

# **C-Interface-Library für DOS und Win 3.11**

Software-Handbuch

Der Inhalt dieser Dokumentation wurde mit größter Sorgfalt erarbeitet und geprüft. esd übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

esd hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.

Sämtliche Rechte an der Dokumentation liegen bei esd. Die Weitergabe an Dritte und Vervielfältigung jeder Art, auch auszugsweise, sind nur mit schriftlicher Genehmigung durch esd gestattet.

**esd electronic system design gmbh**

Vahrenwalder Str. 205

D-30165 Hannover

Tel.: 0511/37298-0

FAX : 0511/633650

Email: [pm@esd.h.eunet.de](mailto:pm@esd.h.eunet.de)

<b>Handbuch-Datei:</b>	I:\TEXTE\DOKU\MANUALS\PROGRAM\CAN\SCHICHT2\DEUTSCH\DOS\UNIDOS11.MA6
<b>Datum des Ausdrucks:</b>	09.02.98

<b>Beschriebene Software:</b>	C-Interface Library für DOS
<b>Revision/Datum:</b>	V1.2

<b>Implementiert auf folgenden Boards</b>	<b>Bestellnummer</b>
CAN-ISA/200	C.2011.xx
CAN-ISA/331	C.2010.xx
CAN-PC104/331	C.2012.xx
-	-
CAN-PCI/331	C.2020.xx
CAN-PCI/360	C.2022.xx
PMC-CAN/331	C.2025.xx
CAN-CPCI/331	C.2027.xx
-	-

### Änderungen in der Software und/oder der Dokumentation

<b>Änderung in diesem Handbuch gegenüber der Vorversion</b>	<b>Änderung in der Software</b>	<b>Änderung in der Dokumentation</b>
Erste Ausgabe des universellen DOS-Handbuches.	-	-



<b>Inhaltsverzeichnis</b>	<b>Seite</b>
<b>1. Referenz</b> .....	3
<b>2. Einleitung</b> .....	4
<b>3. Funktionsbeschreibung der Programmierschnittstelle</b> .....	5
3.1 Initialisierung .....	5
canHwInit() .....	5
canEnableIrq() .....	5
canDisableIrq() .....	6
canInit() .....	7
canInitEx() .....	8
3.2 Lesen und Schreiben von Daten .....	9
canEnableId() .....	9
canDisableId() .....	9
canSetAcceptanceEx() .....	10
canRead() .....	11
canReadEx() .....	12
canWrite() .....	13
canWriteEx() .....	14
<b>4. AC2-Kompatible Funktionsaufrufe der Programmierschnittstelle</b> .....	15
4.1 Initialisierung .....	15
CAN_start_chip() .....	15
CAN_reset_board() .....	15
4.2 Lesen und Schreiben von Daten .....	16
CAN_set_acceptance(), CAN_set_acceptance2() .....	16
CAN_read() .....	17
CAN_send_data(), CAN_send_data2() .....	20
CAN_send_remote(), CAN_send_remote2() .....	21
<b>5. Rückgabewerte</b> .....	22
<b>Anhang - Installation und Implementierung</b> .....	A-1



## 1. Referenz

- /1/: esd electronic system design gmbh, CAL-Slave Manual, December 1996
- /2/: esd electronic system design gmbh, CANopen-Slave Manual, December 1996
- /3/: esd electronic system design gmbh, CAL-Master Manual, December 1996
- /4/: esd electronic system design gmbh, CANopen-Master Manual, December 1996
- /5/: CiA DS-102, CAN Physical Layer for Industrial Applications, April 1994
- /6/: CiA DS-201, CAN Reference Model, February 1996
- /7/: CiA DS-202/1, CMS Service Specification, February 1996
- /8/: CiA DS-202/2, CMS Protocol Specification, February 1996
- /9/: CiA DS-203/3, CMS Data Types and Encoding Rules, February 1996

## 2. Einleitung

Im folgenden Dokument wird die C-Schnittstelle zum Treiber des CAN-Controllers beschrieben.

Das Handbuch besteht aus einem allgemeinen Teil, der identisch für alle von *esd* entwickelten Treiber ist und einem Anhang, der die Installation des Treibers sowie implementierungsspezifische Eigenheiten beschreibt.

