Tiny-CAN API Referenz-Handbuch

MHS Elektronik GmbH & Co. KG
Fuchsöd 4 ~ D-94149 Kößlarn
Tel: +49 (0) 8536/919 740 ~ Fax: +49 (0) 8536/919 738
Email: info@mhs-elektronik.de ~ Internet: www.mhs-elektronik.de

Version: 3.3 vom 26.11.2015

Inhaltsverzeichnis

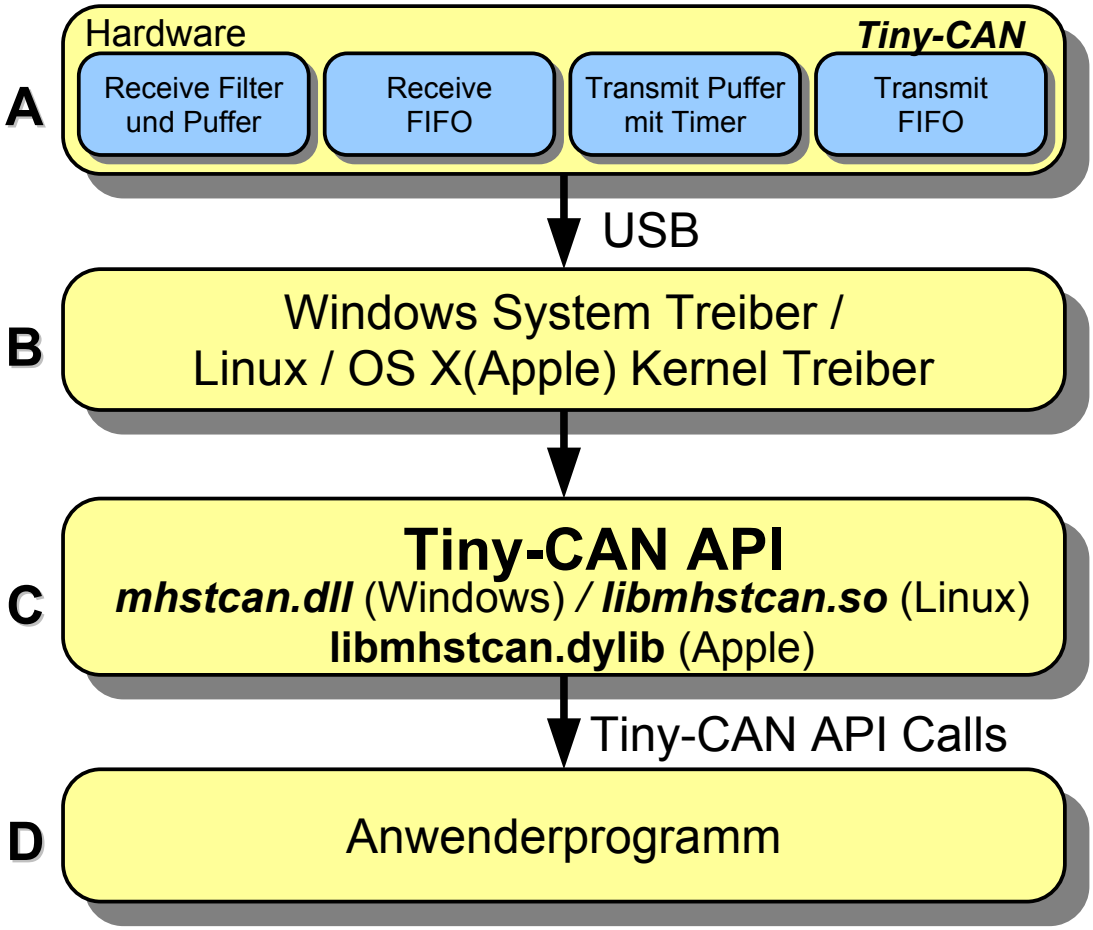
1. Installation / Hardware....4
2. Von der Hardware bis zum Anwenderprogramm....5
3. Dateien....8
4. Die API....9
4.1 Übersicht....9
4.1 Der Parameter „index“....12
4.2 CAN-Filter....14
4.3 Header Files....14
3.5 API Funktionsaufrufe....22
3.6 EX-API Funktionsaufrufe....23
LoadDriver....24
UnloadDriver....25
CanInitDriver....26
CanDownDriver....28
CanSetOptions....29
CanDeviceOpen....30
CanDeviceClose....32
CanSetMode....33
CanTransmit....35
CanTransmitClear....36
CanTransmitGetCount....37
CanTransmitSet....38
CanReceive....40
CanReceiveClear....41
CanReceiveGetCount....42
CanSetSpeed....43
CanSetSpeedUser....44
CanSetFilter....46
CanDrvInfo....48
CanDrvHwInfo....49
CanGetDeviceStatus....50
CanSetPnPEventCallback....52
CanSetStatusEventCallback....53
CanSetRxEventCallback....54
CanSetEvents....55
CanExInitDriver....56
CanExCreateDevice....60
CanExDestroyDevice....61
CanExCreateFifo....62
CanExBindFifo....63
CanExCreateEvent....64
CanExSetObjEvent....65
CanExSetEvent....66
CanExResetEvent....67
CanExWaitForEvent....68
CanExGetDeviceCount....70
CanExGetDeviceList....71
CanExGetDeviceInfo....74

CanExDataFree....78
CanExSetOptions....79
CanExSetAsByte....80
CanExSetAsWord....81
CanExSetAsLong....82
CanExSetAsUByte....83
CanExSetAsUWord....84
CanExSetAsULong....85
CanExSetAsString....86
CanExGetAsByte....87
CanExGetAsWord....88
CanExGetAsLong....89
CanExGetAsUByte....90
CanExGetAsUWord....91
CanExGetAsULong....92
CanExGetAsString....93
5. Fehler-Codes (Error-Codes)....94
6. Parameter....95
7. Config-File....97
8. Log File....98
8. Beispiele....100
8.1 Verwendung der Tiny-CAN API im Polling-Modus....100
8.2 Quellcode des Demoprogramms „ex_sample1“:....101
8.3 Verwendung der Tiny-CAN API im Event-Modus 104
8.4 Quellcode des Demoprogramms „ex_sample6“:....104
8.5 Verwendung von Filtern und Sende-Puffern....110
4.6 Quellcode des Demoprogramms „Sample3“:....110
8.6 Quellcode des Demoprogramms „Sample4“: Verwendung von mehreren Filtern..110

1. Installation / Hardware

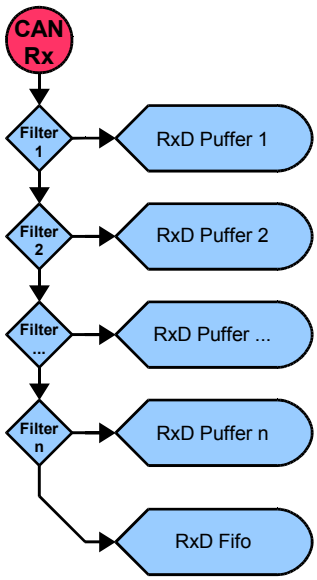
Die Treiber-Installation, eine Beschreibung der Hardware, die Installation von „Tiny-CAN-Monitor und Third Party Tools ist im Dokument „TinyCan.pdf“ beschrieben.

2. Von der Hardware bis zum Anwenderprogramm



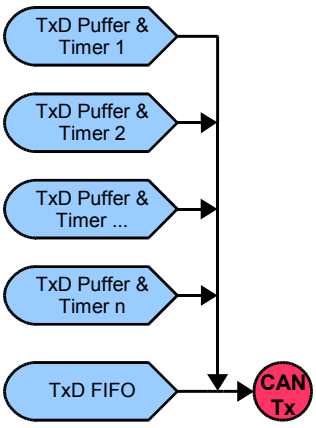
A) Hardware

Die Grafik verdeutlicht die in der Tiny-CAN-Hardware bzw. -Firmware realisierten Funktionen. Eine Besonderheit der Tiny-CAN Module sind die „Receive Filter & Puffer“ und „Transmit Puffer mit Timer“



Funktion der „Receive Filter“ (Hardware Filter):

Je nach Hardware ist eine bestimmte Anzahl von Filtern vorhanden, mit den Filtern ist es möglich, ein hohes Datenaufkommen auf dem USB-Bus zu reduzieren. Eine gefilterte Nachricht wird in dem dazugehörigen Puffer gespeichert, der Puffer wird immer wieder überschrieben. Alles was nach dem Durchlauf der Filter noch übrig bleibt, landet im „RxD FIFO“ des Moduls.



Funktion der „Transmit Puffer mit Timer“:

Je nach Hardware ist eine bestimmte Anzahl von TxD-Puffern mit Intervalltimern vorhanden, diese ermöglichen das Versenden zyklischer Nachrichten in Echtzeit. Der normale Versand von CAN Messages erfolgt über das „TxD FIFO“ des Moduls.

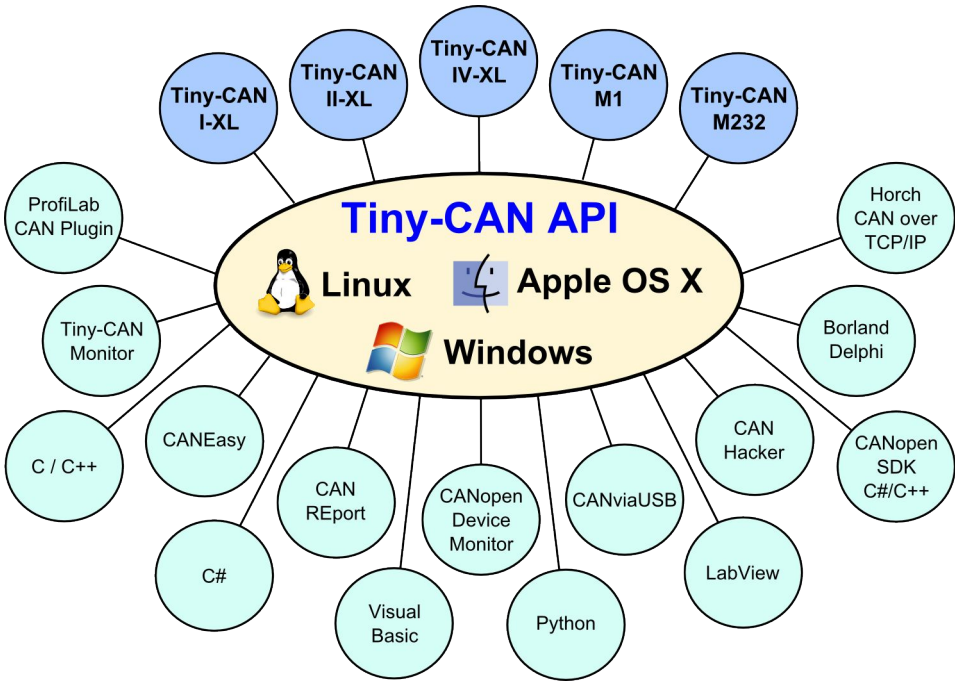
	RxD FIFO [Größe in Messages]	**TxD FIFO** [Größe in Messages]	**Recieve Filter** [Anzahl Filter]	**TxD Puffer mit Intervalltimer** [Anzahl Puffer]
Tiny-CAN I	110	36	4	4
Tiny-CAN I-XL	350	72		
Tiny-CAN II	110	36		
Tiny-CAN II-XL	384	72		
Tiny-CAN III	512		8	16
Tiny-CAN III-XL				
Tiny-CAN IV-XL	900		4	8
Tiny-CAN M1	384		4	4
Tiny-CAN M232			12	

B) Windows System-Treiber / Linux / OS X(Apple) Kernel Treiber

Die Tiny-CAN Module verwenden als USB-Chip einen USB zu RS232 Konverter. Als Treiber wird der Standard-Treiber des Chip-Herstellers verwendet.
Unter Linux steht auch ein „SocketCAN“ Treiber zur Verfügung.

C) Tiny-CAN API

Die Tiny-CAN API ist eine plattformübergreifende Treiberschnittstelle, die es dem Benutzer erlaubt, die Hardware mittels DLL (unter Windows) oder Shared Library (unter Linux und OS X) anzusprechen. Die Funktionsaufrufe für die Betriebssysteme Windows, Linux und OS X sind identisch.



D) Anwenderprogramm

- Dem Anwender stehen zahlreiche Applikationen auch von Drittanbietern zur Verfügung.
- Eigene Applikationen können in C/C++, C#, VisualBasic 6, Python, LabView oder Delphi entwickelt werden.

3. Dateien

API Treiber

.../tiny_can/can_api/...

mhstcan.dll	[2]Tiny-CAN API Treiber für Windows 32Bit
x64	
└ mhstcan.dll	[2]Tiny-CAN API Treiber für Windows 64Bit
libmhstcan.so	[1]Tiny-CAN API Treiber für Linux 32/64Bit
libmhstcan.dylib	[3]Tiny-CAN API Treiber für Apple OS X 64bit

Implementierung der Tiny-CAN API in C/C++, C#, VB6, Delphi und Python

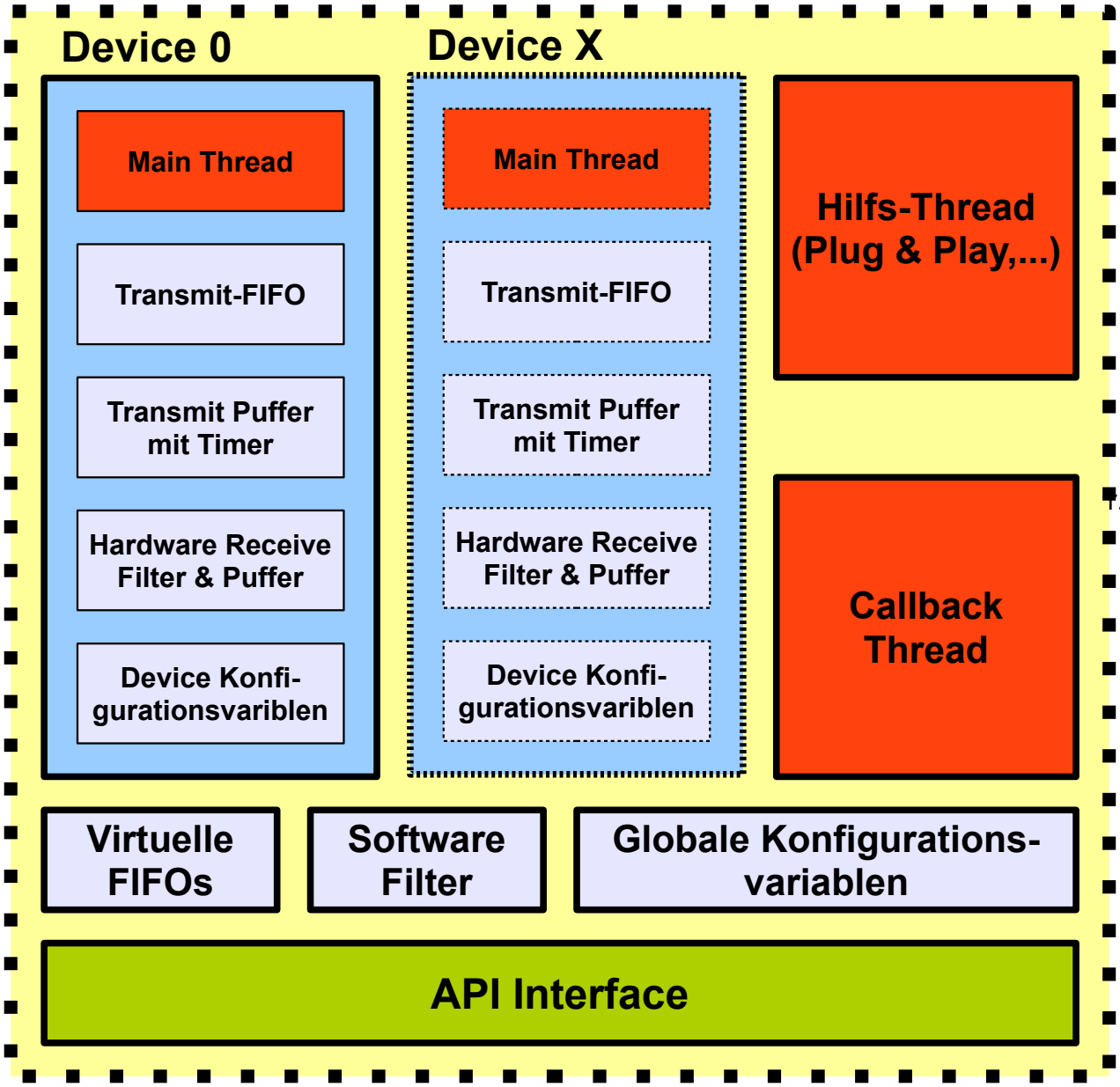
.../tiny_can/dev/...

TinyCanAPI.pdf	Dieses Dokument
lib	Files zum dynamischen Laden eines Tiny-CAN API Treibers
├ can_drv_config.h	Treiber Konfigurationsdatei
├ can_types.h	Allgemeine Definitionen und Datentypen für CAN-Messages, Filter, ...
├ can_drv.h	Header-File der API, bindet auch can_types.h mit ein
├ can_drv_ex.h	Header-File der API, bindet auch can_types.h mit ein
├ mhs_can_drv.c	Modul zum dynamischen Laden der dll oder shared-lib, löst die Dateien „can_drv_win.c“ und „can_drv_linux.c“ ab
├ can_drv_win.c	[2]Modul zum dynamischen Laden der dll
└ can_drv_linux.c	[1]Modul zum dynamischen Laden der shared-lib
static	[2]Files zum statischen linken der mhstcan.dll, inklusive Beispiel
C	Programmbeispiele, die Beispiele „ex_..“ benutzen den neueren „Ex-API“ (Multi-Devices Support) Modus.
├ sample1	Treiber laden, CAN-Bus initialisieren, eine CAN Nachricht versenden, Nachrichten im „Polling“ Modus empfangen
├ sample2	Verwendung der Tiny-CAN API im Event-Modus
├ sample3	Verwendung von Filtern und Sende-Puffern
├ sample4	Verwendung von mehreren Filtern
├ sample5	Versand vieler CAN Nachrichten mit maximaler Geschwindigkeit
├ sample6	Abfrage und Auswertung der „Hardware Info Variablen“, setzen einer Benutzerdefinierten CAN Übertragungsgeschwindigkeit
├ ex_sample1	Treiber laden und im „Ex“ Modus initialisieren, CAN-Bus initialisieren, eine CAN Nachricht versenden, Nachrichten im „Polling“ Modus empfangen
├ ex_sample2	Demonstriert das Öffnen von 2 Devices und dem Empfang von CAN Daten im Polling Modus
├ ex_sample3	Demonstriert die Verwendung eines virtuellen Empfangs-Puffer, 2 Devices schreiben in den Puffer
├ ex_sample4	Beispiel für "CanExGetDeviceInfo" und "CanSetSpeedUser"
├ ex_sample5	Beispiel für „CanExGetDeviceCount“ und „CanExGetDeviceList“
├ ex_sample6	Zeigt das handling mehrere Devices mit Callbackfunktionen (Event-Modus)
└ ex_sample7	Beispiel für „CanExWaitForEvent“
delphi	[2]Komponenten und Beispielprogramme für Delphi
├ Delphi-6	Projekt Dateien für Delphi 6
├ Delphi-EX	Projekt Dateien für Delphi-EX
├ comps	Delphi-Komponenten
├ tiny-can	Ein kleiner CAN-Monitor
└ TinyCANTes	Beispielprogramm
c-sharp	[2]Implementierung der Tiny-CAN API in C-Sharp
├ lib	Interface zur Tiny-CAN API für C-Sharp
├ event	Event basiertes C# Beispiel
└ sample	Beispielprogramm
vb6	[2]Implementierung der Tiny-CAN API in Visual-Basic Version 6.0
vb2013	[2]Implementierung der Tiny-CAN API in Visual-Basic 2013
labview	[2]Implementierung der Tiny-CAN API für LabView von National Instruments
Python	[1]Implementierung der Tiny-CAN in Python, inklusive Beispielen
└ CanMon	GUI basiertes Beispielprogramm in Python unter Verwendung von GTK+
can_monitor	Quellcodes des Tiny-CAN Monitors in GTK+

1 Nur in der Linux Version; 2 Nur in der Windows Version; 3 Nur in der Apple Version

4. Die API

4.1 Übersicht



Treiber Laden
Über die Funktion „*LoadDriver*“ wird ein API Treiber geladen und mit „*CanExInitDriver*“ Initialisiert. Der Hilfs- und wenn entsprechend konfiguriert der Callback Thread werden gestartet.
Ab diesen Zeitpunkt verwaltet der Treiber eine Liste verbundener Tiny-CAN Devices, die mit „*CanExGetDeviceList*“ abgefragt werden kann, die Liste muss mit „*CanExDataFree*“ freigegeben werden. „*CanExGetDeviceCount*“ liefert nur die Anzahl der verbundenen Devices.

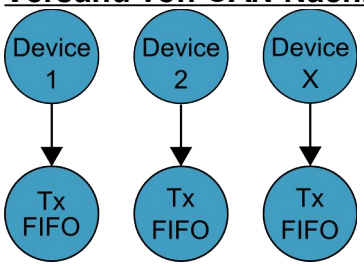
Initialisierung:

Zum öffnen eines Devices muss dies zuerst mit der Funktion „*CanExCreateDevice*“ angelegt und konfiguriert werden, bevor es mit „*CanDeviceOpen*“ geöffnet werden kann. Der Device Thread wird erst beim öffnen des Devices gestartet. Die Funktion „*CanExGetDeviceInfo*“ liefert detaillierte Informationen über das geöffnete Device.

Mit den Funktionen „*CanExSetOptions*“, „*CanExSetAsByte*“, „*CanExSetAsWord*“, „*CanExSetAsLong*“, „*CanExSetAsUByte*“, „*CanExSetAsUWord*“, „*CanExSetAsULong*“ und „CanExSetAsString“ können globale bzw. Device Konfigurationsvarialben gesetzt werden, „*CanExGetAsByte*“, „*CanExGetAsWord*“, „*CanExGetAsLong*“, „*CanExGetAsUByte*“, „*CanExGetAsUWord*“, „*CanExGetAsULong*“ und „*CanExGetAsString*“ dienen deren Abfrage. Die CAN-Übertragungsgeschwindikeit kann auch mit „*CanSetSpeed*“ und „*CanSetSpeedUser*“ gesetzt werden. Filter werden über die Funktion „*CanSetFilter*“ gesetzt.

