Software description
Multi Threaded API

Sontheim Industrie Elektronik GmbH, Diesel Straße 7, 87437 Kempten i. Allgäu
Tel. 0831/575900-0, Fax 0831/575900-8, Email sie@s-i-e.de
Internet http://www.s-i-e.de

History software description SIECA132

Version	description (identification of the author)	Datum
1.00	Documentation created.	08.03.04
1.10	Dilation CANUSB and virtual hardware	30.04.04
1.20	Extensive update and error recovery	02.02.05
1.30	Bugfixes	09.05.05
1.40	Append canWrite information Function canEnableHWExtendedId added	02.11.05
1.50	Unsupported Firmware recognized in canOpen for PowerPCI	13.12.05
1.60	Hardware requirements added	
2.00	Silentmode (canSetBaudrate), Errorframes (canOpen)	17.01.07
2.10	CAN-Level-Measurement	26.03.07
2.20	Extended Filter-Modes Blocked transmitting with canConfirmedTransmit Check for active CANBus with canGetDiffTimeLastFrame	16.07.07
2.30	Function canReadNoWait added.	07.11.07
2.40	Function canSetBaudrateForce, canIsNetOwner, canSetOwner, canGetOwner added	27.05.08
3.00	Support of multiversion installations	02.02.09
3.01	Revised	10.03.09
3.02	Function canSetBridgeFilter, canClearFilter, canGetBridgeFilters (Not in Multiversion API)	25.03.09
3.03	Function canBlinkLED (Not in Multiversion API)	08.06.09
3.04	PowerPCI V2 added	10.12.09
3.05	Added net-numbers for MobiCan and CanFox	28.04.10
3.06	Function canopenSH added	22.10.10
3.07	Table added which show overview of default baudrate settings	30.11.10
3.08	Function canFlush added	07.02.17

Data for software description SIECA132

Author	Bruno Sontheim / Mario Steinhauser
Version	3.0.8
Memory date	2/8/2017 11:08:00 AM
Status	In process (in Bearbeitung / fertiggestellt / geprüft / freigegeben)
Validity	This document is valid for Minimum SIECA132 Version 7.4.74.0 (Multiversion)
Orderer	--
Print date	2/8/2017 11:08:00 AM
Number of pages	142

© 2001 – 2017, Sontheim Industrie Elektronik

Used abbreviations and definitions

CAN	**C**ontroller **A**rea **N**etwork
CANopen	Protocol for CAN on layer 7
CiA	**C**AN **i**n **A**utomation
DLC	**D**ata **L**ength **C**ode = Number of data bytes in a Frame
extended identifier	29 Bit identifier 2.0B
Frame	CAN message
ID	CAN identifier
RTR	**R**emote **T**ransmission **R**equest
SDO	**S**ervice **D**ata **O**bject, service in CANopen-Protocol
standard identifier	11 Bit identifier 2.0A

© 2001 – 2017, Sontheim Industrie Elektronik

Index

© 2001 – 2017, Sontheim Industrie Elektronik

© 2001 – 2017, Sontheim Industrie Elektronik

© 2001 – 2017, Sontheim Industrie Elektronik

© 2001 – 2017, Sontheim Industrie Elektronik

© 2001 – 2017, Sontheim Industrie Elektronik

© 2001 – 2017, Sontheim Industrie Elektronik

1. General things for description

1.1. Demonstration of functions

The functions are described in the following style:

Function name	**Function name**	
description	Short description of the function	
syntax	Data type of return value function (data type1 Parameter name1; data type2 Parameter name2; ……);	
Parameter	Parameter name1 Parameter name2	comment
Return	Error codes	NTCAN_SUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

2. Introduction

Following operating systems are supported:
Windows XP SP 3 (32 Bit), Windows Vista SP 2 (32 Bit and 64 Bit), Windows 7 (32 Bit and 64 Bit)

Below the Software-Interface (API) for our CAN devices is described.
The driver contains functions on layer 2 of the OSI-layer model.
Up to 512 handles (virtual channels) can access the API at the same time.

In the „restricted“-version only 4 handles can access the MT-API at the same time.

The API makes for each handle a receive buffer available, which offers space for 16384 CAN messages for the first eight handles. The further handles offers space for 512 CAN messages.
Furthermore a transmitting buffer for each net number (CAN channel) is created, which is also able to buffer 512 CAN messages.

3. Recommended Hardware Requirements

OS	CPU	RAM	Free HDD Space
Windows 2000/SP4	P3 / 500 MHz or above	>= 128 MB	>=20 MB
Windows XP/SP3	P3 / 500 MHz or above	>= 192 MB	>=20 MB
Windows Vista/SP1	P3 / 500 MHz or above	>= 512 MB	>=20 MB

4. Scope of delivery

The MT-API is delivered on one CD which contains the installation-routine to install all required files, inclusive drivers, on the PC. Products from Sontheim like CanExplorer and MDT will automatically install the API, so it is not required to install an API before.

4.1. Hints for installation

Before CANUSB or PowerPCI are plugged at the system, the setup of the API should be installed. This is done by executing the program “MT_API-Setup.exe” and following the instructions. After successful installation the PowerPCI and CANUSB can be installed to the system. Required drivers are already installed by the setup before, so you can click always “next” during hardware-setup.

© 2001 – 2017, Sontheim Industrie Elektronik

5. Support of multiple installations

Before version 7.0.0.0 it was only possible to install one version of the API at the same time. Due to the fact that many customers want to use their software-system with a specific version of the MT-API it is now possible to install different version of the API at the same time onto the system.

The only limitation to install different versions of the API is, that each version must be version 7.0.0.0 or higher. If there is installed an API with version 7.0.0.0 or higher on the system, the installation of an API with version lower than 7.0.0.0 will fail. E.g if you want to install API 6.5.0.0 on a system where 7.0.0.0 is already installed you must remove at first manually this version before API 6.5.0.0 could be installed.

5.1. Components of multiple installations

Each version will be installed in a separate directory with separate registry-keys. Some components are installed only one-time from the latest installed version.

5.1.1. Switcher

The switcher is a windows-service which starts and stops the different installed API versions. It is installed one-time from the latest installed version of the API. The path where it is installed is: C:\Program Files\Sontheim\ MT_Api\

The required files are:

SIECA132Switcher.exe:	Windows-service which waits for requests to start a specific version of the MT-API
SIECA132ShdSw.dll	Shared DLL which contains routines for switching different versions of API's

5.1.2. Server

The server of the API is a windows-service which opens drivers, starts, stops and initializes the CAN devices and transferring data from and to the devices. It also distributes the data via shared-memory to the different handles. Each version installs an own service which gets started on demand.

The required files are:

SIECE132Svr.exe	Windows-service which opens drivers, starts, stops and initializes the CAN devices and transferring data from and to the devices.
SIECA132.DLL	Shared DLL which contains code for server- and client-side

5.1.3. Client

The client is the application which uses the API.

The required file is:

SIECA132.DLL	Shared DLL which contains code for server- and client-side. The DLL-functions are used by the application.

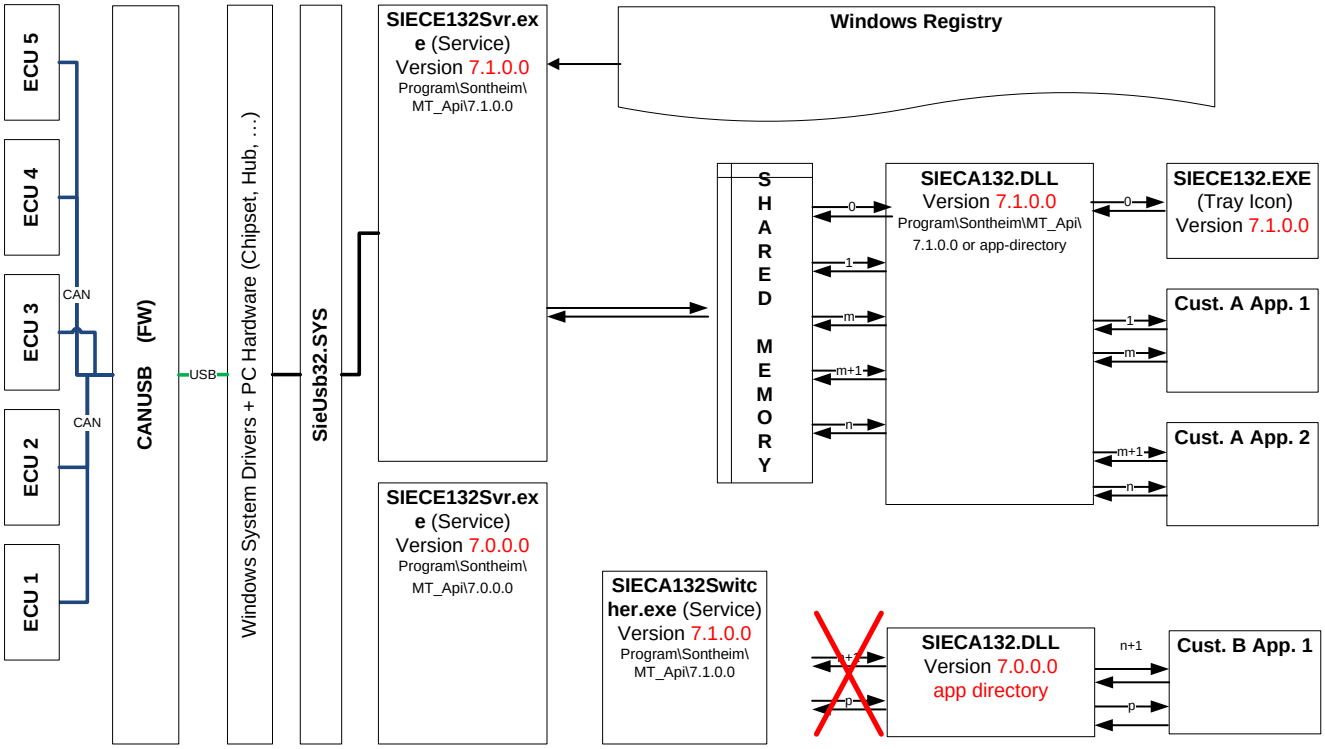
© 2001 – 2017, Sontheim Industrie Elektronik

5.1.4. Tools

There is a tool called SIECE132.EXE which displays the installed CAN-devices. It's running as a tray-icon with a small menu. Because windows-service cannot display information on desktop, this tool is also displaying information like progress of flashing firmware.

5.2. Functionality of multiple installations

To address the requested new functionality to switch between different versions of the mtAPI while the overall architecture remains backward compatible the following two pictures may demonstrate the proposed changes. For this example version 7.1.0.0 and 7.0.0.0 are installed on the system. The first picture shows the situation for the (assumed) most recent version 7.1.0.0



It does not matter in this case, whether the customer application A is using the new multi version feature or not. Applications created for mtAPI < 7.0.0.0 would still use the SIECA132.dll from the program directory of the mtAPI, which is part of the Windows search path. The SIECA132Switcher copies each time the corresponding version of the SIECA132.DLL into the program directory. All these applications would run as before with mtAPI.

In the picture above customer application B (bottom right) might be started. During installation the library SIECA132.dll is put into the application directory. Therefore the first function call to this library will try to connect with the shared memory. This will fail because another version of the mtApi is currently running. The next step is sending a request to the SIECA132Switcher, which should switch to version 7.0.0.0.

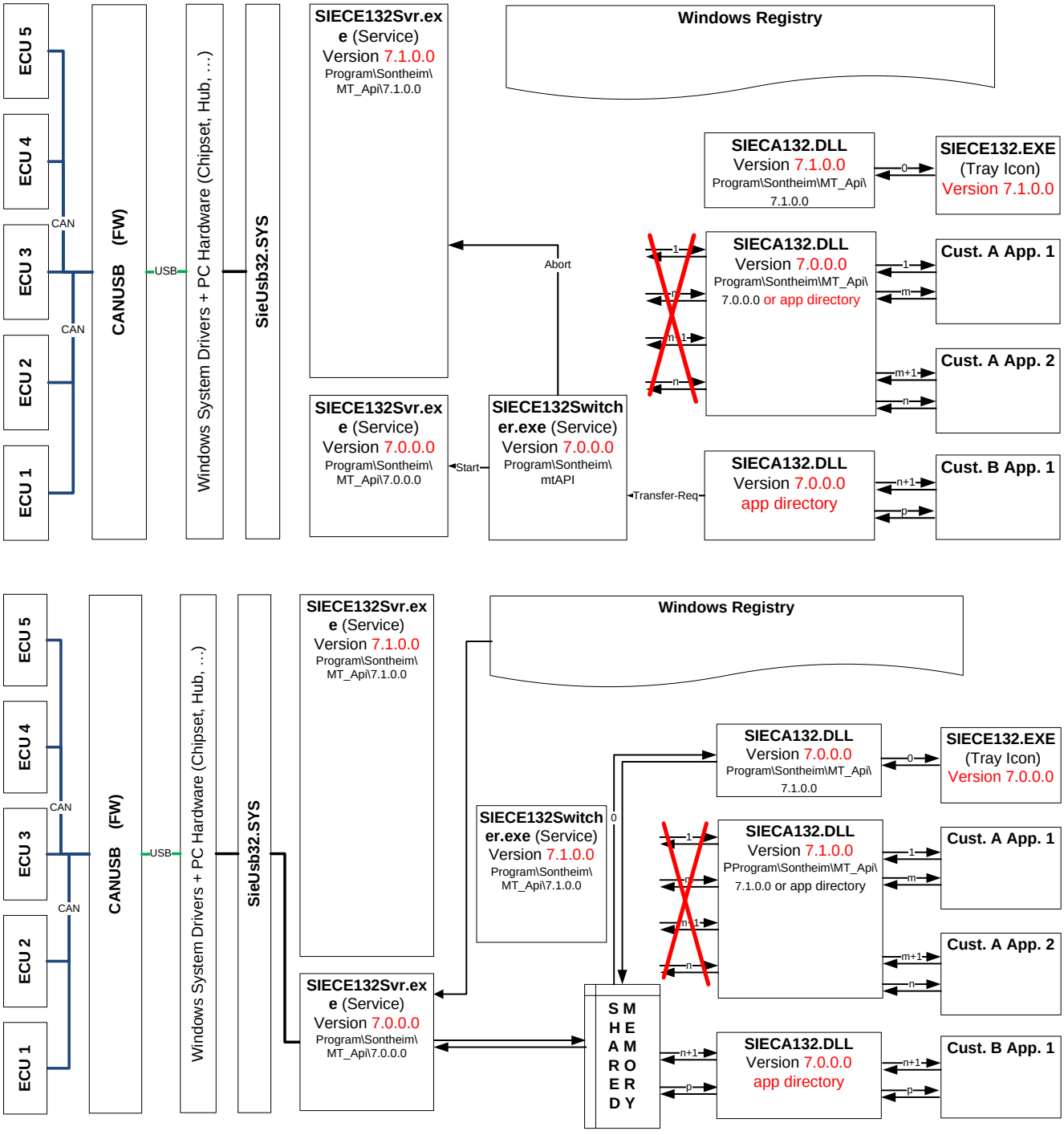
In case other applications are still active, the SIECA132Switcher will reject this request and send back a negative response. The SIECA132.dll version 7.0.0.0 will get this negative response and return a failure to the calling customer application B.

If there is currently no application using mtApi 7.1.0.0, the SIECA132Switcher shut down the SIECE132Svr.exe from version 7.1.0.0 and start the SIECE132Svr.exe from version 7.0.0.0. Also the SIECA132Switcher will stop the associated Tray Icon App, start the Tray Icon App for version 7.0.0.0. After that the SIECA132Switcher copy SIECA132.DLL from version 7.0.0.0 into the main-directory of the mtAPI which is part of the windows-search path. So every application which are created for mtAPI < 7.0.0.0 and application which should be not bound to a

© 2001 – 2017, Sontheim Industrie Elektronik

specifc version (e.g CanExplorer 4) will use this SIECA132.DLL, which is now the version which is currently runiing.

The switch from one API to another version is done within a few seconds (2 to 6 s, depending of hardware interface installed). In this case customer application B is able to get access to the corresponding shared memory and can run data back and forth to the CAN bus. This situation is shown in the following picture



© 2001 – 2017, Sontheim Industrie Elektronik

5.3. Bind application to a specific version

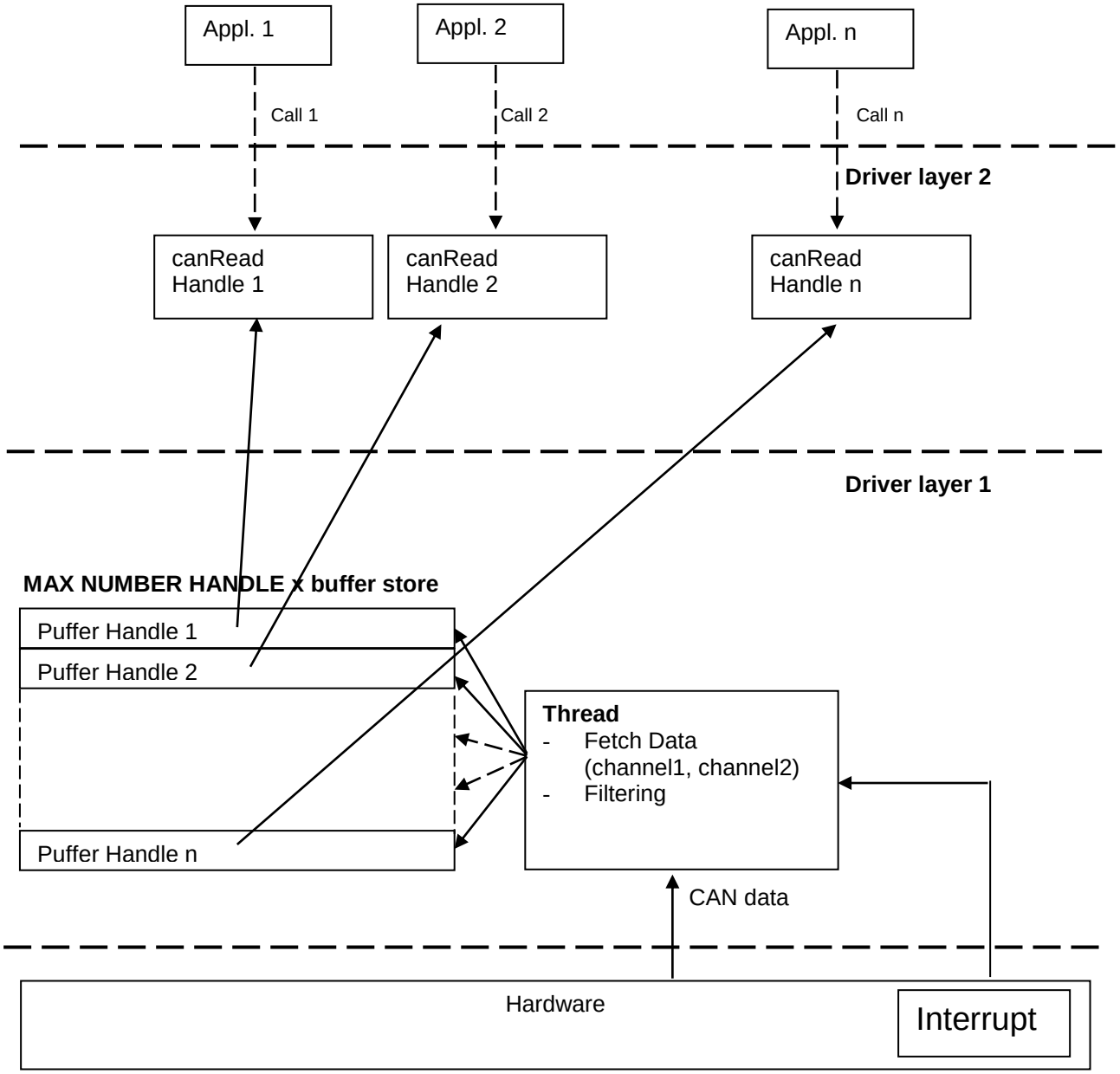
In case an application should run with a specifc mtAPI version you must do the following steps:

a) Install the desired mtAPI version
b) Put the corresponding SIECA132.DLL into the program-directory of your application. **It's not allowed to rename the SIECA132.DLL. Otherwise switching will not work properly.**

If you do not put a SIECA132.DLL in the program-directory the application runs with the newest installed version of the mtAPI or in case another application is running a different version of the mtAPI, your application will run with that version.