Die Prototypen aller Funktionen sowie der *Communication-Handles* sind in der Header-Datei *can.h* zusammengefaßt.

### 3. Funktionsbeschreibung der Programmierschnittstelle

Im nachfolgenden Kapitel wird die C-Programmierschnittstelle zum CAN-Controller beschrieben. Die Bedeutung der Rückgabewerte im Fehlerfall ist in Kapitel 4 beschrieben.

#### 3.1 Initialisierung

---

#### **canHwInit()**

**Name:** `canHwInit()` - Hardware-Initialisierung des CAN-Interfaces

**Aufruf:**

```
int canHwInit
(
    unsigned int base_addr    /* Adresse des Moduls */
)
```

**Beschreibung:** Diese Funktion muß als erste Library-Funktion zu Beginn aufgerufen werden.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

---

#### **canEnableIrq()**

**Name:** `canEnableIrq()` - Einstellen des Interrupt-Levels

**Aufruf:**

```
int canEnableIrq
(
    int level    /* IRQ Level */
)
```

**Beschreibung:** Mit dieser Funktion wird der Interrupt-Level des CAN-Moduls eingestellt.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## **canDisableIrq()**

**Name:** `canDisableIrq()` - Interrupt-Level löschen

**Aufruf:** `int canDisableIrq`  
(  
)

**Beschreibung:** Mit dieser Funktion wird der vorher programmierte Interrupt-Level wieder zurückgenommen. Die Funktion sollte nur aufgerufen werden, wenn vorher erfolgreich *canEnableIrq* aufgerufen worden ist.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## canInit()

**Name:** canInit() - Initialisierung der Baudrate des CAN-Interfaces

**Aufruf:**

```
int canInit
(
  int net,          /* Nummer der CAN-Schnittstelle */
  int baudrate     /* Baudratenindex */
)
```

**Beschreibung:** Diese Funktion initialisiert das CAN-Interface und setzt die Bit-Rate für das Netz *net* auf die Bit-Rate *baudrate* entsprechend nachfolgender Tabelle. Das Verhalten bei Übergabe anderer Werte sowie Zahl und Zuordnung der unterstützten Netze ist implementierungsabhängig.

baud [HEX]	Bit-Rate [KBit/s]
0	1000
1	666.6
2	500
3	333.3
4	250
5	166
6	125
7	100
8	66.6
9	50
A	33.3
B	20
C	12.5
D	10

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## **canInitEx()**

**Name:** **canInitEx()** - Initialisierung der Baudrate über BT0/BT1-Register

**Aufruf:**

```
int canInitEx
(
    int net,          /* Nummer der CAN-Schnittstelle */
    int bt0_bt1      /* Baudrate über Controller-Register */
)
```

**Beschreibung:** Diese Funktion initialisiert das CAN-Interface und setzt die Bit-Rate für das Netz *net* auf den Wert, der über *bt0\_bt1* definiert ist. In *bt0\_bt1* sind die Inhalte der Register von BTR0 und BTR1 des CAN-Controllers kodiert:

$$bt0\_bt1 = BTR0 \cdot 256 + BTR1$$

Die Beschreibung der Registerwerte von BTR0 und BTR1 ist dem Handbuch des CAN-Controllers zu entnehmen.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

### 3.2 Lesen und Schreiben von Daten

Die nachfolgend beschriebenen Aufrufe dienen zum Lesen und Schreiben von Daten auf dem CAN-Bus. Daten können durch Aussenden eines RTR-Frames auf einem Rx-Identifizier angefordert werden und es ist möglich mit einem Tx-Identifizier auf den Empfang eines RTR-Frames zu warten.