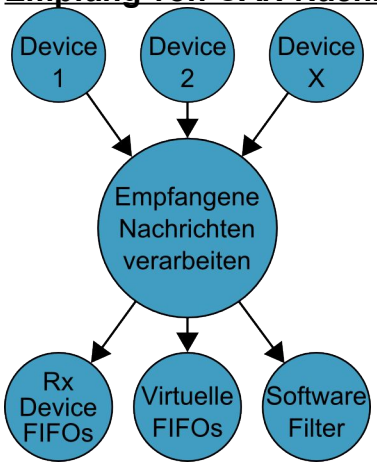
Nach erfolgreicher Konfiguration kann der Bus mit „*CanSetMode*“ gestartet werden.

Versand von CAN Nachrichten



Jedem CAN Device ist ein eigenes Transmitt FIFO zugeordnet, „*CanTransmit*“ schreibt Nachrichten in das angegebene Transmit-FIFO/Puffer, „*CanTransmitClear*“ löscht das FIFO und „*CanTransmitGetCount*“ fragt die Anzahl der Nachrichten im FIFO ab. Die Funktion „*CanTransmitSet*“ konfiguriert die Intervall-Puffer.

Empfang von CAN Nachrichten



Die Nachrichten aller verbundenen Devices werden gemeinsam Verarbeitet und in die entsprechenden FIFOs und Puffer (Filter) geschrieben.
Die Funktion „*CanExCreateFifo*“ erzeugt ein Empfangs-FIFO, „*CanExBindFifo*“ verknüpft einzelne Devices mit dem FIFO falls dies nicht beim erzeugen des FIFOs gemacht wurde. FIFOs werden statisch angelegt und können nicht mehr gelöscht werden. „*CanReceive*“ liest Nachrichten aus einen beliebigen FIFO/Puffer, „*CanReceiveClear*“ löscht ein FIFO/Puffer und „*CanReceiveGetCount*“ fragt die Anzahl der Nachrichten ab.

Ereignisse

Mit den Funktionen „*CanSetPnPEventCallback*“, „*CanSetStatusEventCallback*“, „*CanSetRxEventCallback*“ können Ereignis Callback Funktionen gesetzt werden, über „*CanSetEvents*“ lassen sich Ereignisse enablen/disablen. Voraussetzung für die Nutzung der Event-Funktion ist das der Callback-Thread erzeugt wurde.

Eine andere Möglichkeit auf Ereignisse zu reagieren ist einen Thread schlafen zu lassen bis ein Ereignis eintritt. Mit der Funktion „*CanExWaitForEvent*“ kann auf Ereignisse (Events) bzw. Timeout gewartet werden. Für jeden Thread der „*CanExWaitForEvent*“ benutzt muss ein eigenes Event Objekt mit „*CanExCreateEvent*“ erzeugt werden. Ein Event kann von einen Objekt innerhalb des Treibers das mit der Funktion

„*CanExSetObjEvent*“ verknüpft wurde erzeugt werden. Ein Event kann auch von außen mit der Funktion „*CanExSetEvent*“ erzeugt und mit der Funktion „*CanExResetEvent*“ gecanceld werden.

Beenden

Mit „*UnloadDriver*“ wird der API Treiber Entladen, vor dem Entladen wird automatisch die Funktion „*CanDownDriver*“ aufgerufen, wodurch alle allokierten Ressourcen freigegeben werden. Zuvor müssen jedoch alle geöffneten Devices mit „*CanDeviceClose*“ geschlossen werden, es dürfen keine Callback-Funktionen mehr ausgelöst werden, bzw. alle Callback-Funktionen müssen zurückgekehrt sein.

Kompatibilität

Um eine Kompatibilität zu früheren Versionen der API zu gewährleisten kann die Initialisierung mit „*CanInitDriver*“ erfolgen, die API ist dann im Kompatibilitätsmodus und kann nur ein CAN-Device ansprechen. Die „Ex“-Calls stehen nicht zur Verfügung. Damit können ältere Applikation die neue API ohne Programmänderungen benutzen.

Verwenden Sie in neuen Projekten die API ausschließlich im „Ex“-Modus, siehe „CanExInitDriver“

4.1 Der Parameter „index“

Die meisten Funktionsaufrufe und Callbackfunktionen verwenden den Parameter „index“, deshalb wird der Parameter „index“ hier vorab erläutert

Die einzelnen Bits des „Index“ Parameters:

Bit																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Reserviert					Soft	RxD /TxD	CAN Device				CAN Kanal				Sub-Index															

Virtuelle Empfangs FIFO/Puffer:

Bit																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Sub-Index															

Bei Virtuellen Empfangs-Puffern werden die oberen 16 Bit auf 0x8000 gesetzt, entspricht dem Wertebereich 0x800000000 – 0x80000FFFF

Der Parameter „Source“ in der CAN-Message Struktur:

Bit							
7	6	5	4	3	2	1	0
CAN Device				CAN Kanal			

Die Bits 0 – 7 entsprechen den Bits 16 – 23 des „index“ Parameters.

Der „index“-Wert 0xFFFFFFFF ist Reserviert und wird zum Ansprechen der Globalen Konfigurationsvariablen benutzt.

Reserviert	Reserviert für zukünftige Anwendungen, alle Bits müssen auf 0 gesetzt werden
Soft	0 = Hardware-Filter 1 = Software-Filter
RxD/TxD	0 = Empfangspuffer/FIFO 1 = Sendepuffer/FIFO Das Bit wird in der Regel von den Funktionsaufrufen automatisch gesetzt
CAN Device	Spezifiziert die Hardware, wenn mehr Module am PC angeschlossen sind.
CAN Kanal	Spezifiziert den CAN-Kanal auf einem CAN-Device.
Sub-Index	Index des FIFOs/Puffers im CAN Device/CAN Kanal

Ansprechen der FIFOs und Puffer eines CAN-Devices/CAN-Kanals

Soft	***RxD/TxD***	***Sub-Index***	
0	0	0	RxD FIFO (Empfangsfifo)
0	0	1 - n	Receive Filter, die Anzahl n ist Hardwareabhängig
1	0	1 - 65535	Software Receive Filter
0	1	0	TxD FIFO (Sendefifo)
0	1	1 - n	TxD Puffer mit Intervalltimer, die Anzahl n ist Hardwareabhängig

Das TxD Bit wird automatisch von den Funktionen gesetzt.

4.2 CAN-Filter

Es gibt 3 Möglichkeiten, IDs aus dem CAN-Datenstrom heraus zu filtern, die gefilterte Nachricht wird in dem dazugehörigen Puffer des Filters abgelegt, dieser wird mit jeder neuen Nachricht überschrieben.

Filter-Type	***TMsgFilter***			
	FilIdMode	***Code***	***Maske***	
Maske & Code	0	Code	Maske	Die CAN-IDs mittels Maske Filtern, siehe unter
Start & Stop	1	Start	Stop	Den Bereich zwischen Start und Stop Filtern, nur bei Software Filtern möglich
Single Id	2	Id	-	Eine einzelne ID filtern

Über „FilEFF“ wird festgelegt, ob Extended oder Standard Frames gefiltert werden, „FilEFF = 1“ Extended Frames.

Wird „FilIdMode“ auf 1 gesetzt werden die gefilterten Messages nicht aus dem Datenstrom entfernt, nur bei Software-Filtern möglich.

Ein Filter muss über das Flag „FilEnable“ freigegeben werden

CAN-Filter mit „Maske“ und „Code“, die Bits der Maske entscheiden, welche Bits des CAN-IDs mit Code übereinstimmen müssen, damit der Filter zuschlägt.
Die Tabelle verdeutlicht die Funktionsweise:

Bit	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Maske	1	1	1	1	1	1	1	1	1	0	0
Code	1	0	1	0	0	0	0	0	0	0	0
Filter	1	0	1	0	0	0	0	0	0	X	X

Ein X bedeutet, dass das entsprechende Bit den Wert 0 oder 1 haben kann.
Die Nachrichten mit den CAN-ID: 0x500 – 0x503 werden gefiltert.

Viele Definitionen der Struktur TMsgFilter sind für zukünftige Anwendungen vorgesehen, deshalb sollten, bevor ein Filter verwendet wird, alle Flags gelöscht werden, „FilFlags = 0“.

4.3 Header Files

Die Datei „can_types.h“ stellt alle Grundlegenden Daten-Typen und Definition für den CAN-Bus bereit. Die Datei „can_drv.h“ stellt das CAN-API Interfaces zur Verfügung, es wird automatisch die Datei „can_types.h“ und „can_drv_ex.h“ mit eingebunden. Die Datei „can_drv_ex.h“ bildet die Ex-Funktionen der API ab.

Die Datei „can_types.h“:

```
#ifndef __CAN_TYPES_H__
#define __CAN_TYPES_H__

#ifdef WIN32
```

```
  #ifndef __WIN32__
    #define __WIN32__
  #endif
#endif

#ifdef __WIN32__
// Windows
#include <windows.h>

#define int32_t __int32
#define uint32_t unsigned __int32
#define int16_t __int16
#define uint16_t unsigned __int16
#else
// Linux & Mac
#include <stdint.h>
#endif

#ifdef __cplusplus
  extern "C" {
#endif

#define INDEX_INVALID          0xFFFFFFFF

#define INDEX_FIFO_PUFFER_MASK 0x0000FFFF
#define INDEX_SOFT_FLAG        0x02000000
#define INDEX_RXD_TXT_FLAG     0x01000000
#define INDEX_CAN_KANAL_MASK   0x000F0000
#define INDEX_CAN_DEVICE_MASK  0x00F00000

#define INDEX_USER_MASK        0xFC000000

#define INDEX_CAN_KANAL_A      0x00000000
#define INDEX_CAN_KANAL_B      0x00010000

/*************************************************************/
/*  Typen                                                    */
/*************************************************************/

/****************************************/
/*            CAN Message Type          */
/****************************************/
#define MsgFlags Flags.Long
#define MsgLen Flags.Flag.Len
#define MsgRTR Flags.Flag.RTR
#define MsgEFF Flags.Flag.EFF
#define MsgTxD Flags.Flag.TxD
#define MsgSource Flags.Flag.Source
#define MsgData Data.Bytes

struct TCanFlagsBits
  {
  unsigned Len:4;    // DLC -> Datenlänge 0 - 8 Byte
  unsigned TxD:1;    // TxD -> 1 = Tx CAN Nachricht, 0 = Rx CAN Nachricht
                     //        Eine Erfolgreich versendete Nachricht wird als Bestätigung
                     //        ins Empfangsfifo zurückgeschrieben
                     //        Nicht alle Module unterstützen diese Funktion u. das
                     //        Feature muss aktiveirt sein
  unsigned Error:1;  // Error -> 1 = CAN Bus Fehler Nachricht
                     //        Nicht alle Module unterstützen diese Funktion u. das
                     //        Feature muss aktiveirt sein
  unsigned RTR:1;    // Remote Transmition Request bit -> Kennzeichnet eine RTR Nachricht
  unsigned EFF:1;    // Extended Frame Format bit -> 1 = 29 Bit Id's, 0 = 11 Bit Id's
  unsigned Source:8; // Quelle der Nachricht (Device)
  };

union TCanFlags
  {
  struct TCanFlagsBits Flag;
  uint32_t Long;
  };

union TCanData
  {
  char Chars[8];
  unsigned char Bytes[8];
  uint16_t Words[4];
  uint32_t Longs[2];
  };

struct TTime
  {
  uint32_t Sec;
  uint32_t USec;
  };
```

```
struct TCanMsg
  {
  uint32_t Id;
  union TCanFlags Flags;
  union TCanData Data;
  struct TTime Time;
  };

/*****************************************/
/*         CAN Message Filter Type       */
/*****************************************/
#define FilFlags Flags.Long
#define FilEFF Flags.Flag.EFF
#define FilMode Flags.Flag.Mode
#define FilIdMode Flags.Flag.IdMode
#define FilEnable Flags.Flag.Enable

// * = Reserviert, zur Zeit noch unbenutzt

struct TMsgFilterFlagsBits
  {
  // 1. Byte
  unsigned Len:4;       // * Dlc
  unsigned Res:2;       // Reserviert
  unsigned RTR:1;       // Remote Transmition Request
  unsigned EFF:1;       // Extended Frame Format
  // 2. Byte
  unsigned IdMode:2;    // 0 = Maske & Code
                        // 1 = Start & Stop
                        // 2 = Single Id
  unsigned DLCCheck:1;  // *
  unsigned DataCheck:1; // *
  unsigned Res1:4;
  // 3. Byte
  unsigned Res2:8;
  // 4. Byte
  unsigned Type:4;      // 0 = Single Puffer
  unsigned Res3:2;
  unsigned Mode:1;      // 0 = Message entfernen
                        // 1 = Message nicht entfernen
  unsigned Enable:1;    // 0 = Filter sperren
                        // 1 = Filter freigeben
  };


union TMsgFilterFlags
  {
  struct TMsgFilterFlagsBits Flag;
  uint32_t Long;
  };

struct TMsgFilter
  {                     // IdMode    -> Maske & Code | Start & Stop | Single Id
                        // --------------------------+--------------+-----------
  uint32_t Maske;       // Filter-Id ->    Maske     |     Stop     |
  uint32_t Code;        // Filter-Id ->    Code      |     Start    |    Id
  union TMsgFilterFlags Flags;
  union TCanData Data;  // *
  };

struct TCanIndexSource
  {
  // 1. u. 2 Byte
  unsigned SubIndex:16;
  // 3. Byte
  unsigned Source:8;
  // 4. Byte
  unsigned TxD:1;
  unsigned Soft:1;
  unsigned User:6;
  };

struct TCanIndexBits
  {
  // 1. u. 2 Byte
  unsigned SubIndex:16;
  // 3. Byte
  unsigned Channel:4;
  unsigned Device:4;
  // 4. Byte
  unsigned TxD:1;
  unsigned Soft:1;
  unsigned User:6;
  };

union TCanIndex
```

```
  {
  struct TCanIndexBits Item;
  struct TCanIndexSource SrcItem;
  uint32_t Long;
  };


#ifdef __cplusplus
  }
#endif


#endif
```

Die Datei „can_drv.h“:

```
#ifndef __CAN_DRV_H__
#define __CAN_DRV_H__

#include "can_types.h"

#ifdef __WIN32__
// ****** Windows
#include <windows.h>
#define CALLBACK_TYPE CALLBACK
#else
#define CALLBACK_TYPE
#endif

#ifdef __cplusplus
  extern "C" {
#endif

/*************************************************************/
/* Define Makros                                             */
/*************************************************************/

// CAN Übertragungsgeschwindigkeit
#define CAN_10K_BIT   10    // 10 kBit/s
#define CAN_20K_BIT   20    // 20 kBit/s
#define CAN_50K_BIT   50    // 50 kBit/s
#define CAN_100K_BIT  100   // 100 kBit/s
#define CAN_125K_BIT  125   // 125 kBit/s
#define CAN_250K_BIT  250   // 250 kBit/s
#define CAN_500K_BIT  500   // 500 kBit/s
#define CAN_800K_BIT  800   // 800 kBit/s
#define CAN_1M_BIT    1000  // 1 MBit/s

// Timestamp Mode
#define TIME_STAMP_OFF        0  // keine Time-Stamps
#define TIME_STAMP_SOFT       1  // Software Time-Stamps
#define TIME_STAMP_HW_UNIX    2  // Hardware Time-Stamps, UNIX-Format
#define TIME_STAMP_HW         3  // Hardware Time-Stamps
#define TIME_STAMP_HW_SW_UNIX 4  // Hardware Time-Stamps verwenden wenn verfügbar,
                                 // ansonsten Software Time-Stamps
                                 // Ab Treiber Version 4.08!

// CAN Bus Mode
#define OP_CAN_NO_CHANGE         0  // Aktuellen Zustand nicht ändern
#define OP_CAN_START             1  // Startet den CAN-Bus
#define OP_CAN_STOP              2  // Stopt den CAN-Bus
#define OP_CAN_RESET             3  // Reset CAN Controller (BusOff löschen)
#define OP_CAN_START_LOM         4  // Startet den CAN-Bus im Silent Mode (Listen Only Mode)
#define OP_CAN_START_NO_RETRANS  5  // Startet den CAN-Bus im Automatic Retransmission disable Mode

#define CAN_CMD_NONE                0x0000
#define CAN_CMD_RXD_OVERRUN_CLEAR   0x0001
#define CAN_CMD_RXD_FIFOS_CLEAR     0x0002
#define CAN_CMD_TXD_OVERRUN_CLEAR   0x0004
#define CAN_CMD_TXD_FIFOS_CLEAR     0x0008
#define CAN_CMD_HW_FILTER_CLEAR     0x0010
#define CAN_CMD_SW_FILTER_CLEAR     0x0020
#define CAN_CMD_TXD_PUFFERS_CLEAR   0x0040

#define CAN_CMD_ALL_CLEAR           0x0FFF


// DrvStatus
#define DRV_NOT_LOAD              0  // Die Treiber DLL wurde noch nicht geladen
#define DRV_STATUS_NOT_INIT       1  // Treiber noch nicht Initialisiert (Funktion "CanInitDrv" noch nicht
aufgerufen)
#define DRV_STATUS_INIT           2  // Treiber erfolgrich Initialisiert
#define DRV_STATUS_PORT_NOT_OPEN  3  // Die Schnittstelle wurde nicht geöffnet
#define DRV_STATUS_PORT_OPEN      4  // Die Schnittstelle wurde geöffnet
#define DRV_STATUS_DEVICE_FOUND   5  // Verbindung zur Hardware wurde Hergestellt
```

```
#define DRV_STATUS_CAN_OPEN       6  // Device wurde geöffnet und erfolgreich Initialisiert
#define DRV_STATUS_CAN_RUN_TX     7  // CAN Bus RUN nur Transmitter (wird nicht verwendet !)
#define DRV_STATUS_CAN_RUN        8  // CAN Bus RUN

// CanStatus
#define CAN_STATUS_OK          0     // CAN-Controller: Ok
#define CAN_STATUS_ERROR       1     // CAN-Controller: CAN Error
#define CAN_STATUS_WARNING     2     // CAN-Controller: Error warning
#define CAN_STATUS_PASSIV      3     // CAN-Controller: Error passiv
#define CAN_STATUS_BUS_OFF     4     // CAN-Controller: Bus Off
#define CAN_STATUS_UNBEKANNT   5     // CAN-Controller: Status Unbekannt

// FIFO Status
#define FIFO_OK                 0    // FIFO-Status: Ok
#define FIFO_HW_OVERRUN         1    // FIFO-Status: Hardware FIFO Überlauf
#define FIFO_SW_OVERRUN         2    // FIFO-Status: Software FIFO Überlauf
#define FIFO_HW_SW_OVERRUN      3    // FIFO-Status: Hardware & Software FIFO Überlauf
#define FIFO_STATUS_UNBEKANNT   4    // FIFO-Status: Unbekannt

// Makros für SetEvent
#define EVENT_ENABLE_PNP_CHANGE          0x0001
#define EVENT_ENABLE_STATUS_CHANGE       0x0002
#define EVENT_ENABLE_RX_FILTER_MESSAGES  0x0004
#define EVENT_ENABLE_RX_MESSAGES         0x0008
#define EVENT_ENABLE_ALL                 0x00FF

#define EVENT_DISABLE_PNP_CHANGE         0x0100
#define EVENT_DISABLE_STATUS_CHANGE      0x0200
#define EVENT_DISABLE_RX_FILTER_MESSAGES 0x0400
#define EVENT_DISABLE_RX_MESSAGES        0x0800
#define EVENT_DISABLE_ALL                0xFF00


/*************************************************************/
/*  Typen                                                    */
/*************************************************************/

/*****************************************/
/*            Device Status              */
/*****************************************/
struct TDeviceStatus
  {
  int32_t DrvStatus;         // Treiber Status (Device close / Device open / CAN Bus RUN)
  unsigned char CanStatus;   // Status des CAN Controllers (Ok / ... / Error passiv / Bus off)
  unsigned char FifoStatus;  // FIFO Status (Ok / ... / Hard. u. Soft. FIFO Überlauf)
  };


/*************************************************************/
/*  Funktionstypen                                           */
/*************************************************************/
typedef int32_t (CALLBACK_TYPE *TCanInitDriver)(char *options);
typedef void (CALLBACK_TYPE *TCanDownDriver)(void);
typedef int32_t (CALLBACK_TYPE *TCanSetOptions)(char *options);
typedef int32_t (CALLBACK_TYPE *TCanDeviceOpen)(uint32_t index, char *parameter);
typedef int32_t (CALLBACK_TYPE *TCanDeviceClose)(uint32_t index);

typedef int32_t (CALLBACK_TYPE *TCanSetMode)(uint32_t index, unsigned char can_op_mode,
                uint16_t  can_command);

typedef int32_t (CALLBACK_TYPE *TCanTransmit)(uint32_t index, struct TCanMsg *msg, int32_t count);
typedef void (CALLBACK_TYPE *TCanTransmitClear)(uint32_t index);
typedef uint32_t  (CALLBACK_TYPE *TCanTransmitGetCount)(uint32_t index);
typedef int32_t (CALLBACK_TYPE *TCanTransmitSet)(uint32_t index, uint16_t cmd, uint32_t time);
typedef int32_t (CALLBACK_TYPE *TCanReceive)(uint32_t index, struct TCanMsg *msg, int32_t count);
typedef void (CALLBACK_TYPE *TCanReceiveClear)(uint32_t index);
typedef uint32_t  (CALLBACK_TYPE *TCanReceiveGetCount)(uint32_t index);

typedef int32_t (CALLBACK_TYPE *TCanSetSpeed)(uint32_t index, uint16_t speed);
typedef int32_t (CALLBACK_TYPE *TCanSetSpeedUser)(uint32_t index, uint32_t value);
typedef char * (CALLBACK_TYPE *TCanDrvInfo)(void);
typedef char * (CALLBACK_TYPE *TCanDrvHwInfo)(uint32_t index);
typedef int32_t (CALLBACK_TYPE *TCanSetFilter)(uint32_t index, struct TMsgFilter *msg_filter);

typedef int32_t (CALLBACK_TYPE *TCanGetDeviceStatus)(uint32_t index, struct TDeviceStatus *status);

typedef void (CALLBACK_TYPE *TCanSetPnPEventCallback)(void (CALLBACK_TYPE *event)
   (uint32_t index, int32_t status));
typedef void (CALLBACK_TYPE *TCanSetStatusEventCallback)(void (CALLBACK_TYPE *event)
   (uint32_t index, struct TDeviceStatus *device_status));
typedef void (CALLBACK_TYPE *TCanSetRxEventCallback)(void (CALLBACK_TYPE *event)
   (uint32_t index, struct TCanMsg *msg, int32_t count));

typedef void (CALLBACK_TYPE *TCanSetEvents)(uint16_t events);
typedef uint32_t (CALLBACK_TYPE *TCanEventStatus)(void);
```

```
/*************************************************************/
/*  Tiny-CAN API Funktionen                                  */
/*************************************************************/
extern TCanInitDriver CanInitDriver;
extern TCanDownDriver CanDownDriver;
extern TCanSetOptions CanSetOptions;
extern TCanDeviceOpen CanDeviceOpen;
extern TCanDeviceClose CanDeviceClose;

extern TCanSetMode CanSetMode;

extern TCanTransmit CanTransmit;
extern TCanTransmitClear CanTransmitClear;
extern TCanTransmitGetCount CanTransmitGetCount;
extern TCanTransmitSet CanTransmitSet;
extern TCanReceive CanReceive;
extern TCanReceiveClear CanReceiveClear;
extern TCanReceiveGetCount CanReceiveGetCount;

extern TCanSetSpeed CanSetSpeed;
extern TCanSetSpeedUser CanSetSpeedUser;

extern TCanDrvInfo CanDrvInfo;
extern TCanDrvHwInfo CanDrvHwInfo;
extern TCanSetFilter CanSetFilter;

extern TCanGetDeviceStatus CanGetDeviceStatus;

extern TCanSetPnPEventCallback CanSetPnPEventCallback;
extern TCanSetStatusEventCallback CanSetStatusEventCallback;
extern TCanSetRxEventCallback CanSetRxEventCallback;

extern TCanSetEvents CanSetEvents;
extern TCanEventStatus CanEventStatus;

/*************************************************************/
/*  Funktionen Treiber laden/entladen                        */
/*************************************************************/
int32_t LoadDriver(char *file_name);
void UnloadDriver(void);

#include "can_drv_ex.h"

#ifdef __cplusplus
  }
#endif

#endif
```

