The Embedded I/O Company

TEWS TECHNOLOGIES

TIP903-SW-95

QNX-Neutrino Device Driver

3 Channel Extended CAN Bus

Version 1.0.x

User Manual

Issue 1.0.0

April 2009

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com
Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TIP903-SW-95

QNX-Neutrino Device Driver

3 Channel Extended CAN Bus

Issue	Description	Date
1.0.0	First Issue	April 09, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP903-SW-95 QNX-Neutrino device driver allows the operation of the TIP903 CAN Bus IP systems and requires the TEWS QNX-Neutrino IPAC Carrier Driver Software (CARRIER-SW-95).

The TIP903 device driver is basically implemented as user installable Resource Manager. The standard file (I/O) functions (*open*, *close* and *devctl*) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

This driver invokes a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

To prevent the application program from losing data, incoming messages will be stored in a message FIFO with a depth of 100 messages.

The TIP903-SW-95 device driver supports the following features:

- Transmit and receive messages with extended and standard identifier
- Up to 15 receive queues with user defined size
- Standard bitrates from 20kbps up to 1Mbps
- Message acceptance filtering
- Definition and update of remote objects

The TIP903-SW-95 device driver supports the modules listed below:

TIP903	3 Channel Extended CAN Bus	IndustryPack®

To get more information about the features and use of TIP903 devices it is recommended to read the manuals listed below.

TIP903 User manual

TIP903 Engineering Manual

CARRIER-SW-95 User Manual

Architectural Overview of the Intel 82527 CAN controller

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-95 is part of this TIP903-SW-95 distribution. It is located in directory CARRIER-SW-95 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-95 User Manual for a detailed description how to install and setup the CARRIER-SW-95 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP903-SW-95':

TIP903-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TIP903-SW-95-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TIP903-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tip903':

driver/tip903.c	TIP903 device driver source
driver/tip903def.h	TIP903 driver include file
driver/tip903.h	TIP903 include file for driver and application
driver/i82537.h	CAN controller include file
driver/node.c	Functions supporting node handling
driver/node.h	Include file for node handling
driver/Makefile	Makefile to create TIP903 driver object
example/tip903exa.c	Example application
example/common.mk	Parameters used for example compilation
example/Makefile	Makefile to build example application (recursive Makefile)
example/nto/Makefile	Makefile to build example application (recursive Makefile)
example/nto/x86/Makefile	Makefile to build example application (recursive Makefile)
example/nto/x86/o/Makefile	Makefile to build example application (recursive Makefile)

In order to perform an installation, copy TIP903-SW-95-SRC.tar.gz to /usr/src and extract all files of the archive. The command 'tar -xzvf TIP903-SW-95-SRC.tar.gz' will extract the files into the local directory.

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header files *ipac_*.h*, which are part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-95* on the distribution media.

Its absolute important to extract the TIP903-SW-95-SRC.tar.gz in the /usr/src directory otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the */usr/src/tip903/driver* directory

Copy the header file tip903.h into */usr/include*

```
# cp tip903.h /usr/include
```

Execute the Makefile

```
# make install
```

After successful completion the driver binary will be installed in the */bin* directory.

2.2 Build the example application

Change to the /usr/src/tip903/example directory

Execute the Makefile

```
# make install
```

After successful completion the example binary (*tip903exa*) will be installed in the */bin* directory.

2.3 Start the driver process

To start the TIP903 Resource Manager (Device Driver) you only have to start the TEWS TECHNOLOGIES IPAC Carrier Driver. The Carrier Driver automatically detects installed TEWS IPAC modules and dynamically loads the concerning module driver(s).

The TIP903 Resource Manager registers a device for each TIP903 CAN channel in the QNX-Neutrino pathname space under following name, where *n* specifies the index of the found modules (please refer to the IPAC Carrier Driver Manual):

```
/dev/tip903_n
```

This pathname must be used in the application program to open a path to the desired TIP903 channel device.

For debugging purposes, you can start the IPAC Carrier Driver with the –V (verbose) option. Now the Resource Manager will print versatile information about TIP903 configuration and command execution on the terminal window. For further details about debugging see IPAC Carrier Driver Manual.

2.4 Receive Queue Configuration

Received CAN messages will be stored in receive queues. Each receive queue contains a FIFO and a separate wait queue. The number of receive queues and the depth of the FIFO can be adapted by changing the following symbols in *tip903def.h*.