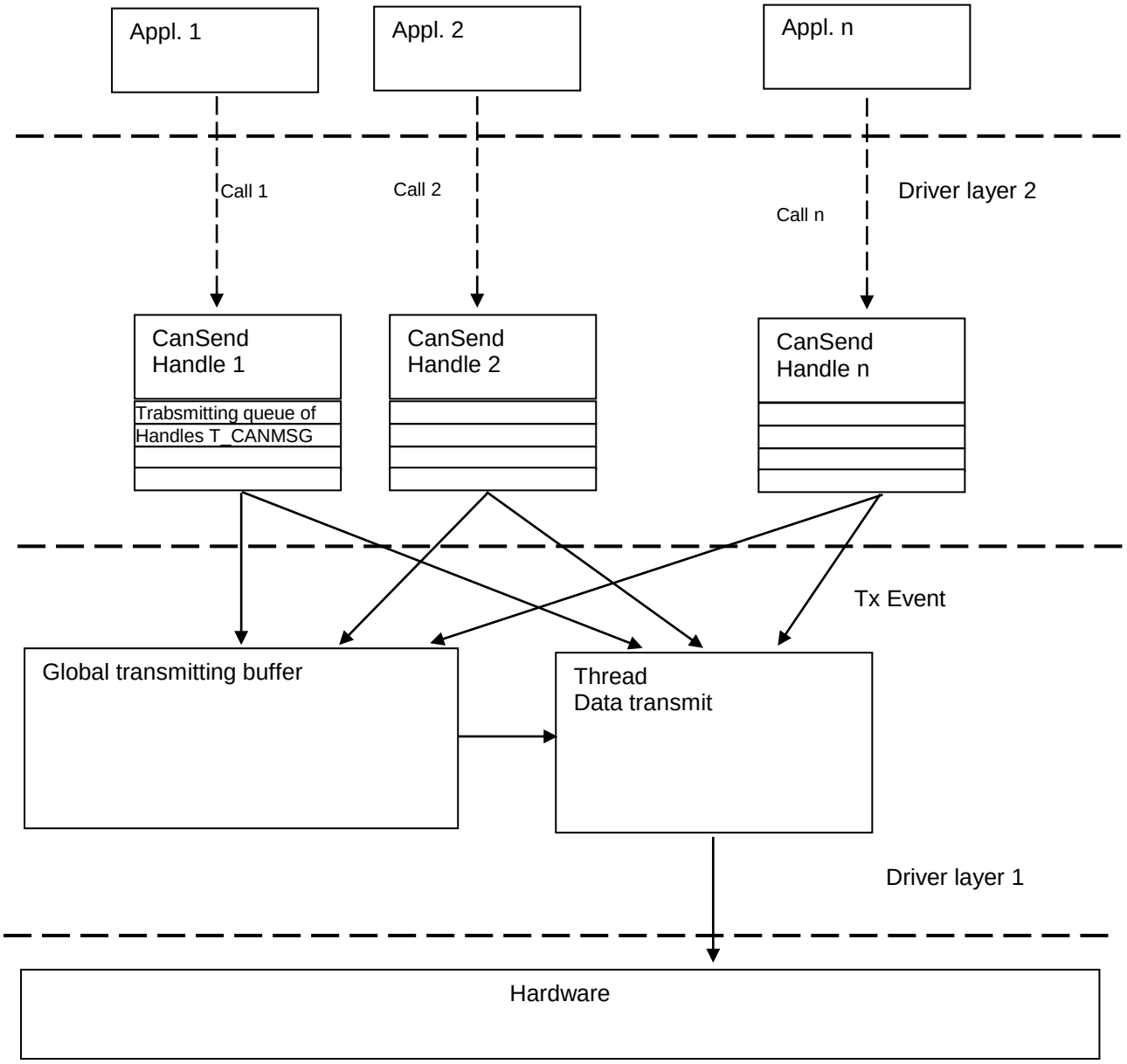
5.4. Functionality of the API

5.4.1. Receiving CAN messages



© 2001 – 2017, Sontheim Industrie Elektronik

5.4.2. Transmitting CAN messages



© 2001 – 2017, Sontheim Industrie Elektronik

5.4.3. Driver layer 1 (Server)

This layer is executed by the windows-service SIECE132Svr.

5.4.3.1. Thread

Several threads are used for receiving and transmitting CAN-messages. Threads which are transmitting messages are event-driven and threads which are receiving messages from the CAN-device are interrupt-driven. Using events and interrupts ensures less CPU consumption and fast reactions during receiving and transmitting.

5.4.3.2. Softwarebased receive buffer

The software-based receive-buffers are organized in form of a ring buffer. Each handle has got an own ring buffer with a capacity of 16384 CAN messages for the first eight handles. The further handles have a buffer of 512 CAN messages. If the messages are not fetched fast enough from the application, the buffer can overflow. If the buffer overflows new messages will be lost.

5.4.3.3. Globale Transmit-Queue

Each physical CAN-channel has a global transmitting buffer. It is organized in form of a ring buffer. The functions canSend, canWrite and canConfirmedTransmit copies messages which should be transmitted on CAN into the corresponding buffer. An event-driven thread is reading the buffer and transmits the messages on CAN. This mechanism ensures maximum throughput if message should be transmitted back-to-back.

5.4.4. Driver layer 2 (Client)

The driver layer 2 contains the DLL interface and is executed by the application which uses the mtAPI.

© 2001 – 2017, Sontheim Industrie Elektronik

6. Definition of the data structures

This chapter describes the data-structures used in API.

6.1. Data structure of the CAN message

The CAN-messages are passed to the functions in following data structure:

```
typedef struct
{
        long            l_id;
        unsigned char   by_len;
        unsigned char   by_msg_lost;
        unsigned char   by_extended;
        unsigned char   by_remote
        unsigned char   aby_data[8];
        unsigned long   ul_tstamp;
} CMSG;
```

Thereby means:	
l_id	11-Bit- or 29-Bit identifier
by_len	Bit 0-3 = number Data-Bytes DLC[3:0] Bit 4-7 = reserved

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
res	res	res	res	DLC3	DLC2	DLC1	DLC0

DLC[3:0]	number Data-Bytes
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8

Thereby means:	
by_msg_lost	<> 0 Flag for lost messages (only Receive)
by_extended	Bit 0: 11-Bit identifier Bit 1: 29-Bit identifier Bit 6: Errorframe (only at CAN Read) Bit 7: Echo (only at CAN Read)
by_remote	<> 0 means remote-frame (rtr set)
aby_data	Data-Bytes 0..8
ul_tstamp	32 Bit time stamp, one tick is 1/10 ms

© 2001 – 2017, Sontheim Industrie Elektronik

6.2. Data structure identifier-Array

This data structure is needed for the function ***canIDStatus***. It contains an array with 2048 (0x7FF) elements of data type unsigned char.

```
typedef struct
{
        unsigned char           aby_ID[2048];

} T_ID_ARRAY;
```

The index of each element represents a standard identifier. Is an identifier for a certain handle activated (by ***canIDAdd***), so the equivalent array-element is set on 1, otherwise on 0.
e.g. identifier 0x100 (dez. 256) approved → t_IDArray.aby_ID[256] = 1;

By call of ***canIDDelete*** the equivalent element is deleted.

6.3. Data structure Status

The status data is packed in following data structure: The information could be fetched with Function: canStatus for each handle.

```
typedef struct
{
        unsigned short          w_hw_rev;
        unsigned short          w_fw_rev;
        unsigned short          w_drv_rev;
        unsigned short          w_dll_rev;
        unsigned long           ul_board_status;
        unsigned char           by_board_id;
        unsigned short          w_busoffctr;
        unsigned short          w_errorflag;
        unsigned short          w_errorframectr;
        unsigned short          w_netctr;
        unsigned short          w_baud;
        unsigned int            ui_epld_rev;
} CAN_IF_STATUS;
```

© 2001 – 2017, Sontheim Industrie Elektronik

Thereby means:	
w_hw_rev	Hardware revision number (if supported, otherwise 0)
w_fw_rev	Firmware revision number
w_drv_rev	Software driver revision number (if supported, otherwise 0)
w_dll_rev	DLL revision number
ul_board_status	Not used
by_board_id	Not used
w_busoffctr	busoff – Counter of the current net, number of “busoffs”
w_errorflag	actual error condition (see definition of constants in SIECA132.h): ✓ error-passive (ERRORPASSIVE 0x08) ✓ busoff (BUSOFF 0x04) ✓ receiver overflow (RBUFOVERFLOW 0x01) ✓ hardware receiver overflow (HWRBUFOVERFLOW 0x02) ✓ txerror (TXERROR 0x10) ✓ USB communication error, maybe data-lost (USBDATAERROR 0x20) ✓ CANUSB has done restart, maybe data-lost (USBRESTART 0x40) ✓ Device is unplugged (HWUNPLUGGED 0x80) ✓ Error during bridging (BRIDGEERROR 0x100)
w_errorframectr	Number of error-frames which are detected on the bus (accumulated) Not all devices support this feature
w_netctr	Not used
w_baud	actual adjusted baudrate (0=1000kBit, 7 = 20kBit)
ui_epld_rev	EPLD revision number, Only PowerPCI

© 2001 – 2017, Sontheim Industrie Elektronik

6.4. Data structure DLL information

The DLL information is packed in following date structure: The data could be fetched with function ***canGetDllInfo***.

```
typedef struct st_InternalDLLInformation {
    unsigned int aui_TxCounter[2];
    unsigned int aui_TxHandleCounter[MAX_NUM_APIHANDLE];
    unsigned int aui_TxCounterRTR[2];
    unsigned int aui_TxHandleCounterRTR[MAX_NUM_APIHANDLE];
    unsigned int aui_TxThreadCounter[2];
    unsigned int aui_TxThreadCounterRTR[2];
    unsigned int aui_RxCounter[MAX_NUM_APIHANDLE];
    unsigned int aui_RxThreadCounter[2];
    unsigned int aui_RxBufferCounter[MAX_NUM_APIHANDLE];
    unsigned int aui_InterfaceCtr[MAX_NUM_APIHANDLE];
    T_STRINGARRAY appNames;
    unsigned int aui_ThreadStatus[MAX_NUM_APIHANDLE];
    unsigned int aui_ZugriffsCounterRead[MAX_NUM_APIHANDLE];
    unsigned int aui_ZugriffsCounterWrite[MAX_NUM_APIHANDLE];
    unsigned int aui_HandleNr[MAX_NUM_APIHANDLE];
    unsigned int aui_NetzZuordnung[MAX_NUM_APIHANDLE];
    unsigned int aui_NetzOwner[2];
    unsigned int aui_Reserve1601[MAX_NUM_APIHANDLE];
    unsigned int aui_Reserve1602[MAX_NUM_APIHANDLE];
    unsigned int aui_Reserve1603[MAX_NUM_APIHANDLE];
    unsigned int aui_Reserve1604[MAX_NUM_APIHANDLE];
    unsigned int ui_NetCount;
    unsigned int ui_Reserve;
    unsigned int aui_AnzahlEchoFrames[2];
    unsigned int ui_CloseZaehler;
    unsigned int ui_OpenZaehler;
    unsigned int ui_CloseFlag;
    unsigned int ui_OpenedHandles;
```

Thereby means:	
Data written from canSend/canWrite or canConfirmedTransmit into the transmitting buffer:	
aui_TxCounter[2]	So many data are written into the transmitting buffer of the equivalent CAN net (0 or 1).
aui_TxHandleCounter [MAX_NUM_APIHANDLE]	So many data were written from the equivalent handle into the buffer.
aui_TxCounterRTR[2]	So many RTR data are written into the transmitting buffer of the equivalent CAN net (0 or 1).
aui_TxHandleCounterRTR [MAX_NUM_APIHANDLE]	So many RTR data were written from the equivalent handle into the buffer.
Data from the transmitting buffer to the PowerCAN-PCI:	
aui_TxThreadCounter[2]	So many data were written into DPRAM.
aui_TxThreadCounterRTR[2]	So many RTR data were written into DPRAM.
Data from canRead read out of the receiving buffer:	
aui_RxCounter	So many data were read from the equivalent application out of the

© 2001 – 2017, Sontheim Industrie Elektronik

[MAX_NUM_APIHANDLE]	receiving buffer.
aui_RxThreadCounter[2]	So many messages were received on the net (0 or 1).
aui_RxBufferCounter [MAX_NUM_APIHANDLE]	So many data were written into the receiving buffer of the equivalent handles.
appNames	Application name of the handles (see function canOpen)
aui_ThreadStatus [MAX_NUM_APIHANDLE]	Defines whether the HW-Thread runs or not. 0: not active ✓ 1: sleeping ✓ 2: running ✓ 3: stopped ✓ 255: error
aui_ZugriffsCounterRead [MAX_NUM_APIHANDLE]	Counts the RD application accesses
aui_ZugriffsCounterWrite [MAX_NUM_APIHANDLE]	counts WR application accesses
aui_HandleNr [MAX_NUM_APIHANDLE]	related handle number
aui_NetzZuordnung [MAX_NUM_APIHANDLE]	Which handle uses which net?
aui_NetzOwner[2]	The owner handles of both nets. (Owner can change baud rate.)
aui_Reserve1601 [MAX_NUM_APIHANDLE]	Still not used
aui_Reserve1602 [MAX_NUM_APIHANDLE]	Still not used
aui_Reserve1603 [MAX_NUM_APIHANDLE]	Still not used
aui_Reserve1604 [MAX_NUM_APIHANDLE]	Still not used
ui_NetCount	Number of the physical existing nets (= 2)
ui_Reserve	Still not used
aui_AnzahlEchoFrames[2]	Number of the Echo-Frames per net
ui_CloseZaehler	Still not used
ui_OpenZaehler	Number of the failured trials at transmitting to achieve the arbitration
ui_CloseFlag	Number of the not correct traversed canClose after the Mutex was still achieved.
ui_OpenedHandles	Number of active (opened) handles

aui_InterfaceCtr[MAX_NUM_APIHANDLE] counter conditions of PowerCAN-PCI for following values:

aui_InterfaceCtr[0]	Timer (in 100 µs-resolution)	Channel 0
aui_InterfaceCtr[1]	Receivercounter	Channel 0
aui_InterfaceCtr[2]	Transmittercounter	Channel 0
aui_InterfaceCtr[3]	Remote Transmitter Counter	Channel 0
aui_InterfaceCtr[4]	Number of transmitted data bytes	Channel 0
aui_InterfaceCtr[5]	Number of received data bytes	Channel 0
aui_InterfaceCtr[6]	reserved	-
aui_InterfaceCtr[7]	Receivercounter	Channel 1
aui_InterfaceCtr[8]	Transmittercounter	Channel 1
aui_InterfaceCtr[9]	Remote Transmitter Counter	Channel 1
aui_InterfaceCtr[10]	Number of transmitted data bytes	Channel 1
aui_InterfaceCtr[11]	Number of received data bytes	Channel 1
aui_InterfaceCtr[12]	CAN1 Statusflags	Channel 0
aui_InterfaceCtr[13]	CAN2 Statusflags	Channel 1
aui_InterfaceCtr[14]	Error Frame Counter	Channel 0
aui_InterfaceCtr[15]	Error Frame Counter	Channel 1
aui_InterfaceCtr[16] bis aui_InterfaceCtr [MAX_NUM_APIHANDLE]	Still not used	-

© 2001 – 2017, Sontheim Industrie Elektronik

6.5. Data structure of the CAN counter data

The CAN counter data is packed in following data structure:

```
typedef struct
{
        unsigned long       ul_timer;
        unsigned long       ul_rxframectr1;
        unsigned long       ul_txframectr1;
        unsigned long       ul_txremframectr1;
        unsigned long       ul_txdatabytectr1;
        unsigned long       ul_rxdatabytectr1;
        unsigned long       reserved0;
        unsigned long       ul_rxframectr2;
        unsigned long       ul_txframectr2;
        unsigned long       ul_txremframectr2;
        unsigned long       ul_txdatabytectr2;
        unsigned long       ul_rxdatabytectr2;
        unsigned long       ul_CAN1ctrlflags;
        unsigned long       ul_CAN2ctrlflags;
        unsigned long       ul_errframectr1;
        unsigned long       ul_errframectr2;
} CTRDATA;
```

Thereby means:	
Variable identifier.	**content:**
ul_timer	Free running timer, resolution 100us
ul_rxframectr1	Counter for received frames channel 1
ul_txframectr1	Counter for transmitted frames channel 1
ul_txremframectr1	Counter for transmitted remote frames channel 1
ul_txdatabytectr1	Counter for all transmitted data bytes channel 1
ul_rxdatabytectr1	Counter for all received data bytes channel 1
reserved0	Reserved
ul_rxframectr2	Counter for received frames channel 2
ul_txframectr2	Counter for transmitted frames channel 2
ul_txremframectr2	Counter for transmitted remote frames channel 1
ul_txdatabytectr2	Counter for all transmitted data bytes channel 2
ul_rxdatabytectr2	Counter for all received data bytes channel 1
ul_CAN1ctrlflags	Statusflags of Full-CAN-Controller 0
ul_CAN2ctrlflags	Statusflags of Full-CAN-Controller 1
ul_errframectr1	Number of error frames (channel 0)
ul_errframectr2	Number of error frames (channel 1)

