

# CAN-LIB

**Dok-Rev. 1.1 vom 06.11.2009**  
**Software-Rev. 2.4 vom 28.05.2009**

---

---

## **Table of Contents**

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Copyright and Disclaimer</b> .....       | <b>3</b>  |
| 1.1      | Handling                                    | 3         |
| 1.2      | Statement                                   | 3         |
| <b>2</b> | <b>Introduction</b> .....                   | <b>4</b>  |
| 2.1      | Purpose                                     | 4         |
| 2.2      | References                                  | 4         |
| 2.3      | Definitions of Terms                        | 4         |
| <b>3</b> | <b>CAN Libraries</b> .....                  | <b>5</b>  |
| 3.1      | C   | 5         |
| 3.2      | CoDeSys                                     | 5         |
| <b>4</b> | <b>CAN Initialization</b> .....             | <b>6</b>  |
| <b>5</b> | <b>CAN Message Data Structure</b> .....     | <b>8</b>  |
| <b>6</b> | <b>Receiving CAN Messages</b> .....         | <b>9</b>  |
| 6.1      | Receiving multiple CAN Messages             | 10        |
| 6.2      | Checking Messages in the Receive Buffer     | 10        |
| <b>7</b> | <b>Sending CAN Messages</b> .....           | <b>12</b> |
| 7.1      | Sending multiple CAN Messages               | 12        |
| 7.2      | Timeout for Sending CAN Messages            | 13        |
| <b>8</b> | <b>Overview of all Error Messages</b> ..... | <b>14</b> |

Revisionsliste:

| Rev. | Datum      | Na. | Change      |
|------|------------|-----|-------------|
| 1.0  | 05.08.2009 | Lu  | First Issue |
| 1.1  | 16.09.2009 | Ko  | Corrections |
|      |            |     |             |

---

## **1 Copyright and Disclaimer**

All rights to these documents are held by the IEP GmbH, Langenhagen.

The reproduction, even partial, is only with our express written approval allowed.

In connection with the purchase of software, the buyer acquires a simple, non-transferable license. This licence is used on or in connection with ONE computer. Creating a copy for archival purposes is only under the supervision of the buyer or his authorized agent permitted. The buyer is liable for damages resulting from the breach of his duty of care, eg case of unauthorized copying, unauthorized distribution of software, etc.. The buyer of the software give its acceptance to the above conditions. For unlicensed copying we assume of theft until subject to a legal declaration. This also applies to documentation and software, which arose through the modification of documents and programs from IEP, regardless of whether the changes are to be regarded as insignificant or significant.

IEP is expressly excluded from any liability for damages resulting from the use of software, hardware, or use of this manuscript, even in the event of faulty software or erroneous information.

The consent of the buyer or user for the exclusion of liability shall apply to the purchase and use of the software and those documents have been given.

### **1.1 Handling**

Please first read this documentation carefully before you start to program. You save time and hassle.

### **1.2 Statement**

We reserve the right to make changes that improve the circuit or the product, without making specific references. The accuracy of any given data, diagrams, descriptions of programs can not be warranted. The suitability of the product for a particular purpose is not guaranteed.

---

## **2 Introduction**

### **2.1 Purpose**

This document describes the CAN API functions implemented in libraries developed by IEP GmbH, Langenhagen, Germany. The libraries can be used to access the CAN bus from C and CoDeSys on a controller board equipped with RTOS-UH and based on a 68k or PowerPC processor architecture.

### **2.2 References**

- /1/ ISO 11898-1 Road vehicles; Controller area network (CAN);  
Data link layer and physical signalling (2003-12)
- /2/ ISO 11898-2 Road vehicles; Controller area network (CAN);  
High-speed medium access unit (2003-12)
- /3/ RTOS RTOS-UH Introduction and Overview (2000)
- /4/ CREST C Programmierung unter RTOS-UH (2000-04)

### **2.3 Definitions of Terms**

|                |   |
|----------------|---|
| CAN            | Controller Area Network                                       |
| CAN message ID | Identification number of a CAN message                        |
| CiA            | CAN in Automation (organisation for promotion of the CAN bus) |

---

## **3 CAN Libraries**

Communication between CAN devices with RTOS-UH is implemented as a set of special function calls implemented in additional libraries for C and CoDeSys:

### **3.1 C**

The CAN implementation for C consists of an object file `canlib.obj` for each processor architecture and a common header file `canproto.h`. These files must be put into their respective folders for the compiler.