---

#### **canEnableId()**

**Name:** **canEnableId()** - Freigabe eines Identifiers für Rx- und RTR-Transfers

**Aufruf:**

```
int canEnableId
(
    int net,          /* Nummer der CAN-Schnittstelle */
    int id           /* Rx-Identifizier */
)
```

**Beschreibung:** Diese Funktion gibt Identifier für Rx- und RTR-Transfers frei.

Mit *net* wird die Netznummer übergeben, falls mehrere Netze verfügbar sind.  
*id* ist der gewünschte CAN-Identifizier im Bereich 0-2047.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

---

#### **canDisableId()**

**Name:** **canDisableId()** - Rücknahme eines Identifiers für Rx- oder RTR-Transfers

**Aufruf:**

```
int canDisableId
(
    int net,          /* Nummer der CAN-Schnittstelle */
    int id           /* Rx-Identifizier */
)
```

**Beschreibung:** Diese Funktion sperrt Identifier für Rx- und RTR-Transfers.

Mit *net* wird die Netznummer übergeben, falls mehrere Netze verfügbar sind.  
*id* ist der gewünschte CAN-Identifizier im Bereich 0-2047.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## canSetAcceptanceEx()

**Name:** canSetAcceptanceEx() - Setzen des Acceptance-Filters für 29-Bit-Identifiers

**Aufruf:**

```
int canSetAcceptanceEx
(
    int net,          /* Nummer der CAN-Schnittstelle */
    int handle,      /* muß immer auf '0' gesetzt werden */
    long mask,       /* Auswahl der zu vergleichenden Id-Bits */
    long id          /* Werte mit denen verglichen werden soll */
)
```

**Beschreibung:** Diese Funktion setzt das Acceptance Filter für 29-Bit-Identifizier.  
Die Funktion ist nicht für 11-Bit-Identifizier vorgesehen.

Mit *net* wird die Netznummer übergeben, falls mehrere Netze verfügbar sind.

*handle* wird zur Zeit noch nicht unterstützt und muß immer auf '0' gesetzt werden.

In *mask* werden die Identifizier-Bits, die mit einem vorgegebenen Wert verglichen werden sollen, auf '1' gesetzt. Identifizier-Bits, deren Wert beliebig sein soll, müssen mit '0' angegeben werden.

In *id* kann für die in *mask* mit einer '1' gekennzeichneten Bits ein Soll-Wert festgelegt werden. Für diese Bits kann in *id* der Wert '0' oder '1' eingetragen werden. Für alle anderen Bits muß in *id* der Wert '0' eingetragen werden.  
Es werden nur die Identifizier in den Receive-Buffer aufgenommen, deren Bits die geforderten Werte aufweisen.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## canRead()

**Name:** canRead() - Lesen des aktuellen Rx-Frames

**Aufruf:**

```
int canRead
(
  int          *net,          /* Nummer der CAN-Schnittstelle */
  int          *id,          /* Rx-Identifizier */
  int          *length,      /* Anzahl der Datenbytes (1..8) */
  unsigned char *data        /* empfangene Daten */
)
```

**Beschreibung:** Diese Funktion liest die letztgültigen Rx-Daten des Rx-Identifiers. Die Funktion wartet nicht.

*\*net* enthält die Nummer des CAN-Netzes ('0' oder '1'). Ist nur ein Netz vorhanden, muß immer '0' eingegeben werden.

In *\*id* wird der gewünschte Rx-Identifizier im Bereich von 0-2047 eingetragen.

Die Zahl der kopierten Bytes wird in *\*length* abgelegt. Werte für *\*length*, die größer als \$0x10 sind, zeigen RTR-Frames an.

z.B. \$0x10 => RTR-Frame mit der Datenlänge Länge '0'

Die empfangenen Daten sind in *\*data* abgelegt.

Status = -1	keine Daten
0	Rx-Daten gelesen
5	Acknowledge für letztes Tx-Telegramm auf diesem Identifizier

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## canReadEx()

**Name:** canReadEx() - Lesen des aktuellen Rx-Frames mit 29-Bit-Identifizier

**Aufruf:**

```
int canReadEx
(
  int          *net,          /* Nummer der CAN-Schnittstelle */
  long         *id,          /* 29-Bit-Rx-Identifizier */
  int          *length,      /* Anzahl der Datenbytes (1...8) */
  unsigned char *data        /* empfangene Daten */
)
```

**Beschreibung:** Diese Funktion liest die letztgültigen Rx-Daten des 29-Bit-Rx-Identifiziers. Die Funktion wartet nicht.