NUM_RX_QUEUES	Defines the number of receive queues for the device (default = 3). Valid numbers are in range between 1 and 15.
RX_FIFO_SIZE	Defines the depth of the message FIFO inside each receive queue (default = 100). Valid numbers are in range between 1 and MAXINT

Enter *make install* to create a new driver and to make the changes persistent.

3 I/O Functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int **open** (const char *pathname, int flags)

DESCRIPTION

The **open** function creates and returns a new file descriptor for the TIP903 channel named by *pathname.* The flags argument controls how the file is to be opened. TIP903 channel devices must be opened *O_RDWR*.

EXAMPLE

```
int  fd;

fd = open(“/dev/tip903_0”, O_RDWR);
if (fd == -1)
{
     /* Handle error */
}
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

SEE ALSO

Library Reference - open()

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int **close** (int *filedes*)

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int  fd;

if (close(fd) != 0)
{
     /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Returns only Neutrino specific error code, see Neutrino Library Reference.

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>
```

int **devctl**(int *filedes*, int *dcmd*, void * *data_ptr*, size_t *n_bytes*, int * *dev_info_ptr*);

DESCRIPTION

The **devctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *dcmd* specifies the control code for the operation.

The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.

The argument *dev_info_ptr* is unused for the TIP903 driver and should be set to NULL.

The following devctl command codes are defined in *tip903.h*:

Value	**Meaning**
DCMD_TIP903_READ	Read CAN message from the specified queue
DCMD_TIP903_READ_NOWAIT	Read CAN message from the specified queue and return immediately if queue is empty
DCMD_TIP903_WRITE	Write message to the CAN bus
DCMD_TIP903_BITTIMING	Setup a new bit timing
DCMD_TIP903_SETFILTER	Setup acceptance filter masks
DCMD_TIP903_BUSON	Enter the bus on state
DCMD_TIP903_BUSOFF	Enter the bus off state
DCMD_TIP903_FLUSH	Flush one or all receive queues
DCMD_TIP903_CANSTATUS	Returns the contents of the CAN controller status register
DCMD_TIP903_DEFRXBUF	Define a receive buffer message object
DCMD_TIP903_DEFRMTBUF	Define a remote transmit buffer message object
DCMD_TIP903_UPDATEBUF	Update a remote or receive buffer message object
DCMD_TIP903_RELEASEBUF	Release an allocated message buffer object
DCMD_TIP903_SETTXOBJ	Define default transmit object

See behind for more detailed information on each control code.

To use these TIP903 specific control codes the header file tip903.h must be included in the application.

RETURNS

On success, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

ENOTTY	Inappropriate I/O control operation. This error code is returned if the requested devctl function is unknown. Please check the argument dcmd.

Other function dependent error codes will be described for each devctl code separately. Note, the TIP903 driver always returns standard QNX Neutrino error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TIP903_READ(_NOWAIT)

NAME

DCMD_TIP903_READ(_NOWAIT) – Read CAN message

DESCRIPTION

This function reads a CAN message from the TIP903 device associated with the file descriptor *filedes.* A pointer to the callers message buffer (*TIP903_MSG_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

If currently no message is available the function will be blocked until a new CAN message is received and this request is in front of the receive queue, or a timeout occurs.

To avoid blocking the *DCMD_TIP903_READ_NOWAIT* command should be used instead. This devctl command will return immediately with the error *ENODATA* if no CAN message is available.

```
typedef struct
{
        unsigned long           Identifier;
        long                    Timeout;
        unsigned char           RxQueueNum;
        unsigned char           Extended;
        unsigned char           Status;
        unsigned char           MsgLen;
        unsigned char           Data[8];
} TIP903_MSG_BUF;
```

Identifier

Receives the message identifier of the read CAN message.

Timeout

Specifies the amount of time (in seconds) the caller is willing to wait for execution of read. A value of 0 means wait indefinitely.

RxQueueNum

Specifies the receive queue number from which the data will be read. Valid receive queue numbers are in range between 1 and n. In which n depends on the definition of *NUM_RX_QUEUES*.

Extended

Receives TRUE for extended CAN messages.

Status

This value returns the status information about overrun conditions either in the CAN controller and intermediate software FIFO.

