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

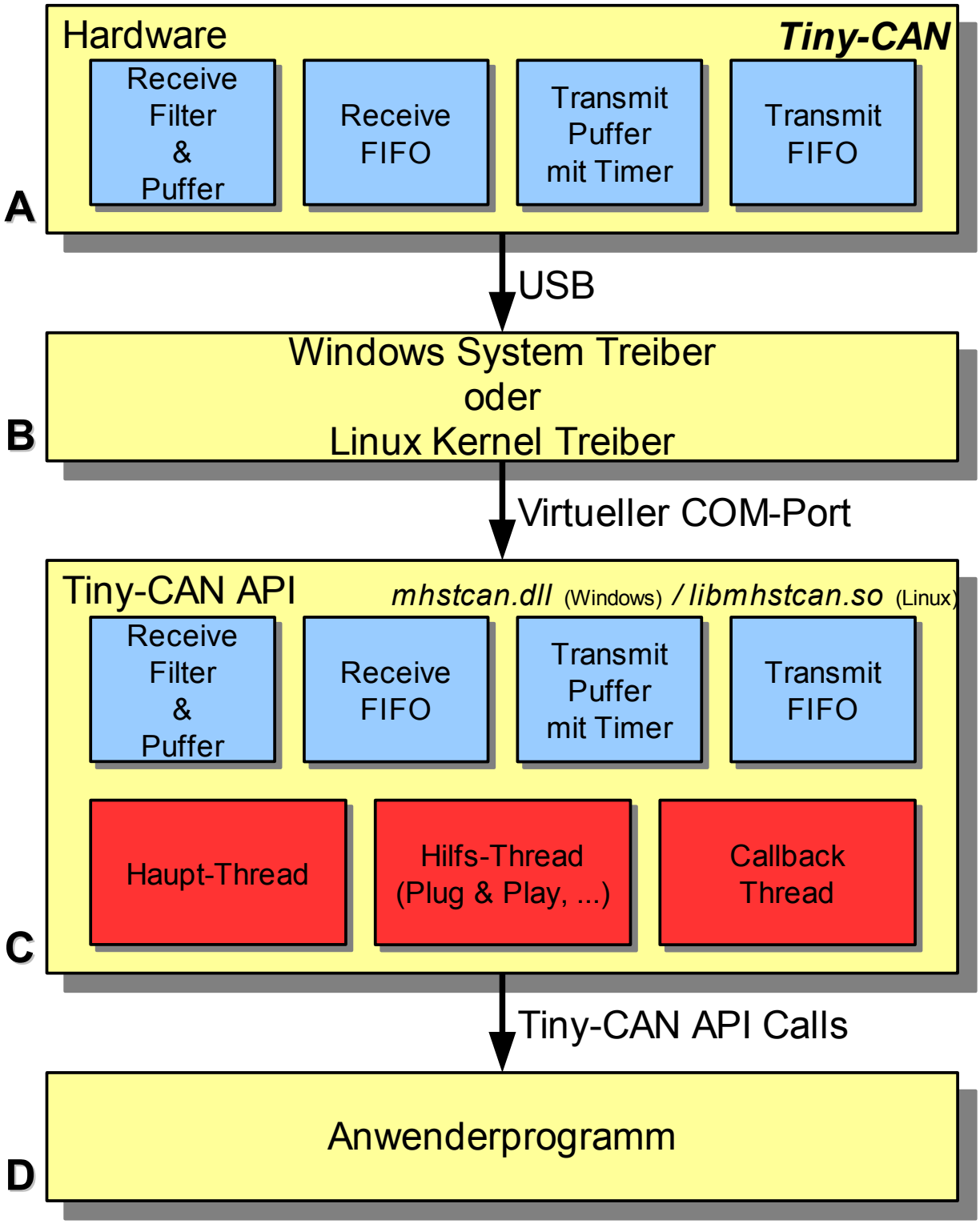
Version: 2.4 vom 06.09.2010

Inhaltsverzeichnis

1. Installation / Hardware

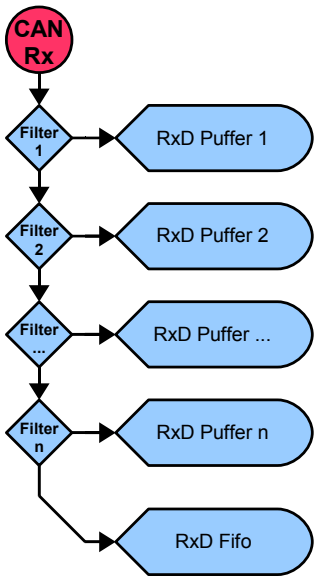
Die Treiber-Installation, eine Beschreibung der Hardware, die Installation von „Tiny-CAN View und Third Party Tools ist im Dokument „TinyCan.pdf“ beschrieben.

