	ulFree:	Reserved parameter, which has to be set to zero.

4.1.14.2. Types

CB_ROUTER_LAN_SERVER:	Connection via LAN as TCP/IP server
CB_ROUTER_LAN_CLIENT:	Connection via LAN as TCP/IP client
CB_ROUTER_WLAN_SERVER:	Connection via WLAN as TCP/IP server
CB_ROUTER_WLAN_CLIENT:	Connection via WLAN as TCP/IP client
CB_FORWARDER_LAN_SERVER:	Connection via LAN as TCP/IP server (enhanced mode)
CB_FORWARDER_LAN_CLIENT:	Connection via LAN as TCP/IP client (enhanced mode)
CB_FORWARDER_WLAN_SERVER:	Connection via WLAN as TCP/IP server (enhanced mode)
CB_FORWARDER_WLAN_CLIENT:	Connection via WLAN as TCP/IP client (enhanced mode)

4.1.14.3. Mode

CB_ROUTER_INACTIVE:	Router is not activated.
CB_ROUTER_ACTIVE:	Router is activated.

4.1.14.4. Options

CB_FLAG_ROUTER_SYNC_MODE:	Flag for activating the latency time consideration
CB_FLAG_FORMAT_29_BIT:	Acceptance filter for 29 bit identifiers in mixed mode
CB_FLAG_FORMAT_11_BIT:	Acceptance filter for 11 bit identifiers in mixed mode
CB_FLAG_FORMAT_MIXED:	Open both 11 and 29 bit acceptance filter

4.1.15.CANBOX_LAN_CONFIG

Structure for setting respectively getting the LAN settings of the CANbox[®].

4.1.15.1.Parameter

ULONG	ullp:	IP address of the CANbox [®] .
ULONG	ulSubnetmask:	Subnetmask of the CANbox [®] .

4.1.16.CANBOX_WLAN_CONFIG

Structure for setting respectively getting the WLAN settings of the CANbox[®].

4.1.16.1.Parameter

ULONG	ullp:	IP address of the CANbox [®] .
ULONG	ulSubnetmask:	Subnetmask of the CANbox [®] .
UCHAR[34]	ucSSID:	Networkname with up to 34 characters.
USHORT	usMode:	Type of connection (AdHoc, Access Point).
USHORT	usOwnChannel:	WLAN channel from 1 to 14.
USHORT	usAuthentication:	Authentication <i>Open System</i> or <i>Shared Key</i> .
USHORT	usEncryption:	Encrypton <i>Switched Off</i> or <i>WEP</i> .
USHORT	usKeyId:	Zero based index of the used key.
UCHAR[16]	ucKey0:	Key 0 (usKeyId=0).
UCHAR[16]	ucKey1:	Key 1 (usKeyId=1).
UCHAR[16]	ucKey2:	Key 2 (usKeyId=2).
UCHAR[16]	ucKey3:	Key 3 (usKeyId=3).

4.1.16.2.Mode

CB_WLAN_ACCESS_POINT	Connection via access point as TCP/IP server.
CB_WLAN_AD_HOC	AdHoc connection.

4.1.16.3.Authentication

CB_WLAN_OPEN_SYSTEM	Open System.
CB_WLAN_SHARED_KEY	Shared Key.

4.1.16.4.Encryption

CB_WLAN_ENCRYPTION_OFF	Switched off.
CB_WLAN_ENCRYPTION_WEP	WEP.

4.1.17.CANBOX_IDENTIFIER_DATA

Structure with CAN identifier data.

4.1.17.1.Parameter

CANBOX_TIME_STAMP	sTime:	Time stamp at time of reception.
-------------------	--------	----------------------------------

ULONG	ulId:	CAN Id of the identifier. In mixed mode the most significant bit is set for 29 bit identifiers if it was set at opening the identifier.
ULONG	ulDataSize:	Number of valid data bytes
UCHAR	ucData[8] (ucData0 bis ucData7 bei VB):	Data bytes

4.1.18. CANBOX_UNIVERSAL_SENDER_DATA

Structure for sending a CAN message by using a general sending object. With a general sending object, any identifier can be sent by declaration of the Id.

4.1.18.1. Parameter

ULONG	ulId:	CAN identifier. In mixed mode the most significant bit can be used for marking 29 bit identifiers.
USHORT	usFlags:	Options.
ULONG	ulDataSize:	Number of data to be sent.
UCHAR	ucData[8] (ucData0 to ucData7 at VB):	Data to be sent.

