



# ***bitcontrol® devn-can1000***

***QNX® 6.x Device Treiber  
für  
Philips 82C200 / SJA1000  
CAN Controller***

BitCtrl Systems GmbH  
Weißenfelser Str. 67  
04229 Leipzig

2007

***[www.bitctrl.de](http://www.bitctrl.de)***

---

### Copyright und Trademark

Die Software und die Dokumentation des **bitcontrol® devn-can1000** sind Eigentum der BitCtrl Systems GmbH und unterliegen bei ihrer Nutzung der lizenzrechtlichen Vereinbarung zwischen dem Endkunden und der BitCtrl Systems GmbH. Das Kopieren in jeglicher Form sowie die Weitergabe und der Verkauf der Software und der Dokumentation vom Endkunden an Dritte sind streng untersagt.

Die Dokumentation entspricht dem jeweiligen Stand der Entwicklung des **bitcontrol® devn-can1000**. Wenden Sie sich bitte bei Fehlern und Ungenauigkeiten in der Beschreibung an:



BitCtrl Systems GmbH  
Weißenfelder Str. 67  
04229 Leipzig, Germany  
Tel. +49-341-49067 0  
Fax +49-341-49067 15  
Email [info@bitctrl.de](mailto:info@bitctrl.de)

Aufgeführte Marken und Produktnamen sind Marken der jeweiligen Rechtsinhaber. QNX® ist registriertes Warenzeichen der kanadischen Firma QNX® Software Systems Ltd.

### Haftung

Die BitCtrl Systems GmbH (im folgenden BitCtrl genannt) übernimmt für die Software **bitcontrol® devn-can1000** und seine Komponenten weder ausdrückliche noch implizite Garantien. Dies schließt einschränkungslos jegliche Ansprüche hinsichtlich Anwendbarkeit und Eignung der Software für einen bestimmten Nutzungszweck ein. BitCtrl haftet in keiner Weise für zufällige, indirekte oder Folgeschäden, die bei richtiger oder falscher Anwendung der Software entstehen. Dies gilt auch, wenn BitCtrl über die Möglichkeit eines solchen Schadens informiert wurde.

Es gelten die Allgemeinen Geschäftsbedingungen der BitCtrl Systems GmbH. Änderungen der Software bzw. der Dokumentation im Sinne des technischen Fortschritts bleiben vorbehalten.

### Release Stände

Dokumentation Version 1.0	erstellt von: BitCtrl Systems GmbH erreichbar unter: <a href="mailto:info@bitctrl.de">info@bitctrl.de</a> Stand: März 2007 Copyright © 2007 by BitCtrl Systems GmbH - Leipzig, Germany
Für Hinweise auf Tipp- oder Formulierungsfehler wären wir dankbar.	

## Inhaltsverzeichnis

Glossar	4
1 Einleitung	5
2 Systemvoraussetzungen	6
3 Installation	6
4 Start des Treibers	6
5 Optionen	8
5.1 -p Wert	8
5.2 -d Name	8
5.3 -v	8
5.4 -t Wert	8
5.3 -r Wert	8
5.4 -f	8
5.5 -isa[n] Beginn der ISA - Spezifikation [für Kanal n]	8
6 Anleitung zur Programmierung	9
6.1 Telegrammstruktur für Senden und Empfangen von CAN-Nachrichten	9
6.1.1 Liste unterstützter Funktionen für den Ressource-Manager	10
6.1.2 caninfo_s - Struktur	14
6.1.3 canstat_s - Struktur	16
6.1.4 canregs_s - Struktur	16
7 Beispiel	17

## Glossar

<i>Begriff</i>	<i>Erklärung</i>

## 1 Einleitung

Der **bitcontrol® devn-can1000** Device Treiber ist ein QNX® Ressource Manager für BasicCAN (basierend auf Philips 82C200/SJA1000 Stand-alone CAN Controller) für PCI/ISA-CAN Karten<sup>1</sup>.

**Achtung:** Die Version 4.00 ist nicht kompatibel zu den vorangegangenen Versionen 3.xx!

Das CAN- Interface ist als Ressource-Manager im OS QNX® 6.x implementiert. Die Funktionalität des CAN- Ressource-Managers ist vergleichbar mit der eines entsprechenden seriellen Gerätetreibers unter anderen UNIX- Betriebssystemen.