Status Value	**Description**
TIP903_SUCCESS	No messages lost
TIP903_FIFO_OVERRUN	One or more messages were overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TIP903_MSGOBJ_OVERRUN	One or more messages have been overwritten in the CAN controller message object because the interrupt latency is too large. Use message object 15 (buffered) to receive this time critical CAN messages, reduce the CAN bit rate or upgrade the system speed.

MsgLen

Receives the number of message data bytes (0..8).

Data[]

This buffer receives up to 8 data bytes. Data[0] receives message Data 0, Data[1] receives message Data 1 and so on.

EXAMPLE

```
#include  <tip903.h>

int                fd;
int                result;
TIP903_MSG_BUF     MsgBuf;

MsgBuf.RxQueueNum =     1;
MsgBuf.Timeout =        10;       /* seconds */

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_READ, &MsgBuf, sizeof(MsgBuf), NULL);
if(result == EOK)
{
    printf("\nRead successful\n");
    printf("    Identifier: %ld\n", MsgBuf.Identifier);
}
else
{
    printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range.
ENOMEM	No memory available to allocate resources to handle the read command.
ECONNREFUSED	The controller is in bus off state and no message is available in the specified receive queue. Note: As long as CAN messages are available in the receive queue FIFO, bus off conditions were not reported by a read function. This means you can read all CAN messages out of the receive queue FIFO during bus off state without an error result.
ENODATA	Currently no CAN message available for read.
ETIMEDOUT	The allowed time to finish the read request has elapsed.

3.3.2 DCMD_TIP903_WRITE

NAME

DCMD_TIP903_WRITE - Write a CAN message

DESCRIPTION

This function writes a CAN message to the TIP903 device associated with the file descriptor *filedes.* A pointer to the callers message buffer (*TIP903_MSG_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

If the CAN controller is busy transmitting another message the function is blocked until all previous pending requests are serviced or a timeout occurs.

Keep in mind to configure the default transmit message object with the control function *DCMD_TIP903_SETTXOBJ* before.

```
typedef struct
{
        unsigned long           Identifier;
        long                    Timeout;
        unsigned char           RxQueueNum;
        unsigned char           Extended;
        unsigned char           Status;
        unsigned char           MsgLen;
        unsigned char           Data[8];
} TIP903_MSG_BUF;
```

Identifier

Contains the message identifier of the CAN message to write.

Timeout

Specifies the amount of time (in seconds) the caller is willing to wait for execution of write. A value of 0 means wait indefinitely.

RxQueueNum

Is unused and can be set to 0.

Extended

Specifies if the message should be sent as an extended (*TRUE*) or standard (*FALSE*) CAN messages.

Status

Is unused.

MsgLen

Specifies the number of message data bytes (0..8).

Data[]

Specifies the buffer with transmit data.

EXAMPLE

```
#include  <tip903.h>

int                 fd;
int                 result;
TIP903_MSG_BUF      MsgBuf;

MsgBuf.Identifier =     1234;
MsgBuf.Timeout =        2;   /* 2 sec*/
MsgBuf.Extended =       TRUE;
MsgBuf.MsgLen =         2;
MsgBuf.Data[0] =        0xaa;
MsgBuf.Data[1] =        0x55;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_WRITE, &MsgBuf, sizeof(MsgBuf), NULL);
if(result == EOK)
{
     printf("\nWrite successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.
ENOMEM	No memory available to allocate resources to handle the write command.
ECONNREFUSED	The controller is in bus off state and unable to transmit messages.
EMSGSIZE	Invalid message size. *MsgBuf.MsgLen* must be in range between 0 and 8.
ETIMEDOUT	The allowed time to finish the write request has elapsed. This occurs if currently no message object is available or if the CAN bus is overloaded and the priority of the message identifier is too low.

3.3.3 DCMD_TIP903_BITTIMING

NAME

DCMD_TIP903_BITTIMING - Setup new bit timing

DESCRIPTION

This function modifies the bit timing register of the CAN controller to setup a new CAN bus transfer speed to the TIP903 device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*TIP903_BITTIMING*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

Keep in mind to setup a valid bit timing value before changing into the Bus On state.

```
typedef struct
{
        unsigned short      TimingValue;
        unsigned short      ThreeSamples;
} TIP903_BITTIMING;
```

TimingValue

This parameter holds the new values for the bit timing register 0 (bit 0..7) and for the bit timing register 1 (bit 8..15). Possible transfer rates are between 20 kbit per second and 1.0 Mbit per second. The include file tip903.h contains predefined transfer rate symbols (TIP903_20KBIT .. TIP903_1MBIT).