2. Von der Hardware bis zum Anwenderprogramm



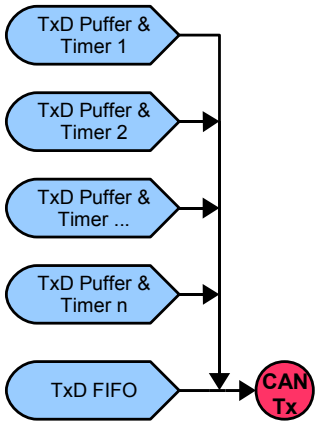
A) Hardware

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.

<table>
<tr><th></th><th>RxD FIFO
[Größe in Messages]</th><th>TxD FIFO
[Größe in Messages]</th><th>Recieve Filter
[Anzahl Filter]</th><th>TxD Puffer mit Intervalltimer
[Anzahl Puffer]</th></tr>
<tr><td>Tiny-CAN I</td><td rowspan="2">110</td><td rowspan="2">36</td><td rowspan="3">4</td><td rowspan="3">4</td></tr>
<tr><td>Tiny-CAN II</td></tr>
<tr><td>Tiny-CAN II-XL</td><td>384</td><td rowspan="5">72</td></tr>
<tr><td>Tiny-CAN III</td><td rowspan="2">512</td><td rowspan="2">8</td><td rowspan="2">16</td></tr>
<tr><td>Tiny-CAN III-XL</td></tr>
<tr><td>Tiny-CAN IV-XL</td><td>900</td><td>4</td><td>8</td></tr>
<tr><td>Tiny-CAN M1</td><td rowspan="2">384</td><td>4</td><td rowspan="2">4</td></tr>
<tr><td>Tiny-CAN M232</td><td>12</td></tr>
</table>

B) Windows System-Treiber oder Linux Kernel-Treiber

Tiny-CAN I - III verwenden als USB-Chip einen USB zu RS232 Konverter. Als Treiber wird der Standard-Treiber des Chip-Herstellers verwendet.

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) anzusprechen. Die Funktionsaufrufe für die Betriebssysteme Windows und Linux sind identisch.

Receive Filter und Puffer

- Es gibt 2 Typen von Filtern: Hardware- und Software-Filter
- Die Funktion der Software-Filter ist mit denen der Hardware-Filter identisch, nur dass die Messages in der Tiny-CAN API gefiltert werden
- Software-Filter werden dynamisch angelegt, es können beliebig viele angelegt werden
- Software-Filter haben mehr Funktionen als die Hardware-Filter

Receive FIFO

- Ein Software-FIFO, dessen Größe frei konfigurierbar ist
- Alle Nachrichten, die sich im Receive FIFO des Tiny-CANs befinden, werden in dieses FIFO übertragen und von der Applikation ausgelesen

Transmit Puffer mit Timer

- Siehe „A) Hardware“, die Tiny-CAN API speichert eine Kopie der Puffer und Timer.

Transmit FIFO

- Ein Software-FIFO, dessen Größe frei konfigurierbar ist
- Ist das Transmit FIFO des Tiny-CAN Moduls zur Hälfte geleert, werden neue Nachrichten des FIFOs zum Senden nachgeladen

Haupt-Thread

- Empfangs-Puffer und FIFO des Tiny-CAN Moduls auslesen
- Sende-FIFO und Puffer ins Tiny-CAN Modul schreiben
- Konfiguration und Einstellungen zum Tiny-CAN Modul übertragen
- Status des CAN Busses vom Tiny-CAN Modul auslesen

Callback Thread

- Vom Haupt-Thread aus werden die Callback-Funktionen getriggert, die dann vom Callback Thread aus aufgerufen werden. Dadurch wird vermieden, dass die Callback-Funktionen den Haupt-Thread blockieren

Hilfs-Thread

- Plug & Play Steuerung, kümmert sich um das Erkennen der Hardware

Übersicht der Tiny-CAN API:

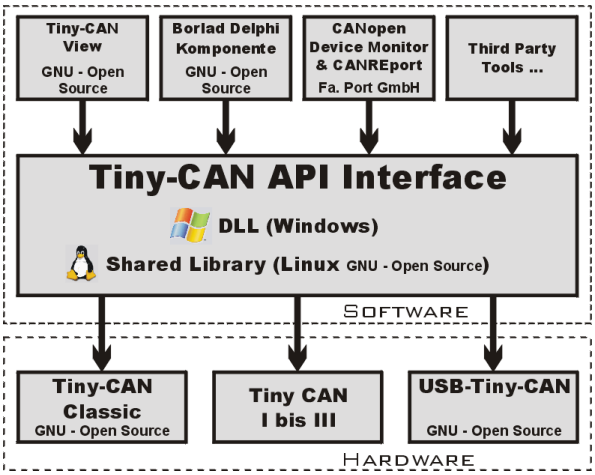


Tabelle welcher Treiber für welche Hardware zu verwenden ist

Hardware	API Treiber	
	Windows	Linux
Tiny-CAN I	mhstcan.dll	libmhstcan.so
Tiny-CAN II		
Tiny-CAN II-XL		
Tiny-CAN III		
Tiny-CAN III-XL		
Tiny-CAN IV-XL		
Tiny-CAN M1		
Tiny-CAN M232		
Abgekündigte Produkte		
Tiny-CAN Classic	mhstcanc.dll	libmhstcanc.so
USB-Tiny-CAN	mhsusbcan.dll	libmhsusbcan.so

D) Anwenderprogramm

- Dem Anwender stehen zahlreiche Applikationen auch von Drittanbietern zur Verfügung.
- Eigene Applikationen können in C, C++ oder Borland Delphi entwickelt werden.