© 2001 – 2017, Sontheim Industrie Elektronik

```
typedef struct
{
        unsigned long       ul_timer;
        unsigned long       ul_rxframectr1;
        unsigned long       ul_txframectr1;
        unsigned long       ul_txremframectr1;
        unsigned long       ul_txdatabytectr1;
        unsigned long       ul_rxdatabytectr1;
        unsigned long       reserved0;
        unsigned long       ul_rxframectr2;
        unsigned long       ul_txframectr2;
        unsigned long       ul_txremframectr2;
        unsigned long       ul_txdatabytectr2;
        unsigned long       ul_rxdatabytectr2;
        unsigned long       ul_CAN1ctrlflags;
        unsigned long       ul_CAN2ctrlflags;
        unsigned long       ul_errctr1;
        unsigned long       ul_errctr2;
        unsigned long       ul_rxremframectr1;
        unsigned long       ul_erxframectr1;
        unsigned long       ul_etxframectr1;
        unsigned long       ul_etxremframectr1;
        unsigned long       ul_erxremframectr1;
        unsigned long       ul_rxremframectr2;
        unsigned long       ul_erxframectr2
        unsigned long       ul_etxframectr2;
        unsigned long       ul_etxremframectr2;
        unsigned long       ul_erxremframectr2;
} CTRDATA2;
```

Thereby means:	
Variable identifier.	**content:**
ul_timer	Free running timer, resolution 100us
ul_rxframectr1	Counter for received frames channel 1
ul_txframectr1	Counter for transmitted frames channel 1
ul_txremframectr1	Counter for transmitted remote frames channel 1
ul_txdatabytectr1	Counter for all transmitted data bytes channel 1
ul_rxdatabytectr1	Counter for all received data bytes channel 1
reserved0	Reserved
ul_rxframectr2	Counter for received frames channel 2
ul_txframectr2	Counter for transmitted frames channel 2
ul_txremframectr2	Counter for transmitted remote frames channel 1
ul_txdatabytectr2	Counter for all transmitted data bytes channel 2
ul_rxdatabytectr2	Counter for all received data bytes channel 1
ul_CAN1ctrlflags	Statusflags of Full-CAN-Controller 0
ul_CAN2ctrlflags	Statusflags of Full-CAN-Controller 1
ul_errctr1	Number of error frames (channel 0)
ul_errctr2	Number of error frames (channel 1)
ul_rxremframectr1	Counter for received remote frames channel 1
ul_erxframectr1	Counter for received extended frames channel 1
ul_etxframectr1	Counter for transmitted extended frames channel 1
ul_etxremframectr1	Counter for transmitted extended remote frames channel 1
ul_erxremframectr1	Counter for received extended remote frames channel 1
ul_rxremframectr2	Counter for received remote frames channel 2
ul_erxframectr2	Counter for received extened frames channel 2

© 2001 – 2017, Sontheim Industrie Elektronik

ul_etxframectr2	Counter for transmitted extended frames channel 2
ul_etxremframectr2	Counter for transmitted extended remote frames channel 2
ul_erxremframectr2	Counter for received extended remote frames channel 2

6.1. Enum for requesting and changing timeouts

Enum is used by Function: canGetTimeout and Function: canSetTimeout to get and set TX and RX timeouts.

```
typedef enum
{
        timeoutType_rx,
        timeoutType_tx
} T_TIMEOUT_TYPE;
```

Thereby means:	
Enum	**Content**
timeoutType_rx	Requesting and changing RX-Timeout
timeoutType_tx	Requesting and changing TX-Timeout

6.1. Structure for optaining the installed devices

```
typedef struct
{
      int Net;
      char Name[20];
      unsigned long ul_Status;
      unsigned long ul_Features;
      long Reserved[18];
 } T_DeviceList;
```

Thereby means:	
Variable	**Content**
Net	Number of the net
Name	Name of the device
ul_Status	Current status of hardware. Following bits are used: Bit 0: Card is ready for operation (otherwise it is not!) Bit 1: Wrong firmware version Bit 2: Wrong driver version
ul_Features	Support features of the hardware. Following bits are used: Bit 0: Error frame detection Bit 1: Receive remote frames Bit 2: High- / lowspeed is selectable Bit 3: Listen only mode (no ACK is send)
Reserved	Reserved

© 2001 – 2017, Sontheim Industrie Elektronik

6.1. Enum for active measuring of CAN-Level

Enum is used by Function: canGetCanLevel to measure active the level on the CAN-bus.

```
typedef enum
{
        canLevelDominant    = 0,
        canLevelRecessive   = 1
} T_CANLEVEL_MODE;
```

Thereby means:	
Enum	**Content**
canLevelDominant	Dominant level
canLevelRecessive	Recessive level

6.1. Structure for passive measuring of CAN-Level

Structure is used by Function: canGetCanLevelHist to measure passive the level on the CAN-bus.

```
typedef struct
{
        unsigned long       ulMeasureCountAll;
        unsigned long       arr_LevelLow[50];
        unsigned long       arr_LevelHigh[50];
        unsigned long       ulMeasureCountLowRez;
        double              dAvgLevelLowRez;
        double              dVariLevelLowRez;
        unsigned long       ulMeasureCountHighRez;
        double              dAvgLevelHighRez;
        double              dVariLevelHighRez;
        unsigned long       ulMeasureCountLowDom;
        double              dAvgLevelLowDom;
        double              dVariLevelLowDom;
        unsigned long       ulMeasureCountHighDom;
        double              dAvgLevelHighDom;
        double              dVariLevelHighDom;
} CANLEVEL_HIST;
```

© 2001 – 2017, Sontheim Industrie Elektronik

Thereby means:	
Variable	**Content**
ulMeasureCountAll	Number of measurements
arr_LevelLow	Each element in array represents the voltage. E.g. element at index 11 represents voltage 1.1V. If CANUSB measures a voltage the current element is raised with value 1. This array contains measurements of CAN LOW.
arr_LevelHigh	Same as arr_LevelLow, but contains measurement from CAN HIGH.
ulMeasureCountLowRez	Number of valid measurements of recessive CAN LOW levels.
dAvgLevelLowRez	Average level of recessive CAN LOW
dVariLevelLowRez	Variance of average level of recessive CAN LOW
ulMeasureCountHighRez	Number of valid measurements of recessive CAN HIGH levels.
dAvgLevelHighRez	Average level of recessive CAN HIGH
dVariLevelHighRez	Variance of average level of recessive CAN HIGH
ulMeasureCountLowDom	Number of valid measurements of dominant CAN LOW levels.
dAvgLevelLowDom	Average level of dominant CAN LOW
dVariLevelLowDom	Variance of average level of dominant CAN LOW
ulMeasureCountHighDom	Number of valid measurements of dominant CAN HIGH levels.
dAvgLevelHighDom	Average level of dominant CAN HIGH
dVariLevelHighDom	Variance of average level of dominant CAN HIGH

6.1. Enum of the different filter-modes

```
typedef enum
{
        filterMode_standard   = 0,
        filterMode_j2534      = 1,
        filterMode_extended   = 2,
        filterMode_j2534_2    = 3,
        filterMode_nofilter   = 4
} T_FILTER_MODE;
```

Thereby means:	
Enum	**Content**
filterMode_standard	Standard-Filtermode (Default)
filterMode_j2534	Range-Based-Filtermode
filterMode_extended	29 Bit-Filtermode
filterMode_j2534_2	Pattern-Mask-Based-Filtermode
filterMode_nofilter	No Filter

© 2001 – 2017, Sontheim Industrie Elektronik

6.1. Enum for include and exclude filter

```
typedef enum
{
        j2534Mode_excl      = 0,
        j2534Mode_incl      = 1
} T_J2534_MODE;
```

Thereby means:	
Enum	**Content**
j2534Mode_excl	Exclude-filter
j2534Mode_incl	Include-filter

6.1. Structure for busload

```
typedef struct
{
        unsigned long ul_intervall;
        unsigned long ul_stdframectr;
        unsigned long ul_stdframebytectr;
        unsigned long ul_extframectr;
        unsigned long ul_extframebytectr;
        double        ObjectTime;
        unsigned long ul_load;
} BUSLOAD;
```

Thereby means:	
Variable	**Content**
ul_intervall	Update interval (in ms)
ul_stdframectr	Number of received 11 bit messages within interval
ul_stdframebytectr	Number of received data from 11 bit messages within interval
ul_extframectr	Number of received 29 bit messages within interval
ul_extframebytectr	Number of received data from 29 bit messages within interval
ObjectTime	Bittime
ul_load	Busload, must be divided with 100

© 2001 – 2017, Sontheim Industrie Elektronik

6.1. Structure for bridging

```
typedef struct st_BridgeFilter
{
        unsigned long ulDestNet;
        unsigned long ulPattern;
        unsigned long ulMask;
        unsigned long ulReserved;
} BRIDGE_FILTER;
```

Thereby means:	
Variable	**Content**
ulDestNet	Netnumer of the destination-net where message are bridged to
ulPattern	Pattern of the filter Bit 30: Bridge also messages which are transmitted by device itself Bit 29: Bridge 29 Bit identifier Bit 28-0: Identifier
ulMask	Mask of the filter Bit 30: Bridge also messages which are transmitted by device itself Bit 29: Bridge 29 Bit identifier Bit 28-0: Identifier
ulReserved	Reserved

© 2001 – 2017, Sontheim Industrie Elektronik

6.2. Return value of the API functions

The return value is always of type "long"

Return value = 0: successful
Return value < 0: error

6.2.1. Standard return values

Return value	decimal value (error number)	description
NTCAN_SUCCESS	0	No error
NTCAN_RX_TIMEOUT	-1	Receive-Timeout
NTCAN_TX_TIMEOUT	-2	Transmit-Timeout
NTCAN_CONTR_BUSOFF	-3	Error on CAN-Bus (Bus-Off)
NTCAN_NO_ID_ENABLED	-4	No filter is enabled.
NTCAN_ID_ALREADY_ENABLED	-5	Identifier already enabled.
NTCAN_ID_NOT_ENABLED	-6	Identifier no enabled.
NTCAN_INVALID_PARAMETER	-7	Parameter is not valid or nullpointer detected.
NTCAN_INVALID_HANDLE	-8	Not a valid handle. Call function canOpen at first.
NTCAN_TOO_MANY_HANDLES	-9	It was tried to open more than 512[1] handles.
NTCAN_INIT_ERROR	-10	Error on initialising the hardware.
NTCAN_RESET_ERROR	-11	Currently not used / reserved.
NTCAN_DRIVER_ERROR	-12	A driver component is missing or failing.
NTCAN_DLL_ALREADY_INIT	-13	Baudrate already set by another handle. No rights (no owner) to change baudrate.
NTCAN_CHANNEL_NOT_INITIALIZED	-14	Baudrate not set before.
NTCAN_TX_ERROR	-15	Currently not used / reserved.
NTCAN_NO_SHAREDMEMORY	-16	Currently not used / reserved.
NTCAN_HARDWARE_NOT_FOUND	-17	Net is not available. Maybe hardware is unplugged.
NTCAN_INVALID_NETNUMBER	-18	Net does not exist. New nets (CAN interfaces) could be available in future versions.
NTCAN_TOO_MANY_J2534_RANGES	-19	Maximum of ten filters already set for this handle.
NTCAN_TOO_MANY_J2534_2_FILTERS	-20	Maximum of ten filters already set for this handle.
NTCAN_DRIVER_NOT_INSTALLED	-21	Windows-driver / DLL not installed.
NTCAN_NO_OWNER_RIGHTS	-22	Handle is not allowed (not owner) to set baudrate. Please take a look at function canSetBaudrateForce.
NTCAN_FIRMWARE_TOO_OLD	-23	Installed firmware not working with this API version.
NTCAN_FIRMWARE_UNSUPPORTED	-24	Automatic firmware-upgrade has detected an invalid firmware

[1] In the „restricted“ version: more than 4 handles.

© 2001 – 2017, Sontheim Industrie Elektronik

		(checksum mismatch). Installed firmware is not working with windows (RMos FW on PowerPCI)
NTCAN_FIRMWAREUPDATE_FAILED	-25	Firmware-upgrade failed.
NTCAN_HARDWARE_NOT_SUPPORTED	-26	Hardware is not longer supported.
NTCAN_FILE_NOT_FOUND	-27	Opening a file failed.
NTCAN_DEVICE_INFO_NOTAVAILABLE	-100	Getting device information failed (PowerPCI & PC104)
NTCAN_DEVICE_NOHW_ADDRESS	-101	Resource issue (PowerPCI & PC104)
NTCAN_NO_INTERRUPT_EVENT	-102	Resource issue, no interrupt (PowerPCI & PC104)
NTCAN_NO_INTERRUPT_EVENT_SET	-103	Resource issue (PowerPCI & PC104)
NTCAN_GET_MUTEX_FAILED	-104	Software-synchronisation failed. OS resource issue.
NTCAN_NO_SHARED_MEMORY	-105	Currently not used / reserved.
NTCAN_NET_NOT_AVAILABLE	-106	Net not available (non plug & play)
NTCAN_SETBAUDRATE_TIMEOUT	-107	Currently not used / reserved.
NTCAN_EXE_ALREADYSTARTED	-108	Currently not used / reserved.
NTCAN_NOTABLE_TOCREATE_SHAREDMEMORY	-109	Currently not used / reserved.
NTCAN_HARDWARE_IN_USE	-110	Currently not used / reserved.
NTCAN_API_NOT_RUNNING	-111	Issues to communicate with windows-service
NTCAN_CHANNEL_CURR_NOT_AVAILABLE	-112	Not possible to initialize more channels.
NTCAN_BUFFER_TOO_SMALL	-113	Passed pointer to variable too small
NTCAN_TOO_MANY_BRIDGE_FILTER	-114	Only 16 bridge-filters can be set for one net. Limit is reached.
NTCAN_HARDWARENOTACTIVE	-200	Communication with hardware is not working properly. Hardware is not responding.
NTCAN_TOO_MANY_APPLICATIONS	-201	More than 512[2] different process connected to API.
NTCAN_FLUSH_TIMEOUT	-202	During an issued canFlush command, the timeout expired. Not all TxMsgs were transmitted.
NTCAN_NOSUCCESS	0xFFFF0000	Function is failing.

[2] In the „restricted“ version: more than 4 handles.

© 2001 – 2017, Sontheim Industrie Elektronik

6.2.1. Error codes of failed version switches

These error codes can only occure if switching to another version of the API failes. To be compatible with older applications which do not know these error-codes, these error-codes are only returned by the function if they are enabled. To enable it use Function: setApplicationFlags

Return value	decimal value (error number)	description
NTCAN_SWITCH_SWITCHER_NOT_RUNNING	-10000	Communication with windows-service SIECE132Switcher failed. Not possible to start or switch API-version.
NTCAN_SWITCH_COMMUNICATION_ERROR	-10001	Communication with windows-service SIECE132Switcher failed. Not possible to start or switch API-version.
NTCAN_SWITCH_DLL_NOT_AVAILABLE	-10002	SIECA132ShdSw.dll not found. Not possible to start or switch API-version.
NTCAN_SWITCH_VERSION_NOT_INSTALLED	-10003	The API-version of SIECA132.DLL is not installed on system. Not possible to start or switch API-version.
NTCAN_SWITCH_COULD_NOT_COPY_DEFAULT	-10004	Not possible to copy SIECA132.DLL in main-directory. Not possible to start or switch API-version.
NTCAN_SWITCH_START_SERVICE_FAILED	-10005	Starting of SIECE132Svr.Exe failed. Not possible to start or switch API-version.
NTCAN_SWITCH_DETECT_VERSION_FAILED	-10006	Not possible to detect the currently running API-version. Not possible to start or switch API-version.
NTCAN_SWITCH_OTHER_VERSION_IN_USE	-10007	Other API-version is running and in use. Not possible to switch API-version.
NTCAN_SWITCH_SYNCHRONIZATION_ERROR	-10008	Synchronisation-issues. Not possible to start or switch API-version.

© 2001 – 2017, Sontheim Industrie Elektronik

7. API functions

7.1. Initialization

7.1.1. Function: canOpen

7.1.1.1. Function name and description

Function name	**canOpen**
Description	This function creates a handle for read and write operations. In full version max. 512 handles and in restricted version max. 4 handles can be created A call of **canOpen** has to be done before other CAN operations are called, because a handle is required for most functions.

7.1.1.2. Syntax

Syntax	long	canOpen (long l_netnumber, long l_mode, long l_echoon, long l_txtimeout, long l_rxtimeout, const char* c_Applicationname const char* c_ReceiverEvent const char* c_ErrorEvent void** handle);

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.1.3. Parameter and Return

Parameter	l_netnumber	Number of desired net. Please look at 7.1.1.4
	l_mode	Bit 0: 1, Errorframes are reported as CANFrame via canRead. Bit 1: 1, If a software-buffer overflow occurs, the error-event is only signaled for the corresponding handle and not for all handles which are bound to the net
	l_echoon	Bit 0: 0, no Echo Bit 0: 1, all transmitted messages are also received.
	l_txtimeout	Timeout in msec for transmitting access (-1..65535) -1,0 -> Timeout is INFINITE
	l_rxtimeout	Timeout in msec for reading access (-1..65535) -1,0 -> Timeout is INFINITE
	c_Applicationname	Name of the application, important for the identification
	c_ReceiverEvent	Name of the “receive event”, which is signale if a messages was received
	c_ErrorEvent	Name of the ‚error event’, wich is signal if an error has occurred,
	handle	If function is succesfull you get back here a pointer (handle), which must be used in further functions.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER NTCAN_TOO_MANY_HANDLES NTCAN_INIT_ERROR NTCAN_DRIVER_ERROR NTCAN_FIRMWARE_UNSUPPORTED NTCAN_HARDWARE_NOT_FOUND NTCAN_INVALID_NETNUMBER NTCAN_DRIVER_NOT_INSTALLED NTCAN_FIRMWARE_TOO_OLD NTCAN_FIRMWARE_UNSUPPORTED NTCAN_FIRMWAREUPDATE_FAILED NTCAN_HARDWARE_NOT_SUPPORTED NTCAN_FILE_NOT_FOUND NTCAN_DEVICE_INFO_NOTAVAILABLE NTCAN_DEVICE_NOHW_ADDRESS NTCAN_NO_INTERRUPT_EVENT NTCAN_NO_INTERRUPT_EVENT_SET NTCAN_GET_MUTEX_FAILED NTCAN_API_NOT_RUNNING NTCAN_CHANNEL_CURR_NOT_AVAILABLE NTCAN_HARDWARENOTACTIVE NTCAN_TOO_MANY_APPLICATIONS

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.1.4. Available net-numbers

The API supports different CAN-interfaces. Each CAN-channel has a unique net-number.

Hardware	Slot	CAN channel	Net-number
PowerPCI, PowerPCI V2, PC104Plus, **(third channel not supported)**	0	1 2 3	0 1 -
CANAS **(no longer supported)**	0	1 2 -	15 16 -
CANUSB	-	1 2 -	21 22 -
CANUSB **(supported in MT-API 6.5.9.0)**	-	1 2 -	24 25 -
Virtual device	-	1 2 -	27 28 29
Virtual device	-	1 2 -	30 31 32
Virtual device	-	1 2 -	33 34 35
Virtual device	-	1 2 -	36 37 38
Virtual device	-	1 2 -	39 40 41
MobiCAN		1 2 -	90 91
CanFox		1 - -	105

Currently, there is only one PowerPCI supported in system. If there is installed a PowerPCI V1 and a PowerPCI V2 than the MT-API will always use PowerPCI V2.