Der CAN- Ressource-Manager ist im QNX® 6.x ein regulärer „user“- Prozess, der sich während der Laufzeit des Systems starten und beenden lässt. Der CAN- Ressource-Manager arbeitet als spezieller Serverprozess mit folgenden Eigenschaften:

- Er bedient Anwendungen (Client-Prozesse) mit typischen Leistungen eines Servers über das Interface der Interprozess-Kommunikation mit Message-Passing (Öffnen, Lesen, Schreiben, Schließen usw.).
- Er fügt sich in den Namensraum für serielle „Gerätetreiber“ ein und unterstützt das im QNX® 6.x übliche Namens-Handling.
- Er ist in der Lage, auf die definierten QNX® 6.x bzw. POSIX Systemaufrufe in Form von Messages zu reagieren und auf diese zu antworten.
- Er unterstützt die typischen POSIX C Funktionen (z.B. read(), write(), u.a.), wobei der Datenbuffer mit einer speziellen Struktur unterlegt ist.

Die Struktur der Namensgebung für den Ressource-Manager im QNX® Filesystem lautet wie folgt:

`/dev/can/<n>`

- `/dev/can` ist der Name des registrierten Ressource-Managers
- `<n>` ist die Nummer des CAN- Controllers im QNX® 6.x System

Der 82C200/SJA1000 - Device-Treiber ist multikanalfähig und unterstützt Interrupt-Sharing. Das bedeutet, dass mehrere Controller dasselbe Interrupt benutzen können.

## 2 Systemvoraussetzungen

Folgende Systemvoraussetzungen müssen zur ordnungsgemäßen Funktion des Treibers erfüllt sein:

- Betriebssystem QNX 6.x
- PCI/ISA-CAN Karte<sup>1</sup>

## 3 Installation

Die Installation des Treibers erfolgt als Superuser (root), mit Hilfe des Programs **qnxinstall** über den Aufruf

```
# /usr/photon/bin/qnxinstall -u dev-can1000-4.0-x86-bitctrl.qpr
```

bzw. mit

```
# /usr/photon/bin/qnxinstall -u http://www.bitctrl.de/repository
```

zur Installation vom „bitctrl.de“ Internet Host.

Folgende Dateien und Verzeichnisse werden installiert (entspricht FHS V.2.2):

/opt/sbin/devn-can1000	Treiber
/opt/bin/caninfo	Info Utility
/opt/include/can.h	API header
/opt/src/bitctrl/can/canping/canping	Anwender – Beispiel
/opt/src/bitctrl/can/caninfo/caninfo	Anwender – Beispiel
/opt/src/bitctrl/can/cansniff/cansniff	Anwender – Beispiel
/opt/src/bitctrl/can/canaccept/canaccept	Anwender – Beispiel
/opt/src/bitctrl/can/canaccept/canstat	Anwender – Beispiel

Folgende Links werden erzeugt:

/usr/sbin/devn-can1000	→	/opt/sbin/devn-can1000
/usr/include/can.h	→	/opt/include/can.h

## 4 Start des Treibers

Zum erfolgreichen Start des Treibers muss folgende Voraussetzung erfüllt sein:

- aktive QNX® - Laufzeitumgebung (Version 6.x).

Die Online-Hilfe zum Start des Treibers kann jederzeit mit Hilfe des Programms **use** angefordert werden.

```
# use devn-can1000
```

```
devn-can1000 [-Options]* [-pci [option[,option ...]]] [-can[n] [option[,option ...]]]
```

oder

```
devn-can1000 [-Options]* [-isa [option[,option ...]]] [-can[n] [option[,option ...]]]
```

<b>Options:</b>	
-p number	priority to run at (default 21, -1 for „don't change priority“)
-d name	device prefix to register (default "/dev/can")
-v	verbose operation (-vv[v..] - debug level)
-t ms	timeout value in 100 ms. (max is 21474836, 0=disable, default=0)
-r value	init can output control register with value (default 0xda)
-f	force driver start even if 82C200 / sj1000 controller is not found