ThreeSamples

If this parameter is *TRUE* (1) the CAN bus is sampled three times per bit time instead of one.

Use one sample point for faster bit rates and three sample points for slower bit rates to make the CAN bus more resistant against noise spikes.

EXAMPLE

```
#include  <tip903.h>

int                 fd;
int                 result;
TIP903_BITTIMING   BitTime;

BitTime.TimingValue =   TIP903_100KBIT;
BitTime.ThreeSamples =  FALSE;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_BITTIMING,
                &BitTime, sizeof(BitTime), NULL);
if(result == EOK)
{
     printf("\nSet bittiming successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

3.3.4 DCMD_TIP903_SETFILTER

NAME

DCMD_TIP903_SETFILTER - Setup acceptance filter masks

DESCRIPTION

This function modifies the acceptance filter masks of the TIP903 device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*TIP903_ACCEPT_MASKS*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The acceptance masks allow message objects to receive messages with a larger range of message identifiers instead of just a single message identifier. A "0" value means "don't care" or accept a "0" or "1" for that bit position. A "1" value means that the incoming bit value "must-match" the corresponding bit in the message identifier.

```
typedef struct
{
    unsigned long     Message15Mask;
    unsigned long     GlobalMaskExtended;
    unsigned short    GlobalMaskStandard;
} TIP903_ACCEPT_MASKS;
```

Message15Mask

This parameter specifies the value for the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier mask appears in bit 3..31 of this parameter.
The Message 15 Mask is "ANDed" with the Global Mask. This means that any bit defined as "don't care" in the Global Mask will automatically be a "don't care" bit for message 15.

GlobalMaskExtended

This parameter specifies the value for the Global Mask-Extended Register. The Global Mask-Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier mask appears in bit 3..31 of this parameter.

GlobalMaskStandard

This parameter specifies the value for the Global Mask-Standard Register. The Global Mask-Standard Register applies only to messages using the standard CAN identifier. The 11 bit identifier mask appears in bit 5..15 of this parameter.

The TIP903 device driver copies the parameter directly into the corresponding registers of the CAN controller, without shifting any bit positions. For further information refer to the Intel 82527 Architectural Overview - *4.7…4.10*

EXAMPLE

```
#include  <tip903.h>

int                     fd;
int                     result;
TIP903_ACCEPT_MASKS     AcceptMask;

/* Standard identifier bits 0..3 don't care */
AcceptMask.GlobalMaskStandard = 0xfe00;

/* Extended identifier bits 0..3 don't care */
AcceptMask.GlobalMaskExtended = 0xffffff80;

/* Message object 15 identifier bits 0..7 don't care */
AcceptMask.Message15Mask = 0xfffff800;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_SETFILTER,
                &AcceptMask, sizeof(AcceptMask), NULL);
if(result == EOK)
{
     printf("\nSet acceptance filter successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

3.3.5 DCMD_TIP903_BUSON

NAME

DCMD_TIP903_BUSON - Enter the bus on state

DESCRIPTION

This function sets the TIP903 device associated with the file descriptor *filedes* into BusOn state. The function does not use the parameters *data_ptr* and *n_bytes*, they must be set to 0.

After an abnormal rate of occurrences of errors on the CAN bus or after driver startup, the CAN controller enters the Bus Off state. This control function resets the init bit in the control register. The CAN controller begins the BusOff recovery sequence and resets both transmit and receive error counters. If the CAN controller counts 128 packets of 11 consecutive recessive bits on the CAN bus, the Bus Off state is exited.

Before the driver is able to communicate over the CAN bus after driver startup, this control function must be executed.