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.1.5. Example

```
HANDLE htEventR;
HANDLE htEventE;

long fn_MyCANFunction ( void ) {

      long  l_netnumber = 1;              // PowerPCI CAN 2
      long  l_txtimeout = 1000;           // msec
      long  l_rxtimeout = 1000;           // msec
      HANDLE      handle;
      long  l_retval;

     // create events
    htEventR = CreateEvent (NULL,TRUE,FALSE,"R2");
    htEventE = CreateEvent (NULL,TRUE,FALSE,"E2");


      // open new handle
      l_retval = canOpen (l_netnumber,
                          0,
                          0,
                          l_txtimeout,
                          l_rxtimeout,
                          "c_MyFirstApplication",
                          "R1",
                          "E1",
                          &handle);
      // successful?
      if ( l_retval != NTCAN_SUCCESS ) {
            // error
            printf ( "error occurred during canOpen!\n");
            return (-1);
      }
      ……
      // useful code
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.1. Function: canOpenSH

7.1.1.1. Function name and description

Function name	**canOpenSH**
Description	This function is used in the same way as function canOpen. The difference to canOpen is that canOpenSH uses only small receive buffers (512 messages)

7.1.1.1. Syntax

Please take a look at Function: canOpen

7.1.1.2. Parameter and Return

Please take a look at Function: canOpen

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.2. Function: canClose

7.1.2.1. Function name and description

Function name	**canClose**
Description	This function closes a handle, which was opened with function canOpen.

7.1.2.2. Syntax

Syntax	long canClose (void* handle);

7.1.2.1. Parameter and Return

Parameter	handle	Pointer (handle) from function canOpen.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.2.2. Example

```
HANDLE htEventR;
HANDLE htEventE;

long fn_MyCANFunction ( void )
{
      long  l_netnumber = 1;              // channel1
      long  l_txtimeout = 1000;           // msec
      long  l_rxtimeout = 1000;           // msec
      void* handle = NULL;
      long  l_retval = 0;

      // create events
      htEventR = CreateEvent (NULL,TRUE,FALSE,"R1");
      htEventE = CreateEvent (NULL,TRUE,FALSE,"E1");


      // open new handle
      l_retval = canOpen (l_netnumber,
                          0,
                          0,
                          l_txtimeout,
                          l_rxtimeout,
                          "c_MyFirstApplication",
                          "R1"
                          "E1",
                          &handle);
      // successful?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "error occurred during canOpen!\n");
            return (-1);
      }
      ……
      // useful code
      ……
      // close handle
      l_retval = canClose ( handle );
      // successful ?
      if (l_retval != NTCAN_SUCCESS)
      {
         // error
         return (-1); // error during can close
      }
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.3. Function: canSetBaudrate

7.1.3.1. Function name and description

Function name	**canSetBaudrate**
Description	This function initializes the CAN interface with the delivered baudrate. Only the owner of a net can initialize or change the baudrate. Normally the first thread which calls canOpen to a specified net gets the master. Please take a look to the following functions: - Function: canSetBaudrateForce - Function: canIsNetOwner - Function: canSetOwner - **Fehler! Verweisquelle konnte nicht gefunden werden.**

7.1.3.2. Syntax, Parameter and Return

Syntax	long canSetBaudrate (void handle, long l_baud);	
Parameter	handle l_baud	Pointer (handle) from function canOpen. Baudrate

© 2001 – 2017, Sontheim Industrie Elektronik

		_baud — Bitrate (kBit/s) Bit 31 = „0“ The predefined baudrates were used. 0x00000000 — 1000 0x00000001 — 800 0x00000002 — 500 0x00000003 — 250 0x00000004 — 125 0x00000005 — 100 0x00000006 — 50 0x00000007 — 20 Bit 31 = „1“ The CAN-timing-register (BTR) could be set byself — Please take a look at chapter 14 Baud rate calculation Bit 30 = „0“ Acknowledge on Bit 30 = „1“ Acknowledge off (silentmode) — Please take a look at chapter 11 Hardware features Bit 29 = „0“ CAN High Speed Bit 29 = „1“ CAN Low Speed — Please take a look at chapter 11 Hardware features Bit 16-28 Value for Errorframe detection — Please take a look at chapter 14 Baud rate calculation Bit 8-15 BTR 1 — Please take a look at chapter 14 Baud rate calculation Bit 0-7 BTR 0 — Please take a look at chapter 14 Baud rate calculation
Return	long	NTCAN_SUCCESS NTCAN_INIT_ERROR NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_DLL_ALREADY_INIT NTCAN_NO_OWNER_RIGHTS NTCAN_HARDWARENOTACTIVE NTCAN_CHANNEL_CURR_NOT_AVAILABLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.3.3. Example: Predefined baudrates

```
long fn_MyCANFunction ( void )
{

      HANDLE      handle;
      long        l_retval;
      long        l_baud = 0x00000002;    // 500kbit/sec

      // open handle
      ……….

      // set baudrate to 500kbit/sec
      l_retval = canSetBaudrate ( handle, l_baud );
      // successful ?
      if ( l_retval != NTCAN_SUCCESS )
      {

            // error
            printf ( "could not initialize CAN!\n");
            return (-1);
      }
      // useful code
      ……
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.3.4. Example: Direct setting of the baudrate with Btr0 and Btr1

Hint:
How to calculate the bitrate-timing-register is explained in chapter: Baud rate calculation

```
long fn_MyCANFunction ( void )
{

      HANDLE            handle;
      long              l_retval;
      long              l_baud = 0;
      long              l_btr0 = 0x13;          // 100 KBit with
      long              l_btr1 = 0x7F;          // 68% Samplerate


      // open handle
      ……….

      // set Bit 31 to 1
      l_baud |= (1<<31);

      // put btr0 and btr1 in variable l_baud
      l_baud |= (l_btr0 << 8);
      l_baud |= (l_btr1);

      // set baudrate
      l_retval = canSetBaudrate ( handle, l_baud );

      // successful ?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not initialize CAN!\n");
            return (-1);
      }

      // useful code
      ……
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.4. Function: canSetBaudrateForce

7.1.4.1. Function name and description

Function name	**canSetBaudrateForce**
Description	This function initializes the CAN interface with the delivered baudrate. In difference to function canSetBaudrate, this function takes not care who is owner of the net. This meens that the baudrate will be set and the currently used handle gets owner of the net. Please take a look to the following functions: - Function: canSetBaudrate - Function: canIsNetOwner - Function: canSetOwner - **Fehler! Verweisquelle konnte nicht gefunden werden.**

7.1.4.1. Syntax, Parameter and Return

Syntax	long canSetBaudrateForce (void handle, long l_baud);	
Parameter	handle l_baud	Pointer (handle) from function canOpen. Baudrate

© 2001 – 2017, Sontheim Industrie Elektronik

		_baud — Bitrate (kBit/s) Bit 31 = „0“ The predefined baudrates were used. 0x00000000 — 1000 0x00000001 — 800 0x00000002 — 500 0x00000003 — 250 0x00000004 — 125 0x00000005 — 100 0x00000006 — 50 0x00000007 — 20 Bit 31 = „1“ The CAN-timing-register (BTR) could be set byself — Please take a look at chapter Baud rate calculation Bit 30 = „0“ Acknowledge on Bit 30 = „1“ Acknowledge off (silentmode) — Please take a look at chapter Hardware features Bit 29 = „0“ CAN High Speed Bit 29 = „1“ CAN Low Speed — Please take a look at chapter Hardware features
Return	long	NTCAN_SUCCESS NTCAN_INIT_ERROR NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_DLL_ALREADY_INIT NTCAN_NO_OWNER_RIGHTS NTCAN_HARDWARENOTACTIVE NTCAN_CHANNEL_CURR_NOT_AVAILABLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.5. Function: canIsNetOwner

7.1.5.1. Function name and description

Function name	**canIsNetOwner**
description	This function checks if the used handle is owner of the net or not. Please take a look to the following functions: - Function: canSetBaudrate - Function: canSetBaudrateForce - Function: canSetOwner - **Fehler! Verweisquelle konnte nicht gefunden werden.**

7.1.5.2. Syntax, Parameter and Return

Syntax	long canIsNetOwner (void* handle);	
Parameter	handle	Handle, which was opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.6. Function: canSetOwner

7.1.6.1. Function name and description

Function name	**canSetOwner**
Description	This function sets the passed handle as owner of the net. Please take a look to the following functions: - Function: canSetBaudrate - Function: canSetBaudrateForce - Function: canIsNetOwner - **Fehler! Verweisquelle konnte nicht gefunden werden.**

7.1.6.2. Syntax, Parameter and Return

Syntax	long canSetBaudrate (long l_netnumber, // NetNumber void* newOwner // Handle);	
Parameter	l_netnumber newOwner	Netnumber on which the owner should be changed Handle from canOpen, which should be the new owner of the net.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_NETNUMBER

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.7. Function: canGetOwner

7.1.7.1. Function name and description

Function name	**canGetOwner**
Description	This function gets the current handle which is owner of the passed net. Please take a look to the following functions: - Function: canSetBaudrate - Function: canSetBaudrateForce - Function: canIsNetOwner - Function: canSetOwner

7.1.7.2. Syntax, Parameter and Return

Syntax	long canGetOwner (long l_netnumber, void** handle);	
Parameter	l_netnumber handle	Netnumber which owner should be determined If function is succesfull you get back here the pointer (handle), which is owner of the net.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_NETNUMBER NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.8. Function: canIdAdd

7.1.8.1. Function name and description

Function name	**canIdAdd**
Description	This function enables an identifier, so it can pass the software-filter.

7.1.8.2. Syntax, Parameter and Return

Syntax	long canIdAdd (void* handle, long l_id);	
Parameter	handle l_id	Handle which is opened with canOpen 11Bit-Identifier (0..2047dez)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_ID_ALREADY_ENABLED NTCAN_INVALID_PARAMETER

To use extended identifier, please take a look at chapter: 8 Extended

There are several modes for filtering messages which are described at chapter: 9 Extended Filter Modes

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.8.3. Example

```
long fn_MyCANFunction ( void )
{

      HANDLE      handle;
      long        l_retval;
      long        l_id;

      // open handle
      ……

      // enable identifier 0x100 for new handle
      l_id = 0x100;
      l_retval = canIdAdd ( handle, l_id );
      // successful ?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not enable identifier!\n");
            return (-1);
      }
      // useful code
      ……
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.9. Function: canIdAddArray

7.1.9.1. Function name and description

Function name	**canIdAddArray**
Description	This function enables an array of identifier, so they can pass the software-filter.

7.1.9.2. Syntax, Parameter and Return

Syntax	long	canIdAddArray
		(
		  void* handle,
		  unsigned char* aby_id
		);

Parameter	handle	Handle which is opened with canOpen
	aby_id	Pointer to an array, which is at least a size of 2048 elements. Each element represents an identifier. If element has value 0, identifier is blocked, if it is 1 identifier could pass filter.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER#

To use extended identifier, please take a look at chapter: 8 Extended

There are several modes for filtering messages which are described at chapter: 9 Extended Filter Modes

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.9.3. Example

```
long fn_MyCANFunction ( void )
{

      HANDLE            handle;
      long              l_retval;
      unsigned char     aby_id[2048] = {0};

      // open handle
      ……

      // enable identifier 0x100 and 0x211 for new handle
      aby_id[0x100] = 1;
      aby_id[0x211] = 1;
      l_retval = canIdAddArray ( handle, aby_id );
      // successful ?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not enable identifier!\n");
            return (-1);
      }
      // useful code
      ……
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.10. Function: canIdDelete

7.1.10.1. Function name and description

Function name	**canIdDelete**
Description	This function disables an identifier which was enabled before with function canIdAdd, so it won't pass any longer the software-filter.

7.1.10.2. Syntax, Parameter and Return

Syntax	long canIdDelete (void* handle, long l_id);	
Parameter	handle l_id	Handle which is opened with canOpen 11Bit-Identifier (0..2047dez)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_ID_NOT_ENABLED NTCAN_INVALID_PARAMETER

To use extended identifier, please take a look at chapter: 8 Extended

There are several modes for filtering messages which are described at chapter: 9 Extended Filter Modes

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.10.3. Example

```
long fn_MyCANFunction ( void )
{

      HANDLE      handle;
      long        l_retval;
      long        l_id;

      // open handle
      ……

      // enable identifier 0x100 for new handle
      l_id = 0x100;
      l_retval = canIdAdd ( handle, l_id );
      // successful ?
      if ( != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not enable identifier!\n");
            return (-1);
      }
      // useful code
      ……
      // delete enabled identifier
      l_id = 0x100;
      l_retval = canIdDelete ( handle, l_id );
      // successful
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not delete identifier!\n");
      }

      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.11. Function: canIdDeleteArray

7.1.11.1. Function name and description

Function name	**canIdDeleteArray**
Description	This function disables all enabled identifier.

7.1.11.2. Syntax, Parameter and Return

Syntax	long canIdDeleteArray (void* handle);	
Parameter	handle	Handle which is opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

To use extended identifier, please take a look at chapter: 8 Extended

There are several modes for filtering messages which are described at chapter: 9 Extended Filter Modes

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.11.3. Example

```
long fn_MyCANFunction ( void )
{

      HANDLE      handle;
      long        l_retval;
      unsigned char aby_id[2048] = {0};

      // open handle
      ……

      // enable identifier 0x100 and 0x211 for new handle
      aby_id[0x100] = 1;
      aby_id[0x211] = 1;
      l_retval = canIdAddArray ( handle, aby_id );
      // successful ?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not enable identifier!\n");
            return (-1);
      }
      // useful code
      ……
      // delete all enabled identifier
      l_retval = canIdDeleteArray ( handle );
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not delete identifier!\n");
            return (-1);
      }

      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.12. Function: canIDStatus

7.1.12.1. Function name and description

Function name	**canIDStatus**
Description	This function fetches the current array of enabled and disabled identifiers.

7.1.12.2. Syntax, Parameter and Return

Syntax	long canIDStatus (void* handle, T_ID_ARRAY* idarray);	
Parameter	handle *idarray	Handle which was opened with canOpen Pointer to a structure T_ID_ARRAY (see 6.2 Data structure identifier-Array) In structure is an unsigned char array with 2048 elements. Each element represents an identifier, If element is 1, the identifier can pass the software-filter. If it is 0, the identifier cannot pass the software-filter.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

To use extended identifier, please take a look at chapter: 8 Extended

There are several modes for filtering messages which are described at chapter: 9 Extended Filter Modes

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.12.3. Example

```
long fn_MyCANFunction ( void )
{

      HANDLE            handle;
      long              l_retval;
      long              l_id;
      T_ID_ARRAY        t_idarray;

      // open handle
      ……

      // enable identifier 0x100 for new handle
      l_id = 0x100;
      l_retval = canIDAdd ( handle, l_id );
      // successful ?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "could not enable identifier!\n");
            return (-1);
      }
      // useful code
      ……

      // get identifier status
      l_retval = canIDStatus ( handle, &t_idarray );
      // successful?
      if ( l_retval != NTCAN_SUCCESS )
      {
            // error
            printf ( "invalid handle for canIDStatus!\n");
            return ( -1 );
      }
      // check, whether identifier 0x100 is enabled
      if ( t_idarray.aby_IDField[0x100] == 1 )
      {
            // identifier enabled
            printf ( "identifier 0x100 enabled!\n);
      }
      else {
            // identifier not enabled
            printf ( "identifier 0x100 not enabled!\n");
      }

      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.13. Function: canEnableAllIds

7.1.13.1. Function name and description

Function name	**canEnableAllIds**
Description	With this function the ID-Filter can be switched-off completely (and switched-on again). If the ID-Filter was switched-off, all CAN-Identifier come always through (independent of that, what was set with canIdAdd or canIdAddArray.

7.1.13.2. Syntax, Parameter and Return

Syntax	long canEnableAllIds (void* handle, bool b_EnableAllIds);	
Parameter	handle b_EnableAllIds	Handle which was opened with canOpen If true, all identifier can pass the software-filter. If it is false, identifier must be enabled with function canIdAdd and/or canIdAddArray to pass the software-filter.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.13.3. Example

```
HANDLE    handle;
long      l_retval;

// open handle
……

// enable all CAN identifiers
l_retval = canEnableAllIds(handle, true);
if (l_retval != NTCAN_SUCCESS)
{
    // Error handling...
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.14. Function: canAreAllIdsEnabled

7.1.14.1. Function name amd description

Function name	**canAreAllIdsEnabled**
Description	This function fetches the current status of the software-filter is enabled or disabled.