<b>Where:</b>	
-isa[n]	begin a ISA description specification [for channel n].
port=0xXXX	i/o port of the interface. The default is 0x300
irq=d	irq of the interface. The default is 12.
mem=0xXXXX	memory address of the interface. The default is port i/o. (3)
-pci	begin a PCI description specification.
vid=0XXXXXXXX	vendor id. The default value is 0x001C
did=0XXXXXXXX	device id. The default value is 0x0001
-can[n]	begin a Can description specification [for channel n].
baud=dddd	bit rate in kBits (10,20,50,100,125,250,500,800,1000 default=125)
obufs=dddd	number of output buffers (min 2, max ..., default=512)
ibufs=dddd	number of input buffers (min 2, max ..., default=512)
sammod=d	sample mode. 1 - the bus is sampled 3 times, 0 - the bus is sampled once (default - 0)

### Beispiele:

Startet den Kanal "/dev/can/0" mit den Standardeinstellungen:

```
# devn-can1000
```

Startet den Treiber mit der Priorität 20 und einem Timeout-Wert von 10\*100ms = 1sec:

```
# devn-can1000 -p20 -t 10
```

Startet den Treiber für zwei Kanäle:

Erster Kanal /dev/can/0:

- ISA Adresse 0x420,
- irq9
- Konfiguriert mit 512 input und output Buffer

Zweiter Kanal /dev/can/1:

- ISA Adresse 0x520,
- irq10
- Konfiguriert mit 512 input and output Buffer

```
# devn-can1000 -isa0 port=0x420 irq=9 -isa1 port=0x520 irq=10
```

## 5 Optionen

### 5.1 -p Wert

Die -p Option erlaubt die Anpassung der Priorität des Treibers in der Kommandozeile. Normalerweise wird der Treiber mit der Priorität 21 gestartet. Der gültige Wert liegt zwischen 0 und 29. Eine Sonderfunktion hat der Wert -1 „Priorität nicht ändern!“. Er erlaubt die Prioritätssteuerung außerhalb des Treibers, z. B. durch das Programm **nice**. Nach dem Treiberstart `nice -2 devn-can1000 -p -1` läuft der Treiber mit der Priorität 12.

### 5.2 -d Name

Die -d Option wird nur gebraucht, wenn der Standardname `/dev/can` durch andere Treiber oder Applikationen in Benutzung ist.

### 5.3 -v

Die -v Option erlaubt dem Treiber die Ausgabe der Debuginformation auf `'stdout'` und die Ausgabe der Fehlerinformation auf `'stderr'`. Mehrere `'-v'` bestimmen den Debug - Level des Treibers.

### 5.4 -t Wert

Die -t Option bestimmt die Wartezeit beim Lesen aus dem Device bis mindestens ein Telegramm angekommen ist. Ausnahme: Aufruf der Funktion `open()` mit dem Flag `O_NONBLOCK`. Die Wartezeit wird in Vielfachen von 0.1s angegeben. Standardwert ist 0, das bedeutet ewig.

### 5.3 -r Wert

Die -r Option bestimmt den Inhalt des „Output Control Register“ (OCR) während der Initialisierung des CAN-Controllers. Unter anderem definiert dieses Register den Output-Modus des Transmitters (Bi-phase Output Mode, Test Output Mode, Normal Output Mode, Clock Output Mode). Für weitere Informationen wenden Sie sich bitte an den Hardwarehersteller.

### 5.4 -f

Die -f Option (force) führt den Start des Treibers durch, auch wenn der Test auf Existenz des Controllers nicht erfolgreich war.

### 5.5 -isa[n] Beginn der ISA - Spezifikation [für Kanal n]

<b>port</b> =0xXXX	Basis I/O – Adresse des Can-Controllers. Standard – Adresse ist 0x300. Falsche Adressangabe führt zu Abbruch des Treibers (Ausnahme: -f Option).
<b>mem</b> =0xXXXX	Basis Memory – Adresse des Can-Controllers. Standard ist Port I/O. Falsche Adressangabe führt zu Abbruch des Treibers (Ausnahme: -f Option).
<b>Irq</b> =d	IRQ-Nummer. IRQ 2-15 werden unterstützt. Standard - IRQ ist 12.