*\*net* enthält die Nummer des CAN-Netzes ('0' oder '1'). Ist nur ein Netz vorhanden, muß immer '0' eingegeben werden.

In *\*id* wird der gewünschte Rx-Identifizier im Bereich von 0...(2<sup>29</sup>-1) eingetragen.

Die Zahl der kopierten Bytes wird in *\*length* abgelegt. Werte für *\*length*, die größer als \$0x10 sind, zeigen RTR-Frames an.

z.B. \$0x10 => RTR-Frame mit der Datenlänge Länge '0'

Die empfangenen Daten sind in *\*data* abgelegt.

Status = -1	keine Daten
0	Rx-Daten gelesen
5	Acknowledge für letztes Tx-Telegramm auf diesem Identifizier
15	Extended-Identifizier: Daten empfangen
16	Extended-Identifizier: Tx-Frame gesendet

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## canWrite()

**Name:** canWrite() - Senden von Tx-Frames

**Aufruf:**

```
int canWrite
(
    int          net,          /* Nummer der CAN-Schnittstelle */
    int          id,          /* Tx-Identifizier */
    int          length,      /* Anzahl der Daten-Bytes (1...8) */
    unsigned const char *data /* zu sendende Daten */
)
```

**Beschreibung:** Diese Funktion führt zum Senden von Tx-Frames.

*net* enthält die Nummer des CAN-Netzes ('0' oder '1'). Ist nur ein Netz vorhanden, muß immer '0' eingegeben werden.

In *id* wird der gewünschte Tx-Identifizier im Bereich von 0-2047 eingetragen.

In *length* wird die gewünschte Anzahl der zu sendenden Daten-Bytes eingetragen. Werte für *length*, die größer als \$0x10 sind, zeigen RTR-Frames an. z.B. \$0x10 => RTR-Frame mit der Datenlänge '0' senden.

Die Daten werden in *\*data* abgelegt.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## canWriteEx()

**Name:** canWriteEx() - Senden von Tx-Frames auf 29-Bit-Identifiers

**Aufruf:**

```
int canWriteEx
(
  int          net,          /* Nummer der CAN-Schnittstelle */
  long         id,          /* 20-Bit-Tx-Identifizier */
  int          length,     /* Anzahl der Daten-Bytes (1...8) */
  unsigned const char *data /* zu sendende Daten */
)
```

**Beschreibung:** Diese Funktion führt zum Senden von Tx-Frames.

*net* enthält die Nummer des CAN-Netzes ('0' oder '1'). Ist nur ein Netz vorhanden, muß immer '0' eingegeben werden.

In *id* wird der gewünschte Tx-Identifizier im Bereich von 0...(2<sup>20</sup>-1) eingetragen.

In *length* wird die gewünschte Anzahl der zu sendenden Daten-Bytes eingetragen. Werte für *length*, die größer als 0x10 sind, zeigen RTR-Frames an.  
z.B. 0x10 => RTR-Frame mit der Datenlänge '0' senden.

Die Daten werden in *\*data* abgelegt.

**Rückgabe:** 0 oder einen im Anhang beschriebenen Fehlercode.

## 4. AC2-Kompatible Funktionsaufrufe der Programmierschnittstelle

Die Software bietet Anwendern, die bisher mit der AC2-Software gearbeitet haben, die im folgenden beschriebenen Funktionsaufrufe. Die Funktionalität dieser Routinen ist kompatibel zu AC1/2-Routinen ähnlichen Namens (z.B. vorher *AC2\_send\_data*, jetzt *CAN\_send\_data*).

**Sie sollten diese Routinen jedoch nur zum Portieren *vorhandener I/O-Funktionen* und *nicht für Neuentwicklungen* verwenden!**

**Wenn diese Routinen verwendet werden, dürfen die Routinen *canEnableId*, *canDisableId*, *canRead* und *canWrite* nicht eingesetzt werden!**

### 4.1 Initialisierung

#### CAN\_start\_chip()

**Name:** CAN\_start\_chip() - Starten der CAN-Bus operationen

**Aufruf:**

```
int CAN_start_chip
( void
)
```

**Beschreibung:** Diese Funktion aktiviert die CAN-bus-Aktionen. Der CAN-Controller verläßt den RESET-Zustand. Ab jetzt können Sendeaufträge gestartet werden und eintreffende Nachrichten werden überwacht.