7.1.14.2. Syntax, Parameter and Return

Syntax	long canAreAllIdsEnabled (void* handle, bool* pb_AllIdsEnabled);	
Parameter	handle	Handle which was opened with canOpen
	pb_AllIdsEnabled	Pointer to the bool variable, in which the status of the software-filter is stored,
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.14.3. Example

```
HANDLE    handle;
long      l_retval;

// open handle
……

// enable all CAN identifiers
l_retval = canEnableAllIds(handle, true);
if (l_retval != NTCAN_SUCCESS)
{
    // Error handling...
}

bool b_allIdsEnabled;

l_retval = canAreAllIdsEnabled(handle, b_allIdsEnabled);
if (l_retval != NTCAN_SUCCESS)
{
    // Error handling...
}
printf("ID filter is %s.\r\n", ((b_allIdsEnabled)? "off" : "on"));
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.15. Function: canSetFilterMode

7.1.15.1. Function name and description

Function name	**canSetFilterMode**
Description	This function is used to switch to a different filter-mode of the CAN-filter in the API. Please take a look at 9 Extended Filter Modes for futher details.

7.1.15.2. Syntax, Parameter and Return

Syntax	long canSetFilterMode (void* handle T_FILTER_MODE t_mode);	
Parameter	handle t_ mode	Handle which was opened with canOpen Desired filter-mode for CAN-messages (see 6.1 Enum of the different filter-modes)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.16. Function: canGetFilterMode

7.1.16.1. Function name and description

Function name	**canGetFilterMode**
Description	This function is used to get the current filter-mode of the CAN-filter in the API. Please take a look at 9 Extended Filter Modes for futher details.

7.1.16.2. Syntax, Parameter and Return

Syntax	long canGetFilterMode (void* handle T_FILTER_MODE* t_mode);	
Parameter	handle t_ mode	Handle which was opened with canOpen Pointer to structure T_FILTER_MODE where current filter-mode is set on success.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.17. Function: canSetFilterJ2534

7.1.17.1. Function name and description

Function name	**canSetFilterJ2534**
Description	This function sets the range of identifiers which should be passed or blocked by the CAN-filter. This function is only valid if filter-mode is set to “filterMode_j2534” with function canSetFilterMode. The parameter t_mode in this function is used to include or exclude the range of identifier. If a CAN-identifer matches an included and an excluded range, than the CAN-message will be blocked. This filter do not differ between 11 Bit / 29 Bit identifier or standard/remote-frames Each logical channel (canOpen) can handle 10 filters. Please take a look at 9 Extended Filter Modes for futher details.

7.1.17.2. Syntax, Parameter and Return

Syntax	long	canSetFilterJ2534 (void* handle T_J2534_MODE t_mode unsigned long ul_rangestart unsigned long ul_rangestop);
Parameter	handle	Handle which was opened with canOpen
	t_ mode	Desired mode for this filter (see 6.1 Enum for include and exclude filter). Value j2534Mode_excl (0) blocks all CAN-identifier in this range by the CAN-filter. Value j2534Mode_incl (1) passes all CAN-identifier in this range by the CAN-filter
	ul_rangestart	Identifier which sets the beginning of the range (0 hex – 1FFFFFFF hex)
	ul_rangestop	Identifier which sets the end of the range (0 hex – 1FFFFFFF hex)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_TOO_MANY_J2534_RANGES

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.18. Function: canDeleteFilterJ2534

7.1.18.1. Function name and description

Function name	**canDeleteFilterJ2534**
Description	This function deletes all filters which are set by the function canSetFilterJ2534. It is not possible to delete a specific filter. This function is only valid if filter-mode is set to “filterMode_j2534” with function canSetFilterMode. Please take a look at 9 Extended Filter Modes for futher details.

7.1.18.2. Syntax, Parameter and Return

Syntax	long canDeleteFilterJ2534 (void* handle);	
Parameter	handle	Handle which was opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.19. Function: canSetFilterJ2534_2

7.1.19.1. Function name and description

Function name	**canSetFilterJ2534_2**
Description	This function filters CAN-messages with a combination of masks and patterns. It filters CAN-messages by identifier and/or by data. This function is only valid if filter-mode is set to “filterMode_j2534_2” with function canSetFilterMode. The parameter t_mode in this function is used to set this filter as include or exclude filter. If a CAN-identifer matches an included and an excluded filter than the CAN-message will be blocked. The CAN message has to comply with the following conditions to be affected by the filter. (((identifier & ul_mask_id) == (ul_pattern_id & ul_mask_id)) && ((candata & ul_mask_data) == (ul_pattern_data & ul_mask_data))) Each logical channel (canOpen) can handle 10 filters. Please take a look at 9 Extended Filter Modes for futher details.

7.1.19.2. Syntax, Parameter and Return

Syntax	long	canSetFilterJ2534_2 (void* handle T_J2534_MODE t_mode unsigned long ul_mask_id, unsigned _int64 ul_mask_data, unsigned long ul_pattern_id, unsigned _int64 ul_pattern_data);

© 2001 – 2017, Sontheim Industrie Elektronik

Parameter	handle	Handle which was opened with canOpen
	t_ mode	Desired mode for this filter (see 6.1 Enum for include and exclude filter). Value j2534Mode_excl (0) blocks all matching CAN-messages Value j2534Mode_incl (1) passes all matching CAN-messages
	ul_mask_id	Mask for the identifier Bit 0 – 28 ist used for identifier itself Bit 30 is used to differ standard/remote frames Bit 31 is used to differ 11/29 bit identifier
	ul_mask_data	Mask for the data-bytes
	ul_pattern_id	Mask for the identifier Bit 0 – 28 ist used for identifier itself Bit 30 is used to differ standard/remote frames Bit 31 is used to differ 11/29 bit identifier
	ul_pattern_data	Pattern for the data-bytes
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_TOO_MANY_J2534_2_FILTERS

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.20. Function: canDeleteFilterJ2534_2

7.1.20.1. Function name and description

Function name	**canDeleteFilterJ2534_2**
Description	This function deletes all filters which are set by the function canSetFilterJ2534_2. It is not possible to delete a specific filter. This function is only valid if filter-mode is set to “filterMode_j2534_2” with function canSetFilterMode.

7.1.20.2. Syntax, Parameter and Return

Syntax	long canDeleteFilterJ2534 _2 (void* handle);	
Parameter	handle	Handle which was opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.21. Function: canSetBridgeFilter

7.1.21.1. Function name and description

Function name	**canSetBridgeFilter**
Description	This function sets a new bridge-filter. If a CAN-message matches the filter, the message is transmitted on the other net. Except the PowerPCI V2 this is done by the MT-API itself. The PowerPCI V2 has implemented this feature in the firmware.

7.1.21.2. Syntax, Parameter and Return

Syntax	long	canSetBridgeFilter (void* HandleSourceNet, void* HandleDestNet, unsigned long ulPattern, unsigned long ulMask, unsigned long ulReserved);
Parameter	HandleSourceNet HandleDestNet ulPattern ulMask ulReserved	Opened handle of the source net Opened handle of the destination net. **It is required that the handle is from the other net of the same device. Otherwise NTCAN_INVALID_PARAMETER is returned.** Pattern Mask Reserved The bits in pattern and mask have following meaning: Bit 30: Bridge also messages which are transmitted by device itself (echo-frame) Bit 29: Bridge 29 Bit identifier Bit 28-0: Identifier
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_CHANNEL_NOT_INITIALIZED NTCAN_GET_MUTEX_FAILED NTCAN_TOO_MANY_BRIDGE_FILTER

If following condition is true the CAN-message is bridged to the other net:

(Identifier & Mask) == (Pattern & Mask)

If CAN-Message is 29 Bit, bit 29 is set in Identifier, if it is an echo-frame, bit 30 is set.

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.22. Function: canGetBridgeFilter

7.1.22.1. Function name and description

Function name	**canGetBridgeFilter**
Description	This function retrieves the current set bridge-filters of the net.

7.1.22.2. Syntax, Parameter and Return

Syntax	long	canGetBridgeFilter (void* HandleSourceNet, unsigned long* pulNumBridgeFilter, st_BridgeFilter* pstBridgeFilter);
Parameter	HandleSourceNet pulNumBridgeFilter pstBridgeFilter	Opened handle of the source net Variable contains the array-size of variable pstBridgeFilter. If function is successful variable contains the numbers of set filters. Pointer to array of type st_BridgeFilter (see 6.1 Structure for bridging)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_GET_MUTEX_FAILED NTCAN_BUFFER_TOO_SMALL

© 2001 – 2017, Sontheim Industrie Elektronik

7.1.23. Function: canClearBridgeFilter

7.1.23.1. Function name and description

Function name	**canClearBridgeFilter**
Description	This function clears a set bridge-filter

7.1.23.2. Syntax, Parameter and Return

Syntax	long canClearBridgeFilter (void* HandleSourceNet, unsigned long ulDestNet, unsigned long ulPattern, unsigned long ulMask, unsigned long ulReserved);	
Parameter	HandleSourceNet ulDestNet ulPattern ulMask ulReserved	Opened handle of the source net Destination net Pattern Mask Reserved
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_GET_MUTEX_FAILED NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.2. Receiving of CAN data

7.2.1. Function: canRead

7.2.1.1. Function name and description

Function name	**canRead**
Description	Blocked reading of received CAN messages.

7.2.1.2. Syntax, Parameter and Return

Syntax	long canRead (void* handle, CMSG* canmsg, long* l_len,);	
Parameter	handle *canmsg l_len	Handle which was opened with canOpen Pointer to an element or an array of CMSG (see 6.1 Data structure of the CAN message). Set here the amount of messages, which could be stored at pointer canmsg. The value must be higher than 0. If function returns with success, the currently amount of read messages is put back.
Return	long	NTCAN_SUCCESS NTCAN_NO_ID_ENABLED NTCAN_INVALID_PARAMETER NTCAN_HARDWARE_NOT_FOUND NTCAN_CONTR_BUSOFF NTCAN_RX_TIMEOUT

Function canRead waits till timeout is expired or at least one messages is read. Timeout must be set in Function: canOpen or could be changed with Function: canSetTimeout.

© 2001 – 2017, Sontheim Industrie Elektronik

7.2.1.3. Example

```
int fn_MyCANFunction ( void )
{

      HANDLE            handle;
      long              l_retval;
      CMSG              t_CANMsg[5];
      long              l_len;
      int i, k;

      // open handle
      ……
      // set baudrate
      ……
      // add identifier
      ……
      // blocked reading
      // read 5 messages, if available
      l_len = 5;
      l_retval = canRead (handle,
                          &t_CANMsg[0],
                          &l_len);
      // successful ?
      if ( l_retval == NTCAN_SUCCESS )
      {
            // at least one message read
            // Parameter l_len contains now number of read frames
            for ( i = 0; i < l_len; i++ )
            {
                  printf ( "message nr. %ld\n", i );
                  printf ( "identifier: %03X\n",
                              t_CANMsg[i].l_id );
                  printf ( "number of data bytes: %ld\n",
                              t_CANMsg[i].by_len & 0x0F );
                  // print data bytes
                  for (k= 0; k < (t_CANMsg[i].by_len&0x0F); k++ )
                  {
                        printf ( "%02X ", t_CANMsg[i].aby_data[k] );
                  }
                  printf ( "\n\n" );
            }
      }
      return ( 0 );
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.2.2. Function: canReadNoWait

7.2.2.1. Function name and description

Function name	**canReadNoWait**
Description	Non-Blocked reading of received CAN messages.

7.2.2.2. Syntax, Parameter and Return

Syntax	long canReadNoWait (void* handle, CMSG* canmsg, long* l_len,);	
Parameter	handle *canmsg l_len	Handle which was opened with canOpen Pointer to an element or an array of CMSG (see 6.1 Data structure of the CAN message). Set here the amount of messages, which could be stored at pointer canmsg. The value must be higher than 0. If function returns with success, the currently amount of read messages is put back.
Return	long	NTCAN_SUCCESS NTCAN_NO_ID_ENABLED NTCAN_INVALID_PARAMETER NTCAN_HARDWARE_NOT_FOUND NTCAN_CONTR_BUSOFF NTCAN_RX_TIMEOUT NTCAN_CHANNEL_NOT_INITIALIZED

Function canReadNoWait is the non-blocking version of the function canRead. It can be used for polling CAN-messages. It should be used only rare cases, because polling causes a high CPU-load.

© 2001 – 2017, Sontheim Industrie Elektronik

7.3. Transmission of CAN data

7.3.1. Function: canConfirmedTransmit

7.3.1.1. Function name and description

Function name	**canConfirmedTransmit**
Description	Function is transmitting CAN messages on the bus and waits till message is acknowledged from another CAN-node.

7.3.1.2. Syntax, Parameter and Return

Syntax	long canConfirmedTransmit (void* handle, CMSG* canmsg, long* l_len);	
Parameter	handle	Handle which was opened with canOpen
	canmsg	Pointer to an element or an array of CMSG (see 6.1 Data structure of the CAN message).
	l_len	Size of stored messages at pointer canmsg. The value must be higher than 0. If functions returns, the number of sucessfull transmitted CAN-messages are stored in the variable.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_CHANNEL_NOT_INITIALIZED NTCAN_INVALID_PARAMETER NTCAN_TX_TIMEOUT

Function canConfirmedTransmit puts the messages into the TX-buffer and waits until the messages are trasnmitted out on bus and are acknowledged from another CAN-node. If timeout has expired before all messages could be send out, error NTCAN_TX_TIMEOUT is returned and variable l_len contains the number of successfull transmitted CAN-messages. Timeout must be set in Function: canOpen or could be changed with Function: canSetTimeout.

© 2001 – 2017, Sontheim Industrie Elektronik

7.3.1.3. Example

```
int fn_MyCANFunction ( void )
{
      HANDLE            handle;
      Long              l_retval;
      CMSG              t_CANMsg[10];
      long              l_len;

      // open handle
      ……
      // set baudrate
      ……
      // set number of CAN frames
      l_len = 2; // two frames
      // fill with CAN data
      t_CANMsg[0].l_id = 0x1FF;
      t_CANMsg[0].by_len = 2;
      t_CANMsg[0].aby_data[0] = 0x11;
      t_CANMsg[0].aby_data[1] = 0x22;
      t_CANMsg[0].by_extended = 0;
      t_CANMsg[0].by_remote = 0;
      t_CANMsg[1].l_id = 0x32;
      t_CANMsg[1].by_len = 5;
      t_CANMsg[1].by_extended = 0;
      t_CANMsg[1].by_remote = 0;
      t_CANMsg[1].aby_data[0] = 0xAA;
      t_CANMsg[1].aby_data[1] = 0xBB;
      t_CANMsg[1].aby_data[2] = 0xCC;
      t_CANMsg[1].aby_data[3] = 0xDD;
      t_CANMsg[1].aby_data[4] = 0xEE;

      // blocked transmit
      l_retval = conConfirmedTrasnmit (   handle,
                                          t_CANMsg[0],
                                          &l_len );

      if ( l_retval == NTCAN_SUCCESS )
      {
            printf ( "all frames sent!\n");
            return ( 0 );
      }
      if ( l_retval == NTCAN_TX_ERROR )
      {
            // error
            printf ( "error, only %ld frames sent!\n", l_len );
            return (-1);
      }
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.3.2. Function: canSend / canWrite

7.3.2.1. Function name and description

Function name	**canSend / canWrite**
Description	Functions are transmitting CAN messages. The messages are only put into the transmit-queue and the function returns. Unlike Transmission of CAN data Function: canConfirmedTransmit these two functions do not wait till messages are transmitted and acknowledged.

7.3.2.2. Syntax, Parameter and Return

Syntax	long canSend / canWrite (void* handle, CMSG* canmsg, long* l_len);	
Parameter	handle	Handle which was opened with canOpen
	canmsg	Pointer to an element or an array of CMSG (see 6.1 Data structure of the CAN message).
	l_len	Size of stored messages at pointer canmsg. The value must be higher than 0. If functions returns, the number of sucessfull transmitted CAN-messages are stored in the variable.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_CHANNEL_NOT_INITIALIZED NTCAN_INVALID_PARAMETER NTCAN_TX_TIMEOUT

Functions canSend / canWrite put the messages into the TX-buffer. If timeout has expired before all messages could be stored into TX-buffer, error NTCAN_TX_TIMEOUT is returned and variable l_len contains the number of successfull stored CAN-messages. Timeout must be set in Function: canOpen or could be changed with Function: canSetTimeout.

© 2001 – 2017, Sontheim Industrie Elektronik

7.3.2.3. Example

```
int fn_MyCANFunction ( void )
{

      HANDLE            handle;
      long              l_retval;
      CMSG              t_CANMsg[10];
      long        l_len;

      // open handle
      ……
      // set baudrate
      ……
      // set number of CAN frames
      l_len = 2; // two frames
      // fill with CAN data
      t_CANMsg[0].l_id = 0x1FF;
      t_CANMsg[0].by_len = 2;
      t_CANMsg[0].by_extended = 0;
      t_CANMsg[0].by_remote = 0;
      t_CANMsg[0].aby_data[0] = 0x11;
      t_CANMsg[0].aby_data[1] = 0x22;
      t_CANMsg[1].l_id = 0x32;
      t_CANMsg[1].by_len = 5;
      t_CANMsg[1].by_extended = 0;
      t_CANMsg[1].by_remote = 0;
      t_CANMsg[1].aby_data[0] = 0xAA;
      t_CANMsg[1].aby_data[1] = 0xBB;
      t_CANMsg[1].aby_data[2] = 0xCC;
      t_CANMsg[1].aby_data[3] = 0xDD;
      t_CANMsg[1].aby_data[4] = 0xEE;
      // blocked transmit
      l_retval = canSend ( handle,
                           &t_CANMsg[0],
                           &l_len );

      if ( l_retval == NTCAN_SUCCESS )
      {
            printf ( "all frames copied to transmit queue!\n");
            return ( 0 );
      }
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.3.3. Function: canFlush

7.3.3.1. Function name and description

Function name	**canFlush**
Description	This function ensures that all TxMessages, which were issued via Function: canSend / canWrite, are completively transmitted over the CanBus. The call to this function will not return until either the defined timeout runs out, or there are no more messages within the transmit-queue and the hardware send-buffer.

7.3.3.2. Syntax, Parameter and Return

Syntax	long canFlush (void* handle, long l_timeout);	
Parameter	handle	Handle which was opened with canOpen
	l_timeout	Timeout in range from -1..65535 ms -1,0 means INFINITE wait time
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER NTCAN_INVALID_HANDLE NTCAN_INIT_ERROR NTCAN_FLUSH_TIMEOUT

Important: This function is currently only supported by CANUSB!

This function is intended to be called only just before the Function: canClose gets called.

© 2001 – 2017, Sontheim Industrie Elektronik

7.3.3.3. Example

```
int fn_MyCANFunction ( void )
{

      HANDLE            handle;
      long              l_retval;
      long              l_timeout = 1000;

      // open handle
      ……
      // set baudrate
      ……
      // transmit a high number of CAN frames
      ……
      // flush channel
      l_retval = canFlush ( handle,
                            l_timeout );

      if ( l_retval == NTCAN_SUCCESS )
      {
            printf ( "all frames sent!\n");
      }
      else if ( l_retval == NTCAN_FLUSH_TIMEOUT )
      {
            // error
            printf ( "error, flush interrupted!" );
      }
      else
      {
            // error
            printf ( "error during canFlush!" );
      }

      // close handle
      canClose ( handle );

}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.4. Read status of CAN device

7.4.1. Function: canStatus

7.4.1.1. Function name and description

Function name	**canStatus**
Description	The function fetches several hardware related information like firmware-version or set baudrate.

7.4.1.2. Syntax, Parameter and Return

Syntax	long	canStatus (void*, handle CAN_IF_STATUS* Status);
Parameter	handle	Handle which was opened with canOpen
	canstatus	Pointer to structure CAN_IF_STATUS (see 6.3 Data structure Status)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

Variable w_errorflag in structure CAN_IF_STATUS is set to 0 after canStatus is called

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.1.3. Example

```
int fn_MyCANFunction ( void )
{

      HANDLE                  handle;
      long                    l_retval;
      CAN_IF_STATUS           t_CANStatus;

      // open handle
      ……
      // set baudrate
      ……
      // add identifier
      ……
      // do anything with the card
      ……
      l_retval = canStatus ( handle, &t_CANStatus );
      if ( l_retval == NTCAN_SUCCESS )
      {
            printf ( "status from card:\n");
            printf ( "HW Rev.  : %04X\n", t_CANStatus.w_hw_recv );
            printf ( "FW Rev.  : %04X\n", t_CANStatus.w_fw_rev );
            printf ( "DLL Rev. : %04X\n", t_CANStatus.w_dll_rev);
            printf ( "Drv Rev. : %04X\n", t_CANStatus.w_drv_rev);
            printf ( "exist. Nets: %02x", t_CANStatus.w_netctr);
            ……
      }
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.2. Function: canGetDllInfo

7.4.2.1. Function name and description

Function name	**canGetDllInfo**
Description	This function is implemented to be compatible with the C167API. The function fetches different information. Because no handle is used in this function it can be used only for PowerPCI (Net 0 and 1)

7.4.2.2. Syntax, Parameter and Return

Syntax	long	canGetDllInfo (T_DllInfo* t_Info, void* Res);
Parameter	*t_Info	Pointer to the data structure T_DllInfo (see 6.4 Data structure DLL information)
	*Res	Reserved
Return	long	NTCAN_SUCCESS NTCAN_CHANNEL_NOT_INITIALIZED