Die Datei „can_drv_ex.h“:

```
#ifndef __CAN_DRV_EX_H__
#define __CAN_DRV_EX_H__


#ifdef __cplusplus
  extern "C" {
#endif

/*************************************************************/
/* Define Makros                                             */
/*************************************************************/
#define CAN_FEATURE_LOM          0x0001  // Silent Mode (LOM = Listen only Mode)
#define CAN_FEATURE_ARD          0x0002  // Automatic Retransmission disable
#define CAN_FEATURE_TX_ACK       0x0004  // TX ACK (Gesendete Nachrichten bestätigen)
#define CAN_FEATURE_HW_TIMESTAMP 0x8000  // Hardware Timestamp

#define VT_BYTE         0x01
#define VT_UBYTE        0x02
#define VT_WORD         0x03
#define VT_UWORD        0x04
#define VT_LONG         0x05
#define VT_ULONG        0x06

#define VT_BYTE_ARRAY   0x07
#define VT_UBYTE_ARRAY  0x08
#define VT_WORD_ARRAY   0x09
#define VT_UWORD_ARRAY  0x0A
#define VT_LONG_ARRAY   0x0B
#define VT_ULONG_ARRAY  0x0C

#define VT_BYTE_RANGE_ARRAY   0x0D
#define VT_UBYTE_RANGE_ARRAY  0x0E
#define VT_WORD_RANGE_ARRAY   0x0F
#define VT_UWORD_RANGE_ARRAY  0x10
```

```
#define VT_LONG_RANGE_ARRAY   0x11
#define VT_ULONG_RANGE_ARRAY  0x12

#define VT_HBYTE  0x40
#define VT_HWORD  0x41
#define VT_HLONG  0x42

#define VT_STREAM 0x80
#define VT_STRING 0x81
#define VT_POINTER 0x82


/*************************************************************/
/*  Typen                                                    */
/*************************************************************/
struct TModulFeatures
  {
  uint32_t CanClock;           // Clock-Frequenz des CAN-Controllers, muss nicht mit
                               // der Clock-Frequenz des Mikrocontrollers übereinstimmen
  uint32_t Flags;              // Unterstützte Features des Moduls:
                               //  Bit  0 -> Silent Mode (LOM = Listen only Mode)
                               //       1 -> Automatic Retransmission disable
                               //       2 -> TX ACK (Gesendete Nachrichten bestätigen)
                               //      15 -> Hardware Timestamp
  uint32_t CanChannelsCount;   // Anzahl der CAN Schnittstellen, reserviert für
                               // zukünftige Module mit mehr als einer Schnittstelle
  uint32_t HwRxFilterCount;    // Anzahl der zur Verfügung stehenden Receive-Filter
  uint32_t HwTxPufferCount;    // Anzahl der zur Verfügung stehenden Transmit Puffer mit Timer
  };

struct TCanDevicesList
  {
  uint32_t TCanIdx;            // Ist das Device geöffnet ist der Wert auf dem Device-Index
                               // gesetzt, ansonsten ist der Wert auf "INDEX_INVALID" gesetzt.
  uint32_t HwId;               // Ein 32 Bit Schlüssel der die Hardware eindeutig Identifiziert.
                               // Manche Module müssen erst geöffnet werden damit dieser Wert
                               // gesetzt wird
  char DeviceName[255];        // Nur Linux: entspricht den Device Namen des USB-Devices,
                               //            z.B. /dev/ttyUSB0
  char SerialNumber[16];       // Seriennummer des Moduls
  char Description[64];                // Modul Bezeichnung, z.B. "Tiny-CAN IV-XL",
                                       // muss in den USB-Controller programmiert sein,
                                       // was zur Zeit nur bei den Modulen Tiny-CAN II-XL,
                                       // IV-XL u. M1 der Fall ist.
  struct TModulFeatures ModulFeatures;  // Unterstützte Features des Moduls, nur gültig
                                       // wenn HwId > 0
  };


struct TCanDeviceInfo
  {
  uint32_t HwId;               // Ein 32 Bit Schlüssel der die Hardware eindeutig Identifiziert.
  uint32_t FirmwareVersion;    // Version der Firmware des Tiny-CAN Moduls
  uint32_t FirmwareInfo;       // Informationen zum Stand der Firmware Version
                               //   0 = Unbekannt
                               //   1 = Firmware veraltet, Device kann nicht geöffnet werden
                               //   2 = Firmware veraltet, Funktionsumfang eingeschränkt
                               //   3 = Firmware veraltet, keine Einschränkungen
                               //   4 = Firmware auf Stand
                               //   5 = Firmware neuer als Erwartet
  char SerialNumber[16];       // Seriennummer des Moduls
  char Description[64];        // Modul Bezeichnung, z.B. "Tiny-CAN IV-XL"
  struct TModulFeatures ModulFeatures; // Unterstützte Features des Moduls
  };


struct TCanInfoVar
  {
  uint32_t Key;      // Variablen Schlüssel
  uint32_t Type;     // Variablen Type
  uint32_t Size;     // (Max)Größe der Variable in Byte
  char Data[255];    // Wert der Variable
  };


typedef struct _TMhsEvent TMhsEvent;

struct _TMhsEvent
  {
  volatile uint32_t Events;
  volatile uint32_t EventsMask;
  volatile int32_t Waiting;
#ifdef __WIN32__
// ****** Windows
  HANDLE Event;
  CRITICAL_SECTION EventLock;
```

```
#else
// ****** Linux

#endif
  };

/*************************************************************/
/*  Funktionstypen                                           */
/*************************************************************/
typedef int32_t (CALLBACK_TYPE *TCanExGetDeviceCount)(int flags);
typedef int32_t (CALLBACK_TYPE *TCanExGetDeviceList)(struct TCanDevicesList **devices_list, int flags);
typedef int32_t (CALLBACK_TYPE *TCanExGetDeviceInfo)(uint32_t index, struct TCanDeviceInfo *device_info,
                struct TInfoVar **hw_info, uint32_t *hw_info_size);
typedef void (CALLBACK_TYPE *TCanExDataFree)(void **data);
typedef int32_t (CALLBACK_TYPE *TCanExCreateDevice)(uint32_t *index, char *options);
typedef int32_t (CALLBACK_TYPE *TCanExDestroyDevice)(uint32_t *index);
typedef int32_t (CALLBACK_TYPE *TCanExCreateFifo)(uint32_t index, uint32_t size, TMhsEvent *event_obj,
                uint32_t event, uint32_t channels);
typedef int32_t (CALLBACK_TYPE *TCanExBindFifo)(uint32_t fifo_index, uint32_t device_index,
                uint32_t bind);
typedef TMhsEvent * (CALLBACK_TYPE *TCanExCreateEvent)(void);
typedef int32_t (CALLBACK_TYPE *TCanExSetObjEvent)(uint32_t index, TMhsEvent *event_obj, uint32_t event);
typedef void (CALLBACK_TYPE *TCanExSetEvent)(TMhsEvent *event_obj, uint32_t event);
typedef void (CALLBACK_TYPE *TCanExResetEvent)(TMhsEvent *event_obj, uint32_t event);
typedef uint32_t (CALLBACK_TYPE *TCanExWaitForEvent)(TMhsEvent *event_obj, uint32_t timeout);
typedef int32_t (CALLBACK_TYPE *TCanExInitDriver)(char *options);
typedef int32_t (CALLBACK_TYPE *TCanExSetOptions)(uint32_t index, char *options);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsByte)(uint32_t index, char *name, char value);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsWord)(uint32_t index, char *name, int16_t value);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsLong)(uint32_t index, char *name, int32_t value);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsUByte)(uint32_t index, char *name, unsigned char value);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsUWord)(uint32_t index, char *name, uint16_t value);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsULong)(uint32_t index, char *name, uint32_t value);
typedef int32_t (CALLBACK_TYPE *TCanExSetAsString)(uint32_t index, char *name, char *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsByte)(uint32_t index, char *name, char *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsWord)(uint32_t index, char *name, int16_t *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsLong)(uint32_t index, char *name, int32_t *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsUByte)(uint32_t index, char *name, unsigned char *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsUWord)(uint32_t index, char *name, uint16_t *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsULong)(uint32_t index, char *name, uint32_t *value);
typedef int32_t (CALLBACK_TYPE *TCanExGetAsString)(uint32_t index, char *name, char **str);

/*************************************************************/
/*  Tiny-CAN API Funktionen                                  */
/*************************************************************/
extern TCanExGetDeviceCount CanExGetDeviceCount;
extern TCanExGetDeviceList CanExGetDeviceList;
extern TCanExGetDeviceInfo CanExGetDeviceInfo;
extern TCanExDataFree CanExDataFree;
extern TCanExCreateDevice CanExCreateDevice;
extern TCanExDestroyDevice CanExDestroyDevice;
extern TCanExCreateFifo CanExCreateFifo;
extern TCanExBindFifo CanExBindFifo;
extern TCanExCreateEvent CanExCreateEvent;
extern TCanExSetObjEvent CanExSetObjEvent;
extern TCanExSetEvent CanExSetEvent;
extern TCanExResetEvent CanExResetEvent;
extern TCanExWaitForEvent CanExWaitForEvent;
extern TCanExInitDriver CanExInitDriver;
extern TCanExSetOptions CanExSetOptions;
extern TCanExSetAsByte CanExSetAsByte;
extern TCanExSetAsWord CanExSetAsWord;
extern TCanExSetAsLong CanExSetAsLong;
extern TCanExSetAsUByte CanExSetAsUByte;
extern TCanExSetAsUWord CanExSetAsUWord;
extern TCanExSetAsULong CanExSetAsULong;
extern TCanExSetAsString CanExSetAsString;
extern TCanExGetAsByte CanExGetAsByte;
extern TCanExGetAsWord CanExGetAsWord;
extern TCanExGetAsLong CanExGetAsLong;
extern TCanExGetAsUByte CanExGetAsUByte;
extern TCanExGetAsUWord CanExGetAsUWord;
extern TCanExGetAsULong CanExGetAsULong;
extern TCanExGetAsString CanExGetAsString;

#ifdef __cplusplus
  }
#endif

#endif
```

3.5 API Funktionsaufrufe

Übersicht:

Treiber laden/entladen				
3.5.1	LoadDriver	Lädt einen Tiny-CAN API Treiber.	**STD** ***Ex***	24
3.5.2	UnloadDriver	Tiny-CAN API Treiber entladen	**STD** ***Ex***	25
Initialisierung und Konfiguration				
3.5.3	CanInitDriver	Initialisiert den Treiber	**STD**	26
3.5.2	CanDownDriver	Deinitialisiert den Treiber	**STD** ***Ex***	28
3.5.3	CanSetOptions	Setzen von Treiber-Optionen	**STD**	29
3.5.4	CanDeviceOpen	Öffnet ein CAN-Device	**STD** ***Ex***	30
3.5.5	CanDeviceClose	Schließt ein CAN-Device	**STD** ***Ex***	32
CAN Betriebsmodus				
3.5.6	CanSetMode	CAN-Bus Mode setzen	**STD** ***Ex***	33
CAN Nachrichten versenden				
3.5.7	CanTransmit	CAN-Messages in FIFO/Puffer schreiben	**STD** ***Ex***	35
3.5.8	CanTransmitClear	Sende-FIFO/Puffer löschen	**STD** ***Ex***	36
3.5.9	CanTransmitGetCount	Anzahl Nachrichten im Sende-FIFO/Puffer abfragen	**STD** ***Ex***	37
3.5.10	CanTransmitSet	Transmit Puffer senden und Intervall setzen	**STD** ***Ex***	38
CAN Nachrichten empfangen				
3.5.11	CanReceive	CAN-Messages von FIFO/Puffer auslesen	**STD** ***Ex***	40
3.5.12	CanReceiveClear	Empfangs-FIFO/Puffer löschen	**STD** ***Ex***	41
3.5.13	CanReceiveGetCount	Anzahl Nachrichten im Empfangs-FIFO/Puffer abfragen	**STD** ***Ex***	42
CAN-Bus Setup				
3.5.14	CanSetSpeed	CAN-Übertragungsgeschwindigkeit einstellen	**STD** ***Ex***	43
3.5.15	CanSetSpeedUser	Eine benutzerdefinierte CAN-Übertragungsgeschwindigkeit einstellen, z.B. 83,3	**STD** ***Ex***	44
3.5.16	CanSetFilter	CAN-Empfangsfilter setzen	**STD** ***Ex***	46
Treiber und Hardware Informationen abfragen				
3.5.17	CanDrvInfo	Treiber-Info-Variablen abfragen	**STD** ***Ex***	48
3.5.18	CanDrvHwInfo	Hardware-Info-Variablen abfragen	**STD**	49
Treiber und CAN Status abfragen				
3.5.19	CanGetDeviceStatus	Device Status abfragen	**STD** ***Ex***	50
Callbackfunktionen Konfiguration				
3.5.20	CanSetPnPEventCallback	Plug & Play Event-Callback-Funktionen setzen	**STD** ***Ex***	52
3.5.21	CanSetStatusEventCallback	Status Event-Callback-Funktionen setzen	**STD** ***Ex***	53
3.5.22	CanSetRxEventCallback	Receive Event-Callback-Funktionen setzen	**STD** ***Ex***	54
3.5.23	CanSetEvents	Event-Maske setzen	**STD** ***Ex***	55

3.6 EX-API Funktionsaufrufe

Übersicht:

Initialisierung und Konfiguration				
3.6.1	CanExInitDriver	Initialisiert den Treiber im Ex-API Modus	Ex	56
3.6.2	CanExCreateDevice	Ein Device erzeugen	Ex	60
3.6.3	CanExDestroyDevice	Ein Device löschen	Ex	61
3.6.4	CanExCreateFifo	Empfangs-FIFO anlegen	Ex	62
3.6.5	CanExBindFifo	Empfangsfifo an Device binden	Ex	63
Event-Funktionen				
3.6.6	CanExCreateEvent	Ein „Event“ Objekt erzeugen	Ex	64
3.6.7	CanExSetObjEvent	Objekt mit Event verknüpfen	Ex	65
3.6.8	CanExSetEvent	Ein Event auslösen	Ex	66
3.6.9	CanExResetEvent	Einen Event zurücksetzten	Ex	67
3.6.10	CanExWaitForEvent	Auf Event(s)/Timeout warten	Ex	68
Info				
3.6.11	CanExGetDeviceCount	Anzahl verbundener Devices abfragen	STD Ex	70
3.6.12	CanExGetDeviceList	Liste verbundener Devices ausgeben	STD Ex	71
3.6.13	CanExGetDeviceInfo	Informationen zur Hardware abfragen	STD Ex	74
Hilfsfunktionen				
3.6.14	CanExDataFree	Dynamisch allokierte Daten freigeben	STD Ex	78
Konfigurationsvariablen lesen/schreiben				
3.6.15	CanExSetOptions	Eine Liste von Variablen setzen	Ex	79
3.6.16	CanExSetAsByte	„byte“ (8 bit) Variable setzen	Ex	80
3.6.17	CanExSetAsWord	„word“ (16 bit) Variable setzen	Ex	81
3.6.18	CanExSetAsLong	„long“ (32 bit) Variable setzen	Ex	82
3.6.19	CanExSetAsUByte	„unsigned byte“ (8 bit) Variable setzen	Ex	83
3.6.20	CanExSetAsUWord	„unsigned word“ (16 bit) Variable setzen	Ex	84
3.6.21	CanExSetAsULong	„unsigned long“ (32 bit) Variable setzen	Ex	85
3.6.22	CanExSetAsString	„String“ Variable setzten	Ex	86
3.6.23	CanExGetAsByte	„byte“ (8 bit) Variable lesen	Ex	87
3.6.24	CanExGetAsWord	„word“ (16 bit) Variable lesen	Ex	88
3.6.25	CanExGetAsLong	„long“ (32 bit) Variable lesen	Ex	89
3.6.26	CanExGetAsUByte	„unsigned byte“ (8 bit) Variable lesen	Ex	90
3.6.27	CanExGetAsUWord	„unsigned word“ (16 bit) Variable lesen	Ex	91
3.6.28	CanExGetAsULong	„unsigned long“ (32 bit) Variable lesen	Ex	92
3.6.29	CanExGetAsString	„String“ Variable lesen	Ex	93

3.5.1 LoadDriver

Aufgabe Lädt einen Tiny-CAN API Treiber.

Syntax `int32_t LoadDriver(char *file_name)`

Parameter

`file_name`	Name des Treibers (DLL / Shared Lib) mit Verzeichnis. Wird „NULL“ übergeben wird „mhstcan.xxx“ geladen.

Rückgabewert Wenn der Treiber erfolgreich geladen worden ist, gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Einen Tiny-CAN API Treiber dynamisch laden. Die Treiber befinden sich im Unterverzeichnis „..\tiny_can\can_api“. Zur Zeit gibt es nur einen Treiber, mhstcan.xxxx der alle Module unterstützt.
Wird unter Windows kein Verzeichnis angegeben wird das Verzeichnis aus der Windows-Registrierungsdatenbank gelesen.

Beispiel, siehe auch „CanDeviceOpen“:

```
if (LoadDriver(NULL) < 0)
  {
  // Treiber konnte nicht geladen werden, Fehler behandeln
  }
```

3.5.2 UnloadDriver

Aufgabe	Tiny-CAN API Treiber entladen
Syntax	`void UnloadDriver(void)`
Parameter	`keine`
Rückgabewert	nichts
Beschreibung	Entlädt einen dynamisch geladenen API Treiber. Vor dem Entladen ruft die Funktion automatisch „CanDownDriver“ auf.

Beispiel siehe „CanDeviceClose“.

3.5.3 CanInitDriver

Aufgabe Initialisiert den Treiber

Syntax `int32_t CanInitDriver(char *options)`

Parameter

`options`	Übergibt einen Optionen-String an den CAN-Treiber. Aufbau des Strings siehe unten. Die Variablen können nur einmalig bei der Initialisierung festgelegt werden

Rückgabewert Wenn der Treiber erfolgreich initialisiert worden ist, gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Die Funktion „CanInitDrv“ initialisiert den Treiber. Alle Funktionen des Treibers sind erst nach erfolgreicher Initialisierung verfügbar. Einzige Ausnahme ist die Funktion „CanDrvInfo“. Die von der DLL verwendeten Systemressourcen werden erst nach Aufruf von CanInitDrv belegt.
Achtung: Um die „Ex-API“ Funktionalität (Multi-Devices Support) nutzen zu können muss die „CanExInitDriver“ Funktion aufgerufen werden.

Aufbau des Option-Strings:

```
[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]
```

Beispiel:

Empfangsfifo auf 10000 CAN-Messages und Sendefifo auf 10 CAN-Messages

```
CanRxDFifoSize=10000;CanTxDFifoSize=10
```

Bezeichner	Beschreibung	Type	Init.
CanRxDFifoSize	Größe des Empfangsfifos in Messages	ULong	32768
CanTxDFifoSize	Größe des Sendefifos in Messages	ULong	2048
CanRxDMode	0 = Die RxD Callbackfunktion übergibt keine CAN-Messages 1 = Die RxD Callbackfunktion übergibt die empfangenen CAN-Messages	UByte	0
CanRxDBufferSize	Größe des Übergabepuffers für RxD Event Proc., nur gültig wenn CanRxDMode = 1.	ULong	50
CanCallThread	0 = Callback Thread nicht erzeugen 1 = Callback Thread erzeugen	UByte	1
MainThreadPriority	0 = THREAD_PRIORITY_NORMAL 1 = THREAD_PRIORITY_ABOVE_NORMAL 2 = THREAD_PRIORITY_HIGHEST 3 = THREAD_PRIORITY_TIME_CRITICAL 4 = THREAD_PRIORITY_REALTIME	ULong	3
CallThreadPriority	0 = THREAD_PRIORITY_NORMAL 1 = THREAD_PRIORITY_ABOVE_NORMAL	ULong	1

Bezeichner	Beschreibung	Type	Init.
	2 = THREAD_PRIORITY_HIGHEST 3 = THREAD_PRIORITY_TIME_CRITICAL		
Hardware	Reserviert, sollte nicht gesetzt werden.	UByte	0
CfgFile	Config File Name das von der DLL geladen wird. Wird der Dateiname ohne Pfad angegeben, so wird der Pfad der DLL benutzt, unter Linux/MacOS X das Verzeichnis „/etc/tiny_can“.	String	
Section	Name der Section, die im Config File gelesen wird	String	„Default“
LogFile	Dateiname des Log-Files, ein leerer String legt kein Log-File an. Wird der Dateiname ohne Pfad angegeben, so wird der Programmpfad der Applikation benutzt, unter Linux/MacOS X das Verzeichnis „/var/log/tiny_can“.	String	„“
LogFlags	Log Flags, siehe Kapitel 7. Log Files	ULong	0
TimeStampMode	0 = Disabled 1 = Software Time-Stamps 2 = Hardware Time-Stamps, UNIX-Format 3 = Hardware Time-Stamps 4 = Hardware Time-Stamps verwenden wenn verfügbar, ansonsten Software Time-Stamps	ULong	1

Nicht in der Tabelle aufgeführte Bezeichner werden ignoriert, die Funktion liefert keinen Fehler zurück.

Beispiel siehe „CanDeviceOpen“.