- `C:\Programs\CREST-C\h\canproto.h`
- `C:\Programs\CREST-C\clib-68k\canlib.obj`
- `C:\Programs\CREST-C\clib-ppc\canlib.obj`

To be able to use the functions in the libraries the header file must be read in by each C file:

```
#include <canproto.h>
```

This provides function declarations, type definitions and symbolic constants for use with CAN communication.

The linker just needs the line

```
canlib.obj
```

to be added to its linker file. Then the CAN library gets linked to the final binary file.

### **3.2 CoDeSys**

The CAN library implementation for CoDeSys consists of a single file named

```
IEPCANLIB20.lib.
```

It is placed into the folder of the CoDeSys project. For example:

```
D:\WorkSpace\CoDeSys\MSC32\library\IEPCANLIB20.lib
```

---

## **4 CAN Initialization**

Before being able to use the CAN interface some initialization of the interface must be performed. This is done with the following functions:

|   |   |
|---|---|
| <b>C function:</b><br>WORD <b>can_init</b> (<br>WORD        can_nr,<br>CAN_INIT *can_init_prt<br>); | <b>CoDeSys function:</b><br>FUNCTION <b>_CAN_INIT</b> : WORD<br>VAR_INPUT<br>chan        : WORD;<br>init_para   : POINTER TO IEP_CAN_INIT_PARA ;<br>END_VAR |
|---|---|

The first parameter `can_nr` (or `chan`) designates the number of the CAN controller unit to be initialized. Valid values for this parameter are 1 and 2 for the first and second CAN controllers in the system.

The second parameter is a pointer to a structure with CAN initialization parameters:

|   |   |
|---|---|
| <b>C type definition:</b><br>Typedef struct <b>CAN_INIT</b><br>{<br>ULONG    can_clock;<br>UWORD    phase_seg1;<br>UWORD    phase_seg2;<br>UWORD    samp;<br>UWORD    sjw;<br>UWORD    baudrate;<br>WORD     anz;<br>}<br><b>CAN_INIT</b> ; | <b>CoDeSys type definition:</b><br>TYPE <b>IEP_CAN_INIT_PARA</b> :<br>STRUCT<br>CAN_CLOCK    : DINT ;<br>PHASE_SEG1   : WORD ;<br>PHASE_SEG2   : WORD ;<br>SAMP         : WORD ;<br>SJW          : WORD ;<br>BAUDRATE     : WORD ;<br>ANZ          : WORD ;<br>END_STRUCT<br>END_TYPE |
|---|---|

The Baudrate of the CAN interface can be defined with custom timing details or chosen from a predefined set of baudrate definitions. If a custom setting is used then the `baudrate` structure element is set to zero and the following structure elements must be filled in:

- `can_clock`: Duration of a bit fraction in nano seconds.
- `phase_seg1`: Number of bit fractions up to the sampling point.
- `phase_seg2`: Number of bit fractions after the sampling point.
- `samp`: Number of samples to perform, valid values: 1 or 3.
- `sjw`: Synchronization jump width.

To use a predefined baudrate setting, choose a value or the symbolic name from the following table. If a value between 1 and 8 is selected then the former structure elements for detailed timing parameters will not be evaluated and a bit timing standardized by the CAN standards group CiA is used.

---

| Value | Baudrate    | C constant: | CoDeSys constant: |
|-------|-------------|-------------|-------------------|
| 0     | Custom      | BAUD_NO     | IEP_CAN_BAUD_NO   |
| 1     | 10kbit/sec  | BAUD_10K    | IEP_CAN_BAUD_10K  |
| 2     | 20kbit/sec  | BAUD_20K    | IEP_CAN_BAUD_20K  |
| 3     | 50kbit/sec  | BAUD_50K    | IEP_CAN_BAUD_50K  |
| 4     | 125kbit/sec | BAUD_125K   | IEP_CAN_BAUD_125K |
| 5     | 250kbit/sec | BAUD_250K   | IEP_CAN_BAUD_250K |
| 6     | 500kbit/sec | BAUD_500K   | IEP_CAN_BAUD_500K |
| 7     | 800kbit/sec | BAUD_800K   | IEP_CAN_BAUD_800K |
| 8     | 1Mbit/sec   | BAUD_1M     | IEP_CAN_BAUD_1M   |

anz:

The last element in the structure is the number of receiving buffers to be provided by the operating system. The maximum value is 64.