3. Dateien

.../tiny_can/can_api/...

libmhstcan.so	[1]Treiber für Tiny-CAN I – III, Tiny-CAN II-XL, Tiny-CAN III-XL, Tiny-CAN M1 und Tiny-CAN M232 Linux Version
mhstcan.dll	[2]Treiber für Tiny-CAN I – III, Tiny-CAN II-XL, Tiny-CAN III-XL, Tiny-CAN M1 und Tiny-CAN M232 Windows Version
doku	Dokumentation der Tiny-CAN API
└ TinyCanAPI.pdf	Dieses Dokument
lib	Files zum dynamischen Laden eines Tiny-CAN API Treibers
├ 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_win.c	[2]Modul zum dynamischen Laden der dll
└ can_drv_linux.c	[1]Modul zum dynamischen Laden der shared-lib
mhstcan	[2]Files zum statischen linken der mhstcan.dll
├ can_types.h	Allgemeine Definitionen und Datentypen für CAN-Messages, Filter, ...
├ mhstcan.h	Header-File der mhstcan.dll, bindet auch can_types.h mit ein
└ mhstcan.lib	Library
sample	Programmbeispiele
├ 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

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

.../Tiny_CAN/...

delphi	[2]Komponenten und Beispielprogramme für Borland Delphi 6.0
├ 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
└ sample	Beispielprogramm
vb6	[2]Implementierung der Tiny-CAN API in Visual-Basic Version 6.0
python	[1]Implementierung der Tiny-CAN in Python

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

4. Die API

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
Reserviert						Soft	RxD /TxD	CAN Device				CAN Kanal				Sub-Index														

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. Der Parameter wird zur Zeit noch nicht unterstützt, die Bits sind immer 0.
CAN Kanal	Spezifiziert den CAN-Kanal auf einem CAN-Device. Der Parameter wird zur Zeit noch nicht unterstützt, die Bits sind immer 0.
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 = 0L“.

4.3 Daten-Typen

Definitionen in „can_types.h“

```
/****************************************/
/*            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 MsgData Data.Bytes

struct TCanFlagsBits
  {
```

```
  unsigned Len:4;   // Dlc
  unsigned TxD:1;   // TxD -> 1 = Tx CAN Message, 0 = Rx CAN Message
  unsigned Res:1;   // Reserviert
  unsigned RTR:1;   // remote transmition request bit
  unsigned EFF:1;   // extended frame bit
  unsigned Res2:8;
  };

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


struct TMsgFilterFlagsBits
  {
  // 1. Byte
  unsigned Len:4;   // Dlc
  unsigned Res:2;   // Reserviert
  unsigned RTR:1;   // remote transmition request bit
  unsigned EFF:1;   // extended frame bit
  // 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;
  };


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

```
struct TMsgFilter
  {
  uint32_t Maske;
  uint32_t Code;
  union TMsgFilterFlags Flags;
  union TCanData Data;
  };
```

Definitionen in „can_drv.h“

```
/****************************************/
/*             Device Status            */
/****************************************/
struct TDeviceStatus
  {
  int32_t DrvStatus;
  unsigned char CanStatus;
  unsigned char FifoStatus;
  };
```

3.4 Define-Makros

Definitionen in „can_types.h“

```
#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_CAN_KANAL_A      0x00000000
#define INDEX_CAN_KANAL_B      0x00010000
```

Definitionen in „can_drv.h“

```
// 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

// 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
#define DRV_STATUS_INIT           2  // Treiber erfolgreich initialisiert
#define DRV_STATUS_PORT_NOT_OPEN  3  // Die Schnittstelle wurde geöffnet
#define DRV_STATUS_PORT_OPEN      4  // Die Schnittstelle wurde nicht 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
```

3.5 API Funktionsaufrufe

Übersicht:

3.5.1 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.

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	Initialisierung
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

Bezeichner	Beschreibung	Initialisierung
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 das Verzeichnis „/etc/tiny_can“	
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, unter Linux/MacOs.	„“
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	1

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

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.

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	Initialisierung
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 1 = CanDeviceOpen wird automatisch aufgerufen, nachdem die Verbindung wiederhergestellt wurde	0
MinEventSleepTime	Min. Wartezeit für das wiederholte Aufrufen von Event Callbacks in ms	25
ExecuteCommandTimeout	Maximale Wartezeit für Kommando Ausführung in ms	6000
LowPollIntervall	Hardware Polling Intervall in ms	250
FilterReadIntervall	Filter Messages alle x ms einlesen	1000

3.5.4 CanDeviceOpen

Aufgabe Öffnet ein CAN-Device

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

Parameter

`index`	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
`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	Initialisierung
Port	Serielle Schnittstelle 1 = COM1... (wird für den USB-Bus nicht verwendet)	1
ComDeviceName	Device Name (Linux: /dev/ttyUSB0)	„“
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 Die Seriennummer muss auch in das Device programmiert sein!	„“

* = Diese Einstellungen sollten nicht geändert werden!

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.

3.5.6 CanApllaySettings

Aufgabe	Alle Einstellungen zur Hardware übertragen	
Syntax	`int32_t CanApplaySettings(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	Alle Einstellungen Speed, Filter, usw werden zum CAN-Interface übermittelt. Diese Funktion wird bei Bedarf automatisch vom Treiber aufgerufen und nicht näher erläutert	

Dieser Funktionsaufruf ist veraltet, nicht mehr verwenden!

3.5.7 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
```