3.5.2 CanDownDriver

Aufgabe	Deinitialisiert den Treiber
Syntax	`void CanDownDriver(void)`
Parameter	`keine`
Rückgabewert	nichts
Beschreibung	Gibt alle Systemressourcen wieder frei. Ein mehrmaliges Aufrufen der Funktionen führt zu keinem Fehler. **Achtung:** Diese Funktion blockiert solange bis alle Callbackfunktionen zurückgekehrt sind! Andere Threads dürfen keine API-Funktionen mehr aufrufen

Beispiel siehe „CanDeviceClose“.

3.5.3 CanSetOptions

Aufgabe Setzen von Treiber-Optionen

Syntax `int32_t CanSetOptions(char *options)`

Parameter

`options`	Übergibt einen Optionen-String an den CAN-Treiber. Aufbau des Strings siehe unten.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Setzen von Treiber-Option-Variablen, im Gegensatz zu den mit „CanInitDriver“ festgelegten Variablen können diese Variablen jederzeit geändert werden.

Aufbau des Option-Strings:

```
[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]
```

Beispiel:

CAN Übertragungsgeschwindigkeit auf 500 kBit/s und Auto Connect Modus ein.

```
CanSpeed1=500;AutoConnect=1
```

Bezeichner	Beschreibung	Type	Init.
CanTxAckEnable	0 = Transmit Message Request sperren 1 = Transmit Message Request freigeben	UByte	0
CanSpeed1	CAN Übertragungsgeschwindigkeit in kBit/s z.B. 100 = 100kBit/s, 1000 = 1MBit/s	UWord	125kBit/s
CanSpeed1User	Wert des BTR Register des CAN-Controllers	ULong	
AutoConnect	0 = Auto Connect Modus aus 1 = Auto Connect Modus ein	UByte	0
AutoReopen	0 = CanDeviceOpen wird nicht automatisch aufgerufen 1 = CanDeviceOpen wird automatisch aufgerufen, nachdem die Verbindung wiederhergestellt wurde	UByte	0
MinEventSleepTime	Min. Wartezeit für das wiederholte Aufrufen von Event Callbacks in ms	ULong	25
ExecuteCommandTimeout	Maximale Wartezeit für Kommando Ausführung in ms	ULong	6000
LowPollIntervall	Hardware Polling Intervall in ms	ULong	250
FilterReadIntervall	Filter Messages alle x ms einlesen	ULong	1000

3.5.4 CanDeviceOpen

Aufgabe Öffnet ein CAN-Device

Syntax `int32_t CanDeviceOpen(uint32_t index, char *parameter)`

Parameter

`index` Muss auf 0 gesetzt werden

`parameter` Der Parameter-String enthält Angaben zur PC-Schnittstelle. Aufbau des Strings siehe unten.

Rückgabewert Wenn das Device erfolgreich geöffnet worden ist, gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Die Funktion öffnet ein CAN-Device, also die Schnittstelle des PCs zur Hardware und baut eine Verbindung zu dieser auf. Die Schnittstelle kann mit „CanDeviceClose“ wieder geschlossen werden.

Aufbau des Parameter-Strings:

```
[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]
```

Beispiel:

Serielle Schnittstelle auf COM 4 und Baudrate auf 38400 Baud.

```
Port=4;BaudRate=38400
```

Bezeichner	Beschreibung	Type	Init.
ComDrvType	0 = RS232 Schnittstelle 1 = USB (FTDI)	Long	1
Port	Serielle Schnittstelle 1 = COM1... (wird für den USB-Bus nicht verwendet)	Long	1
ComDeviceName	Device Name (Linux: /dev/ttyUSB0)*	String	„“
BaudRate	Baudrate, z.B. 38400 = 38400 Baud*	ULong	921600
VendorId	USB-Vendor Id (nur Windows)*	Long	0403
ProductId	USB-Product Id (nur Windows)*	Long	6001
Snr	Seriennummer des CAN Moduls	String	„“

* = Diese Einstellungen sollten nicht geändert werden!

Beispiel für das öffnen und Initialisieren eines CAN-Devices
Der Code ist bei den folgenden Beispielen bei „... Initialisierung ...“ einzufügen.

```
/*****************************/
/*  Initialisierung          */
/*****************************/
// **** Treiber DLL laden, mhstcan.dll, Verzeichnis automatisch bestimmen
if ((err = LoadDriver(NULL)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL initialisieren
if ((err = CanInitDriver(NULL)) < 0)
  {
  printf("CanInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> USB-Tiny öffnen
if ((err = CanDeviceOpen(0, NULL)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
/*************************************/
/*  CAN Speed einstellen & Bus starten */
/*************************************/
// **** Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_125K_BIT);

// Achtung: Um Fehler auf dem Bus zu vermeiden ist die Übertragungsgeschwindigkeit
//          vor dem starten des Busses einzustellen.

// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR);
```

Beispiel für die Initialisierung im „Ex“-Modus siehe „CanExInitDriver“

3.5.5 CanDeviceClose

Aufgabe	Schließt ein CAN-Device	
Syntax	**`int32_t CanDeviceClose(uint32_t index)`**	
Parameter	**`index`**	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
Rückgabewert	Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück	
Beschreibung	Die Funktion schließt ein CAN-Device, also die Schnittstelle des PCs zur Hardware. Die Schnittstelle kann mit „CanDeviceOpen“ wieder geöffnet werden.	

Beispiel für das schließen eins CAN-Devices und beenden/entladen des API Treibers
Der Code ist bei den folgenden Beispielen bei „... beenden ...“ einzufügen.

```
/*****************************/
/*  Treiber beenden          */
/*****************************/
ende :

// **** CAN-Device schließen
CanDeviceClose(0);
// **** Treiber beenden
// CanDownDriver wird auch automatisch von UnloadDriver aufgerufen,
// der separate Aufruf ist nicht zwingend notwendig
CanDownDriver();
// **** DLL entladen
UnloadDriver();
```

3.5.6 CanSetMode

Aufgabe CAN-Bus Mode setzen

Syntax `int32_t CanSetMode(uint32_t index, unsigned char can_op_mode, uint16_t can_command)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`can_op_mode`	Siehe Tabelle, Werte für „can_op_mode“
`can_command`	Siehe Tabelle, Werte für can_command“

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Setzen des CAN Betriebsmodus Start/Stop/Reset

Werte für „can_op_mode“:

Define	Wert	Beschreibung
OP_CAN_NO_CHANGE	0	Zustand des CAN-Buses nicht ändern, nur „can_command“ ausführen
OP_CAN_START	1	Startet den CAN-Bus
OP_CAN_STOP	2	Stoppt den CAN-Bus
OP_CAN_RESET	3	CAN-Bus Reset, zum Löschen des BusOff- Zustandes
OP_CAN_START_LOM	4	Startet den CAN-Bus im Silent Mode (Listen Only Mode)
OP_CAN_START_NO_RETRANS	5	Startet den CAN-Bus im „Automatic Retransmission Disable“ Mode

Werte für „can_command“:

Define	Wert	Beschreibung
CAN_CMD_NONE	0x0000	Keinen Befehl ausführen
CAN_CMD_RXD_OVERRUN_CLEAR	0x0001	Fehler Überlauf Empfangs-FIFO löschen
CAN_CMD_RXD_FIFOS_CLEAR	0x0002	Alle Empfangs-FIFOs löschen, auch das FIFO in der Hardware
CAN_CMD_TXD_OVERRUN_CLEAR	0x0004	Fehler Überlauf Sende-FIFO löschen
CAN_CMD_TXD_FIFOS_CLEAR	0x0008	Alle Sende-FIFOs löschen, auch das FIFO in der Hardware
CAN_CMD_HW_FILTER_CLEAR	0x0010	Alle Hardware Filter löschen
CAN_CMD_SW_FILTER_CLEAR	0x0020	Alle Software Filter löschen
CAN_CMD_TXD_PUFFERS_CLEAR	0x0040	Alle TxD Puffer (Intervall-Messages) löschen
CAN_CMD_ALL_CLEAR	0x0FFF	Alle Befehle ausführen

Mehrere Befehle ausführen:
z.B. Hardware und Software Filter löschen
CAN_CMD_HW_FILTER_CLEAR | CAN_CMD_SW_FILTER_CLEAR

Beispiel:

```
// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
if (CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR) < 0)
  {
  // Fehler
  }
```

3.5.7 CanTransmit

Aufgabe CAN-Messages in FIFO/Puffer schreiben

Syntax `int32_t CanTransmit(uint32_t index, struct TCanMsg *msg, int32_t count)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`msg`	Zeiger auf CAN-Messages
`count`	Anzahl der zu schreibenden CAN-Messages, auf die „msg“ zeigt

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Eine Anzahl von CAN-Nachrichten, in das mit „index“ angegebenen FIFO/Puffer schreiben.

Beispiel:

```
int main(int argc, char **argv)
{
int err;
struct TCanMsg msg;

//
  ... Initialisierung ...
//

// msg Variable Initialisieren
msg.MsgFlags = 0L; // Alle Flags löschen
                   // MsgRTR = 0 -> keine RTR
                   // MsgEFF = 0 -> Stanadrt Frame Format
                   // MsgLen = 0 -> Datenlänge auf 0

//msg.MsgRTR = 1;    // Nachricht als RTR Frame versenden
//msg.MsgEFF = 1;    // Nachricht im EFF (Ext. Frame Format) versenden

msg.Id = 0x100;    // Message Id auf 100 Hex
msg.MsgLen = 5;    // Datenlänge auf 5
memcpy(msg.MsgData, "HALLO", 5);
// **** Nachricht in das Senden-FIFO schreiben
if ((err = CanTransmit(0, &msg, 1)) < 0)
  {
  printf("CanTransmit Error-Code:%d\n\r", err);
  goto ende;
  }

Sleep(500);       // 500 ms warten

ende :
//
  ... beenden ...
//
}
```

3.5.8 CanTransmitClear

Aufgabe Sende-FIFO/Puffer löschen

Syntax `void CanTransmitClear(uint32_t index)`

Parameter `index` Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“

Rückgabewert nichts

Beschreibung Das mit „index“ angegebenen FIFO/Puffer löschen

Beispiel:

```
/*********************************************************************
  Es werden 1000 Nachrichten in das Senden-FIFO geschrieben, der
  Sendevorgangs wird jedoch vorzeitig mit CanTransmitClear abgebrochen.
*********************************************************************/
int main(int argc, char **argv)
{
int32_t err;
//
  ... Initialisierung …
  ... Nachrichten Puffer mit 1000 Nachrichten erzeugen ...
//
if ((err = CanTransmit(0, MsgPuffer, 1000)) < 0)
  {
  printf("CanTransmit Error-Code:%d\n\r", err);
  goto ende;
  }

Sleep(50);  // 50 ms warten

CanTransmitClear(0);

ende :
//
  ... beenden ...
//
}
```

3.5.9 CanTransmitGetCount

Aufgabe	Anzahl Nachrichten im Sende-FIFO/Puffer abfragen	
Syntax	**`uint32_t CanTransmitGetCount(uint32_t index)`**	
Parameter	**`index`**	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
Rückgabewert	Anzahl der Nachrichten	
Beschreibung	Liefert die Anzahl der Nachrichten, in den mit „index“ angegebenen FIFO/Puffer zurück	

Beispiel:

```
int main(int argc, char **argv)
{
int32_t err;

//
  ... Initialisierung …
  ... Nachrichten Puffer mit 1000 Nachrichten erzeugen ...
//

// **** 100 Nachrichten in das Sende-FIFO schreiben
if ((err = CanTransmit(0, MsgPuffer, 100)) < 0)
  {
  printf("CanTransmit Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Warten bis das Sende FIFO leer ist
while (CanTransmitGetCount(0));

ende :
//
  ... beenden ...
//
}
```

3.5.10 CanTransmitSet

Aufgabe Transmit Puffer senden und Intervall setzen

Syntax `int32_t CanTransmitSet(uint32_t index, uint16_t cmd, uint32_t time);`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`cmd`	Bit 0 = CAN-Puffer senden Bit 15 = Transmitt Intervall einstellen
`time`	Transmit Intervall in µS, der Parameter wird nur ausgewertet, wenn Bit 15 von „cmd“ 1 ist.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Einen mit „index“ angegebenen Transmit Puffer senden bzw. Intervall Timer einstellen. Bei FIFOs kann CanTransmitSet nicht angewendet werden.

Beispiel:

```
int main(int argc, char **argv)
{
int32_t err;
struct TCanMsg msg;

//
  ... Initialisierung ...
//

// ****  Sende Puffer 1 laden
// msg Variable Initialisieren
msg.MsgFlags = 0L; // Alle Flags löschen, Stanadrt Frame Format,
                   // keine RTR, Datenlänge auf 0
msg.Id = 0x100;
msg.MsgLen = 5;
memcpy(msg.MsgData, "HALLO", 5);

if ((err = CanTransmit(1, &msg, 1)) < 0)
  {
  printf("CanTransmit Error-Code:%d\n\r", err);
  goto ende;
  }

// **** Intervalltimer auf 100ms setzten
if ((err = CanTransmitSet(1, 0x8000, 100000)) < 0)
  {
  printf("CanTransmitSet Error-Code:%d\n\r", err);
  goto ende;
  }
// **** 1 Sekunde Pause
Sleep(1000);

// **** Intervalltimer aus
if ((err = CanTransmitSet(1, 0x8000, 0)) < 0)
  {
  printf("CanTransmitSet Error-Code:%d\n\r", err);
```

```
  goto ende;
  }
ende :
//
  ... beenden ...
//
}
```

3.5.11 CanReceive

Aufgabe CAN-Messages von FIFO/Puffer auslesen

Syntax `int32_t CanReceive(uint32_t index, struct TCanMsg *msg, int32_t count)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`msg`	Zeiger auf den Puffer, in den die gelesenen Messages kopiert werden
`count`	Gibt die Größe des Puffers auf den „msg“ zeigt an und begrenzt die Anzahl der zu lesenden Messages

Rückgabewert Größer oder gleich 0 entspricht der Anzahl der gelesenen Messages, im Fehlerfall wird ein „Error-Code“ zurückgegeben.

Beschreibung Lesen von CAN-Messages aus dem mit „index“ angegebenen FIFO/Puffer. Die Messages werden im Puffer, auf den „msg“ zeigt abgelegt, es werden maximal „count“ Messages gelesen.

Beispiel:

```
int main(int argc, char **argv)
{
int err;
unsigned long i;
struct TCanMsg msg;

//
  ... Initialisierung ...
//

printf("Empfangene CAN-Messages :\n\r");
while (!KeyHit())
  {
  // Eine CAN Nachricht aus dem Empfangsfifo in "msg" lesen
  if (CanReceive(0, &msg, 1) > 0)
    {
    printf("id:%03lX dlc:%01d data:", msg.Id, msg.MsgLen);
    if (msg.MsgLen)
      {
      for (i = 0; i < msg.MsgLen; i++)
        printf("%02X ", msg.MsgData[i]);
      }
    else
      printf(" keine");
    printf("\n\r");
    }
  }

ende :
//
  ... beenden ...
//
}
```

3.5.12 CanReceiveClear

Aufgabe	Empfangs-FIFO/Puffer löschen	
Syntax	**`void CanReceiveClear(uint32_t index)`**	
Parameter	**`index`**	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
Rückgabewert	nichts	
Beschreibung	Den mit „index“ angegebenen FIFO/Puffer löschen	

Beispiel:

```
/*************************************************************************
  Die ersten 10 Nachrichten werden ignoriert und aus dem Empfangs-FIFO
  gelöscht
*************************************************************************/
int main(int argc, char **argv)
{
unsigned long i;
struct TCanMsg msg;

//
  ... Initialisierung ...
//

printf("Warten auf erste 10 Nachrichten\n");
// **** Warten bis 10 Nachrichten im Empfangs-FIFO
while (CanReceiveGetCount(0) < 10);

// **** Empfangs-FIFO löschen
CanReceiveClear(0);

// **** Nachrichten empfangen
printf("Empfangene CAN-Messages :\n\r");
while (!KeyHit())
  {
  // Eine CAN Nachricht aus dem Empfangs-FIFO in "msg" lesen
  if (CanReceive(0, &msg, 1) > 0)
    {
    printf("id:%03lX dlc:%01d data:", msg.Id, msg.MsgLen);
    if (msg.MsgLen)
      {
      for (i = 0; i < msg.MsgLen; i++)
        printf("%02X ", msg.MsgData[i]);
      }
    else
      printf(" keine");
    printf("\n\r");
    }
  }
//
  ... beenden ...
//
}
```

3.5.13 CanReceiveGetCount

Aufgabe Anzahl Nachrichten im Empfangs-FIFO/Puffer abfragen

Syntax `uint32_t CanReceiveGetCount(uint32_t index)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“

Rückgabewert Anzahl der Messages

Beschreibung Liefert die Anzahl der Messages in den mit „index“ angegebenen FIFO/Puffer zurück

Beispiel:

```
/*********************************************************************
  Es wird gewartet bis 10 Nachrichten im Empfangs-FIFO sind bis dann
  alle 10 gelesen werden.
*********************************************************************/
int main(int argc, char **argv)
{
unsigned long i, ii;
struct TCanMsg msg[10];

//
  ... Initialisierung ...
//

printf("Empfangene CAN-Messages :\n\r");
while (!KeyHit())
  {
  if (CanReceiveGetCount(0) < 10)  // Weniger als 10 Nachrichten im Empfangsfifo
    continue;                      // Ja -> Schleifenanfang
  // 10 CAN Nachrichten aus dem Empfangsfifo in "msg" lesen und ausgeben
  if (CanReceive(0, &msg, 10) == 10)
    {
    for (i = 0; i < 10; i++)
      {
      printf("id:%03lX dlc:%01d data:", msg[i].Id, msg[i].MsgLen);
      if (msg[i].MsgLen)
        {
        for (ii = 0; ii < msg[i].MsgLen; ii++)
          printf("%02X ", msg[i].MsgData[ii]);
        }
      else
        printf(" keine");
      printf("\n\r");
      }
    }
  }
//
  ... beenden ...
//
}
```

3.5.14 CanSetSpeed

Aufgabe CAN-Übertragungsgeschwindigkeit einstellen

Syntax `int32_t CanSetSpeed(uint32_t index, uint16_t speed)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, Der Parameter „index“
`speed`	Übertragungsgeschwindigkeit in kBit/s, 100 = 100kBit/s

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Setzen der Übertragungsgeschwindigkeit für ein CAN-Device, CAN-Kanal entsprechend „index“.

Gültige Werte für „speed“

Define	Wert	Beschreibung
CAN_10K_BIT	10	10 kBit/s
CAN_20K_BIT	20	20 kBit/s
CAN_50K_BIT	50	50 kBit/s
CAN_100K_BIT	100	100 kBit/s
CAN_125K_BIT	125	125 kBit/s
CAN_250K_BIT	250	250 kBit/s
CAN_500K_BIT	500	500 kBit/s
CAN_800K_BIT	800	800 kBit/s
CAN_1M_BIT	1000	1000 kBit/s

Beispiel:

```
// **** Übertragungsgeschwindigkeit auf 125kBit/s einstellen
if (CanSetSpeed(0, CAN_125K_BIT) < 0)
  {
  // Fehler
  }
```

3.5.15 CanSetSpeedUser

Aufgabe Eine benutzerdefinierte CAN-Übertragungsgeschwindigkeit einstellen, z.B. 83,3 kBit/s.

Syntax `int32_t CanSetSpeed(uint32_t index, uint32_t value)`

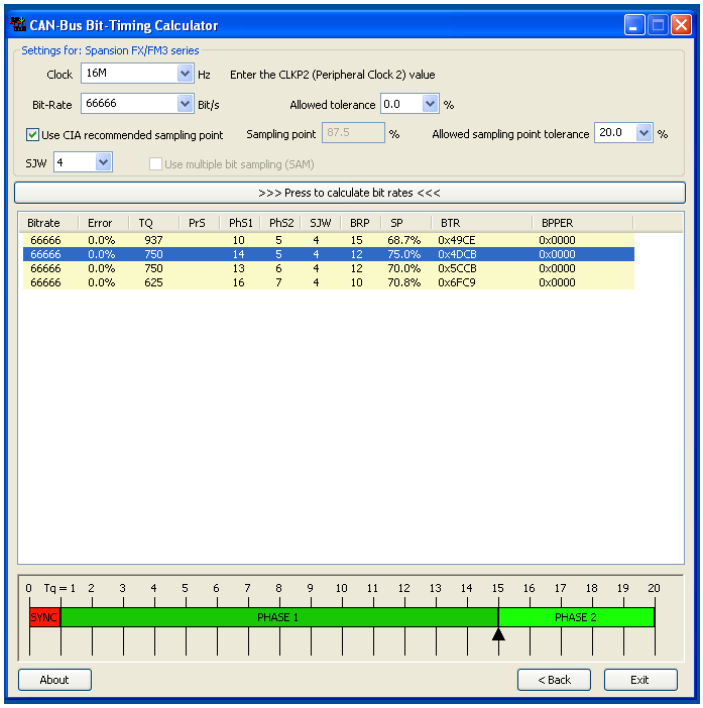
Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`speed`	Übertragungsgeschwindigkeit in kBit/s, 100 = 100kBit/s

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Die Funktion schreibt direkt das BTR Register des CAN-Controllers.

Der BTR Wert lässt sich bequem mit dem Programm „CAN-Bus Bit-Timing Calculator“ berechnen:



1. Manufacturer: Spansion
2. Series: siehe Tabelle
3. Clock auf 24 MHz oder 16MHz einstellen, siehe Tabelle.

Modul	Serie	Clock [MHz]
Tiny-CAN I	LX	24
Tiny-CAN I-XL	FX/FM3	16
Tiny-CAN I-XL Embedded	FX/FM3	16
Tiny-CAN II	LX	24
Tiny-CAN II-XL	FX/FM3	16
Tiny-CAN III	LX	24
Tiny-CAN III-XL	FX/FM3	16
Tiny-CAN M1	FX/FM3	16
Tiny-CAN M2	FX/FM3	16
Tiny-CAN M232	FX/FM3	16
Tiny-CAN IV-XL	FX/FM3	16
Tiny-CAN LS	FX/FM3	16

4. Werte für Bitrate, Tolleranz, … setzen
5. Dem Button „>>> Press to calculate bit rates <<<“ drücken
6. Gewünschten Eintrag in der Tabelle markieren
7. Den BTR-Wert als Hex Zahl ablesen

Beispiel für Benutzerdefinierte CAN Übertragungsraten „sample6“.