EXAMPLE

```
#include  <tip903.h>

int                     fd;
int                     result;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_BUSON, NULL, 0, NULL);
if(result == EOK)
{
    printf("\nSet Bus On successful\n");
}
else
{
    printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

3.3.6 DCMD_TIP903_BUSOFF

NAME

DCMD_TIP903_BUSON - Enter the bus off state

DESCRIPTION

This function sets the TIP903 device associated with the file descriptor *filedes* into BusOff state. The function does not use the parameters *data_ptr* and *n_bytes*, they must be set to 0.

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function *DCMD_TIP903_BUSON* is executed.

Execute this control function before the last close to the CAN controller channel.

EXAMPLE

```
#include  <tip903.h>

int                     fd;
int                     result;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_BUSOFF, NULL, 0, NULL);
if(result == EOK)
{
    printf("\nSet Bus Off successful\n");
}
else
{
    printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

3.3.7 DCMD_TIP903_FLUSH

NAME

DCMD_TIP903_FLUSH - Flush one or all receive queues

DESCRIPTION

This function flushes one or all receive buffers of the TIP903 device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*int*) and the size of the parameter buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

The value in the callers parameter buffer specifies the queue number that shall be flushed. Valid queue numbers are 1 up to the maximum number of defined receive queues. If a 0 is specified as receive queue number all queues will be flushed.

EXAMPLE

```
#include  <tip903.h>

int                 fd;
int                 result;
int                 queueNum;

queueNum =    2;        /* Flush receive queue 2 */

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_FLUSH, &queueNum, sizeof(queueNum), NULL);
if(result == EOK)
{
     printf("\Flush receive successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the specified receive queue is out of range.

3.3.8 DCMD_TIP903_CANSTATUS

NAME

DCMD_TIP903_CANSTATUS - Returns the contents of the CAN status register

DESCRIPTION

This function returns the content of the controller status register of the TIP903 device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*int*) and the size of the parameter buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

The CAN controller status can be used for diagnostic purposes.

EXAMPLE

```
#include  <tip903.h>

int                fd;
int                result;
int                CANstat;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_CANSTATUS,
                &CANstat, sizeof(CANstat), NULL);
if(result == EOK)
{
     printf("\nRead CAN status successful\n");
     printf("CAN controller status: %02Xh\n", CANstat);
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

3.3.9 DCMD_TIP903_DEFRXBUF

NAME

DCMD_TIP903_DEFRXBUF - Define a receive buffer message object

DESCRIPTION

This function defines a CAN message object to receive a single message identifier or a range of message identifiers (see also Acceptance Mask) for the device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*TIP903_BUF_DESC*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

All CAN messages received by this message object are directed to the associated receive queue and can be read with the *DCMD_TIP903_READ* command.

Before the driver can receive CAN messages it is necessary to define at least one receive message object. If only one receive message object is defined at all preferably message object 15 should be used because this message object is double-buffered.

```
typedef struct
{
        unsigned long       Identifier;
        unsigned char       MsgObjNum;
        unsigned char       RxQueueNum;
        unsigned char       Extended;
        unsigned char       MsgLen;
        unsigned char       Data[8];
} TIP903_BUF_DESC;
```

Identifier

Specifies the message identifier the message object shall accept. This value will be combined with the selected acceptance masks.

MsgObjNum

Specifies the number of the message object to be defined. Valid object numbers are in range between 1 and 15.

RxQueueNum

Specifies the associated receive queue for this message object. All CAN messages received by this object are directed to this receive queue. The receive queue number is one based. Valid numbers are in range between 1 and n. In which n depends on the definition of *NUM_RX_QUEUES*.

It is possible to assign more than one receive message object to the same receive queue.

Extended

Set to *TRUE* to accept extended CAN messages or *FALSE* to accept standard CAN messages.

MsgLen

Not used, set to 0.

Data[]

Not used.

EXAMPLE

```
#include  <tip903.h>

int                 fd;
int                 result;
TIP903_BUF_DESC     BufDesc;

BufDesc.MsgObjNum =      15;
BufDesc.RxQueueNum =     1;
BufDesc.Identifier =     1234;
BufDesc.Extended =       TRUE;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_DEFRXBUF,
                &BufDesc, sizeof(BufDesc), NULL);
if(result == EOK)
{
     printf("\nReceive CAN object definition successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the specified receive queue is out of range.
EADDRINUSE	The requested message object is already occupied.

3.3.10 DCMD_TIP903_DEFRMTBUF

NAME

DCMD_TIP903_DEFRMTBUF - Define a remote transmit buffer message object

DESCRIPTION

This function defines a remote transmission CAN message buffer object for the device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*TIP903_BUF_DESC*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

A remote transmission object is similar to normal transmission object except that the CAN message is transmitted only after receiving a remote frame with the same identifier. This type of message object can be used to make process data available for other nodes which can be polled around the CAN bus without any action of the provider node. The message data remain available for other CAN nodes until this message object is updated with the control function *DCMD_TIP903_UPDATEBUF* or cancelled with *DCMD_TIP903_RELEASEBUF*.