3.5.8 CanSet

Aufgabe CAN-Device-Variable setzen

Syntax
```
int32_t CanSet(uint32_t index, uint16_t sub_index,
void *data, int32_t size)
```

Parameter

index	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
sub_index	Index der Variable
data	Zeiger auf Datenpuffer
size	Größe des Speichers, auf den „data“ zeigt.

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

Beschreibung Eine CAN-Device-Variable setzen, die CAN-Device-Variablen sind Hardwareabhängig, mehr in Kapitel 2

3.5.9 CanGet

Aufgabe CAN-Device-Variable abfragen

Syntax
```
int32_t CanSet(uint32_t index, uint16_t sub_index,
void *data);
```

Parameter

index	Erläuterungen zum Parameter „index“, Kapitel 4.1, der Parameter „index“
sub_index	Index der Variable
data	Zeiger auf Daten

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

Beschreibung Eine CAN Device Variable abfragen, die CAN-Device-Variablen sind Hardwareabhängig, mehr in Kapitel 2

3.5.10 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 dem mit „index“ angegebenen Fifo/Puffer schreiben.

3.5.11 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	Den mit „index“ angegebenen Fifo/Puffer löschen	

3.5.12 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

3.5.13 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.

3.5.14 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.

3.5.15 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

3.5.16 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

3.5.17 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

3.5.18 CanSetSpeedUser

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

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

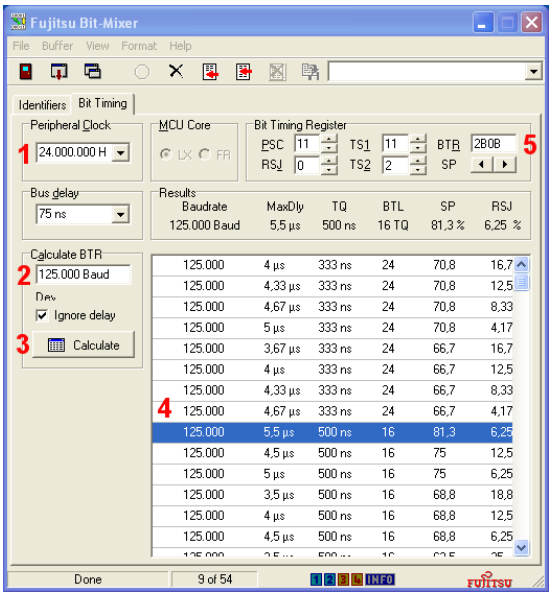
Parameter

`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 „Bit-Mixer“ berechnen:



1. Peripheral Clock auf 24.000.000 (24 MHz) oder 16.000.000 (16MHz) einstellen, siehe Tabelle.

Modul	Clock [MHz]
Tiny-CAN I(B)	24
Tiny-CAN I(B) / 10 kBit	16
Tiny-CAN II	24
Tiny-CAN II / 10 kBit	16
Tiny-CAN II-XL	16
Tiny-CAN III	24
Tiny-CAN III-XL	16
Tiny-CAN M1	16
Tiny-CAN M232	16
Tiny-CAN IX-XL	16

2. Gewünschte Bitrate eingeben, z.B. 125 kBit/s → 125.000 Baud
3. Dem Button „Calculate“ drücken
4. Gewünschten Eintrag in der Tabelle markieren
5. Den BTR-Wert als Hex Zahl ablesen

Ein Beispiel für Benutzerdefinierte CAN Übertragungsraten gibt das „sample6“.

Weitere Informationen, Downloads auf der Internetseite des Controller-Herstellers: mcu.emea.fujitsu.com

3.5.19 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

3.5.20 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
```

3.5.21 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
```

3.5.22 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

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

3.5.23 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.

3.5.24 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.

3.5.25 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.