**Rückgabe:**

- 0: Start erfolgreich
- 3: Timeout-Fehler beim Zugriff auf die Platine
- 4: Timeout-Fehler beim Zugriff auf den CAN-Controller

#### CAN\_reset\_board()

**Name:** CAN\_reset\_board() - Rücksetzen der Karte in den Initialisierungszustand

**Aufruf:**

```
int CAN_reset_board
( void
)
```

**Beschreibung:** Diese Funktion setzt die Karte zurück in den Initialisierungszustand. Die Firmware wird in die Karte geladen, daher benötigt diese Funktion u.U. etwas mehr Zeit.

**Rückgabe:**

- 0: Initialisierung erfolgreich
- 1: Fehler beim Laden der Firmware (Platine oder Firmware nicht gefunden)

## 4.2 Lesen und Schreiben von Daten

Die nachfolgend beschriebenen Aufrufe dienen zum Lesen und Schreiben von Daten auf dem CAN-Bus. Daten können durch Aussenden eines RTR-Frames auf einem Rx-Identifizier angefordert werden und es ist möglich, mit einem Tx-Identifizier auf den Empfang eines RTR-Frames zu warten.

---

### **CAN\_set\_acceptance(), CAN\_set\_acceptance2()**

**Name:**            **CAN\_set\_acceptance()**        - Setzen des Acceptance-Filters für Netz 0  
                  **CAN\_set\_acceptance2()**    - Setzen des Acceptance-Filters für Netz 1

**Aufruf:**

```
int  CAN_set_acceptance
(
    int      AccCode,      /* Acceptance Code Register */
    int      AccMask      /* Acceptance Mask Register */
)

int  CAN_set_acceptance2
(
    int      AccCode,      /* Acceptance Code Register */
    int      AccMask      /* Acceptance Mask Register */
)
```

**Beschreibung:** Diese Funktionen setzen die Acceptance-Register des CAN-Controllers. *CAN\_set\_acceptance* setzt den ersten CAN-Kanal und *CAN\_set\_acceptance2* den zweiten CAN-Kanal. Mit diesen Registern kann ein Empfangsfilter für Daten- und Remote-Frames erzeugt werden. Nur die Daten der Identifizier, deren höherwertige 8 Bits dem Filter entsprechen werden akzeptiert. Alle Bits, die im Acceptance-Mask-Register mit '0' markiert sind, müssen den selben Wert wie die entsprechenden Bits im Acceptance-Code-Register haben, um den Filter zu passieren. Eine '1' im Acceptance-Mask-Register bedeutet, daß das entsprechende Bit nicht überprüft wird.

Die exakte Registerbeschreibung kann dem Datenbuch des CAN-Controllers (82C200, 8xC592 oder SJA100 von Philips) entnommen werden.

*AccCode:*        Acceptance-Code-Register [0 to FF<sub>Hex</sub>]  
*AccMask:*        Acceptance-Mask-Register [0 to FF<sub>Hex</sub>]

**Rückgabe:**    0:    Erfolgreicher Aufruf  
                 -3:    Timeout-Fehler beim Zugriff auf die Platine  
                 -4:    Timeout-Fehler beim Zugriff auf den CAN-Controller

## CAN\_read()

**Name:** CAN\_read() - Nachrichten empfangen und Transfers auswerten

**Aufruf:** int CAN\_read  
(  
 param\_struct \*write\_ac\_param /\* Übergabestruktur \*/  
)

**Beschreibung:** Über diese Funktion wird die Applikation über die Sendung und den Empfang von Nachrichten, sowie verschiedene Fehlerbedingungen informiert.

Es gibt die Option, die Funktion in eine Interrupt-Service-Routine auf dem PC einzubinden, und die Events dem CAN-Modul mittels Interrupt mitzuteilen. Alternativ kann die Funktion per Polling abgefragt werden.

Die Rückgabe-Codes 1...12 (RC1 bis RC12) definieren, welche Elemente der zurückgegebenen Struktur relevant sind.