4.1.18.2. Options

CB_FLAG_FORMAT_29_BIT:	Flag for marking a 29 bit identifier in mixed mode.
------------------------	---

4.1.19. CANBOX_16

Supporting structure for using the CANbox[®] with Visual Basic.

4.1.19.1. Parameter

UCHAR	ucLoByte:	Lower byte.
UCHAR	ucHiByte:	Higher byte.

4.1.20. CANBOX_32

Supporting structure for using the CANbox[®] with Visual Basic.

4.1.20.1. Parameter

UCHAR	ucLoByte:	Lowest byte.
UCHAR	ucByte1:	Higher byte of the lower word.
UCHAR	ucByte2:	Lower byte of the upper word.
UCHAR	ucHiByte:	Highest byte.

4.2. Error codes

CB_ERROR_OK:	No error
--------------	----------

4.2.1. General error messages of the CANbox[®]

CB_ERROR_SIZE_INPUT:	Size of input data invalid
CB_ERROR_SIZE_OUTPUT:	Size of output data invalid

CB_ERROR_DATA:	Error in given data
CB_ERROR_STATE:	Invalid state
CB_ERROR_MEMORY:	Error at accessing RAM memory
CB_ERROR_FLASH:	Error at accessing the FLASH memory
CB_ERROR_EEPROM:	Error at EEPROM access
CB_ERROR_NOT_SUPPORTED:	Function not supported (check library and firmware version)

4.2.2. Error messages concerning initialisation

CB_ERROR_CAN_INIT:	Error at initializing the CAN firmware
CB_ERROR_HARDWARE_INIT:	Error at initializing the CPU
CB_ERROR_MEMORY_INIT:	Error at memory allocation

4.2.3. Error messages concerning release

CB_ERROR_CAN_EXIT:	Error at releasing the CAN mdd
--------------------	--------------------------------

4.2.4. Error messages concerning CAN interfaces

CB_ERROR_INTERFACE_INDEX:	Invalid interface index
CB_ERROR_INTERFACE_STATE:	Invalid interface state (for the desired operation)
CB_ERROR_INTERFACE_OPEN:	Error at opening the interface
CB_ERROR_INTERFACE_OPEN_BITRATE:	The interface has already been opened with a different bit rate.
CB_ERROR_INTERFACE_OPEN_FORMAT:	The interface has already been opened with a different message format (11bits/29 bits)
CB_ERROR_INTERFACE_BITRATE:	Unsupported bit rate
CB_ERROR_INTERFACE_SHUT_OFF:	Invalid value for the terminating resistance
CB_ERROR_INTERFACE_FORMAT:	Invalid value for the CAN format.
CB_ERROR_INTERFACE_INIT:	Error at initializing a CAN interface
CB_ERROR_INTERFACE_CONTROL:	Error at controlling a CAN interface
CB_ERROR_INTERFACE_EXIT:	Error at clearing a CAN interface

4.2.5. Error messages concerning CAN identifier

CB_ERROR_IDENTIFIER_INIT:	Error at initializing a CAN identifier
CB_ERROR_IDENTIFIER_TYPE:	Invalid identifier type
CB_ERROR_IDENTIFIER_EXCLUSIVE:	Invalid value for exclusive flag
CB_ERROR_IDENTIFIER_READ:	Error at reading data
CB_ERROR_IDENTIFIER_WRITE:	Error at writing data

4.2.6. Error messages concerning acceptance filters

CB_ERROR_FILTER_INIT:	Error at initializing an acceptance filter
CB_ERROR_FILTER_TYPE:	Invalid filter type
CB_ERROR_FILTER_SET:	Error at setting the filter configuration

4.2.7. Error messages concerning universal senders

CB_ERROR_SENDER_INIT:	Error at initializing an universal sender
CB_ERROR_SENDER_WRITE:	Error at writing data of an universal sender

4.2.8. General error messages of the library

CB_ERROR_NO_INIT:	Missing initialisation.
CB_ERROR_PARAMETER:	Wrong parameter.
CB_ERROR_HANDLE:	Invalid handle.
CB_ERROR_HANDLE_TYPE:	Invalid handle type.