Beispiel:

```
// BTR Werte für 40kBit/s bei 16 u. 24 MHz
#define CAN_40K_16MHZ 0x3E53
#define CAN_40K_24MHZ 0x3E5D
```

```
int32_t Set40Bits(uint32_t device_index)
{
int32_t err;
struct TCanDeviceInfo device_info;

if ((err = CanExGetDeviceInfo(device_index, &device_info, NULL, NULL)) < 0)
  return(err)
// **** Übertragungsgeschwindigkeit auf 40kBit/s einstellen
if (device_info.ModulFeatures.CanClock == 16)
  err = CanSetSpeedUser(device_index, CAN_40K_16MHZ);
else
  err = CanSetSpeedUser(device_index, CAN_40K_24MHZ);
return(err);
}
```

Hinweis: Die „ModulFeatures“ können auch mit der Funktion „CanExGetDeviceList“ ermittelt werden mit Ausnahme für die Module Tiny-CAN I und Tiny-CAN M232. Beim „CanExGetDeviceList“ Aufruf muss das Device noch nicht geöffnet sein.

3.5.16 CanSetFilter

Aufgabe CAN-Empfangsfilter setzen

Syntax `int32_t CanSetFilter(uint32_t index, struct TMsgFilter *msg_filter)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`msg_filter`	Zeiger auf den zu setzenden Filter

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Setzt einen CAN-Empfangsfilter, mehr im Kapitel CAN-Filter

Aufbau der „TMsgFilter“ Struktur:

```
/*****************************************/
/*         CAN Message Filter Type       */
/*****************************************/
#define FilFlags Flags.Long
#define FilEFF Flags.Flag.EFF
#define FilMode Flags.Flag.Mode
#define FilIdMode Flags.Flag.IdMode
#define FilEnable Flags.Flag.Enable

// * = Reserviert, zur Zeit noch unbenutzt

struct TMsgFilterFlagsBits
  {
  // 1. Byte
  unsigned Len:4;       // * Dlc
  unsigned Res:2;       // Reserviert
  unsigned RTR:1;       // Remote Transmition Request
  unsigned EFF:1;       // Extended Frame Format
  // 2. Byte
  unsigned IdMode:2;    // 0 = Maske & Code
                        // 1 = Start & Stop
                        // 2 = Single Id
  unsigned DLCCheck:1;  // *
  unsigned DataCheck:1; // *
  unsigned Res1:4;
  // 3. Byte
  unsigned Res2:8;
  // 4. Byte
  unsigned Type:4;      // 0 = Single Puffer
  unsigned Res3:2;
  unsigned Mode:1;      // 0 = Message entfernen
                        // 1 = Message nicht entfernen
  unsigned Enable:1;    // 0 = Filter sperren
                        // 1 = Filter freigeben
  };


union TMsgFilterFlags
  {
  struct TMsgFilterFlagsBits Flag;
  uint32_t Long;
  };

struct TMsgFilter
  {                     // IdMode    -> Maske & Code | Start & Stop | Single Id
```

```
                        // --------------------------+--------------+-----------
  uint32_t Maske;       // Filter-Id ->    Maske     |     Stop     |
  uint32_t Code;        // Filter-Id ->    Code      |     Start    |    Id
  union TMsgFilterFlags Flags;
  union TCanData Data;  // *
  };
```

Beispiel:

```
/*****************************/
/*  Filter 1 setzen           */
/*****************************/

//         Bit 11     -     Bit0
// Maske   0 1 1 1 1 1 1 1 1 0 => 0x3FF
// Code    0 0 1 0 0 0 0 0 0 0 => 0x100
// Filter  X 0 0 0 0 0 0 0 0 X
// Die CAN Messages 0x100 u. 0x500 werden gefiltert
msg_filter.Maske = 0x3FF;
msg_filter.Code = 0x100;
msg_filter.Flags.Long = 0L;
msg_filter.FilEnable = 1;   // Filter freigeben

if ((err = CanSetFilter(1, &msg_filter)) < 0)
  {
  printf("CanSetFilter Error-Code:%d\n\r", err);
  goto ende;
  }
```

3.5.17 CanDrvInfo

Aufgabe	Treiber-Info-Variablen abfragen
Syntax	**`char *CanDrvInfo(void)`**
Parameter	**`keine`**
Rückgabewert	Zeiger auf Info-String
Beschreibung	Liefert Informationen zum Treiber

Aufbau des Info Strings:

`[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]`

Beispiel:

`Hardware=Tiny-CAN Classic;Version=1.00;Interface Type=Serial`

```
int main(int argc, char **argv)
{
int err, match;
char *s, *info_str, *key, *val;

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// Zum Abfragen der Treiber-Info-Variablen ist keine Initialisierung
// des Treibers notwendig

printf("Treiber Info:\n\r");
if (!(s = CanDrvInfo()))
  {
  printf("CanDrvInfo Error\n\r");
  goto ende;
  }
// Die Funktion get_item_as_string verändert den ursprünglichen String,
// darum muss eine Kopie des Strings erstellt werden
info_str = mhs_strdup(s);
s = info_str;
do
  {
  // Bezeichner auslesen
  key = get_item_as_string(&s, ":=", &match);
  if (match <= 0)
    break;
  // Value auslesen
  val = get_item_as_string(&s, ";", &match);
  if (match < 0)
    break;
  printf("%-20s : %s\n\r", key, val);
  }
while(1);
save_free(info_str);  // Allokierten Speicher für den String freigeben

// **** DLL entladen
ende :
UnloadDriver();
return(0);
}
```

3.5.18 CanDrvHwInfo

Aufgabe	Hardware-Info-Variablen abfragen	
Syntax	**`char *CanDrvHwInfo(uint32_t index)`**	
Parameter	**`index`**	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
Rückgabewert	Zeiger auf Hardware-Info-String	
Beschreibung	Liefert Informationen zum angeschlossenen CAN-Device	

Aufbau des Hardware Info Strings:

`[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]`

Beispiel:

`Hardware ID String=TINY-CAN;Hardware Snr=00 00 00 01`

```
int main(int argc, char **argv)
{
int match;
char *s, *info_str, *key, *val;

//
  ... Initialisierung ...
   Zur Abfrage der Hardware-Info-Variablen muss das Device geöffnet
   sein, der CAN-Bus muss nicht gestartet sein.
//

printf("Hardware Info:\n\r");
// **** Hardware-Info-Variablen abfragen
if (!(s = CanDrvHwInfo(0)))
  {
  printf("CanDrvHwInfo Error\n\r");
  goto ende;
  }

// Die Funktion get_item_as_string verändert den ursprünglichen String
// darum muss eine Kopie des Strings erstellt werden
info_str = mhs_strdup(s);
s = info_str;
do
  {
  // Bezeichner auslesen
  key = get_item_as_string(&s, ":=", &match);
  if (match <= 0)
    break;
  // Value auslesen
  val = get_item_as_string(&s, ";", &match);
  if (match < 0)
    break;

  printf("%-22s : %s\n\r", key, val);
  }
while(1);
save_free(info_str);  // Allokierten Speicher für den String freigeben

//
  ... beenden ...
//
}
```

3.5.19 CanGetDeviceStatus

Aufgabe Device Status abfragen

Syntax `Int CanGetDeviceStatus(uint32_t index, struct TDeviceStatus *status)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`status`	Zeiger auf Status-Variable, Erläuterung siehe Tabelle.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Abfrage des Device-Status und des CAN-Bus-Status, der mit „index“ angegeben wurde

Aufbau der „TDeviceStatus“ Struktur:

```
struct TDeviceStatus
  {
  int32_t DrvStatus;        // Treiber Status (Device close / Device open / CAN Bus RUN)
  unsigned char CanStatus;  // Status des CAN Controllers (Ok / ... / Error passiv / Bus off)
  unsigned char FifoStatus; // FIFO Status (Ok / ... / Hard. u. Soft. FIFO Überlauf)
  };
```

Interpretation der Variable „status“:

Define	Wert	Beschreibung
Device Status: status->DrvStatus		
DRV_NOT_LOAD	0	Die Treiber DLL wurde noch nicht geladen
DRV_STATUS_NOT_INIT	1	Treiber noch nicht initialisiert (Funktion "CanInitDrv" noch nicht aufgerufen)
DRV_STATUS_INIT	2	Treiber erfolgreich initialisiert
DRV_STATUS_PORT_NOT_OPEN	3	Die Schnittstelle wurde geöffnet
DRV_STATUS_PORT_OPEN	4	Die Schnittstelle wurde nicht geöffnet
DRV_STATUS_DEVICE_FOUND	5	Verbindung zur Hardware wurde hergestellt
DRV_STATUS_CAN_OPEN	6	Device wurde geöffnet und erfolgreich initialisiert
DRV_STATUS_CAN_RUN_TX	7	CAN Bus RUN nur Transmitter (wird nicht verwendet !)
DRV_STATUS_CAN_RUN	8	CAN Bus RUN
CAN-Status: status->CanStatus		
CAN_STATUS_OK	0	CAN-Controller: Ok
CAN_STATUS_ERROR	1	CAN-Controller: CAN Error
CAN_STATUS_WARNING	2	CAN-Controller: Error warning
CAN_STATUS_PASSIV	3	CAN-Controller: Error passiv
CAN_STATUS_BUS_OFF	4	CAN-Controller: Bus Off

CAN_STATUS_UNBEKANNT	5	CAN-Controller: Status Unbekannt
FIFO-Status: status->FifoStatus		
FIFO_OK	0	FIFO-Status: Ok
FIFO_OVERRUN	1	FIFO-Status: Überlauf
FIFO_STATUS_UNBEKANNT	2	FIFO-Status: Unbekannt

Beispiel:

```
// Status abfragen
CanGetDeviceStatus(0, &status);

// BusOff zurücksetzten wenn das Device geöffnet ist
if (status.DrvStatus >= DRV_STATUS_CAN_OPEN)
  {   // CAN Device geöffnet
  if (status.CanStatus == CAN_STATUS_BUS_OFF)
    { // CAN Controller im BusOff
    printf("CAN Status BusOff\n\r");
    CanSetMode(0, OP_CAN_RESET, CAN_CMD_NONE);  // BusOff zurücksetzten
    }
  }
```

3.5.20 CanSetPnPEventCallback

Aufgabe Plug & Play Event-Callback-Funktionen setzen

Syntax `void CanSetPnPEventCallback(void (*event)(unsigned long index, int32_t status))`

Parameter `event` Zeiger auf Funktion

Rückgabewert nichts

Beschreibung Setzt die Plag & Play Event-Callback-Funktion. Damit die Funktion aufgerufen wird, muss sie mit „CanSetEvents“ freigegeben werden und der Treiber muss im Event-Modus arbeiten.

```
// Plug & Play Event-Funktion
static void CALLBACK_TYPE CanPnPEvent(uint32_t index, int32_t status)
{
if (status)
  {
  Online = 1;
  printf(">>> Tiny-CAN Connect\n\r");
  }
else
  {
  Online = 0;
  printf(">>> Tiny-CAN Disconnect\n\r");
  }
}


/*****************************/
/*  Initialisierung          */
/*****************************/
// **** Event Funktionen setzen
CanSetPnPEventCallback(&CanPnPEvent);
// **** Alle Events freigeben
CanSetEvents(EVENT_ENABLE_ALL);

/*
...
   Hauptprogramm
...
*/

/*****************************/
/*  Beenden                  */
/*****************************/
// **** Alle Events sperren
CanSetEvents(EVENT_DISABLE_ALL);
// **** Warten bis alle Event-Funktion zurückgekehrt sind
while (!CanEventStatus())
  {
  }
```

3.5.21 CanSetStatusEventCallback

Aufgabe Status Event-Callback-Funktionen setzen

Syntax `void CanSetStatusEventCallback(void CALLBACK (*event) (uint32_t index, struct TDeviceStatus *DeviceStatus))`

Parameter `event` Zeiger auf Funktion

Rückgabewert nichts

Beschreibung Setzt die Status Event-Callback-Funktion. Damit die Funktion aufgerufen wird, muss sie mit „CanSetEvents“ freigegeben werden und der Treiber muss im Event-Modus arbeiten.

Beispiel:

```
// Status Event-Funktion
static void CALLBACK_TYPE CanStatusEvent(uint32_t index, struct TDeviceStatus *status)
{
printf(">>> Status: %s, %s, %s\n\r", DrvStatusStrings[status->DrvStatus],
      CanStatusStrings[status->CanStatus], CanFifoStrings[status->FifoStatus]);
if (status->DrvStatus >= DRV_STATUS_CAN_OPEN)
  {
  if (status->CanStatus == CAN_STATUS_BUS_OFF)
    {
    printf(">>> CAN Status BusOff clear\n\r");
    CanSetMode(0, OP_CAN_RESET, CAN_CMD_ALL_CLEAR);
    }
  }
}


/*****************************/
/*  Initialisierung          */
/*****************************/
// **** Event Funktionen setzen
CanSetStatusEventCallback(&CanStatusEvent);
// **** Alle Events freigeben
CanSetEvents(EVENT_ENABLE_ALL);

/*
...
   Hauptprogramm
...
*/

/*****************************/
/*  Beenden                  */
/*****************************/
// **** Alle Events sperren
CanSetEvents(EVENT_DISABLE_ALL);
// **** Warten bis alle Event-Funktion zurückgekehrt sind
while (!CanEventStatus())
  {
  }
```

3.5.22 CanSetRxEventCallback

Aufgabe Receive Event-Callback-Funktionen setzen

Syntax `void CanSetRxEventCallback(void CALLBACK (*event) (uint32_t index, struct TCanMsg *msg, int32_t count))`

Parameter `event` Zeiger auf Funktion

Rückgabewert nichts

Beschreibung Setzt die Receive Event-Callback-Funktion. Damit die Funktion aufgerufen wird, muss sie mit „CanSetEvents“ freigegeben werden und Treiber muss im Event-Modus arbeiten.

Beispiel:

```
// RxD Event-Funktion
static void CALLBACK_TYPE CanRxEvent(uint32_t index, struct TCanMsg *msg, int32_t count)
{
struct TCanMsg message;
unsigned long i;

while (CanReceive(0, &message, 1) > 0)
  {
  printf("id:%03lX dlc:%01d data:", message.Id, message.MsgLen);
  if (message.MsgLen)
    {
    for (i = 0; i < message.MsgLen; i++)
      printf("%02X ", message.MsgData[i]);
    }
  else
    printf(" keine");
  printf("\n\r");
  }
}


/*****************************/
/*  Initialisierung          */
/*****************************/
// **** Event Funktionen setzen
CanSetRxEventCallback(&CanRxEvent);
// **** Alle Events freigeben
CanSetEvents(EVENT_ENABLE_ALL);

/*
...
   Hauptprogramm
...
*/

/*****************************/
/*  Beenden                  */
/*****************************/
// **** Alle Events sperren
CanSetEvents(EVENT_DISABLE_ALL);
// **** Warten bis alle Event-Funktion zurückgekehrt sind
while (!CanEventStatus())
  {
  }
```

3.5.23 CanSetEvents

Aufgabe Event-Maske setzen

Syntax `void CanSetEvents(uint16_t events)`

Parameter `events` Event-Maske, siehe Tabelle unten

Rückgabewert nichts

Beschreibung Freigeben und Sperren der Event-Callbackfunktionen. Nur wirksam, wenn sich der Treiber im Event-Modus befindet und die entsprechende Callbackfunktion gesetzt ist.

Define Makro	Beschreibung
EVENT_ENABLE_PNP_CHANGE	Plug & Play Event-Callback-Funktionen freigeben
EVENT_ENABLE_STATUS_CHANGE	Status Event-Callback-Funktionen freigeben
EVENT_ENABLE_RX_FILTER_MESSAGES	Receive Event-Callback-Funktionen für Filter Messages freigeben
EVENT_ENABLE_RX_MESSAGES	Receive Event-Callback-Funktionen für Messages freigeben
EVENT_ENABLE_ALL	Alle Event-Callback-Funktionen freigeben
EVENT_DISABLE_PNP_CHANGE	Plug & Play Event-Callback-Funktionen sperren
EVENT_DISABLE_STATUS_CHANGE	Status Event-Callback-Funktionen sperren
EVENT_DISABLE_RX_FILTER_MESSAGES	Receive Event-Callback-Funktionen für Filter Messages sperren
EVENT_DISABLE_RX_MESSAGES	Receive Event-Callback-Funktionen für Messages sperren
EVENT_DISABLE_ALL	Alle Event-Callback-Funktionen sperren

Beispiel siehe „CanSetPnPEventCallback“, „CanSetStatusEventCallback“ und „CanSetRxEventCallback“.

3.6.1 CanExInitDriver

Aufgabe Initialisiert den Treiber im Ex-API Modus

Syntax `int32_t CanExInitDriver(char *options)`

Parameter

`options`	Übergibt einen Optionen-String an den CAN-Treiber. Aufbau des Strings siehe unten. Die Variablen können nur einmalig bei der Initialisierung festgelegt werden

Rückgabewert Wenn der Treiber erfolgreich initialisiert worden ist, gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Die Funktion „CanExInitDrv“ initialisiert den Treiber im „Ex-API“ (Multi-Devices Support) Modus. Alle Funktionen des Treibers sind erst nach erfolgreicher Initialisierung verfügbar. Einzige Ausnahme ist die Funktion „CanDrvInfo“. Die von der DLL verwendeten Systemressourcen werden erst nach Aufruf von CanExInitDrv belegt.

Aufbau des Option-Strings:

`[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]`

Beispiel:

Empfangsfifo auf 16384 CAN-Messages und Sendefifo auf 16384 CAN-Messages

`CanRxDFifoSize=16384;CanTxDFifoSize=16384`

Bezeichner	Beschreibung	Type	Init.
CanRxDFifoSize	Größe des Empfangsfifos in Messages	ULong	0
CanTxDFifoSize	Größe des Sendefifos in Messages	ULong	2048
CanRxDMode	0 = Die RxD Callbackfunktion übergibt keine CAN-Messages 1 = Die RxD Callbackfunktion übergibt die empfangenen CAN-Messages	UByte	0
CanRxDBufferSize	Größe des Übergabepuffers für RxD Event Proc., nur gültig wenn CanRxDMode = 1.	ULong	50
CanCallThread	0 = Callback Thread nicht erzeugen 1 = Callback Thread erzeugen	UByte	1
CallThreadPriority	0 = THREAD_PRIORITY_NORMAL 1 = THREAD_PRIORITY_ABOVE_NORMAL 2 = THREAD_PRIORITY_HIGHEST 3 = THREAD_PRIORITY_TIME_CRITICAL	ULong	1
LogFile	Dateiname des Log-Files, ein leerer String legt kein Log-File an. Wird der Dateiname ohne Pfad angegeben, so wird der Programmpfad der Applikation benutzt, unter Linux/MacOS das Verzeichnis „/var/log/tiny_can“.	String	„“
LogFlags	Log Flags, siehe Kapitel 7. Log Files	ULong	0

Nicht in der Tabelle aufgeführte Bezeichner werden ignoriert, die Funktion liefert keinen

Fehler zurück.

Beispiel zur Initialisierung im „Ex-Modus“ ohne Callback Unterstützung. Es werden 2 Devices mit eigenem Empfangs-FIFO erzeugt und geöffnet. Der aufgeführte Code ist ein Ausschnitt aus „ex-sample2“.

```
// !!!! Seriennummern an eigene Devices anpassen !!!!
#define DEVICE_OPEN_A "Snr=02000077"
#define DEVICE_OPEN_B "Snr=04000026"


/*
   AUSSCHNITT AUS MAIN
*/
uint32_t device_index_a, device_index_b;

/*******************************/
/*  Initialisierung           */
/*******************************/
device_index_a = INDEX_INVALID;
device_index_b = INDEX_INVALID;

// **** Treiber DLL laden
if ((err = LoadDriver(NULL)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL initialisieren
// Keinen Callback Thread erzeugen, die Callback Funktionen stehen
// nicht zur verfügung
if ((err = CanExInitDriver("CanCallThread=0")) < 0)
  {
  printf("CanInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
/*******************************/
/*  Device A erzeugen u. öffnen */
/*******************************/
// **** Device u. Empfangs-FIFO für das Device erzeugen
if ((err = CanExCreateDevice(&device_index_a, "CanRxDFifoSize=16384")) < 0)
  {
  printf("CanExCreateDevice Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> Tiny-CAN öffnen
if ((err = CanDeviceOpen(device_index_a, DEVICE_OPEN_A)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Übertragungsgeschwindigkeit einstellen
CanSetSpeed(device_index_a, CAN_SPEED);

// Achtung: Um Fehler auf dem Bus zu vermeiden ist die Übertragungsgeschwindigkeit
//          vor dem starten des Busses einzustellen.

// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(device_index_a, OP_CAN_START, CAN_CMD_ALL_CLEAR);
/*******************************/
/*  Device B erzeugen u. öffnen */
/*******************************/
// **** Device u. Empfangs-FIFO für das Device erzeugen
if ((err = CanExCreateDevice(&device_index_b, "CanRxDFifoSize=16384")) < 0)
  {
  printf("CanExCreateDevice Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> Tiny-CAN öffnen
if ((err = CanDeviceOpen(device_index_b, DEVICE_OPEN_B)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
```

```
// **** Übertragungsgeschwindigkeit einstellen
CanSetSpeed(device_index_b, CAN_SPEED);

// Achtung: Um Fehler auf dem Bus zu vermeiden ist die Übertragungsgeschwindigkeit
//          vor dem starten des Busses einzustellen.

// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(device_index_b, OP_CAN_START, CAN_CMD_ALL_CLEAR);
```

Beispiel zur Initialisierung im „Ex-Modus“ ohne Callback Unterstützung mit einem Empfangs-FIFO für alle Devices. Der aufgeführte Code ist ein Ausschnitt aus „ex-sample3“.

```
// !!!! Seriennummern an eigene Devices anpassen !!!!
#define DEVICE_OPEN_A "Snr=02000077"
#define DEVICE_OPEN_B "Snr=04000026"

#define RX_FIFO_INDEX 0x80000000


/*
   AUSSCHNITT AUS MAIN
*/
uint32_t device_index_a, device_index_b;

/*******************************/
/*  Initialisierung           */
/*******************************/
device_index_a = INDEX_INVALID;
device_index_b = INDEX_INVALID;

// **** Initialisierung Utility Funktionen
UtilInit();
// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL initialisieren
// Keinen Callback Thread erzeugen, die Callback Funktionen stehen
// nicht zur verfügung
if ((err = CanExInitDriver("CanCallThread=0")) < 0)
  {
  printf("CanInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
/*******************************/
/*  Device A erzeugen u. öffnen */
/*******************************/
// **** Device erzeugen
if ((err = CanExCreateDevice(&device_index_a, NULL)) < 0)
  {
  printf("CanExCreateDevice Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> Tiny-CAN öffnen
if ((err = CanDeviceOpen(device_index_a, DEVICE_OPEN_A)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Übertragungsgeschwindigkeit einstellen
CanSetSpeed(device_index_a, CAN_SPEED);

// Achtung: Um Fehler auf dem Bus zu vermeiden ist die Übertragungsgeschwindigkeit
//          vor dem starten des Busses einzustellen.

// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(device_index_a, OP_CAN_START, CAN_CMD_ALL_CLEAR);
/*******************************/
/*  Device B erzeugen u. öffnen */
/*******************************/
```

```
// **** Device erzeugen
if ((err = CanExCreateDevice(&device_index_b, NULL)) < 0)
  {
  printf("CanExCreateDevice Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> Tiny-CAN öffnen
if ((err = CanDeviceOpen(device_index_b, DEVICE_OPEN_B)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Übertragungsgeschwindigkeit einstellen
CanSetSpeed(device_index_b, CAN_SPEED);

// Achtung: Um Fehler auf dem Bus zu vermeiden ist die Übertragungsgeschwindigkeit
//          vor dem starten des Busses einzustellen.

// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(device_index_b, OP_CAN_START, CAN_CMD_ALL_CLEAR);

/*************************************************************/
/*  Empfangs FIFO  erzeugen und mit allen Devices verknüpfen */
/*************************************************************/
if ((err = CanExCreateFifo(RX_FIFO_INDEX, 10000, NULL, 0, 0xFFFFFFFF)))
  {
  printf("CanExCreateFifo Error-Code:%d\n\r", err);
  }
```

3.6.2 CanExCreateDevice

Aufgabe Ein Device erzeugen

Syntax
```
int32_t CanExCreateDevice(uint32_t *index, char *options)
```

Parameter

`*index`	Wird auf dem Index-Wert gesetzt unter dem das erzeugte Device angesprochen wird.
`options`	Übergibt einen Optionen-String. Aufbau des Strings siehe unten. Die Variablen können nur einmalig beim Funktionsaufruf festgelegt werden

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Erzeugt ein Device Objekt das Konfiguriert werden kann. Das Device kann nun mit „CanDeviceOpen“ geöffnet werden.
Die Variablen „CanRxDFifoSize“ und „CanTxDFifoSize“ können nur beim Aufruf der Funktion gesetzt werden. Werden die Variablen nicht gesetzt werden die Defaultwerte aus „CanExInitDriver“ verwendet.