The initialization function returns an error level. A zero indicates no error; the other values are listed in the following table:

| Value | C constant:         | Description:                               |
|-------|---------------------|--|
| 0     | E_OK                | No error                                   |
| -1    | E_NO_CAN            | CAN driver not available                   |
| -3    | E_WRONG_PHYS        | Invalid interface number                   |
| -4    | E_WRONG_READ_BUFFER | Invalid size of receiving buffer           |
| -14   | E_NO_MEMORY         | Not enough memory for receiving buffer     |
| -17   | E_SAMP              | Invalid sampling number (1 or 3 are valid) |
| -18   | E_BAUDRATE          | Invalid baudrate                           |
| -19   | E_PHASE_SEG1        | Invalid value of phase seg1                |
| -20   | E_PHASE_SEG2        | Invalid value of phase seg2                |
| -21   | E_CAN_CLOCK         | Invalid value of can clock                 |
| -22   | E_JUMP_WIDTH        | Invalid value of siw                       |
| -23   | E_RESET_IMPOSS      | CAN interface does not support Reset       |

If an error is returned by the initialization function then the CAN interface is not ready for communication. The initialization must be repeated with corrected parameters.

---

## 5 CAN Message Data Structure

Message data is submitted and received from the API functions via data structures:

### C type definition:

```
typedef struct CAN_MESSAGE
{
    ULONG identifier;
    WORD frame_format;
    WORD rtr;
    WORD data_length;
    UBYTE data[8];
} CAN_MESSAGE;
```

### CoDeSys type definition:

```
TYPE IEP_CAN_MESSAGE :
STRUCT
    identifier :    DWORD    ;
    frame_format :  WORD     ;
    rtr         :    WORD    ;
    data_length :   WORD     ;
    data       :  ARRAY[1..8] OF BYTE ;
END_STRUCT
END_TYPE
```

The elements of these structures have the following meaning

Identifier:

This element carries the CAN identifier used for addressing on the CAN bus. The value space for this variable depends on the setting for a width of 11 or 29 bits of CAN addresses in the following `frame_format` element.

`frame_format`:

This flag selects the width of CAN identifiers. The following values are available:

0: 11 bit CAN IDs.

1: 29 bit CAN IDs.

The following symbolic constants are available for this setting:

| Value | C constant: | CoDeSys constant:   |
|-------|-------------|---------------------|
| 0     | FRAME_SHORT | IEP_CAN_FRAME_SHORT |
| 1     | FRAME_LONG  | IEP_CAN_FRAME_LONG  |

`rtr`:

Remote frames are requested with this flag. The following values are available:

0: Regular data transmission.

1: Request of remote frame.

The following symbolic constants are available for this setting:

| Value | C constant: | CoDeSys constant: |
|-------|-------------|-------------------|
| 0     | RTR_NO      | IEP_CAN_RTR_NO    |
| 1     | RTR_YES     | IEP_CAN_RTR_YES   |

`data_length`:

This is the number of bytes received or to be transmitted. Valid values for this element are 0 to 8.

`data`:

This is the memory buffer for the actual data bytes. This array can hold at most 8 bytes.

**Attention:** The valid array indices for the `data` array differ between C and CoDeSys!

**C:** 0 to 7

**CoDeSys:** 1 to 8

---

## 6 Receiving CAN Messages

CAN messages are received by the CAN controller and put in a small buffer inside the controller and then an interrupt is issued. This starts a service task in the operating system to fetch these messages and placing them into a larger buffer in RAM. The size of this buffer in RAM is defined at CAN device initialization, the buffer size inside the controller is fixed in hardware. From this buffer in RAM the user application can read and evaluate the CAN messages.