4.2.9. Error messages of the library concerning the connection to the CANbox[®]

CB_ERROR_CONNECTION_INIT:	Connection could not be established.
CB_ERROR_CONNECTION_IN_USE:	CANbox [®] is already connected to another application.
CB_ERROR_CONNECTION_TIMEOUT:	Timeout while accessing the CANbox [®] .
CB_ERROR_CONNECTION_RECEIVE:	Error at data reception from the CANbox [®] .
CB_ERROR_CONNECTION_SEND:	Error at sending data to the CANbox [®] .
CB_ERROR_CONNECTION_EXIT:	Connection could not be shut down.
CB_ERROR_DEVICE_NOT_INSTALLED:	The addressed CANbox [®] is not installed in the control panel.

4.3. General Functions

4.3.1. canbox_init_lib

Initialisation of the library. This function has to be called once before any other library function. Otherwise all functions return the error CB_NO_INIT.

4.3.1.1. Parameter

None.

4.3.2. canbox_get_driver_version

Delivers information about the driver versions.

4.3.2.1. Parameter

CANBOX_VERSION_INFO*	psInfo:	Pointer to structure for version information
----------------------	---------	--

4.3.3. canbox_get_error_message

Retrieves the error message for an error code.

4.3.3.1. Parameter

CANBOX_ERROR	Error:	Error code.
USHORT*	pusSize:	Size of the character array.
char*	szMessage:	Character array for message text.

4.3.4.canbox_exit_lib

Free allocated resources. This function has to be called at the end.

4.3.4.1.Parameter

None

4.4.Device Functions

4.4.1.canbox_scan_devices

Gives a list of the installed CANboxes.

4.4.1.1.Parameter

USHORT*	pusCount:	Maximum number of device information structures. The driver sets the number to the used count.
CANBOX_SCAN_INFO*	pasInfo:	Pointer to array of info structures for device information.

4.4.2.canbox_open_device

Opens the device with the given Id. The connection to the device is built up.

4.4.2.1.Parameter

USHORT	usId:	Id of the device to be opened.
CANBOX_HANDLE*	phDevice:	Pointer to handle for the device.

4.4.3.canbox_reset_device

Resets the device. All open identifiers and interfaces of the device are closed and the bus communication of the interfaces is stopped. The receive buffers are cleared. The time stamp is reset to zero. The connection maintains alive and the device handle keeps valid. Active routers are restarted after reset.

4.4.3.1.Parameter

CANBOX_HANDLE	hDevice:	Received handle of the device after calling canbox_open_device.
---------------	----------	---

4.4.4.canbox_reset_device_ex

Resets the device. All open identifiers and interfaces of the device are closed and the bus communication of the interfaces is stopped. The receive buffers are cleared. The time stamp is reset to zero. The connection maintains alive and the device handle keeps valid. Additionally it is possible to determine, if active routers are to be restarted after reset.

4.4.4.1.Parameter

CANBOX_HANDLE	hDevice:	Received handle of the device after calling canbox_open_device.
CANBOX_RESET_EX	sResetEx:	Structure for configuring the reset behaviour.

4.4.5.canbox_get_device_info

Gives information about the device.

4.4.5.1. Parameter

CANBOX_HANDLE hDevice: Received handle of the device after calling canbox_open_device.
CANBOX_DEVICE_INFO* psInfo: Pointer to an info structure for device information.

4.4.6. canbox_reset_time_stamp

Resets the time stamp to zero. The time stamp has a resolution of 0.1 micro seconds.

4.4.6.1. Parameter

CANBOX_HANDLE hDevice: Received handle of the device after calling canbox_open_device.

4.4.7. canbox_set_time_stamp

Sets the time stamp to the given value. The time stamp has a resolution of 0.1 micro seconds.

4.4.7.1. Parameter

CANBOX_HANDLE hDevice: Received handle of the device after calling canbox_open_device.
CANBOX_TIME_STAMP sTime: Structure with new time stamp value.

4.4.8. canbox_set_time_stamp_by_ref

Sets the time stamp to the given value. The time stamp has a resolution of 0.1 micro seconds.

4.4.8.1. Parameter

CANBOX_HANDLE hDevice: Received handle of the device after calling canbox_open_device.
CANBOX_TIME_STAMP* psTime: Pointer to structure with new time stamp value.