```
typedef struct
{
        unsigned long       Identifier;
        unsigned char       MsgObjNum;
        unsigned char       RxQueueNum;
        unsigned char       Extended;
        unsigned char       MsgLen;
        unsigned char       Data[8];
} TIP903_BUF_DESC;
```

Identifier

Specifies the message identifier the message object will accept. This value will be combined with the selected acceptance asks.

MsgObjNum

Specifies the number of the message object to be defined. Valid object numbers are in range between 1 and 14.
Keep in mind that message object 15 is only available for receive message objects.

RxQueueNum

Unused for remote transmission message objects. Set to 0

Extended

Set to *TRUE* for a remote CAN messages with extended identifier or *FALSE* for remote CAN messages with standard identifier.

MsgLen

Specifies the number of message data bytes (0..8).

Data[]

This buffer contains up to 8 data bytes that will be transmitted.

EXAMPLE

```
#include <tip903.h>

int                fd;
int                result;
TIP903_BUF_DESC    BufDesc;

/* Define message object 10 to transmit the extended     */
/* message identifier 777 after receiving a remote       */
/* frame with the same identifier                        */
BufDesc.MsgObjNum =     10;
BufDesc.Identifier =    777;
BufDesc.Extended =      TRUE;
BufDesc.MsgLen =        1;
BufDesc.Data[0] =       123;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_DEFRMTBUF,
                &BufDesc, sizeof(BufDesc), NULL);
if(result == EOK)
{
     printf("\nTransmit CAN object definition successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the specified message object number is out of range.
EADDRINUSE	The requested message object is already occupied.
EMSGSIZE	Invalid message size. *BufDesc.MsgLen* must be in range between 0 and 8

3.3.11 DCMD_TIP903_UPDATEBUF

NAME

DCMD_TIP903_UPDATEBUF - Update a remote or receive buffer message object

DESCRIPTION

This function updates a previously defined receive or remote transmission message buffer object for the device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*TIP903_BUF_DESC*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

To update a receive message object a remote frame is transmitted over the CAN bus to request new data from a corresponding remote transmission message object on other nodes. Received message by this message object will be stored in the associated receive queue.

To update a remote transmission object only the message data and message length of the specified message object is changed. No transmission is initiated by this control function.

```
typedef struct
{
        unsigned long       Identifier;
        unsigned char       MsgObjNum;
        unsigned char       RxQueueNum;
        unsigned char       Extended;
        unsigned char       MsgLen;
        unsigned char       Data[8];
} TIP903_BUF_DESC;
```

Identifier

Unused for this control function. Set to 0.

MsgObjNum

Specifies the number of the message object to be updated. Valid object numbers are in range between 1 and 14.
Keep in mind that message object 15 is available only for receive message objects.

RxQueueNum

Unused for this control function. Set to 0.

Extended

Set to *TRUE* for a remote CAN messages with extended identifier or *FALSE* for remote CAN messages with standard identifier.

MsgLen

Contains the number of message data bytes (0..8). This parameter is used only for remote transmission object updates.

Data[]

This buffer contains up to 8 data bytes. This parameter is used only for remote transmission object updates.

EXAMPLE

```
#include  <tip903.h>

int                 fd;
int                 result;
TIP903_BUF_DESC     BufDesc;

/* Update a receive message object */
BufDesc.MsgObjNum =     14;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_UPDATEBUF,
                &BufDesc, sizeof(BufDesc), NULL);
if(result == EOK)
{
     printf("\nUpdate receive remote object successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}

/* Update a remote message object */
BufDesc.MsgObjNum =     10;
BufDesc.MsgLen =        1;
BufDesc.Data[0] =       124;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_UPDATEBUF,
                &BufDesc, sizeof(BufDesc), NULL);
if(result == EOK)
{
     printf("\nUpdate transmit remote object successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if either the message object number is out of range or the requested message object is not defined.
EMSGSIZE	Invalid message size. *BufDesc.MsgLen* must be in range between 0 and 8

3.3.12 DCMD_TIP903_RELEASEBUF

NAME

DCMD_TIP903_RELEASEBUF - Release an allocated message buffer object

DESCRIPTION

This function releases a previously defined CAN message object of the device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*TIP903_BUF_DESC*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

After releasing the message object, it can be defined again with *DCMD_TIP903_DEFRXBUF* or *DCMD_TIP903_DEFRMTBUF* control functions.