### Elemente der Struktur *param\_struct*:

- int *Ident*: Identifier eines empfangenen oder gesendeten Frames (RC1, RC2, RC3, RC8).
- int *DataLength*: Anzahl der empfangenen (RC1) oder angeforderten (RC2) Daten-Bytes.
- int *RecOverrun\_flag*: Die letzten empfangenen Daten wurden nicht vom PC gelesen und wurden von den neuen Daten überschrieben (RC1, RC2).
- int *RCV\_fifo\_lost\_msg*: Anzahl der verlorengegangenen Nachrichten des Rx-FIFOs (RC1, RC2, RC3, RC8).
- byte *RCV\_data[8]*: Empfangene Daten-Bytes (RC1).
- int *AckOverrunFlag*: Die Bestätigung des letzten gesendeten Frames ist von der Applikation bisher nicht gelesen worden (RC3).
- int *XMT\_ack\_fifo\_lost\_acks*: Anzahl der verlorengegangenen Nachrichten im Transmit-Acknowledge-FIFO (RC3).

- int *XMT\_rmt\_fifo\_lost\_remotes*: Anzahl der verlorengegangenen Übertragungsaufträge im Remote-Transmit-FIFO.
  
- int *Bus\_state*: CAN-Bus-Status (RC5):
  - 0: error active
  - 1: error passive
  - 2: bus off
  
- int *Error\_state*: Weitere Fehlerursachen (RC7):
  - 0: kein Fehler
  - 3: Overrun of BASIC CAN 82C200
  
- int *Can*: CAN-Kanal 1 oder 2 (RC1, RC2, RC3, RC4, RC5, RC7, RC8). Wenn *CAN\_read* aufgerufen wird, werden die unten aufgeführten Zustände in der beschriebenen Reihenfolge abgearbeitet.
  
- *Ändern des Bus-Status (RC5)*:

Wenn die Firmware einen Wechsel des Bus-Status entdeckt, gibt sie die Information an die Strukturvariable *Bus\_state* weiter.
  
- *Empfang eines Daten- oder Fehler-Frames (RC1 or RC2)*:

Ist eine Nachricht im Empfangs-FIFO eingetroffen oder wurde der Empfang eines Frames per polling erkannt, so werden die folgenden Parameter in die Struktur übernommen: Identifier, Overrun-Flag, die Anzahl verlorengegangener Nachrichten des Empfangs-FIFOs und die empfangenen Daten.
  
- *Bestätigung der Sendung eines Tx-Datenframes (RC3)*:

Wenn eine Sendebestätigung eines Frames im Transmit-Acknowledge-FIFO oder per polling aller Objekte erkannt wurde, wird der Identifier und, sofern vorhanden, die Anzahl der verlorengegangenen Nachrichten in das Transmit-Acknowledge-FIFO, bzw. in *XMT\_ack\_fifo\_lost\_acks* geschrieben.
  
- *Überlauf des Remote-Transmit-FIFOs (RC4)*:

(Sendeauftrag aufgrund eines empfangenen RTR-Frames)  
Im Falle eines Überlaufs des Remote-Transmit-FIFOs wird die Anzahl der verlorengegangenen Nachrichten in *XMT\_rmt\_fifo\_lost\_remotes* abgelegt.
  
- *Weitere Fehlerursachen (RC7)*:

Die oben aufgeführten Fehlerursachen werden über ihre zugehörigen Codes in der Strukturvariablen *Error\_state* erkannt.
  
- *Nachrichten auf CAN-Netz 1 und CAN-Netz2 , wenn letztere im FIFO-Mode arbeitet (RC1, RC2, RC3, RC5, RC7, RC8)*.

Wenn eine Nachricht erkannt wurde, werden die nachfolgenden Nachrichten erst ausgewertet, wenn *CAN\_read* ein weiteres mal aufgerufen wird.

**Rückgabe:**

(Rückgabe-Codes - RC's des Kommandos)

- 0: Keine neuer Event auf dem CAN-Kopplermodul.
- 1: Standard-Daten-Frame empfangen
- 2: Standard-Remote-Frame empfangen
- 3: Sendung eines Standard-Daten-Frames ist bestätigt.
- 4: Überlauf des Sende-FIFOs.
- 5: Änderung des Bus-Status.
- 7: Eine weitere Fehlerbedingung liegt an.
- 8: Sendung eines Standard-Remote-Frames ist bestätigt.