4.4.9. canbox_close_device

Stops the communication with the device.

4.4.9.1. Parameter

CANBOX_HANDLE hDevice: Received handle of the device after calling canbox_open_device.

4.5. Interface Functions

4.5.1. canbox_open_interface

Opens a CAN interface and configures it with the given parameters. The bit rate is given and the bit timing parameters are calculated from the bit rate value.

4.5.1.1. Parameter

CANBOX_HANDLE hDevice: Received handle of the device after calling canbox_open_device.
CANBOX_INTERFACE sInterface: Structure for interface configuration.
CANBOX_HANDLE* phInterface: Pointer to handle for the handle of the interface.

4.5.2.canbox_open_interface_by_ref

Opens a CAN interface and configures it with the given parameters. The bit rate is given and the bit timing parameters are calculated from the bit rate value.

4.5.2.1.Parameter

CANBOX_HANDLE	hDevice:	Received handle of the device after calling canbox_open_device.
CANBOX_INTERFACE*	psInterface:	Pointer to structure for interface configuration.
CANBOX_HANDLE*	phInterface:	Pointer to handle for the handle of the interface.

4.5.3.canbox_open_interface_ex

Opens a CAN interface and configures it with the given parameters. Bit timing parameters are given directly.

4.5.3.1.Parameter

CANBOX_HANDLE	hDevice:	Received handle of the device after calling canbox_open_device.
CANBOX_INTERFACE_EX	sInterface:	Structure for interface configuration.
CANBOX_HANDLE*	phInterface:	Pointer to handle for the handle of the interface.

4.5.4.canbox_open_interface_ex_by_ref

Opens a CAN interface and configures it with the given parameters. Bit timing parameters are given directly.

4.5.4.1.Parameter

CANBOX_HANDLE	hDevice:	Received handle of the device after calling canbox_open_device.
CANBOX_INTERFACE_EX*	psInterface:	Pointer to structure for interface configuration.
CANBOX_HANDLE*	phInterface:	Pointer to handle for the handle of the interface.

4.5.5.canbox_get_interface_state

Supplies the state of the CAN interface.

4.5.5.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_INTERFACE_STATE*	psState:	Pointer to structure for state information.

4.5.6.canbox_get_last_interface_time

Delivers the last time stamp value where a message has been received.

4.5.6.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_TIME_STAMP*	psTime:	Pointer to a structure for time stamp value.

4.5.7.canbox_clear_interface

Clears the data buffer of an interface.

4.5.7.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
---------------	-------------	---

4.5.8.canbox_start_interface

Starts a stopped CAN interface.

4.5.8.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
---------------	-------------	---

4.5.9.canbox_read_interface

Reads a data set (received message) out of the data buffer of the CAN interface.

4.5.9.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_INTERFACE_READ*	psData:	Pointer to structure for data.

4.5.10.canbox_read_interface_ex

Reads one or more data sets (received messages) out of the data buffer of the CAN interface.

4.5.10.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
USHORT*	pusCount:	Maximum number of data sets to be delivered.
CANBOX_INTERFACE_READ*	pasData:	Pointer to array of structures for data.

4.5.11.canbox_write_interface

Writes the given data of the given identifier to the CAN bus.

4.5.11.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_INTERFACE_WRITE*	psData:	Pointer to structure array with information about the identifier and ist data to be sent.

4.5.12.canbox_write_interface_ex

Writes the given data of the given identifiers to the CAN bus.

4.5.12.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
USHORT*	pusCount:	Number of data sets for sending.
CANBOX_INTERFACE_WRITE*	pasData:	Pointer to structure array with information about the identifiers and their data to be sent.

4.5.13.canbox_stop_interface

Stops a running CAN interface. The CAN interface does not take part at the bus communication any longer.

4.5.13.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
---------------	-------------	---

4.5.14.canbox_close_interface

Closes an open CAN interface. The CAN interface does not take part at the bus communication any longer. All identifier, acceptance filter and universal sender of the interface are also closed.

4.5.14.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
---------------	-------------	---

4.6.Identifier Functions

4.6.1.canbox_open_identifier

Opens an identifier and configures it with the given parameters.

4.6.1.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_IDENTIFIER	sIdentifier:	Structure containing the configuration of the identifier
CANBOX_HANDLE*	phIdentifier:	Pointer to handle for receiving the handle of the identifier

4.6.2.canbox_open_identifier_by_ref

Opens an identifier and configures it with the given parameters.