3.5.26 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

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	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 1 = CanDeviceOpen wird automatisch aufgerufen, nachdem die Verbindung wieder hergestellt wurde	0

Parameter können jederzeit gesetzt werden, Aufruf „CanSetOptions“		
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 Die Seriennummer muss auch in das Device programmiert sein!	„“

* = 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 Path 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 Path 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. Die Schnittstelle PC/Hardware wird geöffnet
4. Die Übertragungsgeschwindigkeit auf den CAN-Bus wird eingestellt, diese Einstellung muss erfolgen, bevor der Bus gestartet wird
5. Der CAN-Bus wird gestartet

```
// **** 1. Treiber DLL laden (Pfad anpassen!)
if (LoadDriver(".\\..\\..\\..\\mhstcan.dll") < 0)
  {
  // Fehler bearbeiten
  }
// **** 2. Treiber DLL initialisieren
if (CanInitDriver(NULL) < 0)
  {
  // Fehler bearbeiten
  }
// **** 3. Schnittstelle PC <-> Tiny-CAN öffnen
if (CanDeviceOpen(NULL) < 0)
  {
  // Fehler bearbeiten
  }
// **** 4. Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_125K_BIT);
// **** 5. 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(0, &msg, 1) < 0)
  {
  // Fehler bearbeiten
  }
```

Empfang einer CAN-Messages:

```
struct TCanMsg msg;


if (CanReceive(0, &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(0, &status);

if (status.CanStatus == CAN_STATUS_BUS_OFF)
  {
  printf("CAN Status BusOff\n");
  CanSetMode(0, OP_CAN_RESET, CAN_CMD_NONE);
  }
```

Programm beenden

```
// **** CAN Bus Stop
CanSetMode(0, OP_CAN_STOP, CAN_CMD_NONE);
// **** Schnittstelle PC <-> Tiny-CAN schließen
CanDeviceClose();
// **** Treiber Recourcen freigeben
CanDownDriver();
// **** DLL entladen
UnloadDriver();
```

Die Funktionen CanSetMode und CanDeviceClose werden von der Funktion UnloadDriver automatisch aufgerufen, daher genügt auch folgender Code zum Beenden des Programms

```


// **** DLL entladen
UnloadDriver();
```

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

8.2 Quellcode des Demoprogramms „Sample1“:

```
/*****************************************************************************/
/*               Tiny-CAN API Demoprogramm "Sample1"                         */
/* ------------------------------------------------------------------------- */
/*  Beschreibung    : - Laden einer Treiber DLL                              */
/*                    - 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           : 01.04.06                                              */
/*  Autor           : Demlehner Klaus, MHS-Elektronik, 94149 Kößlarn        */
/*                    info@mhs-elektronik.de  www.mhs-elektronik.de         */
/* ------------------------------------------------------------------------ */
/*  Compiler        : GNU C Compiler                                        */
/**************************************************************************/
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "..\lib\can_drv.h"


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

/*****************************/
/*  Initialisierung          */
/*****************************/

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n", err);
  goto ende;
  }
// **** Treiber DLL initialisieren
if ((err = CanInitDriver(NULL)) < 0)
  {
  printf("CanInitDrv Error-Code:%d\n", err);
  goto ende;
  }
// **** Schnittstelle PC <-> USB-Tiny öffnen
if ((err = CanDeviceOpen(0, DEVICE_OPEN)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n", err);
  goto ende;
  }
/*****************************/
/*  CAN Speed einstellen     */
/*****************************/
// **** Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_125K_BIT);

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

/*****************************/
/*  Message versenden        */
/*****************************/
// 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;
memcpy(msg.MsgData, "HALLO", 5);

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

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

```
while (!kbhit())
  {
  /*****************************/
  /*  Status abfragen          */
  /*****************************/
  CanGetDeviceStatus(0, &status);

  if (status.DrvStatus >= DRV_STATUS_CAN_OPEN)
    {
    if (status.CanStatus == CAN_STATUS_BUS_OFF)
      {
      printf("CAN Status BusOff\n");
      CanSetMode(0, OP_CAN_RESET, CAN_CMD_NONE);
      }
    }
  else
    {
    printf("CAN Device nicht geöffnet\n");
    goto ende;
    }

  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");
    }
  }

/*****************************/
/*  Treiber beenden          */
/*****************************/
ende :
// **** 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)
{
if (status)
  { // status = 1
  // Die Tiny-CAN Hardware wurde mit dem PC verbunden
  }
else
  { // status = 0
  // Die Tiny-CAN Hardware wurde vom PC getrennt
  }
}
```

Die Status Event-Funktion

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

Die Receive Event-Funktion

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

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

8.4 Quellcode des Demoprogramms „Sample2“:

```
/*****************************************************************************/
/*               Tiny-CAN API Demoprogramm "Sample2"                         */
/* ------------------------------------------------------------------------- */
/*  Beschreibung    : - Laden einer Treiber DLL und Initialisierung des      */
/*                      CAN-Buses                                            */
/*                    - Setzen der Callbackfunktionen                        */
/*                    - An- und Abstecken der Hardware, Statusänderungen     */
/*                      und den Empfang von CAN-Messages in Callback-        */
```