### 5.6 -pci Beginn der PCI - Spezifikation

<b>vid</b> =0XXXXXXXX	Hersteller Id. Standard Wert ist 0x001C. Falsche Wertangabe führt zu Abbruch des Treibers
<b>did</b> =0XXXXXXXX	Device Id. Standard Wert ist 0x0001. Falsche Wertangabe führt zu Abbruch des Treibers

## 5.7 -can[n] Beginn der CAN - Spezifikation [für Kanal n]

<b>baud</b> =dddd	Baudrate in Kilobits. Gültige Werte sind 10, 20, 50, 100, 125, 250, 500, 800, 1000. Standard - Baudrate ist 125.
<b>obufs</b> =dddd	Größe des Ausgabepuffers; Anzahl der Telegramme. Minimum 2, Standard-Wert ist 512.
<b>ibufs</b> =dddd	Größe des Eingabepuffers; Anzahl der Telegramme. Minimum 2, Standard-Wert ist 512.
<b>sammod</b> =d	Sample Mode : 1 – Bus wird 3 mal abgetastet 0 – Bus wird 1 mal abgetastet

Bei der Parameterangabe dürfen keine Leerzeichen vorkommen, zum Beispiel „baud = 125“ ist nicht erlaubt. Korrekt ist „baud=125“.

Ausgelassene Parameter werden auf Standard Werte eingestellt.

## 6 Anleitung zur Programmierung

Nach dem Treiberstart stehen ein oder mehrere Kanäle zu Verfügung, welche die typischen POSIX C-Funktionen wie *open()*, *read()*, *write()* unterstützen. Kern der Kommunikation zwischen der Anwenderapplikation und dem Treiber ist ein Telegramm (Message), das eine spezielle Struktur unterliegt. Nach der Installation des Treibers findet man diese Struktur in der Header-Datei `/usr/include/can.h`

Jedes Telegramm besteht aus einer bestimmten Anzahl von Daten-Bytes, einem Identifier, der zum einen die Priorität des Telegramms bestimmt, zum anderen ist er auch die logische Adresse des Telegramms. Diese logische Adresse ist 11 (Standard-CAN-Frame) bzw. 29 (Extended-CAN-Frame) Bits lang. Informationen über den Typ des Telegramms werden über ein EXT-Bit (für Extended-CAN-Frame) im Feld 'flags' übertragen<sup>ii</sup>. Der nächste Eintrag ist die Länge des Datenfeldes. Jedes Telegramm kann bis zu 8 Daten-Bytes transportieren, es können aber auch Telegramme mit 0 Bytes Daten-Länge sein. In dem Fall dient der Identifier selbst als der Informationsträger.

RTR-Bit im Feld 'flags': CAN benutzt das so genannte „Broadcast Transmission Protocol“ - d. h., dass alle Teilnehmer am Bus hören können, wie andere sich unterhalten. Es gibt keine direkte Möglichkeit ein Telegramm von einem bestimmten Teilnehmer zu bekommen. Dafür ist ein spezielles Telegramm mit gesetztem RTR-Bit (Remote Transmission Request) erforderlich. Dieses Telegramm fordert den Teilnehmer auf, seine Daten an den Bus zu senden.

### 6.1 Telegrammstruktur für Senden und Empfangen von CAN-Nachrichten

```
typedef struct {
    uint32_t          flags;           /* message flags      */
    uint32_t          reserved;       /* reserved           */
    uint32_t          id;             /* message id         */
    struct timespec   timestamp;     /* receive timestamp  */
    uint16_t          length;        /* message length     */
    uint8_t           data[CAN_MSG_LENGTH]; /* data              */
} canmsg_t;
```

<b>flags -</b>	Flags bestimmen der Kommunikationsstatus und die Kommunikationsart:	
	MSG_RTR -	Remote Transssmission Request Bit