4.6.2.1. Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_IDENTIFIER*	psIdentifier:	Pointer to a structure containing the configuration of the identifier
CANBOX_HANDLE*	phIdentifier:	Pointer to handle for receiving the handle of the identifier.

4.6.3. canbox_read_identifier

Reads a data set (CAN message) of the CAN identifier.

4.6.3.1. Parameter

CANBOX_HANDLE	hIdentifier:	Received handle of the identifier after calling canbox_open_identifier.
CANBOX_IDENTIFIER_DATA*	psData:	Pointer to a structure for the data.

4.6.4. canbox_write_identifier

Writes identifier data to the CAN bus.

4.6.4.1. Parameter

CANBOX_HANDLE	hIdentifier:	Received handle of the identifier after calling canbox_open_identifier.
CANBOX_IDENTIFIER_DATA*	psData:	Pointer to a structure with the data to be sent.

4.6.5. canbox_close_identifier

Closes an existing CAN identifier.

4.6.5.1. Parameter

CANBOX_HANDLE	hIdentifier:	Received handle of the identifier after calling canbox_open_identifier.
---------------	--------------	---

4.7. Acceptance Filter Functions

4.7.1. canbox_open_acceptance_filter

Opens an acceptance filter and configures it with the given parameters.

4.7.1.1. Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_ACCEPTANCE_FILTER	sFilter:	Structure containing the configuration of the filter.
CANBOX_HANDLE*	phFilter:	Pointer to handle for receiving the handle of the filter.

4.7.2.canbox_open_acceptance_filter_by_ref

Opens an acceptance filter and configures it with the given parameters.

4.7.2.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively can_box_open_interface_ex.
CANBOX_ACCEPTANCE_FILTER*psFilter:		Pointer to a structure containing the configuration of the filter.
CANBOX_HANDLE*	phFilter:	Pointer to handle for receiving the handle of the filter.

4.7.3.canbox_read_acceptance_filter

Reads data of an acceptance filter.

4.7.3.1.Parameter

CANBOX_HANDLE	hFilter:	Received handle of the filter after calling canbox_open_acceptance_filter.
CANBOX_IDENTIFIER_DATA* psData:		Pointer to a structure for received data.

4.7.4.canbox_close_acceptance_filter

Closes an existing acceptance filter.

4.7.4.1.Parameter

CANBOX_HANDLE	hFilter:	Received handle of the filter after calling canbox_open_acceptance_filter.
---------------	----------	--

4.8.Universal Sender Functions

4.8.1.canbox_open_universal_sender

Opens an universal sender. An universal sender is an object for sending any CAN message by specifying the Id and the data set.

4.8.1.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively can_box_open_interface_ex.
CANBOX_HANDLE*	phSender:	Pointer to handle for receiving the handle of the universal sender.

4.8.2.canbox_write_universal_sender

Sends one message by using an universal sender.

4.8.2.1.Parameter

CANBOX_HANDLE	hSender:	Received handle of the universal sender after calling canbox_open_universal_sender.
---------------	----------	---

CANBOX_UNIVERSAL_SENDER_DATA* psData: Pointer to the data for sending.

4.8.3.canbox_write_universal_sender_ex

Sends one or more messages by using a universal sender. If not all messages can be sent directly, those not sent are saved in a software buffer and are sent when the CAN bus is free.

4.8.3.1.Parameter

CANBOX_HANDLE	hSender:	Received handle of the universal sender after calling canbox_open_universal_sender.
USHORT*	pusCount:	Number of data sets to be sent.
CANBOX_UNIVERSAL_SENDER_DATA*	pasData:	Pointer to the data for sending.

4.8.4.canbox_close_universal_sender

Closes an existing universal sender.

4.8.4.1.Parameter

CANBOX_HANDLE	hSender:	Received handle of the universal sender after calling canbox_open_universal_sender.
---------------	----------	---

4.9.Router Functions

4.9.1.canbox_read_router_config

Gives the current configuration of the interfaces router.

4.9.1.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively can_box_open_interface_ex.
CANBOX_ROUTER*	psRouter:	Pointer to structure for receiving the router parameters.

4.9.2.canbox_read_router_config_ex

Gives the current configuration of the interfaces router.