```
/*                     Funktionen verarbeiten                             */
/*                   - Alle 2 Sekunden eine CAN-Message versenden          */
/*                                                                        */
/*  Version         : 1.10                                                */
/*  Datei Name      : main.c                                              */
/* ---------------------------------------------------------------------- */
/*  Datum           : 07.03.07                                            */
/*  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"


const char *DrvStatusStrings[] =
     {"DRV_NOT_LOAD",              // Die Treiber DLL wurde noch nicht geladen
      "DRV_STATUS_NOT_INIT",       // Treiber noch nicht initialisiert (Funktion "CanInitDrv"
noch nicht aufgerufen)
      "DRV_STATUS_INIT",           // Treiber erfolgreich initialisiert
      "DRV_STATUS_PORT_NOT_OPEN", // Die Schnittstelle wurde nicht geöffnet
      "DRV_STATUS_PORT_OPEN",      // Die Schnittstelle wurde geöffnet
      "DRV_STATUS_DEVICE_FOUND",  // Verbindung zur Hardware wurde hergestellt
      "DRV_STATUS_CAN_OPEN",       // Device wurde geöffnet und erfolgreich initialisiert
      "DRV_STATUS_CAN_RUN_TX",     // CAN Bus RUN nur Transmitter (wird nicht verwendet !)
      "DRV_STATUS_CAN_RUN"};       // CAN Bus RUN

const char *CanStatusStrings[] =
     {"CAN_STATUS_OK",             // Ok
      "CAN_STATUS_ERROR",          // CAN Error
      "CAN_STATUS_WARNING",        // Error warning
      "CAN_STATUS_PASSIV",         // Error passiv
      "CAN_STATUS_BUS_OFF",        // Bus Off
      "CAN_STATUS_UNBEKANNT"};     // Status Unbekannt

const char *CanFifoStrings[] =
     {"FIFO_OK",
      "CAN_FIFO_HW_OVERRUN",
      "CAN_FIFO_SW_OVERRUN",
      "CAN_FIFO_HW_SW_OVERRUN",
      "CAN_FIFO_STATUS_UNBEKANNT"};


unsigned char Online;   // Hardware Online


// Plug & Play Event-Funktion
void CALLBACK_TYPE CanPnPEvent(uint32_t index, int32_t status)
{
if (status)
  {
  CanDeviceOpen(0, DEVICE_OPEN);
  // **** CAN Bus Start
  CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR);
  Online = 1;
  printf(">>> Tiny-CAN Connect\n\r");
  }
else
  {
  Online = 0;
  printf(">>> Tiny-CAN Disconnect\n\r");
  }
}


// Status Event-Funktion
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);
    }
  }
}


// RxD Event-Funktion
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");
  }
}


/*************************************************************/
/*                        M A I N                            */
/*************************************************************/
int main(int argc, char **argv)
{
int err;
struct TCanMsg msg;

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

Online = 0;
/*****************************/
/*  Initialisierung          */
/*****************************/

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 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;
  }
/*****************************/
/*  CAN Speed einstellen     */
/*****************************/
// **** Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_125K_BIT);
// **** AutoConnect auf 1
CanSetOptions("AutoConnect=1");
// **** Event Funktionen setzen
CanSetPnPEventCallback(&CanPnPEvent);
CanSetStatusEventCallback(&CanStatusEvent);
CanSetRxEventCallback(&CanRxEvent);
```

```
// **** Alle Events freigeben
CanSetEvents(EVENT_ENABLE_ALL);
// **** Schnittstelle PC <-> Tiny-CAN öffnen
if ((err = CanDeviceOpen(0, DEVICE_OPEN)) < 0)
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
else
{
  if (!CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR))
    Online = 1;
}
// **** CAN Bus Start

while (!KeyHit())
  {
  // msg Variable initialisieren
  msg.MsgFlags = 0L; // Alle Flags löschen, Standart Frame Format,
                     // keine RTR, Datenlänge auf 0
  msg.Id = 0x100;
  msg.MsgLen = 5;
  memcpy(msg.MsgData, "HALLO", 5);

  if (Online)
    CanTransmit(0, &msg, 1);
  Sleep(2000);   // 2 Sekunden warten
  }

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

// **** Alle Events sperren
CanSetEvents(EVENT_DISABLE_ALL);
// **** DLL entladen
UnloadDriver();

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
  }
```

4.6 Quellcode des Demoprogramms „Sample3“:

```
/*****************************************************************************/
/*               Tiny-CAN API Demoprogramm "Sample3"                         */
/* ------------------------------------------------------------------------- */
/*  Beschreibung    : - Laden einer Treiber DLL                              */
/*                    - Initialisierung des CAN-Buses                        */
/*                    - Alle 100ms eine CAN-Message versenden                */
/*                      (Sende Puffer 1)                                     */
/*                    - Setzen eines CAN Filters                             */
/*                    - Empfang von CAN-Messages                             */
/*                    - Bus-Status abfragen und BusOff löschen               */
/*                                                                           */
/*  Version         : 1.10                                                   */
/*  Datei Name      : main.c                                                 */
/* ------------------------------------------------------------------------- */
/*  Datum           : 07.03.07                                               */
/*  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"
```

```
#define mS(t) (t * 1000)  // Wandelt ms in us um


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

/*****************************/
/*  Initialisierung          */
/*****************************/