Beispiel siehe „CanExInitDriver“.

3.6.3 CanExDestroyDevice

Aufgabe Ein Device löschen

Syntax `int32_t CanExDestroyDevice(uint32_t *index)`

Parameter

`*index`	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Löscht ein Device Objekt. Das Device muss geschlossen sein!

3.6.4 CanExCreateFifo

Aufgabe Empfangs-FIFO anlegen

Syntax
```
int32_t CanExCreateFifo(uint32_t index, uint32_t
size, TMhsEvent *event_obj, uint32_t event,
uint32_t channels)
```

Parameter

index	FIFO Index des zu erzeugenden FIFOs. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
size	Größe des FIFOs in Messages
event_obj	Verknüpft ein „Ereignis-Objekt“ mit den FIFO das mit „CanExCreateEvent“ erzeugt wurde. Wird NULL übergeben wird die Verknüpfung mit den internen Callback System hergestellt, falls der Callback-Thread erzeugt worden ist, „CanCallThread=1“ bei der Treiber-Initialisierung.
event	Gibt das Event Bit an was bei auslösen des Events gesetzt wird. Ist „event_obj“ NULL wird „event“ ignoriert.
channels	Gibt die Devices an mit dem das FIFO verknüpft wird, siehe auch „CanExBindFifo“

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Ein Empfangs-FIFO mit der Größe „size“ erzeugen. Das FIFO kann mit ein oder mehreren Devices verknüpft werden, auch wenn diese noch nicht erzeugt wurden. Verknüpfungen zwischen Devices können auch mit „CanExBindFifo“ erstellt/aufgehoben werden.

Beispiel siehe „CanExInitDriver“.

3.6.5 CanExBindFifo

Aufgabe Empfangsfifo an Device binden

Syntax **`int32_t CanExBindFifo(uint32_t fifo_index, uint32_t device_index, uint32_t bind)`**

Parameter

`fifo_index`	FIFO Index (Ziel), das mit „CanExCreateFifo“ erzeugt wurde.
`device_index`	Device Index (Quelle), es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`bind`	1 = Verbindung herstellen, 0 = Verbindung aufheben.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Das FIFO „fifo_index“ wird an das Device „device_index“ gebunden wenn der wert für bind > 0 ist, bei 0 wird die Verbindung getrennt. Ein „FIFO“ kann mit einer beliebigen Anzahl von Devices verbunden werden. Das Device muss für die Verknüpfung noch nicht erzeugt worden sein.

Beispiel:

```
device_index_a = INDEX_INVALID;
device_index_b = INDEX_INVALID;

// **** Device A erzeugen
if (CanExCreateDevice(&device_index_a, NULL) < 0)
  {
  // Fehler behandeln
  }
// **** Device B erzeugen
if (CanExCreateDevice(&device_index_a, NULL) < 0)
  {
  // Fehler behandeln
  }
// **** FIFO erzeugen
if (CanExCreateFifo(RX_FIFO_INDEX, 10000, NULL, 0, 0x0) < 0)
  {
  // Fehler behandeln
  }
// **** Device A an FIFO binden
if (CanExBindFifo(RX_FIFO_INDEX, device_index_a, 1) < 0)
  {
  // Fehler behandeln
  }
// **** Device B an FIFO binden
if (CanExBindFifo(RX_FIFO_INDEX, device_index_b, 1) < 0)
  {
  // Fehler behandeln
  }
```

3.6.6 CanExCreateEvent

Aufgabe Ein „Event“ Objekt erzeugen

Syntax `TMhsEvent *CanExCreateEvent(void)`

Parameter `keine`

Rückgabewert Das erzeugte „Event“ Objekt, bei Fehler gibt die Funktion NULL zurück.

Beschreibung Ein Event Objekt erzeugen, mit der Funktion „CanExWaitForEvent“ kann auf Ereignisse gewartet werden. Für jeden Thread der „CanExWaitForEvent“ benutzt muss ein eigenes Event Objekt erzeugt werden. Ein Event kann von einen Objekt innerhalb des Treibers das mit der Funktion „CanExSetObjEvent“ verknüpft wurde erzeugt werden. Ein Event kann auch von außen mit der Funktion „CanExSetEvent“ erzeugt und mit der Funktion „CanExResetEvent“ gecanceld werden.

3.6.7 CanExSetObjEvent

Aufgabe Objekt mit Event verknüpfen

Syntax
```
int32_t CanExSetObjEvent(uint32_t index, TMhsEvent
*event_obj, uint32_t event)
```

Parameter

index	Index des zu verknüpfenden Objekts.
event_obj	Event Objekt
event	Event (Ereignis) das vom Objekt ausgelöst wird.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Ein Objekt mit einen Event-Objekt verknüpfen. Ein Objekt kann nur mit einen Event-Objekt verknüpft werden.

3.6.8 CanExSetEvent

Aufgabe Ein Event auslösen

Syntax `void CanExSetEvent(TMhsEvent *event_obj, uint32_t event)`

Parameter

`event_obj`	Event Objekt
`event`	Event (Ereignis) das ausgelöst wird.

Rückgabewert nichts

Beschreibung Einen Event (Ereignis) auslösen.

3.6.9 CanExResetEvent

Aufgabe Einen Event zurücksetzten

Syntax
```
void CanExResetEvent(TMhsEvent *event_obj, uint32_t event)
```

Parameter

`event_obj` Event Objekt

`event` Event (Ereignis) das gelöscht wird.

Rückgabewert nichts

Beschreibung Einen Event löschen. Events werden automatisch beim Verlassen der Funktion „CanExWaitForEvent“ gelöscht.

3.6.10 CanExWaitForEvent

Aufgabe Auf Event(s)/Timeout warten

Syntax `uint32_t CanExWaitForEvent(TMhsEvent *event_obj, uint32_t timeout)`

Parameter

`event_obj`	Event Objekt
`timeout`	Rückkehr der Funktion nach „timeout“ ms, 0 = Timeout Funktion aus.

Rückgabewert Aufgetretene „Events“, 0 = Timeout

Beschreibung Warten auf das Eintreten von Ereignissen oder Timeout. Jedem Event ist ein Bit einer 32 Bit Variable zugeordnet, das letzte Bit (31) ist für das beenden reserviert.

Beispiel, Auszüge aus „ex_sample7“

```
/*****************************************************/
/* Events definieren                                 */
/*****************************************************/
#define RX_EVENT     0x00000001
#define PNP_EVENT    0x00000002
#define STATUS_EVENT 0x00000004
#define CMD_EVENT    0x00000008


/*****************************************************/
/* Selbst definierten Event setzten                  */
/*****************************************************/
static void ProcessOpenClose(int32_t key_idx)
{
// ...
CanExSetEvent(Event, CMD_EVENT);
}


/****************/
/* Event Thread */
/****************/
static DWORD __stdcall thread_execute(void *data)
{
uint32_t event;

do
  {
  event = CanExWaitForEvent(Event, 0);  // Auf Events warten
  if (event & 0x80000000)        // Beenden Event, Thread Schleife verlassen
    break;
  else if (event & RX_EVENT)     // CAN Rx Event
    RxMsg();
  else if (event & PNP_EVENT)    // Pluy &  Play Event
    CanPnPEvent();
  else if (event & STATUS_EVENT) // Event Status Änderung
    CanStatusEvent();
  else if (event & CMD_EVENT)    // Kommando Event
    {
    ProcessCmd(Cmd);
    Cmd = 0;
    }
  }
```

```
while (1);
return(0);
}

/*****************************************************/
/*  Event Thread erzeugen & Events konfigurieren     */
/*****************************************************/
Event = CanExCreateEvent();   // Event Objekt erstellen
// Event Thread erzeugen und starten
Thread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread_execute,
                              NULL, 0, NULL);
// Events mit API Ereignissen verknüfpen
CanExSetObjEvent(0x80000000, MHS_EVS_OBJECT, Event, RX_EVENT);
CanExSetObjEvent(INDEX_INVALID, MHS_EVS_PNP, Event, PNP_EVENT);
for (i = 0; i < 4; i++)
  CanExSetObjEvent(DeviceIndex[i], MHS_EVS_STATUS, Event, STATUS_EVENT);

/*
    Hauptprogramm Loop
*/

/*****************************************************/
/*  Event Thread beenden                             */
/*****************************************************/
if (Thread)
  {
  // Terminate Event setzen
  CanExSetEvent(Event, MHS_TERMINATE);
  // Warten bis der Event Thread beenden
  WaitForSingleObject(Thread, INFINITE);
  // Thread freigeben
  CloseHandle(Thread);
  }
```

3.6.11 CanExGetDeviceCount

Aufgabe Anzahl verbundener Devices abfragen

Syntax `int32_t CanExGetDeviceCount(int flags)`

Parameter

`flags`	0 = Alle FTDI USB-Devices zählen, 1 = Nur Tiny-CAN Devices zählen. **Achtung:** Wenn „flags“ auf 1 gesetzt ist dann werden NUR Tiny-CAN I-XL, Tiny-CAN II-XL, Tiny-CAN IV-XL, Tiny-CAN M1 und neuere Module erkannt!

Rückgabewert Anzahl der verbundenen Devices

Beschreibung Gibt die Anzahl angeschlossener Tiny-CAN Devices zurück.

Beispiel:

```
int main(int argc, char **argv)
{
int32_t err;

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL im extended Mode initialisieren
if ((err = CanExInitDriver(NULL)) < 0)
  {
  printf("CanExInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Anzahl verbundener Devices abfragen
if ((err = CanExGetDeviceCount(0)) < 0)
  {
  printf("CanExGetDeviceCount Error-Code:%d\n\r", err);
  goto ende;
  }
else
  printf("Anzahl Devices: %d\n\r", err);
// **** DLL entladen
ende :
UnloadDriver();
return(0);
}
```

3.6.12 CanExGetDeviceList

Aufgabe Liste verbundener Devices ausgeben

Syntax `int32_t CanExGetDeviceList(struct TCanDevicesList **devices_list, int flags)`

Parameter

`devices_list` Pointer Pointer auf die Devices Liste, die Liste wird von der Funktion dynamisch allokiert und muss mit der Funktion „CanExDataFree“ wieder freigegeben werden.

`flags` 0 = Alle FTDI USB-Devices zählen, 1 = Nur Tiny-CAN Devices zählen
Achtung: Wenn „flags“ auf 1 gesetzt ist dann werden NUR Tiny-CAN I-XL, Tiny-CAN II-XL, Tiny-CAN IV-XL, Tiny-CAN M1 und neuere Module erkannt!

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Gibt eine Liste aller an den PC angeschlossenen Tiny-CAN Devices aus. Die Anzahl der gelieferten Devices kann von der mit „CanExGetDeviceCount“ ermittelten abweichen da die Liste bei jeder Abfrage neu erstellt wird.
Hinweis: Der von der Funktion allokierte Speicher wird nicht automatisch freigegeben.

Aufbau der „TCanDevicesList“ Struktur:

```
/************************************************************/
/* Define Makros                                            */
/************************************************************/
#define CAN_FEATURE_LOM          0x0001  // Silent Mode (LOM = Listen only Mode)
#define CAN_FEATURE_ARD          0x0002  // Automatic Retransmission disable
#define CAN_FEATURE_TX_ACK       0x0004  // TX ACK (Gesendete Nachrichten bestätigen)
#define CAN_FEATURE_HW_TIMESTAMP 0x8000  // Hardware Timestamp

/************************************************************/
/*  Typen                                                   */
/************************************************************/
struct TModulFeatures
  {
  uint32_t CanClock;           // Clock-Frequenz des CAN-Controllers, muss nicht mit
                               // der Clock-Frequenz des Mikrocontrollers übereinstimmen
  uint32_t Flags;              // Unterstützte Features des Moduls:
                               //  Bit  0 -> Silent Mode (LOM = Listen only Mode)
                               //       1 -> Automatic Retransmission disable
                               //       2 -> TX ACK (Gesendete Nachrichten bestätigen)
                               //      15 -> Hardware Timestamp
  uint32_t CanChannelsCount;   // Anzahl der CAN Schnittstellen, reserviert für
                               // zukünftige Module mit mehr als einer Schnittstelle
  uint32_t HwRxFilterCount;    // Anzahl der zur Verfügung stehenden Receive-Filter
  uint32_t HwTxPufferCount;    // Anzahl der zur Verfügung stehenden Transmit Puffer mit
Timer
  };
```

```
struct TCanDevicesList
  {
  uint32_t TCanIdx;       // Ist das Device geöffnet ist der Wert auf dem Device-Index
                          // gesetzt, ansonsten ist der Wert auf "INDEX_INVALID" gesetzt.
  uint32_t HwId;          // Ein 32 Bit Schlüssel der die Hardware eindeutig Identifiziert.
                          // Manche Module müssen erst geöffnet werden damit dieser Wert
                          // gesetzt wird
  char DeviceName[255];   // Nur Linux: entspricht den Device Namen des USB-Devices,
                          //            z.B. /dev/ttyUSB0
  char SerialNumber[16];  // Seriennummer des Moduls
  char Description[64];   // Modul Bezeichnung, z.B. "Tiny-CAN IV-XL"
  struct TModulFeatures ModulFeatures;  // Unterstützte Features des Moduls, nur gültig
                                        // wenn HwId > 0
  };
```

Beispiel:

```
void PrintDevices(void)
{
int32_t i, num_devs;
uint32_t idx;
char *str;
char str_puf[100];
struct TCanDevicesList *l;

if ((num_devs = CanExGetDeviceList(&l, 0)) > 0)  // Device Liste lesen
  {
  for (i = 0; i < num_devs; i++)  // Alle Devices durchlaufen
    {
#ifndef __WIN32__
// Der "DeviceName" ist nur unter Linux gesetzt
    printf("Dev:%s ->", l[i].DeviceName);
#endif
    idx = l[i].TCanIdx;
    if (idx == INDEX_INVALID)  // Device geöffnet ?
      {                        // -> Nein
      printf("%s [%s]\n\r", l[i].Description, l[i].SerialNumber);
      }
    else
      {                        // -> Ja, Device Index mit anzeigen
      printf("%s [%s] Open: 0x%08lX\n\r", l[i].Description, l[i].SerialNumber, idx);
      }
    printf("   Id               : 0x%08lX\n\r", l[i].HwId);
    if (l[i].HwId)             // Nachfolgende Werte nur gültig wenn HwId > 0
      {
      printf("   CanClock         : %u\n\r", l[i].ModulFeatures.CanClock);
      str = str_puf;
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_LOM) // Silent Mode (LOM = Listen only Mode)
        str = mhs_stpcpy(str, "LOM ");
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_ARD) // Automatic Retransmission disable
        str = mhs_stpcpy(str, "ARD ");
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_TX_ACK)
        str = mhs_stpcpy(str, "TX_ACK ");            // TX ACK
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_HW_TIMESTAMP)
        str = mhs_stpcpy(str, "HW_TIMESTAMP ");      // Hardware Time-Stamps
      printf("   Features-Flags   : %s\n\r", str_puf);
      printf("   CanChannelsCount : %u\n\r", l[i].ModulFeatures.CanChannelsCount);
      printf("   HwRxFilterCount  : %u\n\r", l[i].ModulFeatures.HwRxFilterCount);
      printf("   HwTxPufferCount  : %u\n\r", l[i].ModulFeatures.HwTxPufferCount);
      }
    printf("\n\r");
    }
  }
else
  printf("keine Devices gefunden.\n\r");
// WICHTIG: Den allokierten Speicher mit CanExDataFree wieder freigeben
CanExDataFree(&l);
}


/*************************************************************/
/*                        M A I N                            */
/*************************************************************/
int main(int argc, char **argv)
```

```
{
int32_t err;

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL im extended Mode initialisieren
if ((err = CanExInitDriver(NULL)) < 0)
  {
  printf("CanExInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
// ****  Device Liste erzeugen & ausgeben
PrintDevices();
// **** DLL entladen
ende :
UnloadDriver();
return(0);
}
```

Terminal Ausgabe des Beispielprogramms, es sind ein Tiny-CAN II-XL und Tiny-CAN IV-XL angeschlossen.

```
Tiny-CAN II-XL [02000077]
   Id               : 0x43414E42
   CanClock         : 16
   Features-Flags   : LOM TX_ACK
   CanChannelsCount : 1
   HwRxFilterCount  : 4
   HwTxPufferCount  : 4

Tiny-CAN IV-XL [04000026]
   Id               : 0x43414E06
   CanClock         : 16
   Features-Flags   : LOM ARD TX_ACK HW_TIMESTAMP
   CanChannelsCount : 1
   HwRxFilterCount  : 4
   HwTxPufferCount  : 4
```

3.6.13 CanExGetDeviceInfo

Aufgabe Informationen zur Hardware abfragen

Syntax `int32_t CanExGetDeviceInfo(uint32_t index, struct TCanDeviceInfo *device_info, struct TInfoVar **hw_info, uint32_t *hw_info_size)`

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`device_info`	Pointer auf die „TCanDeviceInfo“ Struktur oder NULL um die Device Info Daten nicht abzufragen.
`hw_info`	Pointer auf einen Zeiger der die Hardware Info Variablen enthält, der Speicher wird von der Funktion dynamisch allokiert und muss mit „CanExDataFree“ wieder freigegeben werden. Wird ein NULL übergeben werden die Hardware Info Variablen nicht abgefragt.
`hw_info_size`	Anzahl der Einträge des Arrays in „hw_info“

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Die Funktion fragt die „Device Info“ und/oder die „Hardware Info“ Variablen ab.

Aufbau der „TCanDeviceInfo“ Struktur:

```
struct TCanDeviceInfo
  {
  uint32_t HwId;            // Ein 32 Bit Schlüssel der die Hardware eindeutig Identifiziert.
  uint32_t FirmwareVersion; // Version der Firmware des Tiny-CAN Moduls
  uint32_t FirmwareInfo;    // Informationen zum Stand der Firmware Version
                            //   0 = Unbekannt
                            //   1 = Firmware veraltet, Device kann nicht geöffnet werden
                            //   2 = Firmware veraltet, Funktionsumfang eingeschränkt
                            //   3 = Firmware veraltet, keine Einschränkungen
                            //   4 = Firmware auf Stand
                            //   5 = Firmware neuer als Erwartet
  char SerialNumber[16];    // Seriennummer des Moduls
  char Description[64];     // Modul Bezeichnung, z.B. "Tiny-CAN IV-XL"
  struct TModulFeatures ModulFeatures; // Unterstützte Features des Moduls
  };
```

Struktur „TModulFeatures“ siehe „CanExGetDeviceList“.