4.9.2.1.Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively can_box_open_interface_ex.
CANBOX_ROUTER_EX*	psRouter:	Pointer to structure for receiving the router parameters.

4.9.3.canbox_write_router_config

Configures the router with the given values.

4.9.3.1. Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_ROUTER	sRouter:	Structure for configuring the router.

4.9.4. canbox_write_router_config_ex

Configures the router with the given values.

4.9.4.1. Parameter

CANBOX_HANDLE	hInterface:	Received handle of the interface after calling canbox_open_interface respectively canbox_open_interface_ex.
CANBOX_ROUTER_EX	sRouter:	Structure for configuring the router.

4.10. Supporting Functions for Visual Basic

As Visual Basic does not allow the correct alignment of some CANbox structures, the structures CANBOX_16 and CANBOX_32 have been created. These replace 16 bit respectively 32 bit values in several structures in order to ensure correct alignment. The following functions can be used for conversion from respectively into the structures.

4.10.1. canbox_convert_to_CANbox16

Conversion of a 16 bit value into a CANBOX_16 structure.

4.10.1.1. Parameter

USHORT	usValue:	16 bit value for conversion
--------	----------	-----------------------------

4.10.2. canbox_convert_to_CANbox16_by_ref

Conversion of a 16 bit value into a CANBOX_16 structure.

4.10.2.1. Parameter

USHORT	usValue:	16 bit value for conversion
CANBOX_16*	psValue:	Pointer to a CANBOX_16 structure

4.10.3. canbox_convert_from_CANbox16

Conversion of a CANBOX_16 structure to a 16 bit value.

4.10.3.1. Parameter

CANBOX_16*	psValue:	Pointer to a CANBOX_16 structure for conversion
------------	----------	---

4.10.4. canbox_convert_to_CANbox32

Conversion of a 32 bit value into a CANBOX_32 structure.

4.10.4.1. Parameter

USHORT	usValue:	32 bit value for conversion
--------	----------	-----------------------------

4.10.5. canbox_convert_to_CANbox32_by_ref

Conversion of a 32 bit value into a CANBOX_32 structure.

4.10.5.1. Parameter

ULONG	ulValue:	32 bit value for conversion
CANBOX_32*	psValue:	Pointer to a CANBOX_32 structure

4.10.6. canbox_convert_from_CANbox32

Conversion of a CANBOX_32 structure to a 32 bit value.

4.10.6.1. Parameter

CANBOX_32*	psValue:	Pointer to a CANBOX_32 structure for conversion
------------	----------	---

5. Socket Interface

The communication with the CANbo[®] can take place by directly using the socket interface instead of using the library. For this the CANbox[®] offers the port 1024 as server port for the LAN as well as the WLAN interface.

Furthermore, the port 1040 can be used as server port for resetting the connection on port 1024. This can be done by building up a connection to port 1040, if the IP addresses of both client ports are identical.

5.1. Structures and Definitions

After building up a connection to the CANbox, the communication takes place by using the structures CANBOX_SOCKET_IN and CANBOX_SOCKET_OUT.

5.1.1. CANBOX_SOCKET_IN

Structure for sending data to the CANbox.

5.1.1.1. Parameter

USHORT	usMacro:	0xEE44.
USHORT	usSizeIn:	Number of given bytes + 8.
USHORT	usSizeOut:	Maximum number of expected bytes in answer.
USHORT	usTask:	0x0090.
USHORT	usFunc:	Index of function to be called (see below).
UCHAR	usData[1440]	Data.

5.1.2. CANBOX_SOCKET_OUT

Structure for reception of data from the CANbox.

5.1.2.1. Parameter

USHORT	usMacro:	0x0144, if no error occurred.
USHORT	usSizeOut/ usError:	Number of delivered bytes + 6 respectively error code in case of error.
USHORT	usUnused:	Reserved parameter.
UCHAR	usData[1440]	Data.

5.2. Functions

5.2.1. canbox_init (Index 4)

Init of the CANbox. This function has to be called once before calling any other function. Otherwise all functions return the error CB_NO_INIT.

5.2.1.1. Parameter

To	None
From	None

5.2.2. canbox_reset (Index 5)

Closes all identifiers, acceptance filters and interfaces. Sets the time stamp to zero. Clears the software buffer.