```
typedef struct
{
        unsigned long       Identifier;
        unsigned char       MsgObjNum;
        unsigned char       RxQueueNum;
        unsigned char       Extended;
        unsigned char       MsgLen;
        unsigned char       Data[8];
} TIP903_BUF_DESC;
```

Identifier

Unused for this control function. Set to 0.

MsgObjNum

Specifies the number of the message object to be released. Valid object numbers are in range between 1 and 15.

RxQueueNum

Unused for this control function. Set to 0.

Extended

Unused for this control function. Set to 0.

MsgLen

Unused for this control function. Set to 0.

Data[]

This buffer is unused.

EXAMPLE

```
#include  <tip903.h>

int                 fd;
int                 result;
TIP903_BUF_DESC     BufDesc;


/* Release message object */
BufDesc.MsgObjNum =     14;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_RELEASEBUF,
                 &BufDesc, sizeof(BufDesc), NULL);
if(result == EOK)
{
     printf("\nrelease object successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the message object number is out of range.
EBADMSG	The requested message object is not defined.
EBUSY	The message object is currently busy transmitting data.

3.3.13 DCMD_TIP903_SETTXOBJ

NAME

DCMD_TIP903_SETTXOBJ - Define a default transmit message object

DESCRIPTION

This function defines the message object, which is used to transmit CAN messages with the *DCMD_TIP903_WRITE* control function on the device associated with the file descriptor *filedes.* A pointer to the callers parameter buffer (*int*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

The callers parameter buffer specifies the number of the message object that shall be used for transmit requests. Valid transmission object numbers are in range between 1..14.

As long as the default transmission object is unknown no messages can be sent with the *DCMD_TIP903_WRITE* control function.

EXAMPLE

```
#include  <tip903.h>

int            fd;
int            result;
int            txObjNum;

/* Init transmit message object */
txObjNum =     1;

/* Send request to the device driver */
result = devctl(fd, DCMD_TIP903_SETTXOBJ,
                &txObjNum, sizeof(txObjNum), NULL);
if(result == EOK)
{
     printf("\Define default transmit object successful\n");
}
else
{
     printf("devctl failed (Error = %d) : %s\n", result, strerror(result));
}
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the specified message object number is out of range or the desired message object is already occupied.

3.4 Step by Step Driver Initialization

The following code example illustrates all necessary steps to initialize a CAN device for communication.

```
/* (1) Setup CAN bus bit timing */
BitTimingParam.TimingValue  =    TIP903_100KBIT;
BitTimingParam.ThreeSamples =    FALSE;

result = devctl( fd,
                 DCMD_TIP903_BITTIMING,
                 &BitTimingParam,
                 sizeof(BitTimingParam),
                 NULL);

/* (2) Setup acceptance filter masks */
AcceptMasksParam.GlobalMaskStandard    =    0xFFFF;
AcceptMasksParam.GlobalMaskExtended    =    0xFFFFFFFF;
AcceptMasksParam.Message15Mask         =    0;

result = devctl( fd,
                 DCMD_TIP903_SETFILTER,
                 &AcceptMasksParam,
                 sizeof(AcceptMasksParam),
                 NULL);

/* (3) Define msg obj 15 for reception (receive all identifiers) */
BufDesc.MsgObjNum      =    15;
BufDesc.RxQueueNum     =    1;
BufDesc.Identifier     =    0;
BufDesc.Extended       =    TRUE;

result = devctl( fd,
                 DCMD_TIP903_DEFRXBUF,
                 &BufDesc,
                 sizeof(BufDesc),
                 NULL);

/* (4) Define default transmit message object */
TxObjectNum =    1;

result = devctl( fd,
                 DCMD_TIP903_SETTXOBJ,
                 &TxObjectNum,
                 sizeof(TxObjectNum),
                 NULL);

/* (5) Enter Bus On State */
result = devctl(fd, DCMD_TIP903_BUSON, NULL, 0, NULL);
```