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.2.3. Example

```
int fn_MyCANFunction ( void )
{

      long                    l_retval;
      T_DllInfo               t_DllInfo;

      // open handle
      ……
      // set baudrate
      ……
      // add identifier
      ……
      // do anything with the card
      ……
      l_retval = canGetDllInfo ( t_DllInfo,NULL);
      if ( l_retval == NTCAN_SUCCESS )
      {
            printf ( "status from DLL\n");
            printf ( "TxCtr1. %4X\n",t_DllInfo.aui_TxCounter[0]);
            printf ( "TxCtr1. %4X\n",t_DllInfo.aui_TxCounter[1]);
            ……
      }
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.3. Function: canGetCounter

7.4.3.1. Function name and description

Function name	**canGetCounter**
Description	This function is implemented to be compatible with the C167API. The function fetches counter information like number of received messages. Because no handle is used in this function it can be used only for PowerPCI (Net 0 and 1). Please use Function: canGetCounterExtended instead.

7.4.3.2. Syntax, Parameter and Return

Syntax	long	canGetCounter (CTRDATA* data);
Parameter	data	Pointer to structure CTRDATA (see 6.5 Data structure of the CAN counter data)
Return	long	NTCAN_SUCCESS NTCAN_CHANNEL_NOT_INITIALIZED

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.4. Function: canGetCounterExtended

7.4.4.1. Function name and description

Function name	**canGetCounterExtended**
Description	The function fetches counter information like number of received messages.

7.4.4.2. Syntax, Parameter and Return

Syntax	long	canGetCounterExtended
		(
		  void* handle,
		  CTRDATA2* data
		);

Parameter	handle *data	Handle which was opened with canOpen Pointer to structure CTRDATA2 (see 6.5 Data structure of the CAN counter data)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.1. Function: canResetCounter

7.4.1.1. Function name and description

Function name	**canResetCounter**
Description	The function resets the counter, which could be requested with Function: canGetCounterExtended.

7.4.1.2. Syntax, Parameter and Return

Syntax	long canResetCounter (void* handle);	
Parameter	handle	Handle which was opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.4.2. Function: canGetBusloadExtended

7.4.2.1. Function name and description

Function name	**canGetBusloadExtended**
Description	The function retrieves the current busload

7.4.2.2. Syntax, Parameter and Return

Syntax	long	canGetBusloadExtended
		(
		  void* handle,
		  BUSLOAD* data
		);

Parameter	Handle data	Handle which was opened with canOpen Pointer to structure BUSLOAD, where the current busload is stored. (See 6.1 Structure for busload)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5. Request and change settings

7.5.1. Function: canGetTimeout

7.5.1.1. Function name and description

Function name	**canGetTimeout**
Description	This function reads out the current values of the RX- and TX-Timeouts.

7.5.1.2. Syntax, Parameter and Return

Syntax	long	canGetTimeout (void* handle, T_TIMEOUT_TYPE t_type, long* pl_timeout);
Parameter	handle	Handle which was opened with canOpen
	t_type	timeoutType_rx = Rx-Timeout timeoutType_tx = Tx-Timeout (see 6.1 Enum for requesting and changing timeouts)
	pl_timeout	Pointer to a variable of type of long, in which the wanted Timeout-value is returned.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.1.3. Example

```
// open handle
...
// set baud rate
...
// add identifier
...
// useful code
...
// Get Rx timeout
long l_ret;
long l_rxTimeout;

l_ret = canGetTimeout(
    handle,            // Handle from canOpen.
    timeoutType_rx,    // Set Rx timeout value.
    &l_rxTimeout);     // Variable to hold the returned value.

if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}

// useful code
...
// close handle
...
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.2. Function: canSetTimeout

7.5.2.1. Function name and description

Function name	**canSetTimeout**
Description	This function sets new values for RX- or TX-Timeout

7.5.2.2. Syntax, Parameter and Return

Syntax	long	canSetTimeout (void* handle T_TIMEOUT_TYPE t_type long l_timeout);
Parameter	handle	Handle which was opened with canOpen
	t_type	timeoutType_rx = Rx-Timeout timeoutType_tx = Tx-Timeout (see 6.1 Enum for requesting and changing timeouts)
	l_timeout	New timeout in range from -1..65535 ms -1,0 means INFINITE wait time
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.2.3. Example

```
// open handle
...
// set baud rate
...
// add identifier
...
// useful code
...
// change Rx timeout to 1 second
long l_ret;
l_ret = canSetTimeout(
    handle,            // Handle from canOpen.
    timeoutType_rx,    // Set Rx timeout value.
    1000);             // New value.
if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}

// useful code
...
// close handle
...
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.3. Function: canBreakcanRead

7.5.3.1. Function name and description

Function name	**canBreakcanRead**
Description	This function aborts a blocking canRead of the passed handle.

7.5.3.2. Syntax, Parameter and Return

Syntax	long canBreakcanRead (void* handle);	
Parameter	handle	Handle which was opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.3.3. Example

In a thread canRead is called in a loop. canRead waits on every call the set RX-timeout.
In a second thread the actual waiting canRead shall be aborted after the change of the Rx-Timeout (with the function canSetTimeout), so the new RX-timeout could be used

```
// change Rx timeout
   ...
// break waiting canRead
long l_ret = canBreakcanRead(handle);
if (l_ret != NTCAN_SUCCESS)
{
   // handle error
   ...
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.4. Function: canClearBuffer

7.5.4.1. Function name and description

Function name	**canClearBuffer**
Description	This function deletes all received messages in the RX-buffer of the passed handle.

7.5.4.2. Syntax, Parameter and Return

Syntax	long canClearBuffer (void* handle);	
Parameter	handle	Handle which was opened with canOpen
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.4.1. Example

```
// open handle
...
// set baud rate
...
// add identifier
...
// useful code
...
// clear RX-buffer
long l_ret;
l_ret = canClearBuffer(
    handle)            // Handle from canOpen.

if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}

// useful code
...
// close handle
...
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.5. Function: canGetNumberOfConnectedDevices

7.5.5.1. Function name and description

Function name	**canGetNumberOfConnectedDevices**
Description	This function returns the number of connected devices. Call this function before **Fehler! Verweisquelle konnte nicht gefunden werden.** to find out how large the buffer must be.

7.5.5.2. Syntax, Parameter and Return

Syntax	long canGetNumberOfConnectedDevices (long* devices);	
Parameter	devices	Pointer to the variable where the number of connected devices should be stored.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.5.1. Example

```
long l_ret;
long l_NumDevices;

l_ret = canGetNumberOfConnectedDevices (
    &l_NumDevices)
if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.6. Function: canGetDeviceList

7.5.6.1. Function name and description

Function name	**canGetDeviceList**
Description	This function will return all available devices with there net numbers. Call Function: canGetNumberOfConnectedDevices to find out how many devices are available and what status and features they have.

7.5.6.2. Syntax, Parameter and Return

Syntax	long canGetDeviceList (T_DeviceList * list);	
Parameter	list	Pointer to the structure array where all available devices will be stored. The array must be large enough to store the data. (see 6.1 Structure for optaining the installed devices)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.6.1. Example

```
long l_ret;
long l_NumDevices;

T_DeviceList* pDeviceList = NULL;

l_ret = canGetNumberOfConnectedDevices (
    &l_NumDevices)
if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}

pDeviceList = new DeviceList[NumDevices];

l_ret = canGetDeviceList (
    pDeviceList)
if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}


delete [] pDeviceList;
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.7. Function: canGetSyncTimer

7.5.7.1. Function name and description

Function name	**canGetSyncTimer**
Description	This function shows the CAN timestamp in relation to the PC System time.

7.5.7.2. Syntax, Parameter and Return

Syntax	long	canGetSyncTimer (void* handle unsigned long* timer __int64* perfCounter);
Parameter	handle	Handle which was opened with canOpen
	timer	CAN timer of the hardware
	perfCounter	The value of the 'QueryPerformanceCounter' which fits to the CAN timer.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_GET_MUTEX_FAILED NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.8. Function: canGetDeviceTimestampBase

7.5.8.1. Function name and description

Function name	**canGetDeviceTimestampBase**
Description	This function returnsreturns the timestamp base of the device.

7.5.8.2. Syntax, Parameter and Return

Syntax	long	canGetDeviceTimestampBase
		(
		  long netnr
		  unsigned long * Base
		);

Parameter	netnr	Net number of the device.
	Base	Timestamp base in ns (nanoseconds). E.g. a value of 100000 means 100 ms (milliseconds) per CAN timer tick.
Return	long	NTCAN_SUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.9. Function: canEnableHWExtendedId

7.5.9.1. Function name and description

Function name	**canEnableHWExtendedId**
Description	This function enables or disables the receiving of extended identifier in the firmware of the PowerPCI. CAN channel 0 and 1 can be configured separately. The can channel is bound to the handle of canOpen.

7.5.9.2. Syntax, Parameter and Return

Syntax	long canEnableHWExtendedId (void * handle bool bEnableExtID);	
Parameter	handle bEnableExtID	Handle which was opened with canOpen Enable / disable of receiving extended identifier. true = enable extended identifier in firmware false = disable extended identifier in firmeare
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_NETNUMBER NTCAN_GET_MUTEX_FAILED

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.10. Function: canGetCanLevel

7.5.10.1. Function name and description

Function name	**canGetCanLevel**
Description	This function is used to do an active CAN level measurement (This function is only supported by a CANUSB with installed CAN level measurement)

7.5.10.2. Syntax, Parameter and Return

Syntax	long canGetDeviceList (void* handle T_CANLEVEL_MODE t_canLevelMode long l_Ident double* canLevelLow double* canLevelHigh);	
Parameter	handle	Handle which was opened with canOpen
	t_canLevelMode	Mode of the used CAN level measurement (see 6.1 Enum for active measuring of CAN-Level) If set to canLevelDominant, a measurement of the dominant level is done. If set to canLevelRecessive a recessive measurement is done.
	l_Ident	Identifier which will be sent out, to measure the dominant CAN level (not required to mesure the recessive CAN level) Valid values are from 0x0 to 0x7FF
	canLevelLow	In this variable is stored the low level of the CAN after measuring
	canLevelHigh	In this variable is stored the high level of the CAN after measuring
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_GET_MUTEX_FAILED NTCAN_CHANNEL_NOT_INITIALIZED

! The dominant measurement is done on a different baudrate. This cause errorframes on the bus, which might disturb some ECU, switching from operation mode into error mode.

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.11. Function: canGetCanLevelHist

7.5.11.1. Function name and description

Function name	**canGetCanLevelHist**
Description	This function is used to do a passive CAN level measurement (This function is only supported by CANUSB with installed CAN level measurement and PowerPCI V2)

7.5.11.2. Syntax, Parameter and Return

Syntax	long	canGetDeviceList (void* handle bool bReset CANLEVEL_HIST* pstCanLevelHist);
Parameter	Handle bReset CANLEVEL_HIST*	Handle which was opened with canOpen true = Reset level measurement false = No reset of level measurement Pointer to structure which contains the current values of the passive level measurement. (see 6.1 Structure for passive measuring of CAN-Level)
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_GET_MUTEX_FAILED NTCAN_CHANNEL_NOT_INITIALIZED

The difference to Function: canGetCanLevel is that this function does not transmit anything on the CAN bus. The first call of the function activates the cyclic level measurement in the firmware of the CANUSB. It measures approx. every 5ms the levels on the CAN bus. Due this fact there must be CAN traffic on the bus, to get a valid dominant level. The higher the baudrate more messages are required to get valid levels from the function. Variable bReset clears all measurements.

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.11.3. Example

```
// open handle
...
// set baud rate
...
// useful code
...
// activate cyclic level measurement
long l_ret;
CANLEVEL_HIST stHist;

l_ret = canGetCanLevelHist (
      handle,                // handle from canOpen.
      true,                  // Reset level measurement
      &stHist)               // Pointer to structure

if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}

// If there is CAN traffic on the bus, some time is required to get the CAN
// levels. If there is no traffic on the bus, the user must transmit here
// message

// Do a wait of 3 second here
Sleep(3000)

l_ret = canGetCanLevelHist (
      handle,                // handle from canOpen.
      false,                 // Reset level measurement
      &stHist)               // Pointer to structure

if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}

// useful code
...
// close handle
...
```

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.12. Function: canGetDiffTimeLastFrame

7.5.12.1. Function name and description

Function name	**canGetDiffTimeLastFrame**
Description	Function checks if CAN-traffic is on the CANBus.

7.5.12.2. Syntax, Parameter and Return

Syntax	long canGetDiffTimeLastFrame (void* handle long* pl_diffTime);	
Parameter	handle pl_diffTime	Handle which was opened with canOpen In this variable is stored the difference of time (ms) since the last received CAN-message and the call of this function. If there was no CAN-message at all the variable is set to -1. Example: Last CAN-message was received at 600000 ms system-uptime. This function is called at 6001000 ms system-uptime. The value of this variable is 1000 ms. If there is a lot of traffic on the bus, the value of this variable could be 0.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_CHANNEL_NOT_INITIALIZED

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.13. Function: canGetHWSerialNumber

7.5.13.1. Function name and description

Function name	**canGetHWSerialNumber**
Description	Function reads out the serial-number of the device. Not all devices support this feature.

7.5.13.2. Syntax, Parameter and Return

Syntax	long	canGetHWSerialNumber (void* handle unsigned long* pulSNHigh unsigned long* pulSNLow);
Parameter	handle	Handle which was opened with canOpen
	pulSNHigh	Pointer, where the high bytes of the serial-number is stored
	pulSNLow	Pointer, where the low bytes of the serial-number is stored
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_GET_MUTEX_FAILED NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.14. Function: canGetSystemTime

7.5.14.1. Function name and description

Function name	**canGetSystemTime**
Description	Function gets the current system-time, which is used for the software-synchronized timestamp.

7.5.14.2. Syntax, Parameter and Return

Syntax	long	canGetSystemTime (unsigned __int64* pui64CurrSysTime unsigned __int64* pui64StartSysTime);
Parameter	pui64CurrSysTime	Pointer, where the current system-time is stored. Each tick is 1/10 ms.
	pui64StartSysTime	Pointer, where the system-time of API-Start (service) is stored.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.15. Function: queryRunningVersion

7.5.15.1. Function name and description

Function name	**queryRunningVersion**
Description	Function queries switcher which version is currently runnning

7.5.15.2. Syntax, Parameter and Return

Syntax	long queryRunningVersion (unsigned long arrVersion[4]);	
Parameter	arrVersion	Pointer to array, where version of running API is stored.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER NTCAN_SWITCH_DLL_NOT_AVAILABLE NTCAN_SWITCH_COMMUNICATION_ERROR NTCAN_SWITCH_DETECT_VERSION_FAILED

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.16. Function: setApplicationFlags

7.5.16.1. Function name and description

Function name	**setApplicationFlags**
Description	Function sets flags

7.5.16.2. Syntax, Parameter and Return

Syntax	long	setApplicationFlags (unsigned long ulFlags);
Parameter	ulFlags	Flags Bit 0: If switching to another MT-API version fails, more specific errors are returned from the functions. (See 6.2.1 Error codes of failed version switches)
Return	long	NTCAN_SUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.17. Function: getApplicationFlags

7.5.17.1. Function name and description

Function name	**getApplicationFlags**
Description	Function gets flags

7.5.17.1. Syntax, Parameter and Return

Syntax	long getApplicationFlags (unsigned long* pulFlags);	
Parameter	pulFlags	Pointer, where Flags are stored.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_PARAMETER

© 2001 – 2017, Sontheim Industrie Elektronik

7.5.18. Function: canBlinkLED

7.5.18.1. Function name and description

Function name	**canBlinkLED**
Description	Function alters the status of the LED's of the CANUSB.

7.5.18.2. Syntax, Parameter and Return

Syntax	long	canBlinkLED (void* handle, unsigned long ulMode, unsigned long ulStatus, unsigned long ulPattern,);
Parameter	handle	Handle which was opened with canOpen
	ulMode	Control-mode of the LED's 0 = Standard, LED's displaying the current status of the CANUSB. 1 = Direct, LED's are controlled by user. The LED's are set by ulStatus 2 = Blink, LED's are controlled by user. Pattern which is set in ulPattern is displayed.
	ulStatus	Status of the LED's. Required for control-mode 1. If bit is set the corresponding LED is on. Bit1 = CAN1 Status, Bit 2 = CAN1 Daten, Bit3 = CAN2 Status, Bit 4 = CAN2 Daten
	ulPattern	Pattern of the LED's. Required for control-mode 2. 0 = CAN1 Status, CAN1 Daten, CAN2 Status, CAN2 Daten are blinking with 2 Hz 1 = CAN1 Status, CAN1 Daten, CAN2 Status, CAN2 Daten are blinking with 0,5 Hz 2 = CAN1 Status, CAN1 Daten and CAN2 Status, CAN2 Daten are blinking alternately with 0,5Hz 3 = CAN1 Status, CAN1 Status and CAN1 Daten, CAN2 Daten are blinking alternately with 0,5Hz 4 = Running light with CAN1 Status, CAN1 Daten, CAN2 Status, CAN2 Daten clockwise with 4Hz 5 = Running light with CAN1 Status, CAN1 Daten, CAN2 Status, CAN2 Daten counter-clockwise with 4Hz
Return	long	NTCAN_SUCCESS NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.6. Function to read and alter the internal eeprom of the CANUSB

The following functions can be used only with the CANUSB

7.6.1. Function: canGetEepromAccess

7.6.1.1. Function name and description

Function name	**canGetEepromAccess**
Description	Function is required to get access to the the eeprom for reading and writing data. Access is granted if a key is calculated from a seed with the right algorithm. The algorithm and the different areas are not document in this documentation.

7.6.1.2. Syntax, Parameter and Return

Syntax	long	canGetEepromAccess (void* handle, unsigned long area, key* key);
Parameter	handle area key	Handle which was opened with canOpen Area of the eeprom which should be read or altered. If contain of pointer is 0, the function stores the seed. If contain of pointer is unequal 0, it is the calculated key.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.6.2. Function: canReadEeprom

7.6.2.1. Function name and description

Function name	**canReadEeprom**
Description	Function read out the eeprom of the device. Access must be granted before with Function: canGetEepromAccess

7.6.2.2. Syntax, Parameter and Return

Syntax	long	canReadEeprom (void* handle, long address, long quantity, unsigned char* value);
Parameter	handle	Handle which was opened with canOpen
	address	Address of the eeprom which should be read
	quantity	Number of bytes to read
	value	Pointer of array where data is stored after success.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_GET_MUTEX_FAILED NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.6.3. Function: canWriteEeprom

7.6.3.1. Function name and description

Function name	**canWriteEeprom**
Description	Function writes into the eeprom of the device. Access must be granted before with Function: canGetEepromAccess

7.6.3.2. Syntax, Parameter and Return

Syntax	long	canWriteEeprom (void* handle, long address, long quantity, unsigned char* value);
Parameter	handle	Handle which was opened with canOpen
	address	Address of the eeprom which should be altered
	quantity	Number of bytes to write
	value	Pointer of array where the data is stored which should be written into the eeprom.
Return	long	NTCAN_SUCCESS NTCAN_INVALID_HANDLE NTCAN_INVALID_PARAMETER NTCAN_GET_MUTEX_FAILED NTCAN_NOSUCCESS

© 2001 – 2017, Sontheim Industrie Elektronik

7.6.4. Example

This C++ example shows only how to call the functions. It does not contain a valid area and the right seed-key algorithm.