To fetch the CAN messages from the RAM buffer the application calls a function:

|   |   |
|---|---|
| <b>C function:</b>  | <b>CoDeSys function:</b>  |
| WORD <b>can_read</b> (<br>WORD can_nr,<br>CAN_MESSAGE *rd_ptr<br>); | FUNCTION <b>_CAN_READ</b> : WORD<br>VAR_INPUT<br>chan : WORD ;<br>message : POINTER TO IEP_CAN_MESSAGE ;<br>END_VAR |

The parameter `can_nr` submits the number of the CAN controller, the message data is put into a structure the pointer parameter `rd_ptr` points to. If no CAN message is left in the receiving buffer and the function is called again then the function will not return until a CAN message had been received. To avoid this blocking see the function `can_nr_of_messages` in the next section.

The return value of the function has a dual use: Values greater or equal than zero tell the number of remaining messages in the receiving buffer. In case of an error negative values indicate the error number according to the table below:

| Value | C constant:        | Description:                                  |
|-------|--------------------|---|
| >0    |                    | Number of remaining messages in buffer        |
| 0     | E_OK               | No error                                      |
| -1    | E_NO_CAN           | CAN driver not available                      |
| -2    | E_ERROR_LEVEL      | CAN controller reached internal warning level |
| -5    | E_NO_INIT          | No initialization                             |
| -6    | E_OFF_BUS          | CAN not connected to bus                      |
| -7    | E_OVERRUN          | Receiving buffer overrun in CAN controller    |
| -8    | E_READ_BUF_OVERRUN | Receiving buffer overrun in operating system  |
| -9    | E_RESET            | CAN reset, receiving buffer cleared           |

If an error value has been returned it depends on the case of error if data values had been successfully read. The errors `E_OVERRUN` and `E_READ_BUF_OVERRUN` indicate the loss of messages because of buffer overflows but the current function call returned a valid CAN message. The error `E_ERROR_LEVEL` indicates multiple transmission errors on the CAN bus but a valid CAN message is returned nevertheless.

---

## 6.1 Receiving multiple CAN Messages

The function `can_n_read` can be used to read multiple CAN messages from the buffer in one function call:

### C function:

```
WORD can_n_read (  
    WORD can_nr,  
    WORD max,  
    WORD *p_anz,  
    CAN_MESSAGE *rd_ptr  
);
```

### CoDeSys function:

```
FUNCTION _CAN_N_READ : WORD  
VAR_INPUT  
    chan      : WORD ;  
    maxanzahl : WORD ;  
    anzahl    : POINTER TO WORD ;  
    message   : POINTER TO IEP_CAN_MESSAGE ;  
END_VAR
```

The parameter `max` (or `max_anzahl`) submits the maximum number of can messages to be read and `p_anz` (or `anzahl`) returns the actually available and read messages. `p_anz` will be less or equal `max`. If not at least one message is in the buffer then the function call blocks.

The pointer `rd_ptr` (or `message`) is array of message structures. At least `max` number of entries must be present in this array, otherwise memory corruption will occur.

| Value | C constant:                     | Description:                                  |
|-------|---------------------------------|---|
| >0    |                                 | Number of remaining messages in buffer        |
| 0     | <code>E_OK</code>               | No error                                      |
| -1    | <code>E_NO_CAN</code>           | CAN driver not available                      |
| -2    | <code>E_ERROR_LEVEL</code>      | CAN controller reached internal warning level |
| -5    | <code>E_NO_INIT</code>          | No initialization                             |
| -6    | <code>E_OFF_BUS</code>          | CAN not connected to bus                      |
| -7    | <code>E_OVERRUN</code>          | Receiving buffer overrun in CAN controller    |
| -8    | <code>E_READ_BUF_OVERRUN</code> | Receiving buffer overrun in operating system  |
| -9    | <code>E_RESET</code>            | CAN reset, receiving buffer cleared           |

The error conditions are the same as for the `can_read` function.

## 6.2 Checking Messages in the Receive Buffer

If the function `can_read` is called and no message had been currently received, then the function will block until a message is available. To avoid this blocking the function `can_nr_of_messages` can be used beforehand to ask for the currently stored number of messages in the receiving buffer.