5.2.2.1. Parameter

To	CANBOX_RESET_EX
From	None

5.2.3. canbox_info (Index 6)

Delivers information about the device.

5.2.3.1. Parameter

To	None	
From	CANBOX_DEVICE_INFO	Info structure for taking device information.

5.2.4. canbox_set_time_stamp (Index 7)

Sets the time stamp. The time stamp has a resolution of 0.1 micro seconds.

5.2.4.1. Parameter

To	CANBOX_TIME_STAMP	New value for the time stamp.
From	None	

5.2.5. canbox_exit (Index 12)

Release of used resources. This function is to be called in the end.

5.2.5.1. Parameter

To None
From None

5.2.6. canbox_open_interface (Index 15)

Opens a CAN interface and configures it with the given parameters. The bit rate is given and the bit timing parameters are calculated from the given bit rate.

5.2.6.1. Parameter

To CANBOX_INTERFACE Configuration for the interface.
From None

5.2.7. canbox_open_interface_ex (Index 16)

Opens a CAN interface and configures it with the given parameters. Bit timing parameters are given directly.

5.2.7.1. Parameter

To CANBOX_INTERFACE_EX Configuration for the interface.
From None

5.2.8. canbox_get_interface_state (Index 17)

Supplies the state of the CAN interface.

5.2.8.1. Parameter

To USHORT Zero based index of the interface.
From CANBOX_INTERFACE_STATE Structure for retrieving the interface state.

5.2.9. canbox_start_interface (Index 18)

Starts a stopped CAN interface.

5.2.9.1. Parameter

To USHORT Zero based index of the interface.
From None

5.2.10. canbox_clear_interface (Index 19)

Clears the data buffer of the interface.

5.2.10.1. Parameter

To USHORT Zero based index of the interface.
From None

5.2.11. canbox_read_interface (Index 20)

Reads the oldest data out of the data buffer of the interface.

5.2.11.1. Parameter

To	USHORT	Zero based index of the interface.
From	CANBOX_IDENTIFIER_DATA	Array of identifier data for taking the received data.

5.2.12. canbox_write_interface (Index 21)

Sets the given data onto the CAN bus. Parameter ulId has to be set with the handle of the respective identifier instead of the Id.

5.2.12.1. Parameter

To	CANBOX_IDENTIFIER_DATA	Array of identifier data to be sent.
From	None	

5.2.13. canbox_stop_interface (Index 22)

Stops a running CAN interface.

5.2.13.1. Parameter

To	USHORT	Zero based index of the interface.
From	None	

5.2.14. canbox_close_interface (Index 23)

Closes an open CAN interface.

5.2.14.1. Parameter

To	USHORT	Zero based index of the interface.
From	None	

5.2.15. canbox_open_identifier (Index 25)

Opens an identifier and configures it with the given configuration.

5.2.15.1. Parameter

To	CANBOX_IDENTIFIER	Configuration of identifier.
From	CANBOX_HANDLE	Handle of the identifier for accessing.

5.2.16. canbox_read_identifier_buffer (Index 26)

Reads the last received data of an identifier. The time stamp value is the actual time stamp value. This function should not be used if large amounts of data are to be exchanged between the CANbox and another connection participant, because the efficiency is reduced by the small packet size. Instead, the function canbox_read_interface with maximum amount of data should be used in this case.

5.2.16.1. Parameter

To	CANBOX_HANDLE	Handle of the identifier.
From	CANBOX_IDENTIFIER_DATA	Structure for received identifier data.

5.2.17.canbox_read_identifier_actual (Index 27)

Reads the last received data of an identifier. The time stamp value is the actual time stamp value. This function should not be used if large amounts of data are to be exchanged between the CANbox and another connection participant, because the efficiency is reduced by the small packet size. Instead, the function canbox_read_interface with maximum amount of data should be used in this case.

5.2.17.1.Parameter

To	CANBOX_HANDLE	Handle of the identifier.
From	CANBOX_IDENTIFIER_DATA	Structure for received identifier data.

5.2.18.canbox_write_identifier (Index 28)

Writes data of an identifier to the CAN bus.

5.2.18.1.Parameter

To	CANBOX_IDENTIFIER_DATA	Structure with identifier data to send.
From	None	

5.2.19.canbox_close_identifier (Index 29)

Closes an open CAN identifier.