```
// open handle
...
// Request seed

long l_ret;
unsigned long ulArea = 0x1000; // Only example, not valid.
unsigned long ulSeed = 0;
unsigned long ulKey = 0;

l_ret = canGetEepromAccess (
      handle,                 // handle from canOpen.
      ulArea,                 // Area
      &ulSeed)                // Pointer to variable which contains seed
                              // afterwards

if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}
...
// Calculate the right seed
...
// Send key
l_ret = canGetEepromAccess (
      handle,                 // handle from canOpen.
      ulArea,                 // Area
      &ulKey)                 // Pointer which contains the calculated key

if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}


// Read eeprom
long address = 0x0000;
long quantity = 8;
unsigned char arrData[8] = {0};

l_ret = canReadEeprom (
      handle,                 // handle from canOpen.
      address,                // Address
      arrData)                // Data


if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}
...
// Alter Eeprom
...
```

© 2001 – 2017, Sontheim Industrie Elektronik

```
// Write eeprom
l_ret = canWriteEeprom (
      handle,                 // handle from canOpen.
      address,                // Address
      arrData)                // Data


if (l_ret != NTCAN_SUCCESS)
{
    // handle error
    ...
}
// close handle
...
```

7.7. Internal functions with no support

The following functions are used only for internal purposes and they are not supported by Sontheim.

7.7.1. Function: canSetTxDelay

7.7.1.1. Function name and description

Function name	**canSetTxDelay**
Description	Sets globally a delay between each transmitted CAN-message

7.7.2. Function: canIsVirtualHw

7.7.2.1. Function name and description

Function name	**canIsVirtualHw**
Description	Dummy function

7.7.3. Function: canInstruction

7.7.3.1. Function name and description

Function name	**canInstruction**
Description	Activates several features in device

© 2001 – 2017, Sontheim Industrie Elektronik

7.7.4. Function: canFlashDevice

7.7.4.1. Function name and description

Function name	**canFlashDevice**
Description	Flash firmware of the device

7.7.5. Function: canSetHWFilterEmu

7.7.5.1. Function name and description

Function name	**canSetHWFilterEmu**
Description	Emulates the HW-filter of the CAN-controller for each handle.

7.7.6. Function: canEnableBusloadFiltered

7.7.6.1. Function name and description

Function name	**canEnableBusloadFiltered**
Description	Change the calculation of the busload.

7.7.7. Function: canGetBusloadFiltered

7.7.7.1. Function name and description

Function name	**canGetBusloadFiltered**
Description	Get the filtered busload if enabled.

© 2001 – 2017, Sontheim Industrie Elektronik

7.7.8. Function: canSetTxTimeout

7.7.8.1. Function name and description

Function name	**canSetTxTimeout**
Description	Set the timeout, how long a CAN message should be tried to transmit on the bus if there is no acknowledge from another CAN node.

7.7.9. Function: SetPrioritySIECE132

7.7.9.1. Function name and description

Function name	**SetPrioritySIECE132**
Description	Set the process-priority of the service.

7.7.10. Function: GetPrioritySIECE132

7.7.10.1. Function name and description

Function name	**GetPrioritySIECE132**
Description	Get the process-priority of the service.

© 2001 – 2017, Sontheim Industrie Elektronik

7.7.11. Function: canSetTxDelay

7.7.11.1. Function name and description

Function name	**createCustomSharedMemory**
Description	Create a shared memory which is hold by the service

7.7.12. Function: canSetTxDelay

7.7.12.1. Function name and description

Function name	**canSetVirtual**
Description	Routes physical channels to virtual channels

© 2001 – 2017, Sontheim Industrie Elektronik

8. Extended identifier

8.1. The Extended identifier difficulty of the SIECA132

The SIECA132 - API works as follows:
Which IDs are enabled for a handle, is stored in a 2048 Byte large array for every handle. Thereby the index of the array accords the ID and the content of the subscripted array-field (0 or 1) defines whether the ID is disabled (0) or enabled (1).

8.2. Solution for the conditioning of Extended IDs

To the 2048 Byte large array an offset is stored yet. Thus optional big IDs can be used.

Disadvantage: It is only the constricted range 0+Offset..2047+Offset useable.

Advantages:

1. At the existing SIECA132 function calls `canIdAdd`, `canIdAddArray`, `canIdDelete`, `canIdDeleteArray` and `canIDStatus` do not change anything (compatibility to existing applications).

2. No essential performance-loss, because furthermore it is only one array access necessary to find out, whether a handle has got enabled a certain identifier.

0 .. Offset – 1	**Offset .. Offset + 7FF $_{hex}$**	**Offset + 800 $_{hex}$ .. 1FFF FFFF $_{hex}$**
Not usable IDs	**usebale IDs**	**Not usable IDs**

8.2.1.1. Example

If the Offset 10000 $_{hex}$ is chosen, the identifiers 10000 $_{hex}$ till 107FF $_{hex}$ can be used.
In a program it could be look like that:

```
HANDLE handle;
long   l_ret;

// Like so far:

// canOpen
// ...

// canSetBaudrate
// ...

// New: identifier Offset 0x10000 set.
l_ret = canSetIdOffset(handle, 0x10000);
if (l_ret != NTCAN_SUCCESS)
{
   // Here error handle...
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

```
// identifier 0x10200 enable.
l_ret = canIdAdd(handle, 0x200); // Offset is added automatically internal.
if (l_ret != NTCAN_SUCCESS)
{
   // Here error handle...
}

// or also with array...
// identifier 0x10201 till 0x102FF enable.
BYTE aby_id[0x800];
memset(&aby_id[0], 0, 0x800); // With 0 initialize.
for (int i_id = 0x201; i_id < 0x300; ++i_id)
{
   aby_id[i_id] = 1;
}
l_ret = canIdAddArray(handle, &aby_id[0]);
if (l_ret != NTCAN_SUCCESS)
{
   // Here error handle...
}

// A CAN message transmit with ID 0x10201.
CMSG t_CMSG[1];
long l_len = 1;
t_CMSG[0].l_id = 0x10201; // canSend expect the right ID!
// ...
l_ret = canSend(handle, &t_CMsg[0], &l_len);
if (l_ret != NTCAN_SUCCESS)
{
   // Here error handle...
}
```

8.2.2. Hints for the application

The IDs which are handed over to following functions or returned by them, always corresponds **ID-Offset**:

- ✓ canIdAdd
- ✓ canIdAddArray
- ✓ canIdDelete
- ✓ canIdDeleteArray
- ✓ canIDStatus.

Example:

If the IDs 10220_{hex} and 10221_{hex} are enabled at Offset 10000_{hex}, canIDStatus in idarray [0x220] and idarray[0x221] return a 1.
At canSend/canWrite, canConfirmedTransmit and canRead, the real IDs are handed over in contrast to that.

8.2.3. Functions: canSetIdOffset, canGetIdOffset

extern long _declspec (dllexport) FAR WINAPI canSetIdOffset(HANDLE handle, DWORD dw_offset)
and extern long _declspec (dllexport) FAR WINAPI canGetIdOffset(HANDLE handle, LPDWORD lpdw_offset) in lpdw_offset the actual adjusted offset is returned.

The **Default-Offset** is **0**.
The offset which is adjusted with canSetIdOffset applies for the functions canIdAdd, canIdAddArray, canIdDelete, canIdDeleteArray and canIDStatus. IDs which are handed over to this functions or are returned

© 2001 – 2017, Sontheim Industrie Elektronik

from this functions, always correspond *ID - Offset*. At canSend/canWrite, canConfirmedTransmit and canRead, the real IDs are handed over in contrast to that.

9. Extended Filter Modes

To avoid problems by setting filters for 29 Bit identifiers (take a look at chapter 8) there are several filtermodes, which could be set with the function canSetFilterMode.

9.1. Modes

Filter-Mode	Description
filterMode_standard (0)	Standard-Filtermode (Default)
filterMode_j2534 (1)	Range-Based-Filtermode
filterMode_extended (2)	29 Bit-Filtermode
filterMode_j2534_2 (3)	Pattern-Mask-Based-Filtermode
filterMode_nofilter (4)	No Filter

9.1.1. Standard Filtermode

This is the standard filter mode, which is set after a new handle is opended with function canOpen. In this mode CAN-messages can by filtered by identifier. 11 bit CAN-messages, can be filtered individually by each identifier. 29 bit CAN-message (> 7FF hex) can be filtered only with the usage of offsets. (Please take a look at chapter 8).

The following functions affect the filter configuration:

- canIdAdd
- canIdDelete
- canIdAddArray
- canIdDeleteArray
- canEnableAllIds
- canAreAllIdsEnabled
- canIDStatus
- canSetIdOffset
- canGetIdOffset

9.1.2. Range-Based-Filtermode

This filtermode filters CAN-message by identifier. The filter differs not between 11 and 29 bit identifiers or standard- and remote-frames. It could be set a range of identifier which should be passed or blocked. If there is a overlapping of passed and blocked ranges than the CAN-messages will be blocked which are matching the overlapping part of the range

The following functions affect the filter configuration:

- canSetFilterJ2534
- canDeleteFilterJ2534

© 2001 – 2017, Sontheim Industrie Elektronik

9.1.2.1. Example

- CAN-messages with identifiers 1000 hex to 10000 hex should be passed by the filter
- CAN-messages with identifiers 9000 hex to 20000 hex should be blocked by the filter

In this case you must call two times the function canSetFilterJ2534 with the following configuration:

- t_mode = j2534Mode_incl;
  ul_rangestart = 0x1000;
  ul_rangestop = 0x10000;

- t_mode = j2534Mode_excl;
  ul_rangestart = 0x9000;
  ul_rangestop = 0x20000;

If there is for example a CAN-message with identifier 9500 hex, it will be blocked by the filter. All other CAN-identifiers which are not matching one of the defined ranges will also be blocked.

9.1.3. 29Bit Filtermode

This filtermode is working in the same way as the Standard Filtermode, except all CAN-message with 29 Bit identifiers will be passed by the filter. 11 Bit CAN-message can be filtered individually by each identifier. The function canSetIdOffset does not affect the filter configuration.

The following functions affect the filter configuration for 11 Bit identifiers:

- canIdAdd
- canIdDelete
- canIdAddArray
- canIdDeleteArray
- canEnableAllIds
- canAreAllIdsEnabled
- canIDStatus

9.1.4. Pattern-Mask-Based Filtermode

This filtermode filters CAN-message by identifier and/or data. It uses a combination of a mask and a pattern. The filter can differ between 11 and 29 bit identifiers and/or standard- and remote-frames.

The CAN message has to comply with the following conditions to be affected by the filter:

(((identifier & ul_mask_id) == (ul_pattern_id & ul_mask_id)) &&
((candata & ul_mask_data) == (ul_pattern_data & ul_mask_data)))

The following functions affect the filter configuration:

- canSetFilterJ2534_2
- canDeleteFilterJ2534_2

© 2001 – 2017, Sontheim Industrie Elektronik

9.1.4.1. Examples

a). We want to filter J1939-messages with a source address (SA) of 2A hex. In this case all 29 Bit CAN-messages which have the last byte of the identifier set to 2A hex should be passed by the filter. The data of the CAN-message itself should not be check by the filter.

In this case you must call the function canSetFilterJ2534_2 with the following configuration:

```
t_mode = j2534Mode_incl;

// Bit 31 and Bit 30 must be set, so filter checks 11/29 Bit and standard/remote frames
// Last byte must be set to 0xFF, so only this byte is checked
ul_mask_id = 0xC00000FF;

// Set to 0, so data is not checked
ul_mask_data = 0x0000000000000000;

// Bit 31 must be set, so only 29 Bit can pass
ul_pattern_id = 0x8000002A;

// Set to 0
ul_pattern_data = 0x0000000000000000;
```

Check in API for 29 Bit Standard CAN-message with identifier CDA002A hex
Data of the message is: 01 02 03 FF FF FF FF FF

Internal value of identifier: 8CDA002A hex (Bit 31 is set because of 29 Bit identifier)

Masking Identifer: `0x8CDA002A & 0xC00000FF = 0x8000002A`

Masking Id-Pattern: `0x8000002A & 0xC00000FF = 0x8000002A`

Compare results: `0x8000002A == 0x8000002A`

Masking Data: `0x010203FFFFFFFFFF & 0x000000000000 = 0x0000000000000000`

Masking Data-Pattern: `0x0000000000000000 & 0x000000000000 = 0x0000000000000000`

Compare results: `0x0000000000000000 == 0x0000000000000000`

=> CAN-message message is passed by filter, because both results are matching

© 2001 – 2017, Sontheim Industrie Elektronik

Check in API for 11 Bit Standard CAN-message with identifier 2A hex
Data of the message is: 01 02 03 FF FF FF FF FF

Internal value of identifier: 0000002A hex (Bit 31 is not set because of 11 Bit identifier)

Masking Identifer: `0x0000002A & 0xC00000FF = 0x0000002A`

Masking Pattern: `0x8000002A & 0xC00000FF = 0x8000002A`

Compare results: `0x0000002A != 0x8000002A`

Masking Data: `0x010203FFFFFFFFFF & 0x000000000000 = 0x0000000000000000`

Masking Data-Pattern: `0x0000000000000000 & 0x000000000000 = 0x0000000000000000`

Compare results: `0x0000000000000000 == 0x0000000000000000`

=> CAN-message message is blocked by filter, because only one result is matching

b). We want to filter a 11 Bit Standard CAN-message with identifier 100 hex
The first byte of the data must be set to 30 hex

In this case you must call the function canSetFilterJ2534_2 with the following configuration:

```
t_mode = j2534Mode_incl;

// Set all bits except Bit 29, because it is not used. If Bit 29 is set it doesen't take
// care, because it is removed in API during filter check.
ul_mask_id = 0xDFFFFFFF;

// Set all bits for the first data-byte
ul_mask_data = 0xFF00000000000000;

// Set bits for identifier 100 hex
ul_pattern_id = 0x00000100;

// Set bits for the first byte to 30 hex
ul_pattern_data = 0x3000000000000000
```

© 2001 – 2017, Sontheim Industrie Elektronik

Check in API for 11 Bit Standard CAN-message with identifier 100 hex
Data of the message is: 30 01 02 03 04 05 06 07

Internal value of identifier: 00000100 hex

Masking Identifer: `0x00000100 & 0xDFFFFFFF = 0x00000100`

Masking Id-Pattern: `0x00000100 & 0xDFFFFFFF = 0x00000100`

Compare results: `0x00000100 == 0x00000100`

Masking Data: `0x3001020304050607 & 0xFF00000000000000 = 0x3000000000000000`

Masking Data-Pattern: `0x3000000000000000 & 0xFF00000000000000 = 0x3000000000000000`

Compare results: `0x3000000000000000 == 0x3000000000000000`

=> CAN-message is passed by filter, because both results are matching

Check in API for 29 Bit Standard CAN-message with identifier 100 hex
Data of the message is: 20 01 02 03 04 05 06 07

Internal value of identifier: 80000100 hex (Bit 31 is set because of 29 Bit identifier)

Masking Identifer: `0x80000100 & 0xDFFFFFFF = 0x80000100`

Masking Id-Pattern: `0x00000100 & 0xDFFFFFFF = 0x00000100`

Compare results: `0x80000100 != 0x00000100`

Masking Data: `0x2001020304050607 & 0xFF00000000000000 = 0x2000000000000000`

Masking Data-Pattern: `0x3000000000000000 & 0xFF00000000000000 = 0x3000000000000000`

Compare results: `0x2000000000000000 != 0x3000000000000000`

=> CAN-message is blocked by filter, because both results are not matching

9.1.5. No Filter

When this mode is set, no filter is active for this logical channel (canOpen). All CAN-messages are passed to the application. There is nothing to configure in this mode.

© 2001 – 2017, Sontheim Industrie Elektronik

10. Events

The driver activates the following events:

- ✓ Data were received
- ✓ The error status has been changed

It is required to create an event with the SDK-function “CreateEvent” before the event-handle is passed to canOpen.

10.1. Receiving event

A receiving event is always activated if messages are received for the equivalent handle.

This shall be clarified by a small example (see next page)

© 2001 – 2017, Sontheim Industrie Elektronik

10.1.1. Example receiving event

```
// im Main-Programm
char ac_ReceiverEvent[32];
char ac_ErrorEvent[32];
HANDLE htEventR1;
HANDLE htEventE1;
HANDLE htEventR2;
HANDLE htEventE2;

strcpy(ac_ReceiverEvent,"R1");
strcpy (ac_ErrorEvent,"E1");
htEventR1 = CreateEvent (NULL, TRUE, FALSE, ac_EventText);
htEventE1 = CreateEvent (NULL, TRUE, FALSE, ac_ErrorText);
l_retval = canOpen (      0,                // net number
                          0,                // l_mode
                          0,                // l_echoon
                          20000,            // txtimeout
                          20000,            // rxtimeout
                          "name 1",         // application name
                          ac_ReceiverEvent, // recv event string
                          ac_ErrorEvent,    // error event string
                          &handle );

// Start of the error handling
hManager = _beginthread (th_canErrorManager1,0x8000,NULL);

strcpy(ac_ReceiverEvent,"R2");
strcpy (ac_ErrorEvent,"E2");
htEventR2 = CreateEvent (NULL, TRUE, FALSE, ac_EventText);
htEventE2 = CreateEvent (NULL, TRUE, FALSE, ac_ErrorText);
l_retval = canOpen (      1,                // net number
                          0,                // l_mode
                          0,                // l_echoon
                          20000,            // txtimeout
                          20000,            // rxtimeout
                          "name 2",         // application name
                          ac_ReceiverEvent, // recv event string
                          ac_ErrorEvent,    // error event string
                          &handle );

// Start of the error handling
hManager = _beginthread (th_canErrorManager2,0x8000,NULL);
  …


// im Receiver-Thread:

WaitForSingleObject (htEventR1,INFINITE);
ResetEvent(htEventR1);
// beginning here further handling of data, e.g. canRead etc.
..
```

© 2001 – 2017, Sontheim Industrie Elektronik

10.2. Error event

It is recommended to start an own thread for the error handling in the application. The following example shall comment the handling of this process:

10.2.1. Example error event

```
HANDLE handle, handle1;
//handle and handle1 are acquired with the function canOpen.

void _cdecl th_canErrorManager1 (void * dummy)
{
   CAN_IF_STATUS t_CAN1Status;
   ui_threadrun = 1;
   while (ui_threadrun)
   {
      WaitForSingleObject (htEventE1,INFINITE);
      ResetEvent(htEventE1);
      canStatus ( handle, &t_CAN1Status );
      if (t_CAN1Status.w_errorflag & RBUFOVERFLOW)
      {
         MessageBox (GetFocus(),"Error 1",
                           "Overflow Receiver Buffer",MB_OK);
      }
      if (t_CAN1Status.w_errorflag & HWRBUFOVERFLOW)
      {
         MessageBox (GetFocus(),"Error 1",
                     "Overflow Hardware Receiver Buffer",MB_OK);
      }
      if (t_CAN1Status.w_errorflag & BUSOFF)
      {
                MessageBox (GetFocus(),"Error 1","Bus-Off",MB_OK);
      }
      if (t_CAN1Status.w_errorflag & ERRORPASSIVE)
      {
         MessageBox (GetFocus(),"Error 1",
                           "error passive flag",MB_OK);
      }
      if (t_CAN1Status.w_errorflag & TXERROR)
      {
         MessageBox (GetFocus(),"Error 1",
                            "transmit error",MB_OK);
      }
   }
}
```

© 2001 – 2017, Sontheim Industrie Elektronik

```
void _cdecl th_canErrorManager2 (void * dummy)
{
   CAN_IF_STATUS t_CAN2Status;
   ui_threadrun = 1;
   while (ui_threadrun)
   {
      WaitForSingleObject (htEventE2,INFINITE);
      ResetEvent(htEventE2);
      canStatus ( handle1, &t_CAN2Status );
      if (t_CAN2Status.w_errorflag & RBUFOVERFLOW)
      {
          MessageBox (GetFocus(),"Error 2",
                      "Overflow Receiver Buffer",MB_OK);
      }
      if (t_CAN2Status.w_errorflag & HWRBUFOVERFLOW)
      {
         MessageBox (GetFocus(),"Error 2",
                     "Overflow Hardware Receiver Buffer",MB_OK);
      }
      if (t_CAN2Status.w_errorflag & BUSOFF)
      {
                MessageBox (GetFocus(),"Error 2","Bus-Off",MB_OK);
      }
      if (t_CAN2Status.w_errorflag & ERRORPASSIVE)
      {
         MessageBox (GetFocus(),"Error 2",
                     "error passive flag",MB_OK);
      }
      if (t_CAN2Status.w_errorflag & TXERROR)
      {
         MessageBox (GetFocus(),"Error 2",
                               "transmit error",MB_OK);
      }
   }
}
```

11. Hardware features

	CANUSB	PowerPCI	CANAS	PC104	PowerPCI V2
Report errorframes as canframe (canRead)	✓	✗	✗	✗	✓
Silent Mode (Acknowledge off)	✓	✗	✗	✗	✓
Low Speed CAN	✗	✗	✗	✓	✗

© 2001 – 2017, Sontheim Industrie Elektronik

12. Hardware specific abnormality

12.1. PowerPCI (V1)

The paramter by_remote in the structure CMSG is valid only at transmit (canSend/canWrite or canConfirmedTransmit) of CAN messages. The PowerPCI is not able to receive remote frames for its own.