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 0)
  {
  printf("LoadDriver Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Treiber DLL initialisieren
// Größe des Sende-Fifos auf 10 und des Empfangs-Fifos auf 100
if ((err = CanInitDriver("CanTxDFifoSize=10;CanRxDFifoSize=100")) < 0)
  {
  printf("CanInitDrv Error-Code:%d\n\r", err);
  goto ende;
  }
// **** Schnittstelle PC <-> USB-Tiny öffnen
// COM Port 1 auswählen
if ((err = CanDeviceOpen(0, DEVICE_OPEN)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
/*****************************/
/*  CAN Speed einstellen     */
/*****************************/
// **** Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_SPEED);

// **** CAN Bus Start
CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR);

/*****************************/
/*  Sende Puffer 1 laden     */
/*****************************/
// msg Variable initialisieren
msg.MsgFlags = 0L; // Alle Flags löschen, Standart 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, mS(100))) < 0)
  {
  printf("CanTransmitSet Error-Code:%d\n\r", err);
  goto ende;
  }


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

//         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 - 0x401 werden gefiltert
msg_filter.Maske = 0x3FE;
msg_filter.Code = 0x000;
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;
  }

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

while (!KeyHit())
  {
  /*****************************/
  /*  Status abfragen          */
  /*****************************/
  CanGetDeviceStatus(0, &status);

  if (status.DrvStatus >= DRV_STATUS_CAN_OPEN)
    {
    if (status.CanStatus == CAN_STATUS_BUS_OFF)
      {
      printf("CAN Status BusOff\n\r");
      CanSetMode(0, OP_CAN_RESET, CAN_CMD_NONE);
      }
    }
  else
    {
    printf("CAN Device nicht geöffnet\n\r");
    goto ende;
    }

  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");
    }
  }

/*****************************/
/*  Treiber beenden          */
/*****************************/
ende :
// **** DLL entladen
UnloadDriver();

return(0);
}
```

8.6 Quellcode des Demoprogramms „Sample4“: Verwendung von mehreren Filtern

```
/*****************************************************************************/
/*               Tiny-CAN API Demoprogramm "Sample4"                         */
/* ------------------------------------------------------------------------- */
/*  Beschreibung    : - Laden einer Treiber DLL                              */
```

```
/*                    - Initialisierung des CAN-Buses                    */
/*                    - Setzen mehrerer CAN Filter                       */
/*                    - Empfang von CAN-Messages                         */
/*                    - Bus-Status abfragen und BusOff löschen           */
/*                                                                       */
/*  Version         : 1.10                                               */
/*  Datei Name      : main.c                                             */
/* --------------------------------------------------------------------- */
/*  Datum           : 07.03.07                                           */
/*  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;
struct TMsgFilter msg_filter;

/*****************************/
/*  Initialisierung          */
/*****************************/

// **** Treiber DLL laden
if ((err = LoadDriver(TREIBER_NAME)) < 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
// COM Port 1 auswählen
if ((err = CanDeviceOpen(0, DEVICE_OPEN)) < 0)
  {
  printf("CanDeviceOpen Error-Code:%d\n\r", err);
  goto ende;
  }
/*****************************/
/*  CAN Speed einstellen     */
/*****************************/
// **** Übertragungsgeschwindigkeit auf 125kBit/s einstellen
CanSetSpeed(0, CAN_SPEED);

// **** CAN Bus Start
CanSetMode(0, OP_CAN_START, CAN_CMD_ALL_CLEAR);


/*********************************************************************/
/*               Filter 1 setzen  (Index = 0x00000001L)              */
/* ================================================================= */
/* Type: Hardware Filter, Single Id                                  */
/* CAN Nachrichten mit der ID 0x010 sollen gefiltert werden          */
/*********************************************************************/
msg_filter.FilFlags = 0L;    // Alle Flags mit 0 initialisieren
                             // Dlc = 0, RTR = 0, EFF = 0, Mode = 0
```

```
msg_filter.Code = 0x010;     // ID = 0x10
msg_filter.Maske = 0x000;    // Wird nicht verwendet

msg_filter.FilIdMode = 2;    // 2 = Single Id

msg_filter.FilEnable = 1;    // Filter freigeben

if ((err = CanSetFilter(0x00000001L, &msg_filter)) < 0)  // Filter mit Index 0x00000001L
setzen
  {
  printf("CanSetFilter Error-Code:%d\n\r", err);
  goto ende;
  }


/*********************************************************************/
/*               Filter 2 setzen  (Index = 0x00000002L)              */
/* ================================================================= */
/* Type: Hardware Filter, Maske & Code                               */
/* Die CAN Nachrichten mit der ID 0x000 - 0x001 und 0x400 - 0x401    */
/* sollen gefiltert werden                                           */
/*********************************************************************/
msg_filter.FilFlags = 0L;    // Alle Flags mit 0 initialisieren
                             // Dlc = 0, RTR = 0, EFF = 0, Mode = 0

//         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
//
msg_filter.Code = 0x000;    // Code = 0x000
msg_filter.Maske = 0x3FE;   // Maske = 0x3FE

msg_filter.FilIdMode = 0;   // 0 = Maske & Code

msg_filter.FilEnable = 1;   // Filter freigeben

if ((err = CanSetFilter(0x00000002L, &msg_filter)) < 0)  // Filter mit Index 0x00000002L
setzen
  {
  printf("CanSetFilter Error-Code:%d\n\r", err);
  goto ende;
  }