		Anwendung: Senden / Empfangen
	MSG_EXT -	Bit gesetzt: Telegramm unterliegt der CAN 2.0B Spezifikation (Extended-CAN-Frame)  Anwendung: Senden / Empfangen.
	MSG_OVR -	Data Overrun Flag  Bit gesetzt: Mindestens ein Telegramm ist beim Empfang verloren gegangen  Anwendung: nur Empfangen
	MSG_HIP -	High Priority Message  Telegramme dieses Typs haben einen eigenen FIFO-Sendebuffer, der als erstes abgearbeitet wird. Falls noch ein anderes Telegramm im Sendebuffer des Controllers steht, wird der Sendevorgang abgebrochen, um das Senden des HIP-Telegrammes durchzusetzen. Dabei können ein oder mehrere Sendetelegramme verloren gehen. Diese Sorte von Telegrammen wird z. B. bei der Kalibrierung der CAN – Module, die ohne Quarz arbeiten und bei denen der interne Oszillator von außen über den CAN-Bus auf die gewünschte Baudrate eingestellt werden muss, eingesetzt.
	reserved -	Reserviert
	id -	Identifiziert die logische Adresse des Senders / Empfängers und das Arbitrations - Feld. Die niedrigere Adresse hat höhere Priorität.
	timestamp -	Timestamp ist der Zeitstempel des angekommenen Telegramms. Der Treiber setzt die timespec – Struktur – Elemente folgendermaßen (siehe auch QNX Library clock_gettime): <ul style="list-style-type: none"> <li>• tv_sec – Anzahl der Sekunden seit 1970 (GMT)</li> <li>• tv_nsec – Nanosekunden der nächsten Sekunde. Der Wert ist ein Vielfaches von einer Nanosekunde, bezogen auf die Auflösung der System-Uhr.</li> </ul>
	length -	Anzahl der Bytes im Daten-Feld (0 - 8), z.B. CAN_MSG_LENGTH ist 8.
	data	Nutzerdaten zum Senden und Empfangen

### 6.1.1 Liste unterstützter Funktionen für den Ressource-Manager

Die genaue Beschreibung der Funktionen findet man in der C-Library Reference und der POSIX.4 – Dokumentation.

Die Beschreibung der Funktionen beruht auf der Annahme, dass der Treiber mit der Standard –d Option gestartet wurde.

<b>open()</b>	Öffnet den Filedescriptor „/dev/can/<n>“  <i>int open(const char *path, int oflags, ...);</i>
---------------	---

	<p>Unterstützte oflags:</p> <p>O_RDONLY O_WRONLY O_RDWR O_NONBLOCK</p> <p>O_NONBLOCK – hat extra Bedeutung.</p> <p><u>Flag nicht gesetzt:</u> Lesen aus dem Device blockiert den „Client“ um die mit <code>-t &lt; Timeout &gt;</code> definierte Wartezeit, solange kein Telegramm im Eingangsbuffer steht. Ist <code>-t</code> nicht oder auf „0“ gesetzt, wird der Client so lange blockiert, bis ein Telegramm empfangen wurde.</p> <p><u>Flag gesetzt:</u> Der „Client“ bekommt sofort eine Antwort - die geforderte oder kleinere Anzahl von Telegrammen oder -1 wenn kein Telegramm zur Verfügung steht, <b>errno</b> wird auf 11 (Resource temporarily unavailable) gesetzt.</p> <p>Nicht unterstützte Flags:</p> <p>O_APPEND O_CREATE O_EXCL O_NOCTTY O_TRUNC O_DSYNC O_SYNC O_TEMP O_CACHE</p>
<b>close()</b>	Schließt den Filedescriptor für „/dev/can/<n>“
<b>read()</b>	<p>Liest vom CAN-Device „/dev/can/&lt;n&gt;“</p> <p><i>ssize_t read(int fields, void *buffer, size_t len);</i></p> <p>- <b>len</b> muss in Bytes und nicht in Anzahl der Telegramme angegeben werden.</p> <p>- <b>len</b> soll in Vielfach von <i>sizeof(canmsg_t)</i> angegeben werden.</p>
<b>write()</b>	<p>Schreibt auf CAN-Device „/dev/can/&lt;n&gt;“</p> <p><i>ssize_t write(int fields, void *buffer, size_t len);</i></p> <p>- <b>len</b> muss in Bytes und nicht in der Anzahl der Telegramme angegeben werden.</p> <p>- <b>len</b> muss in Vielfachen von <i>sizeof(canmsg_t)</i> angegeben werden, sonst kehrt <i>write()</i> mit <code>-1</code> zurück, <b>errno</b> wird auf 22 (Invalid argument) gesetzt.</p>
<b>lseek()</b>	Nicht unterstützt
<b>fpathconf()</b>	Nicht unterstützt
<b>dup()</b>	Descriptor duplizieren