12.2. Virtual device

At the use of virtual CAN hardware following parameter have to be considered:

Please select the net number which is handed over with canOpen from following chart:

Hardware	Slot	CAN channel	Net number
Virtual device	-	1 2 -	27 28 29
Virtual device	-	1 2 -	30 31 32
Virtual device	-	1 2 -	33 34 35
Virtual device	-	1 2 -	36 37 38
Virtual device	-	1 2 -	39 40 41

The third CAN channel is not supported yet at the time of this documentation creation.

© 2001 – 2017, Sontheim Industrie Elektronik

12.3. Timeout

The handing over of the values l_txtimeout and l_rxtimeout are limited on 65535. The handing over value 0 effects that the timeout is set internal to the value 4 294 967 295 (= hex 0xFFFFFFFF).

At the function canRead this value is set on INFINITE.

13. Predefined baudrate settings

13.1. CANUSB / CANUSB 2X4

Define	Baudrate (KBit)	BTR0	BTR1	SamPoint (%)	Time Seg 1	Time Seg 2	Prescaler	SJW	Sampling
0	1000	0x44	0x16	80	7	2	5	2	1
1	800	Not supported							
2	500	0xC4	0x3E	80	15	4	5	4	1
3	250	0xC9	0x3E	80	15	4	10	4	1
4	125	0xD3	0x3E	80	15	4	20	4	1
5	100	0xD8	0x3E	80	15	4	25	4	1
6	50	0xF1	0x3E	80	15	4	50	4	1
7	20	Not supported							

13.2. PowerPCI V1 (40 MHz)

Define	Baudrate (KBit)	BTR0	BTR1	SamPoint (%)	Time Seg 1	Time Seg 2	Prescaler	SJW	Sampling
0	1000	0xC0	0x7A	60	11	8	1	4	1
1	800	0xC0	0x7F	68	16	8	1	4	1
2	500	0xC1	0x7A	60	11	8	2	4	1
3	250	0xC3	0x7A	60	11	8	4	4	1
4	125	0xC7	0x7A	60	11	8	8	4	1
5	100	0xC9	0x7A	60	11	8	10	4	1
6	50	0xD3	0x7A	60	11	8	20	4	1
7	20	0xF1	0x7A	60	11	8	50	4	1

13.3. PowerPCI V1 (33 MHz)

Define	Baudrate (KBit)	BTR0	BTR1	SamPoint (%)	Time Seg 1	Time Seg 2	Prescaler	SJW	Sampling
0	1000	0xC0	0x58	62,5	9	6	1	4	1
1	800	0xC0	0x7A	60,0	11	8	1	4	1
2	500	0xC1	0x58	62,5	9	6	2	4	1
3	250	0xC3	0x58	62,5	9	6	4	4	1
4	125	0xC7	0x58	62,5	9	6	8	4	1
5	100	0xC7	0x7A	60,0	11	8	8	4	1
6	50	0xD3	0x58	62,5	9	6	20	4	1
7	20	0xE7	0x7A	60,0	11	8	40	4	1

© 2001 – 2017, Sontheim Industrie Elektronik

13.4. PowerPCI V2

Define	Baudrate (KBit)	PSEG1	PSEG2	SamPoint (%)	Time Seg 1	Time Seg 2	PROP SEG	Prescaler	SJW	Sampling
0	1000	8	3	85,0	16	3	8	5	3	1
1	800	8	8	68,0	18	8	8	5	3	1
2	500	8	3	85,0	16	3	8	10	3	1
3	250	8	2	87,5	13	2	5	25	3	1
4	125	8	2	85,0	16	3	5	50	3	1
5	100	8	3	85,0	16	3	8	50	3	1
6	50	8	3	85,0	16	3	8	100	3	1
7	20	8	3	85,0	16	3	8	250	3	1

© 2001 – 2017, Sontheim Industrie Elektronik

14. Baud rate calculation

This chapter deals with the determination of both baud rate parameters Btr0 and Btr1. Via setting of the 31st Bit at the long variable l_baud in the function canSetBaudrate, the baud rate can be given directly via the parameter Btr0 and Btr1. Thereby baud rates or different samplepoints are also able to be used (presumed that the CAN controller supports that), which are not set directly with the predefined values via the function canSetBaudrate.

14.1. CANUSB

Popular baud rate adjustments:

The values which are accented bold are already predefined in the API and are able to be used directly in the function canSetBaudrate.

Baud rate / Samplepoint	50 KBd		100 KBd		125 KBd		250 KBd		500 KBd		1000 KBd	
	BTR0	BTR1	BTR0	BTR1	BTR0	BTR1	BTR0	BTR1	BTR0	BTR1	BTR0	BTR1
80 %			0x31	0x16	0x27	0x16	0x13	0x16	0x09	0x16	0x04	0x16
75 %					**0x31**	**0x14**	**0x18**	**0x14**				
70 %			0x31	0x25	0x27	0x25	0x13	0x25	0x09	0x25	0x04	0x25
68 %	**0x27**	**0x7F**	**0x13**	**0x7F**	0x0F	0x7F	0x07	0x7F	**0x03**	**0x7F**	**0x01**	**0x7F**
62 %					0x31	0x23	0x18	0x23				

The Bus-Timing-Register 0 is constructed as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SJW 1	SJW 0	BRP 5	BRP 4	BRP 3	BRP 2	BRP 1	BRP 0

SJW = Synchronization Jump Width

SJW 1	SJW 0	Synchonization Jump Width
0	0	1 Tq Cycle
0	1	2 Tq Cycle
1	0	3 Tq Cycle
1	1	4 Tq Cycle

With both SJW entries it is defined how many tq (tq = 1 / fTQ) the period for a transmitting bit may be corrected. On this the adjustment of the BTR1 is also to attend!

BRP = Baud Rate Prescaler

$$fTq = \frac{fCANCLK}{(Prescaler\ Value)}$$

Prescaler Value = [(BRP5 * 32) + (BRP4 * 16) + (BRP3 * 8) + (BRP2 * 4) + (BRP1 * 2) + (BRP0 * 1)] + 1
fCANCLK = 50 MHz
Example for the calculation of fTq:

Register content BTR0 = 0x01

$$fTq = \frac{50\ MHz}{2} = 25\ MHz \quad Tq = 40ns$$

© 2001 – 2017, Sontheim Industrie Elektronik

The Bus Timing Register 1 is constructed as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SAMP	TSEG 2.2	TSEG 2.1	TSEG 2.0	TSEG 1.3	TSEG 1.2	TSEG 1.1	TSEG 1.0

SAMP Function
0 One bit is scanned three times at the reception
1 One bit is scanned once at the reception

Values for the Time-Segment 2:

TSEG 2.2	TSEG 2.1	TSEG 2.0	Time Segment 2
0	0	0	1 Tq clock cycles (1)
0	0	1	2 Tq clock cycles
0	1	0	3 Tq clock cycles
.	.	.	.
1	1	0	7 Tq clock cycles
1	1	1	8 Tq clock cycles

(1) This adjustment is not valid, please look for this at chart Valid adjustments for BTR1

Values for the Time-Segment 1

TSEG 1.3	TSEG 1.2	TSEG 1.1	TSEG 1.0	Time Segment 1
0	0	0	0	1 Tq clock cycle (1)
0	0	0	1	2 Tq clock cycle (1)
0	0	1	0	3 Tq clock cycle (1)
0	0	1	1	4 Tq clock cycle
.	.	.	.	.
1	1	1	0	15 Tq clock cycle
1	1	1	1	16 Tq clock cycle

(1) This adjustment is not valid, please look for this at chart Valid adjustments for BTR1

Valid adjustments for BTR1 after Bosch CAN specification 2.0A/B from September 1991

Time Segment	TSEG 1	Time Segement	TSEG 2	Synchronisation Jump Width	SJW
5..10	4..9	2	1	1..2	0..1
4..11	3..10	3	2	1..3	0..2
5..12	4..11	4	3	1..4	0..3
6..13	5..12	5	4	1..4	0..3
7..14	6..13	6	5	1..4	0..3
8..15	7..14	7	6	1..4	0..3
9..16	8..15	8	7	1..4	0..3

Calculation of the bit rate

$$BR = \frac{fTq}{\text{Number Tq}}$$

The number of Tq calculates as follows
Number of Tq = SyncSeg + TSEG1 + 1 + TSEG2 + 1
SyncSeg = Synchronization segment is always one Tq long
Example:
BTR0 = 0x01, BTR1 = 0x7F
Number of Tq = 1 + (15 + 1) + (7 + 1) = 25
BR = 25MHz / 25 = 1 MHz = 1000 kBit / sec

© 2001 – 2017, Sontheim Industrie Elektronik

14.2. PowerPCI / PowerPCI V2

14.2.1. Baudrate calculation

To be fully compatible with PowerPCI V2 to PowerPCI V1 the parameters for BTR0 and BTR1 are calculated for both cards in the same manner. The MT-API is recalculating the parameters for the PowerPCI V2.

Popular baud rate adjustments:

Baudrate / Samplepoint	20 KBd		50 KBd		100 KBd		125 KBd		250 KBd		500 KBd		800 KBd		1 MBd	
	BTR0	BTR1	BTR0	BTR1	BTR0	BTR1	BTR0	BTR1	BTR1	BTR0	BTR1	BTR0	BTR1	BTR0	BTR0	BTR1
80 %	0x31	0x3E	0x13	0x3E	0x09	0x3E	0x07	0x3E	0x03	0x3E	0x01	0x3E			0x00	0x3E
75 %	0x31	0x4D	0x13	0x4D	**0x09**	**0x4D**	**0x07**	**0x4D**	**0x03**	**0x4D**	**0x01**	0x4D			0x00	0x4D
70 %	0x31	0x5C	0x13	0x5C	0x09	0x5C	0x07	0x5C	0x03	0x5C	0x01	0x5C			0x00	0x5C
68 %	**0x27**	**0x7F**	**0x0F**	**0x7F**	0x07	0x7F							**0x00**	**0x7F**		
62 %	0x31	0x7A	0x13	0x7A	0x09	0x7A	0x07	0x7A	0x03	0x7A	0x01	0x7A			0x00	0x7A

The Bus-Timing-Register 0 (BTR0) is constructed as follows:

These are bits 0-7 in parameter l_baud for function canSetBaudrate.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SJW 1	SJW 0	BRP 5	BRP 4	BRP 3	BRP 2	BRP 1	BRP 0

SJW = Synchronization Jump Width

SJW 1	SJW 0	Synchonization Jump Width
0	0	1 Tq Cycle
0	1	2 Tq Cycle
1	0	3 Tq Cycle
1	1	4 Tq Cycle

With both SJW entries it is defined how many tq (tq = 1 / fTQ) the period for a transmitting bit may be corrected On this the adjustment of the BTR1 is also to attend!

BRP = Baud Rate Prescaler

$$fTq = \frac{fCANCLK}{(Prescaler\ Value)}$$

Prescaler Value = [(BRP5 * 32) + (BRP4 * 16) + (BRP3 * 8) + (BRP2 * 4) + (BRP1 * 2) + (BRP0 * 1)] + 1
fCANCLK = 40 MHz
Example for the calculation of fTq:

Register content BTR0 = 0x01

$$fTq = \frac{40\ MHz}{2} = 25\ MHz \quad Tq = 50ns$$

© 2001 – 2017, Sontheim Industrie Elektronik

The Bus Timing Register 1 (BTR1) is constructed as follows:

These are bits 8-15 in parameter l_baud for function canSetBaudrate.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SAMP	TSEG 2.2	TSEG 2.1	TSEG 2.0	TSEG 1.3	TSEG 1.2	TSEG 1.1	TSEG 1.0

SAMP Function
0 One bit is scanned three times at the reception
1 One bit is scanned once at the reception

Values for the Time-Segment 2:

TSEG 2.2	TSEG 2.1	TSEG 2.0	Time Segment 2
0	0	0	1 Tq clock cycles (1)
0	0	1	2 Tq clock cycles
0	1	0	3 Tq clock cycles
.	.	.	.
1	1	0	7 Tq clock cycles
1	1	1	8 Tq clock cycles

(1) This adjustment is not valid, please look for this at chart Valid adjustments for BTR1

Values for the Time-Segment 1

TSEG 1.3	TSEG 1.2	TSEG 1.1	TSEG 1.0	Time Segment 1
0	0	0	0	1 Tq clock cycle (1)
0	0	0	1	2 Tq clock cycle (1)
0	0	1	0	3 Tq clock cycle (1)
0	0	1	1	4 Tq clock cycle
.	.	.	.	.
1	1	1	0	15 Tq clock cycle
1	1	1	1	16 Tq clock cycle

(1) This adjustment is not valid, please look for this at chart Valid adjustments for BTR1

Valid adjustments for BTR1 after Bosch CAN specification 2.0A/B from September 1991

Time Segment	TSEG 1	Time Segement	TSEG 2	Synchronisation Jump Width	SJW
5..10	4..9	2	1	1..2	0..1
4..11	3..10	3	2	1..3	0..2
5..12	4..11	4	3	1..4	0..3
6..13	5..12	5	4	1..4	0..3
7..14	6..13	6	5	1..4	0..3
8..15	7..14	7	6	1..4	0..3
9..16	8..15	8	7	1..4	0..3

© 2001 – 2017, Sontheim Industrie Elektronik

Calculation of the bit rate

$$BR = \frac{fTq}{Anzahl\ Tq}$$

The number of Tq calculates as follows
Number of Tq = SyncSeg + TSEG1 + 1 + TSEG2 + 1
SyncSeg = Synchronization Segment is always one Tq long
Example:
BTR0 = 0x00, BTR1 = 0x4D
Number of Tq = 1 + (13 + 1) + (4 + 1) = 20
BR = 20MHz / 20 = 1 MHz = 1000 kBit / sec

14.2.1. Calculation for error-frame detection

To be fully compatible with PowerPCI V2 to PowerPCI V1 the parameters for BTR0 and BTR1 are calculated for both cards in the same manner. The MT-API is recalculating the parameters for the PowerPCI V2.

The PowerPCI is able to recognize error-frames. To do this a parameter must be passed if used-defined baudrate is used. There is a formula to calculate the right value for this.

ParaErrorFrame = (5000000 * 53) / Baudrate / 10

List of used values if standard-baudrate is used:

```
(5000000 * 53   / 1000000 / 10  = 0x1A        // 1000 Kbit
(5000000 * 53)  / 800000   / 10  = 0x21        //  800  KBit
(5000000 * 53)  / 500000   / 10  = 0x35        //  500  KBit
(5000000 * 53)  / 250000   / 10  = 0x6A        //  250  KBit
(5000000 * 53)  / 125000   / 10  = 0xD4        //  125  Kbit
(5000000 * 53)  / 100000   / 10  = 0x0109      //  100  KBit
(5000000 * 53)  / 50000    / 10  = 0x0212      //  50   KBit
(5000000 * 53)  / 20000    / 10 =  0x052D      //  20   KBit
```

The value 53 in formula is standing for the bit-time. In this case it is a bit-time of 5,3. This means that if the CAN-level is longer than 5,3 bit-times dominant the FPGA on the PowerPCI is detecting an error-frame.

It is also possible to raise this value.

The result should be set in bits 16-28 in parameter l_baud of function canSetBaudrate.

© 2001 – 2017, Sontheim Industrie Elektronik

14.3. Functionality of the CAN hardware

It is possible to set the API without having installed a physical CAN card in the system. On this you have to give the equivalent net number of the virtual hardware at the function canOpen, as described in chapter 9.3.Via the parameter l_echoon it is additionally possible to give whether CAN channel 1 is connected with CAN channel 2. The parameter l_echoon is analyzed as follows:

Bit 0 of the parameter:

Value 0 : transmitted data are not received at the same CAN channel
Value 1 : transmitted data are received at the same channel at the same time.

Bit 1 of the parameter:

Value 0 : CAN channel 1 is not connected with CAN channel 2.
Value 1 : CAN channel 1 is connected with CAN channel 2.

The parameter must be set as follows:

Value 0 : No reception of transmitted data at the same time & CAN channel 1 is not connected with CAN channel 2.
Value 1 : Reception of data at the same time & CAN channel 1 is not connected with CAN channel 2.
Value 2 : No reception of transmitted data & CAN channel 1 is connected with CAN channel 2.
Value 3 : Reception of transmitted data at the same time & CAN channel 1 is connected with CAN channel 2.

© 2001 – 2017, Sontheim Industrie Elektronik