Aufbau der „TinfoVar“ Struktur:

```
struct TInfoVar
  {
  uint32_t Key;      // Variablen Schlüssel
  uint32_t Type;     // Variablen Type
  uint32_t Size;     // (Max)Größe der Variable in Byte
  char Data[255];    // Wert der Variable
```

```
  };
```

Beispiel:

```
#define VALUE_TYPE_DESC_SIZE 20

struct TValueTypeDesc
  {
  uint32_t Type;
  const char *Bezeichner;
  };

static const struct TValueTypeDesc ValueTypeDesc[VALUE_TYPE_DESC_SIZE] = {
  {VT_BYTE  , "BYTE"},
  {VT_UBYTE , "UBYTE"},
  {VT_WORD  , "WORD"},
  {VT_UWORD , "UWORD"},
  {VT_LONG  , "LONG"},
  {VT_ULONG , "ULONG"},
  {VT_HBYTE , "HBYTE"},
  {VT_HWORD , "HWORD"},
  {VT_HLONG , "HLONG"},
  {VT_STRING, "STRING"}};


static const char *GetValueTypeString(uint32_t type)
{
int32_t i;
const struct TValueTypeDesc *item;

for (i = 0; i < VALUE_TYPE_DESC_SIZE; i++)
  {
  item = &ValueTypeDesc[i];
  if (item->Type == type)
    return(item->Bezeichner);
  }
return(NULL);
}


static void PrintHwInfo(struct TInfoVar *info, uint32_t size)
{
uint32_t i;

printf(" Key   |  Type  | Size | Value\n\r");
printf("-------+--------+------+-------------------------------\n\r");
for (i = 0; i < size; i++)
  {
  printf("0x%04X | %-6s |  %-3u | ", info[i].Key, GetValueTypeString(info[i].Type),
info[i].Size);

  switch (info[i].Type)
    {
    case VT_BYTE   : {
                     printf("%d\n\r", *((signed char*)(info[i].Data)));
                     break;
                     }
    case VT_UBYTE  : {
                     printf("%u\n\r", *((unsigned char *)(info[i].Data)));
                     break;
                     }
    case VT_HBYTE  : {
                     printf("0x%02X\n\r", *((unsigned char *)(info[i].Data)));
                     break;
                     }
    case VT_WORD   : {
                     printf("%d\n\r", *((signed short *)(info[i].Data)));
                     break;
                     }
    case VT_UWORD  : {
                     printf("%u\n\r", *((uint16_t *)(info[i].Data)));
                     break;
                     }
    case VT_HWORD  : {
```

```
                    printf("0x%04X\n\r", *((uint16_t *)(info[i].Data)));
                    break;
                    }
    case VT_LONG   : {
                    printf("%ld\n\r", *((int32_t *)(info[i].Data)));
                    break;
                    }
    case VT_ULONG  : {
                    printf("%lu\n\r", *((uint32_t *)(info[i].Data)));
                    break;
                    }
    case VT_HLONG  : {
                    printf("0x%08lX\n\r", *((uint32_t *)(info[i].Data)));
                    break;
                    }
    case VT_STRING : {
                    printf("%s\n\r", info[i].Data);
                    break;
                    }
    }
  }
printf("-------+--------+------+------------------------------\n\r\n\r");
}


static void PrintDeviceInfo(struct TCanDeviceInfo *info)
{
char *str;
char str_puf[100];
uint32_t ver, ver2;

printf("Device:%s [Snr:%s]\n\r", info->Description, info->SerialNumber);
printf("   Hardware Id      : 0x%08X\n\r", info->HwId);
ver = info->FirmwareVersion / 1000;
ver2 = info->FirmwareVersion % 1000;
printf("   Firmware Version : %u.%03u\n\r", ver, ver2);
printf("   CanClock         : %u\n\r", info->ModulFeatures.CanClock);
str = str_puf;
if (info->ModulFeatures.Flags & CAN_FEATURE_LOM)      // Silent Mode (LOM = Listen only Mode)
  str = mhs_stpcpy(str, "LOM ");
if (info->ModulFeatures.Flags & CAN_FEATURE_ARD)      // Automatic Retransmission disable
  str = mhs_stpcpy(str, "ARD ");
if (info->ModulFeatures.Flags & CAN_FEATURE_TX_ACK)   // TX ACK
  str = mhs_stpcpy(str, "TX_ACK ");
if (info->ModulFeatures.Flags & CAN_FEATURE_HW_TIMESTAMP)
  str = mhs_stpcpy(str, "HW_TIMESTAMP ");
printf("   Features-Flags   : %s\n\r", str_puf);
printf("   CanChannelsCount : %u\n\r", info->ModulFeatures.CanChannelsCount);
printf("   HwRxFilterCount  : %u\n\r", info->ModulFeatures.HwRxFilterCount);
printf("   HwTxPufferCount  : %u\n\r", info->ModulFeatures.HwTxPufferCount);
printf("\n\r");
}


/************************************************************/
/*                        M A I N                           */
/************************************************************/
int main(int argc, char **argv)
{
int err;
uint32_t device_index;
struct TInfoVar *hw_info;
uint32_t hw_info_size;
struct TCanDeviceInfo device_info;

hw_info = NULL;
//
  ... Initialisierung im EX-Modus ...
//

/**********************************************/
/*  Device u. Hardware Info Variablen abfragen   */
/**********************************************/
if ((err = CanExGetDeviceInfo(device_index, &device_info, &hw_info, &hw_info_size)) < 0)
  {
  printf("CanExGetDeviceInfo Error-Code:%d\n\r", err);
```

```
  goto ende;
  }
PrintDeviceInfo(&device_info);
PrintHwInfo(hw_info, hw_info_size);

// WICHTIG: Den allocierten Speicher mit CanExDataFree wieder freigeben
CanExDataFree(&hw_info);

//
  ... beenden ...
//
}
```

Terminal Ausgabe des Beispielprogramms:

```
Device:TINY-CAN IV-XL [Snr:04000026]
   Hardware Id      : 0x43414E06
   Firmware Version : 1.530
   CanClock         : 16
   Features-Flags   : LOM ARD TX_ACK HW_TIMESTAMP
   CanChannelsCount : 1
   HwRxFilterCount  : 4
   HwTxPufferCount  : 8

 Key   |  Type  | Size | Value
-------+--------+------+-------------------------------
0x1000 | HLONG  |  4   | 0x43414E06
0x1001 | STRING |  15  | Tiny-CAN IV-XL
0x1002 | UWORD  |  2   | 1530
0x1003 | STRING |  5   | 1.53
0x1004 | STRING |  16  | Klaus Demlehner
0x1005 | STRING |  15  | keine Optionen
0x1006 | ULONG  |  4   | 0
0x8000 | UBYTE  |  1   | 1
0x8001 | UBYTE  |  1   | 7
0x8010 | STRING |  9   | TJA1050T
0x8020 | UBYTE  |  1   | 1
0x8030 | UBYTE  |  1   | 0
0x8040 | UBYTE  |  1   | 0
0x8050 | UBYTE  |  1   | 8
0x8060 | UBYTE  |  1   | 4
0x8100 | UBYTE  |  1   | 0
0x8200 | UBYTE  |  1   | 0
0x0000 | HLONG  |  4   | 0x04000026
0x0001 | STRING |  15  | TINY-CAN IV-XL
0x0002 | STRING |  47  | Fujitsu FLASH Bios, Ver. 4.40 - MHS Elektronik
-------+--------+------+-------------------------------
```

3.6.14 CanExDataFree

Aufgabe	Dynamisch allokierte Daten freigeben	
Syntax	`void CanExDataFree(void **data)`	
Parameter	`data`	Pointer Pointer auf Daten, der Daten Pointer wird auf NULL gesetzt.
Rückgabewert	nichts	
Beschreibung	Von der API Dynamisch allokierte Daten wieder freigeben, wie z.B. von „CanExGetDeviceList“. Die Funktion prüft vor Freigabe ob der „Daten Pointer“ ungleich NULL ist, nach Freigabe wird der Pointer auf NULL gesetzt.	

Beispiel: Siehe „CanExGetDeviceList“

3.6.15 CanExSetOptions

Aufgabe Eine Liste von Variablen setzen

Syntax `int32_t CanExSetOptions(uint32_t index, char *options)`

Parameter

`index`	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`options`	Übergibt einen Parameter-String an das mit „index“ angegebene Device. Aufbau des Strings siehe unten.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Setzen von Parametern als Parameter String.

Aufbau des Parameter-Strings:

```
[Bezeichner]=[Wert];[Bezeichner]=[Wert];[..]=[..]
```

Beispiel:

CAN Übertragungsgeschwindigkeit auf 500 kBit/s und Transmit Message Request freigeben.

```
CanSpeed=500;CanTxAckEnable=1
```

3.6.16 CanExSetAsByte

Aufgabe „byte“ (8 bit) Variable setzen

Syntax
```
int32_t CanExSetAsByte(uint32_t index, char *name,
char value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Zu setzender Wert.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „byte“ sein.

3.6.17 CanExSetAsWord

Aufgabe „word“ (16 bit) Variable setzen

Syntax
```
int32_t CanExSetAsWord(uint32_t index, char *name,
int16_t value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Zu setzender Wert.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „word“ sein.

3.6.18 CanExSetAsLong

Aufgabe „long“ (32 bit) Variable setzen

Syntax
```
int32_t CanExSetAsLong(uint32_t index, char *name,
int32_t value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Zu setzender Wert.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „long“ sein.

3.6.19 CanExSetAsUByte

Aufgabe „unsigned byte“ (8 bit) Variable setzen

Syntax
```
int32_t CanExSetAsUByte(uint32_t index, char *name,
unsigned char value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Zu setzender Wert.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „unsigned byte“ sein.

3.6.20 CanExSetAsUWord

Aufgabe „unsigned word“ (16 bit) Variable setzen

Syntax
```
int32_t CanExSetAsUWord(uint32_t index, char *name,
uint16_t value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Zu setzender Wert.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „unsigned word“ sein.

3.6.21 CanExSetAsULong

Aufgabe „unsigned long“ (32 bit) Variable setzen

Syntax
```
int32_t CanExSetAsULong(uint32_t index, char *name,
uint32_t value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Zu setzender Wert.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „unsigned long“ sein.

3.6.22 CanExSetAsString

Aufgabe „String“ Variable setzten

Syntax
```
int32_t CanExSetAsString(uint32_t index, char *name, char *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters.
value	Zu schreibender String.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben setzen, sofern das „schreiben“ erlaubt ist.
Der Type des zu setzenden Parameters muss „String“ sein.

3.6.23 CanExGetAsByte

Aufgabe „byte“ (8 bit) Variable lesen

Syntax
```
int32_t CanExGetAsByte(uint32_t index, char *name,
char *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Pointer auf Variable die den Parameter aufnimmt.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „byte“ sein.

3.6.24 CanExGetAsWord

Aufgabe „word“ (16 bit) Variable lesen

Syntax

```
int32_t CanExGetAsWord(uint32_t index, char *name,
int16_t *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Pointer auf Variable die den Parameter aufnimmt.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „word“ sein.

3.6.25 CanExGetAsLong

Aufgabe „long“ (32 bit) Variable lesen

Syntax
```
int32_t CanExGetAsULong(uint32_t index, char *name,
int32_t *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Pointer auf Variable die den Parameter aufnimmt.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „long“ sein.

3.6.26 CanExGetAsUByte

Aufgabe „unsigned byte“ (8 bit) Variable lesen

Syntax
```
int32_t CanExGetAsUByte(uint32_t index, char *name,
unsigned char *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Pointer auf Variable die den Parameter aufnimmt.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „unsigned byte“ sein.

3.6.27 CanExGetAsUWord

Aufgabe „unsigned word“ (16 bit) Variable lesen

Syntax
```
int32_t CanExGetAsUWord(uint32_t index, char *name,
uint16_t *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Pointer auf Variable die den Parameter aufnimmt.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „unsigned word“ sein.

3.6.28 CanExGetAsULong

Aufgabe „unsigned long“ (32 bit) Variable lesen

Syntax
```
int32_t CanExGetAsULong(uint32_t index, char *name,
uint32_t *value)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters, z.B „CanSpeed“.
value	Pointer auf Variable die den Parameter aufnimmt.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „unsigned long“ sein.

3.6.29 CanExGetAsString

Aufgabe „String“ Variable lesen

Syntax
```
int32_t CanExGetAsString(uint32_t index, char *name, char **str)
```

Parameter

index	Device Index, es werden nur die Bits „CAN Device“ ausgewertet. Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
name	Name des Parameters.
str	Übergibt einen Pointer auf einen von der Funktion übergebenen String. Der String muss mit der Funktion „CanExDataFree“ freigegeben werden.

Rückgabewert Bei fehlerfreier Ausführung gibt die Funktion 0, andernfalls einen „Error-Code“ zurück

Beschreibung Den mit „name“ angegebenen Parameter des Devices, mit „index“ angegeben lesen, sofern das lesen erlaubt ist.
Der Type des zu lesenden Parameters muss „String“ sein.
Die Funktion allokiert einen eigenen String der mit „CanExDataFree“ freigegeben werden muss.

5. Fehler-Codes (Error-Codes)

Error-Code	*Erklärung*
-1	Treiber nicht initialisiert
-2	Es wurden ungültige Parameter-Werte übergeben
-3	Ungültiger Index-Wert
-4	Ungültiger CAN-Kanal
-5	Allgemeiner Fehler
-6	In das FIFO kann nicht geschrieben werden
-7	Der Puffer kann nicht geschrieben werden
-8	Das FIFO kann nicht gelesen werden
-9	Der Puffer kann nicht gelesen werden
-10	Variable nicht gefunden
-11	Lesen der Variable nicht erlaubt
-12	Lesepuffer für Variable zu klein
-13	Schreiben der Variable nicht erlaubt
-14	Der zu schreibende String/Stream ist zu groß
-15	Min Wert unterschritten
-16	Max Wert überschritten
-17	Zugriff verweigert
-18	Ungültige CAN-Speed
-19	Ungültige Baudrate
-20	Wert nicht gesetzt
-21	Keine Verbindung zur Hardware
-22	Kommunikationsfehler zur Hardware
-23	Hardware sendet falsche Anzahl Parameter
-24	Zu wenig Arbeitsspeicher
-25	Das System kann die benötigten Resourcen nicht bereitstellen
-26	Ein System-CALL kehrt mit Fehler zurück
-27	Der Main-Thread ist beschäftigt

6. Parameter

Bezeichner	Beschreibung	Initialisierung
Parameter können nur beim Initialisieren des Treibers gesetzt werden, Aufruf „CanInitDriver“		
CanRxDFifoSize	Größe des Empfangsfifos in Messages	32768
CanTxDFifoSize	Größe des Sendefifos in Messages	2048
CanRxDMode	0 = Die RxD Callbackfunktion übergibt keine CAN-Messages 1 = Die RxD Callbackfunktion übergibt die empfangenen CAN-Messages	0
CanRxDBufferSize	Größe des Übergabepuffers für RxD Event Proc., nur gültig wenn CanRxDMode = 1.	50
CanCallThread	0 = Callback Thread nicht erzeugen 1 = Callback Thread erzeugen	1
MainThreadPriority	0 = THREAD_PRIORITY_NORMAL 1 = THREAD_PRIORITY_ABOVE_NORMAL 2 = THREAD_PRIORITY_HIGHEST 3 = THREAD_PRIORITY_TIME_CRITICAL 4 = THREAD_PRIORITY_REALTIME	3
CallThreadPriority	0 = THREAD_PRIORITY_NORMAL 1 = THREAD_PRIORITY_ABOVE_NORMAL 2 = THREAD_PRIORITY_HIGHEST 3 = THREAD_PRIORITY_TIME_CRITICAL	1
Hardware	Reserviert, sollte nicht gesetzt werden	0
CfgFile	Config File Name, das von der DLL geladen wird. Wird der Dateiname ohne Pfad angegeben, so wird der Pfad der DLL benutzt.	
Section	Name der Section, die im Config File gelesen wird	
LogFile	Dateiname des Log-Files, ein leerer String legt kein Log-File an. Wird der Dateiname ohne Pfad angegeben, so wird der Programmpfad der Applikation benutzt.	„“
LogFlags	Log Flags, siehe Kapitel 7. Log Files	0
TimeStampMode	0 = Disabled 1 = Software Time-Stamps 2 = Hardware Time-Stamps, UNIX-Format 3 = Hardware Time-Stamps 4 = Hardware Time-Stamps verwenden wenn verfügbar, ansonsten Software Time-Stamps	1
Parameter können jederzeit gesetzt werden, Aufruf „CanSetOptions“		
CanTxAckEnable	0 = Transmit Message Request sperren 1 = Transmit Message Request freigeben	0
CanSpeed1	CAN Übertragungsgeschwindigkeit in kBit/s z.B. 100 = 100kBit/s, 1000 = 1MBit/s	125kBit/s
CanSpeed1User	Wert des BTR Register des CAN-Controllers	
AutoConnect	0 = Auto Connect Modus aus 1 = Auto Connect Modus ein	0
AutoReopen	0 = CanDeviceOpen wird nicht automatisch aufgerufen	0

Parameter können jederzeit gesetzt werden, Aufruf „CanSetOptions“		
	1 = CanDeviceOpen wird automatisch aufgerufen, nachdem die Verbindung wieder hergestellt wurde	
MinEventSleepTime	Min. Wartezeit für das wiederholte Aufrufen von Event Callbacks	300
ExecuteCommandTimeout	Maximale Wartezeit für Kommando Ausführung in ms	4000
LowPollIntervall	Hardware Polling Intervall in ms	250
FilterReadIntervall	Filter Messages alle x ms einlesen	1000
Paramter können nur beim „Öffnen“ des Devices gesetzt werden, Aufruf „CanDeviceOpen“		
Port	Serielle Schnittstelle 1 = COM1... (wird für den USB-Bus nicht verwendet)	1
ComDeviceName	Device Name (Linux: /dev/ttyS0)	„“
BaudRate	Baudrate, z.B. 38400 = 38400 Baud*	921600
VendorId	USB-Vendor Id (nur Windows)*	0403
ProductId	USB-Product Id (nur Windows)*	6001
Snr	Seriennummer des CAN Moduls	„“

* = Diese Einstellungen sollten nicht geändert werden!

7. Config-File

Beim Aufruf der Funktion „CanInitDriver“ wird nach der Datei „tiny_can.cfg“ gesucht, alternativ kann über den Parameter „CfgFile“ eine andere Konfigurationsdatei geladen werden. Die Datei wird ohne Pfadangabe unter Windows im Pfad der DLL gesucht und unter Linux/MacOS im Verzeichnis „/etc/tiny_can“.

Der Aufbau von Config-Files entspricht denen von Windows INI Dateien. Sofern nicht über den Parameter „Section“ anders definiert wird die Section „Default“ geladen. Alle in Kapitel 6 beschriebenen Parameter können in der Konfigurationsdatei gesetzt werden.

Beispiel für eine Konfigurationsdatei zum Erzeugen eines „Log-Files“ mit dem Namen „test.log“, alle LogFlags sind gesetzt.

```
[Default]
LogFile = test.log
LogFlags = 0xFFFF
```

8. Log File

Wird beim Aufruf der Funktion „CanInitDriver“ oder im Config-File der Parameter „LogFile“ gesetzt, so erzeugt die DLL ein Log-File. Die Datei wird ohne Pfadangabe unter Windows im Pfad der Applikation angelegt und unter Linux/MacOS im Verzeichnis „/var/log/tiny_can“.
Die Log-File Funktion ist nur für Debug-Zwecke vorgesehen!! Einträge ins Log-File werden immer sofort geschrieben, damit bei einem Programmabsturz alle Einträge bis zuletzt vorhanden sind. Log-Files können mit einen Texteditor geöffnet werden.

Der Parameter „LogFlags“ bestimmt, was ins Log-File geschrieben wird:
Bit

- 0 → Messages vom Treiber, z.B. CAN-Device geöffnet...
- 1 → Änderungen des „DeviceStatus“, Treiber Status, CAN-Bus, FIFOs
- 2 → Empfangene CAN-Messages
- 3 → Gesendete CAN-Messages (ist jedoch keine Bestätigung für den erfolgreichen Versand der Messages)
- 4 → API-Call, z.B. CanSetPnPEventCallback, result: Ok
- 5 → Fehler Meldungen

Beispiel für ein Log-File:

```
API-Call Enter: CanInitDriver, Parameter-Liste:
API-Call Exit: , result: Ok
API-Call: CanSetPnPEventCallback, result: Ok
API-Call: CanSetStatusEventCallback, result: Ok
API-Call: CanSetRxEventCallback, result: Ok
API-Call Enter: CanSetEvents, events: 0XFF
API-Call Exit: CanSetEvents, result: Ok
API-Call Enter: CanDeviceOpen, index: 00000000, Parameter-Liste:
STATUS: Ch 0: Drv=INIT, Can=UNBEKANNT, Fifo=UNBEKANNT
MESSAGE: CAN-Device erfolgreich geöffnet (Port: Baudrate:921600):
API-Call Exit: CanDeviceOpen, result: Ok
API-Call Enter: CanSetSpeed, index: 00000000, speed: 125
STATUS: Ch 0: Drv=CAN_OPEN, Can=UNBEKANNT, Fifo=UNBEKANNT
API-Call Exit: CanSetSpeed, result: Ok
API-Call Enter: CanSetFilter, index: 0X000001
API-Call Exit: CanSetFilter, result: Ok
API-Call Enter: CanSetOptions, Options-Liste:
   AutoConnect = 0
   FilterReadIntervall = 1000
API-Call Exit: CanSetOptions, result: Ok
API-Call Enter: CanSet, index: 00000000, obj_index: 0X03, obj_sub_index: 0000
API-Call Exit: CanSet, result: Ok
API-Call Enter: CanGet, index: 00000000, obj_index: 0X03, obj_sub_index: 0000
API-Call Exit: CanGet, result: Ok
API-Call Enter: CanApplaySettings, index: 00000000
API-Call Exit: CanApplaySettings, result: Ok
API-Call Enter: CanSetMode, index: 00000000, can_op_mode: 0X1, can_command: 0XFFF
API-Call Exit: CanSetMode, result: Ok
API-Call Enter: CanTransmit, index: 00000000, Messages: 1
   STD     |      100 | 5 | 48 41 4C 4C 4F
API-Call Exit: CanTransmit, result: Ok
API-Call Enter: CanTransmit, index: 00000000
API-Call Exit: CanTransmit, result: Ok
API-Call Enter: CanTransmitGetCount, index: 00000000
API-Call Exit: CanTransmitGetCount, count: 0
```

```
API-Call Enter: CanTransmitSet, index: 0X000001, cmd: 0X8000, 100000
STATUS: Ch 0: Drv=CAN_RUN, Can=OK, Fifo=OK
API-Call Exit: CanTransmitSet, result: Ok
API-Call Enter: CanReceive, index: 00000000, count: 1
API-Call Exit: CanReceive, count: 0
API-Call Enter: CanReceiveClear, index: 00000000
API-Call Exit: CanReceiveClear, result: Ok
API-Call Enter: CanReceiveGetCount, index: 00000000
API-Call Exit: CanReceiveGetCount, count: 0
API-Call Enter: CanGetDeviceStatus, index: 00000000
API-Call Exit: CanGetDeviceStatus, result: Ok
API-Call:  CanDrvInfo
   Description = Tiny-CAN API Treiber fuer die Module Tiny-CAN I - III
   Hardware = Tiny-CAN I, Tiny-CAN II, Tiny-CAN III
   Hardware IDs = 0x43414E01, 0x43414E02, 0x43414E03
   Version = 3.01
   Interface Type = USB
   API Version = 1.20
   Autor = Klaus Demlehner
   Homepage = www.mhs-elektronik.de
API-Call Enter: CanDrvHwInfo
Hardware Info Variablen:
   ID = 0X43414E01
   ID String = Tiny-CAN I
   Version = 1210
   Version String = 1.21
   Autor = Klaus Demlehner
   Optionen = keine Optionen
   Snr = 0
   Anzahl CAN Interfaces = 1
   Treiber = PCA82C251T
   Opto = 0
   Term = 0
   HighSpeed = 0
   Anzahl Interval Puffer = 4
   Anzahl Filter = 4
   Anzahl I2C Interfaces = 0
   Anzahl SPI Interfaces = 0
   Hardware Snr = 0
   Hardware ID String = Tiny-CAN I
   Bios ID String = Fujitsu FLASH Bios, Ver. 4.10 - MHS Elektronik
API-Call Exit: CanDrvHwInfo, result: Ok
API-Call Enter: CanDeviceClose, index: 00000000
API-Call Exit: CanDeviceClose, result: Ok
API-Call: CanDownDriver
```

8. Beispiele

8.1 Verwendung der Tiny-CAN API im Polling-Modus

Die Dateien can_drv.h und can_drv.c sind dem Projekt hinzuzufügen.

Das Include File „can_drv.h“ ist jeder Datei hinzuzufügen, die die API verwendet