### C function:

```
WORD can_nr_of_messages (  
    WORD can_nr,  
    WORD *p_anz  
);
```

### CoDeSys function:

```
FUNCTION _CAN_NR_MESSAGE: WORD  
VAR_INPUT  
    chan      : WORD ;  
    anzahl    : POINTER TO WORD ;  
END_VAR
```

The pointer parameter `p_anz` or `anzahl` will contain the current number of messages in the buffer. The following error conditions can be returned:

---

| <b>Value</b> | <b>C constant:</b> | <b>Description:</b>                          |
|--------------|--------------------|--|
| 0            | E_OK               | No error                                     |
| -1           | E_NO_CAN           | CAN driver not available                     |
| -5           | E_NO_INIT          | No initialization                            |
| -6           | E_OFF_BUS          | CAN not connected to bus                     |
| -7           | E_OVERRUN          | Receiving buffer overrun in CAN controller   |
| -8           | E_READ_BUF_OVERRUN | Receiving buffer overrun in operating system |

The ...overrun errors indicate loss of messages sometimes in the past, this error condition will not be remove. Nevertheless the returned number of messages in the buffer is correct.

---

## 7 Sending CAN Messages

CAN messages are being sent by calling the `can_write` function. The message data to be sent is to be prepared in the structure the pointer `wr_ptr` (or `message`) points to.

### **C function:**

```
WORD can_write (  
    WORD can_nr,  
    CAN_MESSAGE *wr_ptr  
);
```

### **CoDeSys function:**

```
FUNCTION _CAN_WRITE : WORD  
VAR_INPUT  
    chan      : WORD ;  
    message   : POINTER TO IEP_CAN_MESSAGE ;  
END_VAR
```

If the message can not be sent immediately because of other bus activity then the function blocks until the message got sent or an error condition occurred. If this blocking without limit is not acceptable, then the function `can_set_timeout` is available to define a maximum waiting time.

The following error conditions can be returned:

| <b>Value</b> | <b>C constant:</b>         | <b>Description:</b>                           |
|--------------|----------------------------|---|
| 0            | <code>E_OK</code>          | No error                                      |
| -1           | <code>E_NO_CAN</code>      | CAN driver not available                      |
| -2           | <code>E_ERROR_LEVEL</code> | CAN controller reached internal warning level |
| -5           | <code>E_NO_INIT</code>     | No initialization                             |
| -6           | <code>E_OFF_BUS</code>     | CAN not connected to bus                      |
| -9           | <code>E_RESET</code>       | CAN reset, receiving buffer cleared           |
| -10          | <code>E_IDENTIFIER</code>  | Invalid identifier                            |
| -11          | <code>E_LENGTH</code>      | Invalid data length submitted                 |
| -13          | <code>E_RTR</code>         | Invalid remote frame flag                     |
| -15          | <code>E_TIMEOUT</code>     | Timeout while sending message                 |

If `E_ERROR_LEVEL` or `E_OK` is returned then the message had been sent successfully. In all other cases the message had not been sent.

### 7.1 Sending multiple CAN Messages

Similar to receiving multiple messages, multiple messages can be sent, too.

### **C function:**

```
WORD can_n_write (  
    WORD can_nr,  
    WORD anz,  
    WORD *p_anz,  
    CAN_MESSAGE *wr_ptr  
);
```

### **CoDeSys function:**

```
FUNCTION _CAN_N_WRITE : WORD  
VAR_INPUT  
    chan      : WORD ;  
    maxanzahl : WORD ;  
    anzahl    : POINTER TO WORD ;  
    message   : POINTER TO IEP_CAN_MESSAGE ;  
END_VAR
```

An array pointer `wr_ptr` of message structures submits the messages to be sent. The number of messages is submitted in `anz`. In case of an error the parameter `p_anz` contains the number of messages successfully sent up until the error occurred.

---

The following error values are returned by this function:

| Value | C constant:   | Description:                                  |
|-------|---------------|---|
| 0     | E_OK          | No error                                      |
| -1    | E_NO_CAN      | CAN driver not available                      |
| -2    | E_ERROR_LEVEL | CAN controller reached internal warning level |
| -5    | E_NO_INIT     | No initialization                             |
| -6    | E_OFF_BUS     | CAN not connected to bus                      |
| -9    | E_RESET       | CAN reset, receiving buffer cleared           |
| -10   | E_IDENTIFIER  | Invalid identifier                            |
| -11   | E_LENGTH      | Invalid data length submitted                 |
| -13   | E_RTR         | Invalid remote frame flag                     |
| -15   | E_TIMEOUT     | Timeout while sending message                 |

If E\_ERROR\_LEVEL or E\_OK is returned then the message had been sent successfully. In all other cases some or all messages had not been sent. See parameter p\_anz for the number of messages successfully sent.