5.2.19.1.Parameter

To	CANBOX_HANDLE	Handle of the identifier.
From	None	

5.2.20.canbox_open_filter (Index 30)

Opens an acceptance filter and configures it with the given parameters.

5.2.20.1.Parameter

To	CANBOX_ACCEPTANCE_FILTER	Configuration for acceptance filter.
From	CANBOX_HANDLE	Handle of acceptance filter for accessing.

5.2.21.canbox_read_filter_buffer (Index 31)

Reads the oldest data of an acceptance filter from the data buffer of the corresponding interface. This function should not be used if large amounts of data are to be exchanged between the CANbox and another connection participant, because it scans the whole interface data buffer for the oldest data of the filter. Instead, the function canbox_read_interface with maximum amount of data should be used in this case.

5.2.21.1.Parameter

To	CANBOX_HANDLE	Handle of the acceptance filter.
From	CANBOX_IDENTIFIER_DATA	Structure for received acceptance filter data.

5.2.22.canbox_read_filter_actual (Index 27)

Reads the last received data of an acceptance filter, the time stamp value is the actual time stamp value. This function should not be used if large amounts of data are to be exchanged between the CANbox and another

connection participant, because the efficiency is reduced by the small packet size. Instead, the function `canbox_read_interface` with maximum amount of data should be used in this case.

5.2.22.1. Parameter

To	CANBOX_HANDLE	Handle of the acceptance filter
From	CANBOX_IDENTIFIER_DATA	Structure for received acceptance filter data

5.2.23. `canbox_close_filter` (Index 32)

Closes an open CAN acceptance filter.

5.2.23.1. Parameter

To	CANBOX_HANDLE	Handle of the acceptance filter
From	None	

5.2.24. `canbox_open_sender` (Index 13)

Opens a universal sender.

5.2.24.1. Parameter

To	USHORT	Zero based index of the interface.
From	CANBOX_HANDLE	Handle of the universal sender for accessing

5.2.25. `canbox_write_sender` (Index 24)

Writes the data onto the CAN bus. The parameter `ulId` has to be set with the handle of the respective sender instead of the Id.

5.2.25.1. Parameter

To	CANBOX_UNIVERSAL_SENDER_DATA	Array of identifier data to be sent. In parameter <code>hSender</code> the respective handle of the universal sender has to be set.
From	None	

5.2.26. `canbox_close_sender` (Index 14)

Closes an open universal sender.

5.2.26.1. Parameter

To	CANBOX_HANDLE	Handle of universal sender
From	None	

5.2.27. `canbox_read_router_config` (Index 34)

Supplies the CAN router configuration.

5.2.27.1. Parameter

To	USHORT	Zero based index of the interface
From	CANBOX_ROUTER	Structure for delivering the router configuration

5.2.28.canbox_write_router_config (Index 35)

Configures the router using the given values.

5.2.28.1.Parameter

To CANBOX_ROUTER Router configuration
From None

5.2.29.canbox_read_lan_config (Index 36)

Supplies the actual EEPROM settings for the LAN connection.

5.2.29.1.Parameter

To None
From CANBOX_LAN_CONFIG Structure for delivering the LAN settings

5.2.30.canbox_write_lan_config (Index 37)

Sets the actual EEPROM settings for the LAN connection. The settings take effect after a restart of the CANbox.

5.2.30.1.Parameter

To CANBOX_LAN_CONFIG LAN configuration
From None

5.2.31.canbox_read_wlan_config (Index 38)

Supplies the actual EEPROM settings for the WLAN connection.

5.2.31.1.Parameter

To None
From CANBOX_WLAN_CONFIG Structure for delivering the WLAN settings

5.2.32.canbox_write_wlan_config (Index 39)

Sets the actual EEPROM settings for the WLAN connection. The settings take effect after a restart of the CANbox.

5.2.32.1.Parameter

To CANBOX_WLAN_CONFIG WLAN configuration
From None

6. Technical Data

Parameter	min.	typ.	max.	Unit	Remark
Power supply	6	12	60	V	
Power consumption (depending on the used MAX modules)	1	5	12	W	min.: without modules max.: limited by DC/DC
Galvanic isolation		--		V	Only by suitable I/O module
Temperature	0	20	70	°C	
Dimensions					Complete device
Width		83		mm	
Height		33		mm	
Depth		113		mm	
Protection type	IP54				