<b>opendir()</b>	Nicht unterstützt
<b>fstat()</b>	Filedescriptor Status
<b>chmod()</b>	Rechte zuweisen
<b>chown()</b>	Eigentümer zuweisen
<b>utime()</b>	Nicht unterstützt
<b>fcntl()</b>	File-Steuerung
<b>readdir()</b>	Nicht unterstützt
<b>rewinddir()</b>	Nicht unterstützt
<b>stat()</b>	Status
<b>ionotify()</b>	<p>Ereignisüberwachung für ein Device</p> <p><i>int ionotify(int fd, int action, int flags, const struct sigevent *event)</i></p> <p>Unterstützte Flags:</p> <p style="text-align: center;">_NOTIFY_COND_INPUT, _NOTIFY_COND_OUTPUT</p>
<b>select()</b>	Nicht unterstützt

<b>ioctl()</b>	<p>IO-Steuerung laut POSIX.4 Standard:</p> <p style="text-align: center;"><i>int ioctl (int fd, long cmd, void *buf),</i></p> <ul style="list-style-type: none"> <li>- Unterstützung der Funktion <i>ioctl()</i> ist vom Hersteller abhängig</li> <li>- <i>ioctl()</i> erlaubt der Applikation nur das zu machen, was der Treiber unterstützt.</li> </ul> <p>devn-can1000 Treiber unterstützt den Aufruf von <i>ioctl()</i> mit folgenden Parametern:</p> <p style="text-align: center;"><u>Information über den Device:</u></p> <p style="text-align: center;"><i>int ioctl (int fd, CNGETA, void *buf),</i></p> <p>wobei</p> <p>fd      - Descriptor cmd     - CNGETA Kommando "Get Can Attributs" buf     - Zeiger auf die Struktur caninfo_s;</p> <p style="text-align: center;"><u>Device anhalten:</u></p> <p style="text-align: center;"><i>int ioctl (int fd, CNSTOP)</i></p> <p>wobei</p> <p>fd      - Descriptor cmd     - CNSTOP Kommando "Can Stop"</p>
----------------	---

	<p><u>Device starten:</u></p> <p><i>int ioctl (int fd, CNSTART)</i></p> <p>wobei</p> <p>fd - Descriptor cmd - CNSTART Kommando "Can Start"</p> <p><u>Akzeptanz Code für Device setzen:</u></p> <p><i>int ioctl (int fd, CNSETACC, int *code)</i></p> <p>wobei</p> <p>fd - Descriptor cmd - CNSETACC Kommando "Set Acceptance Code" code - Zeiger auf Akzeptanz Code (8 bit MSB)</p> <p><u>Akzeptanz Maske für Device setzen:</u></p> <p><i>int ioctl (int fd, CNSETACM, int *mask)</i></p> <p>wobei</p> <p>fd - Descriptor cmd - CNSETACM Kommando "Set Acceptance Mask" mask - Zeiger auf Akzeptanz Mask (8 bit MSB)</p> <p><u>Lese- und Schreibebuffer von Device leeren:</u></p> <p><i>int ioctl (int fd, CNFLUSH)</i></p> <p>wobei</p> <p>fd - Descriptor cmd - CNFLUSH Kommando "Flush Device"</p>
	<p><u>Information über den aktuellen Device Status:</u></p> <p><i>int ioctl (int fd, CNGETSTAT, void *buf),</i></p> <p>wobei</p> <p>fd - Descriptor cmd - CNGETSTAT Kommando "Get Can State" buf - Zeiger auf die Struktur canstat_s;</p> <p><u>Information über den aktuellen Registerzustand:</u></p> <p><i>int ioctl (int fd, CNGETR, void *buf),</i></p> <p>wobei</p> <p>fd - Descriptor cmd - CNGETR Kommando "Get Can Registers" buf - Zeiger auf die Struktur canregs_s;</p>
<b>devctl()</b>	Device-Steuerung

Die Struktur und die Kommando(s) für ioctl() sind in der Header-Datei /usr/include/can.h deklariert.