```
#include "can_drv.h"
```

Die einzelnen Schritte zur Initialisierung

1. Treiber DLL laden, die angegebene Treiber DLL wird dynamisch geladen
2. Treiber DLL initialisieren, Speicher und Systemressourcen werden allokiert
3. Ein Device mit Empfangs-FIFO erzeugen
4. Die Schnittstelle PC/Hardware wird geöffnet
5. Die Übertragungsgeschwindigkeit auf den CAN-Bus wird eingestellt, diese Einstellung muss erfolgen, bevor der Bus gestartet wird
6. Der CAN-Bus wird gestartet

```
// **** 1. Treiber DLL laden
if (LoadDriver(NULL) < 0)
  {
  // Fehler bearbeiten
  }
// **** 2. Treiber DLL initialisieren
if (CanExInitDriver("CanCallThread=0") < 0)
  {
  // Fehler bearbeiten
  }
// **** 3. Device u. Empfangs-FIFO für das Device erzeugen
if (CanExCreateDevice(&device_index, "CanRxDFifoSize=16384") < 0)
  {
  // Fehler bearbeiten
  }
// **** 4. Schnittstelle PC <-> Tiny-CAN öffnen
if (CanDeviceOpen(device_index, NULL) < 0)
  {
  // Fehler bearbeiten
  }
// **** 5. Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_125K_BIT);
// **** 6. CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR);
```

Versand einer CAN-Message:

```
struct TCanMsg msg;

// msg Variable initialisieren
msg.MsgFlags = 0L;   // Alle Flags löschen, Standart Frame Format,
                     // keine RTR, Datenlänge auf 0

//msg.MsgRTR = 1;    // Nachricht als RTR Frame versenden
//msg.MsgEFF = 1;    // Nachricht im EFF (Ext. Frame Format) versenden

msg.Id = 0x100;    // Message Id auf 100 Hex
memcpy(msg.msgdata, "HALLO", 5);

if (CanTransmit(device_index, &msg, 1) < 0)
  {
```

```
  // Fehler bearbeiten
  }
```

Empfang einer CAN-Messages:

```
struct TCanMsg msg;


if (CanReceive(device_index, &msg, 1) > 0)
  {
  printf("id:%03lX dlc:%01d data:", msg.id, msg.flags);
  if (msg.flags)
    {
    for (i = 0; i < msg.flags; i++)
      printf("%02X ", msg.msgdata[i]);
    }
  else
    printf(" keine");
  }
```

Den Device-Status auslesen, bei Bedarf den CAN-Controller resetten

```
struct TDeviceStatus status;


CanGetDeviceStatus(device_index, &status);

if (status.CanStatus == CAN_STATUS_BUS_OFF)
  {
  printf("CAN Status BusOff\n");
  CanSetMode(device_index, OP_CAN_RESET, CAN_CMD_NONE);
  }
```

Programm beenden

```
// **** CAN Bus Stop
CanSetMode(device_index, OP_CAN_STOP, CAN_CMD_NONE);
// **** Schnittstelle PC <-> Tiny-CAN schließen
CanDeviceClose(device_index);
// **** Treiber Ressourcen freigeben
CanDownDriver();
// **** DLL entladen
UnloadDriver();
```

Die Funktionen „CanDownDriver“ wird von der Funktion „UnloadDriver“ automatisch aufgerufen.

Hinweis: Da der Treiber Systemressourcen beansprucht, wird die DLL möglicherweise nicht automatisch vom Betriebssystem entladen, der Aufruf von UnloadDriver ist daher notwendig!

8.2 Quellcode des Demoprogramms „ex_sample1“:

```
/*****************************************************************************/
/*               Tiny-CAN API Demoprogramm "Sample1"                         */
/* ------------------------------------------------------------------------- */
/*  Beschreibung    : - Laden einer Treiber DLL und Initialisierung          */
/*                      im EX-Modus                                          */
/*                    - Initialisierung des CAN-Buses                        */
/*                    - Versand einer CAN-Message                            */
/*                    - Empfang von CAN-Messages                             */
/*                    - Bus-Status abfragen und BusOff löschen               */
/*                                                                           */
/*  Version         : 1.00                                                   */
/*  Datei Name      : main.c                                                 */
/* ------------------------------------------------------------------------- */
```

```
/*  Datum           : 29.09.11                                              */
/*  Autor           : Demlehner Klaus, MHS-Elektronik, 94149 Kößlarn        */
/*                    info@mhs-elektronik.de  www.mhs-elektronik.de         */
/* ------------------------------------------------------------------------ */
/*  Compiler        : GNU C Compiler                                        */
/**************************************************************************/
#include "config.h"
#include "global.h"
#include <string.h>
#include <stdio.h>
#ifdef __WIN32__
#include <conio.h>
#endif
#include "can_drv.h"


/************************************************************/
/*                        M A I N                           */
/************************************************************/
int main(int argc, char **argv)
{
int err;
unsigned long i;
struct TDeviceStatus status;   // Status
struct TCanMsg msg;
uint32_t device_index;

device_index = INDEX_INVALID;
/*****************************/
/*  Initialisierung          */
/*****************************/

// **** Initialisierung Utility Funktionen
UtilInit();
// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL initialisieren
// Keinen Callback Thread erzeugen, die Callback Funktionen stehen
// nicht zur verfügung
if ((err = CanExInitDriver("CanCallThread=0")) < 0)
  {
  printf("CanInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Device u. Empfangs-FIFO für das Device erzeugen
if ((err = CanExCreateDevice(&device_index, "CanRxDFifoSize=16384")) < 0)
  {
  printf("CanExCreateDevice Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> Tiny-CAN öffnen
if ((err = CanDeviceOpen(device_index, DEVICE_OPEN)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
/***************************************/
/*  CAN Speed einstellen & Bus starten   */
/***************************************/
// **** Übertragungsgeschwindigkeit einstellen
CanSetSpeed(device_index, CAN_SPEED);

// Achtung: Um Fehler auf dem Bus zu vermeiden ist die Übertragungsgeschwindigkeit
//          vor dem starten des Busses einzustellen.

// **** CAN Bus Start, alle FIFOs, Filter, Puffer und Fehler löschen
CanSetMode(device_index, OP_CAN_START, CAN_CMD_ALL_CLEAR);

/*****************************/
/*  Message versenden        */
/*****************************/
```

```
// msg Variable Initialisieren
msg.MsgFlags = 0L; // Alle Flags löschen, Stanadrt Frame Format,
                   // keine RTR, Datenlänge auf 0

//msg.MsgRTR = 1;    // Nachricht als RTR Frame versenden
//msg.MsgEFF = 1;    // Nachricht im EFF (Ext. Frame Format) versenden

msg.Id = 0x100;    // Message Id auf 100 Hex
msg.MsgLen = 5;    // Datenlänge auf 5
memcpy(msg.MsgData, "HALLO", 5);
if ((err = CanTransmit(device_index, &msg, 1)) < 0)
  {
  printf("CanTransmit Error-Code:%d\n\r", err);
  goto ende;
  }

printf("Tiny-CAN API Demoprogramm\n\r");
printf("=========================\n\r\n\r");
printf("Empfangene CAN-Messages :\n\r");

while (!KeyHit())
  {
  /*****************************/
  /*  Status abfragen          */
  /*****************************/
  CanGetDeviceStatus(device_index, &status);

  if (status.DrvStatus >= DRV_STATUS_CAN_OPEN)
    {
    if (status.CanStatus == CAN_STATUS_BUS_OFF)
      {
      printf("CAN Status BusOff\n\r");
      CanSetMode(device_index, OP_CAN_RESET, CAN_CMD_NONE);
      }
    }
  else
    {
    printf("CAN Device nicht geöffnet\n\r");
    goto ende;
    }

  if (CanReceive(device_index, &msg, 1) > 0)
    {
    //printf("%10lu.%10lu ", msg.Time.Sec, msg.Time.USec);
    printf("id:%03lX dlc:%01d data:", msg.Id, msg.MsgLen);
    if (msg.MsgLen)
      {
      for (i = 0; i < msg.MsgLen; i++)
        printf("%02X ", msg.MsgData[i]);
      }
    else
      printf(" keine");
    printf("\n\r");
    }
  }

/*****************************/
/*  Treiber beenden          */
/*****************************/
ende :

// Device schließen
(void)CanDeviceClose(device_index);
// Device löschen
(void)CanExDestroyDevice(&device_index);
// CanDownDriver wird auch automatisch von UnloadDriver aufgerufen,
// der separate Aufruf ist nicht zwingend notwendig
CanDownDriver();
// **** DLL entladen
UnloadDriver();

return(0);
}
```

8.3 Verwendung der Tiny-CAN API im Event-Modus

Plug & Play Funktionalität aktivieren

```
CanSetOptions("AutoConnect=1");
```

Wird der Paramteter AutoConnect nicht auf 1 gesetzt, wird nur ein Disconnect der Hardware erkannt.

Die Eventhandler für „Plug & Play, Status und Receive“ setzen und freigeben.

```
// **** Event-Funktionen setzen
CanSetPnPEventCallback(&CanPnPEvent);
CanSetStatusEventCallback(&CanStatusEvent);
CanSetRxEventCallback(&CanRxEvent);
// **** Alle Events freigeben
CanSetEvents(EVENT_ENABLE_ALL);
```

Die Plug & Play Event-Funktion

```
// Plug & Play Event-Funktion
void CALLBACK CanPnPEvent(uint32_t index, int32_t status)
{
// Im Ex-Modus haben die Variablen index und status keine Bedeutung mehr
}
```

Die Status Event-Funktion

```
// Status Event-Funktion
void CALLBACK CanStatusEvent(uint32_t index, struct TDeviceStatus *status)
{
// Die Variable status des mit index angegebenen Devices auswerten
}
```

Die Receive Event-Funktion

```
// RxD Event Funktion
void CALLBACK CanRxEvent(uint32_t index, struct TCanMsg *msg, int32_t count)
{
// Empfangene CAN-Messages des mit index angegebenen Devices verarbeiten
}
```

Wenn der Parameter „CanRxDMode“ auf 1 gesetzt ist, übergibt die Funktion die empfangenen CAN-Messages, andernfalls müssen die Messages mit „CanReceive“ gelesen werden. Der index-Wert wird immer gesetzt.

8.4 Quellcode des Demoprogramms „ex_sample6“:

```
/*************************************************************************/
/*                Tiny-CAN API Demoprogramm "ex_sample2"                 */
/* --------------------------------------------------------------------- */
/*  Beschreibung    : - Laden einer Treiber DLL und Initialisierung      */
/*                      im EX-Modus                                      */
/*                    - Setzen der Callbackfunktionen                    */
/*                    - An- und Abstecken der Hardware, Statusänderungen */
/*                      und den Empfang von CAN-Messages in Callback-    */
/*                      Funktionen verarbeiten                           */
/*                    - Device Liste auf dem Bildschim ausgeben          */
/*                    - Öffnen u. schließen angeschlossener Devices      */
/*                                                                       */
/*  Version         : 1.00                                               */
/*  Datei Name      : main.c                                             */
/* --------------------------------------------------------------------- */
```

```
/*  Datum           : 09.08.11                                              */
/*  Autor           : Demlehner Klaus, MHS-Elektronik, 94149 Kößlarn        */
/*                    info@mhs-elektronik.de  www.mhs-elektronik.de         */
/* ------------------------------------------------------------------------ */
/*  Compiler        : GNU C Compiler                                        */
/**************************************************************************/
#include "config.h"
#include "global.h"
#include <string.h>
#include <stdio.h>
#include "util.h"
#ifdef __WIN32__
#include <conio.h>
#else
#include "linux_util.h"
#endif
#include "can_drv.h"


const char *DrvStatusStrings[] =
     {"NOT_LOAD",      // Die Treiber DLL wurde noch nicht geladen
      "NOT_INIT",      // Treiber noch nicht Initialisiert (Funktion "CanInitDrv" noch nicht
aufgerufen)
      "INIT",          // Treiber erfolgreich Initialisiert
      "PORT_NOT_OPEN", // Die Schnittstelle wurde nicht geöffnet
      "PORT_OPEN",     // Die Schnittstelle wurde geöffnet
      "DEVICE_FOUND",  // Verbindung zur Hardware wurde Hergestellt
      "CAN_OPEN",      // Device wurde geöffnet und erfolgreich Initialisiert
      "CAN_RUN_TX",    // CAN Bus RUN nur Transmitter (wird nicht verwendet !)
      "CAN_RUN"};      // CAN Bus RUN

const char *CanStatusStrings[] =
     {"OK",            // Ok
      "ERROR",         // CAN Error
      "WARNING",       // Error warning
      "PASSIV",        // Error passiv
      "BUS_OFF",       // Bus Off
      "UNBEKANNT"};    // Status Unbekannt

const char *CanFifoStrings[] =
     {"OK",
      "HW_OVERRUN",
      "SW_OVERRUN",
      "HW_SW_OVERRUN",
      "STATUS_UNBEKANNT"};


CRITICAL_SECTION Lock;
uint32_t DeviceIndex[4];
int32_t DevicesListCount = 0;
struct TCanDevicesList *DevicesList = NULL;


void PrintUsbDevices(void)
{
int32_t i;
uint32_t idx;
char *str;
char str_puf[100];
struct TCanDevicesList *l;

l = DevicesList;
if (DevicesListCount)
  {
  printf("================================================================================\n");
  for (i = 0; i < DevicesListCount; i++)
    {
    idx = l[i].TCanIdx;
    if (idx == INDEX_INVALID)
      printf("Key '%d', %s [%s]\n\r", i+1, l[i].Description, l[i].SerialNumber);
    else
      printf("Key '%d', %s [%s] Open: 0x%08lX\n\r", i+1, l[i].Description, l[i].SerialNumber,
idx);
    printf("   Id               : 0x%08lX\n\r", l[i].HwId);
    if (l[i].HwId)
      {
```

```
      printf("   CanClock         : %u\n\r", l[i].ModulFeatures.CanClock);
      str = str_puf;
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_LOM) // Silent Mode (LOM = Listen only Mode)
        str = mhs_stpcpy(str, "LOM ");
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_ARD) // Automatic Retransmission disable
        str = mhs_stpcpy(str, "ARD ");
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_TX_ACK)  // TX ACK
        str = mhs_stpcpy(str, "TX_ACK ");
      if (l[i].ModulFeatures.Flags & CAN_FEATURE_HW_TIMESTAMP)
        str = mhs_stpcpy(str, "HW_TIMESTAMP ");
      printf("   Features-Flags   : %s\n\r", str_puf);
      printf("   CanChannelsCount : %u\n\r", l[i].ModulFeatures.CanChannelsCount);
      printf("   HwRxFilterCount  : %u\n\r", l[i].ModulFeatures.HwRxFilterCount);
      printf("   HwTxPufferCount  : %u\n\r", l[i].ModulFeatures.HwTxPufferCount);
      }
    printf("\n\r");
    }
  printf("================================================================================\n");
  }
else
  printf("keine Devices gefunden.\n");
}


int32_t GetUsbDevices(void)
{
int32_t num_devs;

DevicesListCount = 0;
CanExDataFree(&DevicesList);
if ((num_devs = CanExGetDeviceList(&DevicesList, 0)) > 0)
  DevicesListCount = num_devs;
return(num_devs);
}


// Plug & Play Event-Funktion
void CALLBACK_TYPE CanPnPEvent(uint32_t index, int32_t status)
{
EnterCriticalSection(&Lock);
GetUsbDevices();
PrintUsbDevices();
LeaveCriticalSection(&Lock);
}


// Status Event-Funktion
void CALLBACK_TYPE CanStatusEvent(uint32_t index, struct TDeviceStatus *status)
{
EnterCriticalSection(&Lock);
printf(">>> Status: Drv:%s, Can:%s, FIFO:%s\n\r", DrvStatusStrings[status->DrvStatus],
      CanStatusStrings[status->CanStatus], CanFifoStrings[status->FifoStatus]);
LeaveCriticalSection(&Lock);
}


// RxD Event-Funktion
void CALLBACK_TYPE CanRxEvent(uint32_t index, struct TCanMsg *msg, int32_t count)
{
struct TCanMsg message;
unsigned long i;

EnterCriticalSection(&Lock);
while (CanReceive(0x80000000, &message, 1) > 0)
  {
  printf("[%02X] ", message.MsgSource);
  printf("id:%03lX dlc:%01d data:", message.Id, message.MsgLen);
  if (message.MsgLen)
    {
    for (i = 0; i < message.MsgLen; i++)
      printf("%02X ", message.MsgData[i]);
    }
  else
    printf(" keine");
  printf("\n\r");
  }
```

```
LeaveCriticalSection(&Lock);
}


void ProcessOpenClose(int32_t key_idx)
{
int32_t i, err, open;
uint32_t dev_idx;
char str[30];

EnterCriticalSection(&Lock);
dev_idx = DeviceIndex[key_idx];
// Device in der Liste suchen
open = 0;
for (i = 0; i < DevicesListCount; i++)
  {
  if (dev_idx == DevicesList[i].TCanIdx)
    {
    open = 1;
    break;
    }
  }
if (open)
  {
  (void)CanDeviceClose(dev_idx);
  printf("CanDeviceClose [Key '%d']\n\r", key_idx+1);
  }
else
  {
  if (key_idx < DevicesListCount)
    {
    sprintf(str, "Snr=%s", DevicesList[key_idx].SerialNumber);
    if ((err = CanDeviceOpen(dev_idx, str)) < 0)
      printf("CanDeviceOpen [Key '%d'] Error-Code:%d\n\r", key_idx+1, err);
    else
      {
      printf("CanDeviceOpen [Key '%d'] ok\n\r", key_idx+1);
      // **** CAN Bus Start
      (void)CanSetMode(dev_idx, OP_CAN_START, CAN_CMD_ALL_CLEAR);
      }
    }
  }
LeaveCriticalSection(&Lock);
}


/*************************************************************/
/*                        M A I N                            */
/*************************************************************/
int main(int argc, char **argv)
{
int i, err;
char ch;

/*****************************************************/
/*  Message versenden                                */
/*****************************************************/
printf("Tiny-CAN \"GetDeviceList\" Demo\n\r");
printf("============================\n\r\n\r");

InitializeCriticalSection(&Lock);
/*****************************************************/
/*  Treiber laden & Initialisieren                   */
/*****************************************************/

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL im extended Mode initialisieren
if ((err = CanExInitDriver(NULL)) < 0)
  {
  printf("CanExInitDrv Error-Code:%d\n\r", err);
  goto ende;
```

```
  }
/*****************************************************/
/*  Device Objekte erzeugen                          */
/*****************************************************/
for (i = 0; i < 4; i++)
  {
  if ((err = CanExCreateDevice(&DeviceIndex[i], NULL)))
    printf("CanExCreateDevice [%u] Error-Code:%d\n\r", i, err);
  else
    printf("CanExCreateDevice [%u] Index:0x%08X\n\r", i, DeviceIndex[i]);
  }
/*****************************************************/
/*  Devices konfigurieren                            */
/*****************************************************/
for (i = 0; i < 4; i++)
  {
  // **** Übertragungsgeschwindigkeit auf 1MBit/s einstellen
  if ((err = CanExSetAsUWord(DeviceIndex[i], "CanSpeed1", 1000)) < 0)
    printf("Set \"CanSpeed1\" Fehler: %d\n\r", err);
  // **** 0 = Transmit Message Request sperren
  if ((err = CanExSetAsUByte(DeviceIndex[i], "CanTxAckEnable", 0)) < 0)
    printf("Set \"CanTxAckEnable\" Fehler: %d\n\r", err);
  // **** 3 = Hardware Time-Stamps
  if ((err = CanExSetAsUByte(DeviceIndex[i], "TimeStampMode", 3)) < 0)
    printf("Set \"TimeStampMode\" Fehler: %d\n\r", err);
  }
/*****************************************************/
/*  Empfangs FIFO  erzeugen                          */
/*****************************************************/
if ((err = CanExCreateFifo(0x80000000, 10000, NULL, 0, 0xFFFFFFFF)))
  {
  printf("CanExCreateFifo Error-Code:%d\n\r", err);
  }
else
  printf("CanExCreateFifo ok\n\r");
printf("\n\r");
/*****************************************************/
/*  Device Liste erzeugen & ausgeben                 */
/*****************************************************/
GetUsbDevices();
PrintUsbDevices();

// **** Event Funktionen setzen
CanSetPnPEventCallback(&CanPnPEvent);
CanSetStatusEventCallback(&CanStatusEvent);
CanSetRxEventCallback(&CanRxEvent);
// **** Alle Events freigeben
CanSetEvents(EVENT_ENABLE_ALL);

ch = '\0';
do
  {
  if (KeyHit())
    {
    ch = getch();
    switch (ch)
      {
      case '1' : {
                 ProcessOpenClose(0);
                 break;
                 }
      case '2' : {
                 ProcessOpenClose(1);
                 break;
                 }
      case '3' : {
                 ProcessOpenClose(2);
                 break;
                 }
      case '4' : {
                 ProcessOpenClose(3);
                 break;
                 }
      }
    }
  }
```

```
while (ch != 'q');

/*****************************/
/*  Treiber beenden          */
/*****************************/
ende :

// **** Alle Events sperren
CanSetEvents(EVENT_DISABLE_ALL);
// **** Device Liste löschen
CanExDataFree(&DevicesList);
// **** DLL entladen
UnloadDriver();

DeleteCriticalSection(&Lock);
return(0);
}
```

8.5 Verwendung von Filtern und Sende-Puffern

Einen CAN Sende-Puffer laden und den Intervalltimer auf 100ms setzen

```
#define mS(t) (t * 1000)  // Wandelt ms in us um
struct TCanMsg msg;


if (CanTransmit(1, &msg, 1) < 0)
  {
  // Fehler bearbeiten
  }
// **** Intervalltimer auf 100ms setzten
if (CanTransmitSet(1, 0x8000, mS(100)) < 0)
  {
  // Fehler bearbeiten
  }
```

Einen Message-Filter laden

```
struct TMsgFilter msg_filter;


//         Bit 11     -     Bit0
// Maske   0 1 1 1 1 1 1 1 1 1 0 => 0x3FE
// Code    0 0 0 0 0 0 0 0 0 0 0 => 0x000
// Filter  X 0 0 0 0 0 0 0 0 0 X
// Die CAN Messages 0x000 - 0x001 und 0x400 - 0x4001 werden gefiltert
msg_filter.Maske = 0x3FE;
msg_filter.Code = 0x000;
msg_filter.Flags.Long = 0L;
msg_filter.FilEnable = 1;   // Filter freigeben

if (CanSetFilter(1, &msg_filter) < 0)
  {
  // Fehler bearbeiten
  }
```