## **7.2 Timeout for Sending CAN Messages**

A simple case of an otherwise infinitely blocking can\_write function call is established if no other CAN controller is connected to the bus and therefore the message being sent is not acknowledged.

The function can\_set\_timeout sets a maximum time after which an attempt to sent a CAN message is aborted and a

|                        |                                  |
|------------------------|----------------------------------|
| <b>C function:</b>     | <b>CoDeSys function:</b>         |
| WORD can_set_timeout ( | FUNCTION _CAN_SET_TIMEOUT : WORD |
| WORD can_nr,           | VAR_INPUT                        |
| LONG timeout           | chan        : WORD ;             |
| );                     | timeout     : DWORD ;            |
|                        | END_VAR                          |

The parameter timeout specifies the amount of time in units of milliseconds.

| Value | C constant: | Description:             |
|-------|-------------|--------------------------|
| 0     | E_OK        | No error                 |
| -1    | E_NO_CAN    | CAN driver not available |
| -5    | E_NO_INIT   | No initialization        |

## 8 Overview of all Error Messages

| Value | C constant:         | Description:                                  |
|-------|---------------------|---|
| 0     | E_OK                | No error                                      |
| -1    | E_NO_CAN            | CAN controller not available                  |
| -2    | E_ERROR_LEVEL       | CAN controller reached internal warning level |
| -3    | E_WRONG_PHYS        | Invalid interface number                      |
| -4    | E_WRONG_READ_BUFFER | Invalid size of receiving buffer              |
| -5    | E_NO_INIT           | CAN controller not initialized                |
| -6    | E_OFF_BUS           | CAN not connected to bus                      |
| -7    | E_OVERRUN           | Receiving buffer overrun in CAN controller    |
| -8    | E_READ_BUF_OVERRUN  | Receiving buffer overrun in operating system  |
| -9    | E_RESET             | CAN reset, receiving buffer cleared           |
| -10   | E_IDENTIFIER        | Invalid identifier                            |
| -11   | E_LENGTH            | Invalid data length submitted                 |
| -12   | E_WRONG_COMMAND     | Invalid command                               |
| -13   | E_RTR               | Invalid remote frame flag                     |
| -14   | E_NO_MEMORY         | Not enough memory for receiving buffer        |
| -15   | E_TIMEOUT           | Timeout while sending message                 |
| -16   | E_FRAME             | Invalid frame format                          |
| -17   | E_SAMP              | Invalid sampling number (1 or 3 are valid)    |
| -18   | E_BAUDRATE          | Invalid baudrate                              |
| -19   | E_PHASE_SEG1        | Invalid value of phase seg1                   |
| -20   | E_PHASE_SEG2        | Invalid value of phase seg2                   |
| -21   | E_CAN_CLOCK         | Invalid value of can clock                    |
| -22   | E_JUMP_WIDTH        | Invalid value of sjw                          |
| -23   | E_RESET_IMPOSS      | CAN controller does not support Reset         |
| -24   | E_SET_ACCEPT        | Invalid acceptance mask for CAN controller    |
| -25   | E_ETH_MASK          | Invalid ethernet mask                         |
| -26   | E_DUP_INIT          | Multiple initializations not supported        |
| -27   | E_WRONG_ETH_BUFFER  | Invalid Ethernet buffer                       |
| -28   | E_NO_SEMA           | Semaphore error on initialization             |
| -29   | E_NO_FREE_ACCEPT    | No free space found for acceptance id         |
| -30   | E_ACCEPT_NOT_FOUND  | Submitted acceptance id not found in list     |
| -31   | E_NIMPL             | Function is not implemented                   |
| -32   | E_RTR_OVERFLOW      | No further free RTR answer buffer             |
| -33   | E_RTR_NFOUND        | No RTR answer buffer found on Clear           |

Symbolic error constants are not defined for CoDeSys. It definitely eases the readability of the code if they are manually added to the application code.