## 6.1.2 caninfo\_s - Struktur

```
typedef struct caninfo_s {
    uint32_t      chn;           /* channel number          */
    uint32_t      iobase;       /* i/o base addr           */
    uint32_t      irq;          /* hardware interrupt      */
    uint32_t      baud;         /* baud rate               */
    uint32_t      ocr;          /* output control register */
    uint32_t      tov;         /* timeout in 100ms       */
    uint32_t      tx_ok;        /* total packets Txd OK   */
    uint32_t      rx_ok;        /* total packets Rxd OK   */
    uint32_t      tx_bad;       /* total packets Txd Bad  */
    uint32_t      rx_err;       /* total Rx errors         */
    uint32_t      rx_sovr;      /* Software FIFO overruns during Rx */
    uint32_t      rx_hovr;      /* Hardware FIFO overruns during Rx */
    uint32_t      er_int;       /* error interrupts        */
    uint32_t      er_bus;       /* bus errors              */
    char          driver_name   [ INFO_NAME_MAX ]; /* driver name */
    char          process_name [ INFO_NAME_MAX ]; /* process name */
    char          driver_vers   [ INFO_NAME_MAX ]; /* driver version */
    uint32_t      driver_pid;    /* driver pid */
    char          card_name     [ INFO_NAME_MAX ]; /* card name */
    char          contr_name    [ INFO_NAME_MAX ]; /* controller */
} caninfo_t;
```

uint32_t	chn;	Kanalnummer.
uint32_t	iobase;	Base I/O-Adresse des CAN- Controllers
uint32_t	irq;	Hardware - Interrupt (IRQ - Nummer)
uint32_t	baud;	Baudrate
uint32_t	ocr;	Inhalt des OCRs (Output Control Register)
uint32_t	toV;	Wartezeit in 100ms (timeout value)
uint32_t	tx_ok;	Insgesamt erfolgreich gesendete Telegramme  <i>tx_ok</i> wird in der Laufzeit hoch gezählt, wenn ein Telegramm durch den Can-Controller abgeschickt und mindestens von einem Bus – Teilnehmer als fehlerfrei signiert wurde.
uint32_t	rx_ok;	Insgesamt erfolgreich empfangene Telegramme  <i>rx_ok</i> wird in der Laufzeit hoch gezählt, wenn ein Telegramm durch Can-Controller empfangen und vom Treiber abgeholt wurde.
uint32_t	tx_bad;	Insgesamt verlorene Telegramme beim Senden  <i>tx_bad</i> wird in der Laufzeit hoch gezählt, wenn ein Sendevorgang durch

		den Treiber abgebrochen wird.  (Siehe dazu „flag MSG_HIP“, Kapitel „Anleitung zur Programmierung“)
uint32_t	rx_err;	gesamte Anzahl aufgetretene Fehler während des Empfangs  $rx\_err = rx\_sovr + rx\_hovr$
uint32_t	rx_sovr;	Software FIFO Überlauf während des Empfangs  $rx\_sovr$ wird in der Laufzeit hochgezählt, wenn ein Telegramm durch den Can-Controller empfangen wurde, aber durch den Treiber nicht abgespeichert werden konnte, weil Empfangsbuffer voll ist.  (Siehe dazu Parameter <b>ibufs</b> in der Kanalspezifikation)
uint32_t	rx_hovr;	Hardware FIFO Überlauf während des Empfangs  $rx\_hovr$ wird in der Laufzeit hochgezählt, wenn ein ankommendes Telegramm durch Can-Controller nicht empfangen worden sein kann, weil der Controller-Empfangsbuffer von dem Treiber noch nicht entleert und freigegeben wurde.
uint32_t	er_int;	Anzahl der Fehler-meldungen – Interrupts  $er\_int$ wird in der Laufzeit hochgezählt, wenn ein interner Controller-Fehlerzähler den Wert 96 erreicht hat.
uint32_t	er_bus;	Anzahl der "Bus errors"  $er\_bus$ wird in der Laufzeit hochgezählt, wenn der Controller-Transmitfehlerzähler den Wert 255 erreicht hat.
char	driver_name[INFO_NAME_MAX];	Name des Treibers
char	process_name[INFO_NAME_MAX];	Prozessname
char	driver_vers [INFO_NAME_MAX];	Version des Treibers
uint32_t	driver_pid;	Pid (Process Identifier) des Treibers
char	card_name [INFO_NAME_MAX];	Name der Karte (Optional)
char	contr_name [INFO_NAME_MAX];	Name des Controllers

INFO\_NAME\_MAX ist 64 Bytes lang.