/*********************************************************************/
/*               Filter 3 setzen  (Index = 0x02000001L)              */
/* ================================================================= */
/* Type: Software Filter, Single Id                                  */
/* CAN Nachrichten mit der ID 0x100 sollen gefiltert werden          */
/*********************************************************************/
msg_filter.FilFlags = 0L;    // Alle Flags mit 0 initialisieren
                             // Len = 0, RTR = 0, EFF = 0, Mode = 0

msg_filter.Code = 0x100;     // ID = 0x100
msg_filter.Maske = 0x000;    // wird nicht verwendet

msg_filter.FilIdMode = 2;    // 2 = Single Id

msg_filter.FilEnable = 1;    // Filter freigeben

if ((err = CanSetFilter(0x00000001L | INDEX_SOFT_FLAG, &msg_filter)) < 0)  // Filter mit
Index 0x02000001L setzen
  {
  printf("CanSetFilter Error-Code:%d\n\r", err);
  goto ende;
  }


/*********************************************************************/
/*                Filter 4 setzen  (Index = 0x02000002L)             */
/* ================================================================= */
/* Type: Software Filter, Start & Stop (Einen Bereich von bis filtern)   */
/* CAN Nachrichten mit der ID ab 0x200 bis 0x215 sollen gefiltert werden */
/*********************************************************************/
```

```
msg_filter.FilFlags = 0L;    // Alle Flags mit 0 initialisieren
                             // Len = 0, RTR = 0, EFF = 0, Mode = 0

msg_filter.Code = 0x200;     // Start ID = 0x200
msg_filter.Maske = 0x215;    // Stop ID = 0x215

msg_filter.FilIdMode = 1;    // 1 = Start & Stop

msg_filter.FilEnable = 1;    // Filter freigeben

if ((err = CanSetFilter(0x00000002L | INDEX_SOFT_FLAG, &msg_filter)) < 0)  // Filter mit
Index 0x02000002L setzen
  {
  printf("CanSetFilter Error-Code:%d\n\r", err);
  goto ende;
  }


/*********************************************************************/
/*               Filter 5 setzen  (Index = 0x02000003L)              */
/* ================================================================= */
/* Type: Software Filter, Single Id                                  */
/* CAN Nachrichten mit der ID 0x100 sollen gefiltert werden und im   */
/* Empfangs-Fifo verbleiben                                          */
/*********************************************************************/
msg_filter.FilFlags = 0L;    // Alle Flags mit 0 initialisieren
                             // Len = 0, RTR = 0, EFF = 0, Mode = 0

msg_filter.Code = 0x100;     // ID = 0x100
msg_filter.Maske = 0x000;    // wird nicht verwendet

msg_filter.FilIdMode = 2;    // 2 = Single Id
msg_filter.FilMode = 1;      // 1 = Die Nachricht wird nicht aus dem Datenstrom gelöscht,
                             //     die Nachricht steht im Puffer und im FIFO

msg_filter.FilEnable = 1;    // Filter freigeben

if ((err = CanSetFilter(0x00000003L | INDEX_SOFT_FLAG, &msg_filter)) < 0)  // Filter mit
Index 0x02000003L setzen
  {
  printf("CanSetFilter Error-Code:%d\n\r", err);
  goto ende;
  }

/*****************************************************************************/
/* Die Anzahl der zur Verfügung stehenden Hardware Filter ist abhängig von der */
/* verwendeten Hardware                                                      */
/*                                                                           */
/* Wird ein Filter mit dem gleichen Index noch einmal geschrieben,           */
/* so wird das Filter überschrieben                                          */
/*                                                                           */
/* Hardware Filter können als "Single Id (IdMode = 2)" und "Maske & Code (IdMode = 0)"*/
/* gesetzt werden, "Start & Stop (IdMode = 1)" ist nicht möglich             */
/* Software Filter können in allen drei Modis gesetzt werden                 */
/* Aufbau der Struktur "TMsgFilter" ist in der Datei "can_types.h" zu finden */
/* Die Felder Len, DLCCheck, DataCheck sind für zukünftige Anwendungen reserviert */
/*****************************************************************************/


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

while (!KeyHit())
  {
  /*****************************/
  /*  Status abfragen          */
  /*****************************/
  CanGetDeviceStatus(0, &status);

  if (status.DrvStatus >= DRV_STATUS_CAN_OPEN)
    {
    if (status.CanStatus == CAN_STATUS_BUS_OFF)
      {
      printf("CAN Status BusOff\n\r");
      CanSetMode(0, OP_CAN_RESET, CAN_CMD_ALL_CLEAR);
```

```
      }
    }
  else
    {
    printf("CAN Device nicht geöffnet\n\r");
    goto ende;
    }

  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");
    }
  }

/*****************************/
/*  Treiber beenden          */
/*****************************/
ende :
// **** DLL entladen
UnloadDriver();

return(0);
}
```