## CAN\_send\_data(), CAN\_send\_data2()

**Name:**            **CAN\_send\_data()**     - Daten senden auf Netz 0  
                   **CAN\_send\_data2()**   - Daten senden auf Netz 1

**Aufruf:**

```

int  CAN_send_data
(
  int          Ident,          /* Tx-Identifizier */
  int          DataLength,    /* Anzahl zu sendender Daten-Bytes */
  byte        *pData          /* Pointer auf Datenstruktur */
)

int  CAN_send_data2
(
  int          Ident,          /* Tx-Identifizier */
  int          DataLength,    /* Anzahl zu sendender Daten-Bytes */
  byte        *pData          /* Pointer auf Datenstruktur */
)
  
```

**Beschreibung:** Dieses Kommando sendet ein Daten-Frame mit den angegebenen Parametern auf dem CAN-Bus.

*Ident* übergibt den Tx-Identifizier.  
 In *DataLength* wird die Anzahl der zu sendenden Bytes eingetragen.  
*pData* ist der Pointer auf das Adreßfeld der Daten.

**Rückgabe:**     0:   Funktion erfolgreich  
                  -1:   Sende-FIFO ist voll

## CAN\_send\_remote(), CAN\_send\_remote2()

**Name:** CAN\_send\_remote() - RTR senden auf Netz 0  
CAN\_send\_remote2() - RTR senden auf Netz 1

**Aufruf:**

```
int CAN_send_remote
(
  int      Ident,          /* Tx-Identifizier */
  int      DataLength     /* Anzahl zu sendender Daten-Bytes */
)

int CAN_send_remote2
(
  int      Ident,          /* Tx-Identifizier */
  int      DataLength     /* Anzahl zu sendender Daten-Bytes */
)
```

**Beschreibung:** Dieses Kommando sendet ein Remote-Frame mit den angegebenen Parametern auf den CAN-Bus. Das gesendete Remote-Frame hat immer die Länge 0. Trotzdem wird im Parameter *DataLength* die Anzahl der gesendeten Daten-Bytes (hier eben 0) mit übertragen.

*Ident* übergibt den Tx-Identifizier.

In *DataLength* wird die Anzahl der zu sendenden Bytes eingetragen (hier immer '0').

**Rückgabe:** 0: Funktion erfolgreich  
-1: Sende-FIFO ist voll

## 5. Rückgabewerte

Eine Funktion mit einem Rückgabewert vom Typ *int* wird ein Fehlercode entsprechend nachfolgender Tabelle zurückgegeben, wobei nicht alle Codes für alle Funktionen sinnvoll sind.

<b>Fehler</b>	<b>Beschreibung</b>
CAN_OK	Kein Fehler
CAN_NO_DEVICE	CAN-Controller nicht vorhanden oder nicht initialisiert
CAN_PARA_ERROR	Ungültiger Parameter beim Funktionsaufruf
CAN_SYS_ERROR	Interner Fehler

## Anhang - Installation und Implementierung

Inhalt	Seite
A 1 Installation .....	A-2
A 1.1 Installation der Hardware .....	A-2
A 1.2 Installation der Software .....	A-2
A 2 Implementierung .....	A-2
A 2.1 Zahl und Zuordnung der Netze .....	A-2

## **A 1 Installation**

### **A 1.1 Installation der Hardware**

Die Hardware-Installation wird in der Dokumentation 'Hardware-Installation und technische Daten' beschrieben.

### **A 1.2 Installation der Software**

Die Software umfaßt zur Zeit zwei Files, die lediglich in ein gemeinsames Verzeichnis, dessen Namen frei wählbar ist, kopiert werden müssen..

## **A 2 Implementierung**

### **A 2.1 Zahl und Zuordnung der Netze**

Der Treiber unterstützt bis zu 2 Netze. Das erste Netz hat die logische Netz-Nummer '0' und das zweite Netz die logische Netznummer '1'.