### 6.1.3 canstat\_s - Struktur

```
typedef struct canstat_s {
    uint32_t      busoff;      /* 1 - Bus off          */
    uint32_t      error;       /* 1 - Error            */
    uint32_t      dataovr;     /* 1 - Data Overrun    */
    int32_t rxbt;             /* rx buffers total    */
    int32_t rxbu;            /* rx buffers used     */
    int32_t rxbf;            /* rx buffers free     */
    int32_t txbt;            /* tx buffers total    */
    int32_t txbu;            /* tx buffers used     */
    int32_t txbf;            /* tx buffers free     */
} canstat_t;
```

uint32_t	busoff;	1 – Device in „busoff“ Zustand
uint32_t	error;	1 – Device in „error“ Zustand
uint32_t	dataovr;	Empfang - Überlauf
int32_t	rxbt	Anzahl der Empfangsbuffer insgesamt
int32_t	rxbu;	Anzahl der Empfangsbuffer benutzt
int32_t	rbf;	Anzahl der Empfangsbuffer frei
int32_t	txbt	Anzahl der Sendebuffer insgesamt
int32_t	txbu;	Anzahl der Sendebuffer benutzt
int32_t	txbf;	Anzahl der Sendebuffer frei

### 6.1.4 canregs\_s - Struktur

```
typedef struct canregs_s {
    uint32_t cr; /* control register*/
    uint32_t sr; /* status register*/
    uint32_t isr; /* interrupt status register */
    uint32_t ier; /* interrupt enable register */
} canregs_t;
```

uint32_t	cr;	Control Register
uint32_t	sr;	Status Register
uint32_t	isr;	Interrupt Status register
uint32_t	ier	Interrupt Enable register

Als Rückgabewert liefert *ioctl()* den Wert 0 beim Erfolg, -1 beim Fehler und **errno** wird entsprechend gesetzt:

- EBADF - Parameter „fd“ - falscher Filedescriptor
- EINVAL - Parameter „cmd“ ist falsch.

## 7 Beispiel

Ein typisches Beispiel mit read/write:

```
#include <libc.h>
#include <can.h>

#define DATA_ID    1234

int main(int argc, char *argv[])
{
    int          fd, ret, len;
    canmsg_t     msg;
    volatile int  done = 0;
    int          myid = 1313;
    char         mydata[CAN_MSG_LENGTH] = "Hi Can!";

    if ((fd = open("/dev/can/0", O_RDWR)) == -1){
        perror("open() failed");
        exit(EXIT_FAILURE) ;
    }

    while(!done){
        ret = read(fd, &msg , 1 * sizeof(canmsg_t));

        switch(ret){
            case 0:
                continue;
            case -1:
                fprintf(stderr, "read() failed\n");
                continue;
            default:
                break;
        }

        if (msg.flags & MSG_OVR){
            /*
             * ...
             * Strategie zur Schadenbegrenzung fahren
             * ...
             */
        }
        switch (msg.id){
            case 0x80:          /* Sync - message.  Sende meine Daten
*/
                msg.flags      = 0;
                msg.id         = myid;
                msg.timestamp   = 0;
                len = strlen(mydata);
                len = len < CAN_MSG_LENGTH ? len : CAN_MSG_LENGTH;
                memcpy(&msg.data, &mydata, len);
                if (write(fd, &msg, 1 * sizeof(canmsg_t)) <= 0){
```

```
        fprintf(stderr, "write() failed\n");
    }
    break;
case DATA_ID:
    /*
        ...
        Daten auswerten.
        ...
    */
default:
    continue;
    }
}
return 0;
}
```

---

<sup>i</sup> Getestete KARTEN:

- PC-ISA-Karten:
  - PC-ISA-CAN von PEAK-Service GmbH
  - CANISA-DN von Contemporary Control Systems, Inc.
  
- PC/104-Karten:
  - CAN104-DN von Contemporary Control Systems, Inc.
  
- PC-PCI-Karten:
  - PCCAN-PCI von PEAK-Service GmbH

<sup>ii</sup> Der Treiber unterstützt nur die CAN 2.A Spezifikation (11 bit Identifier).