The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV010-SW-95

QNX - Neutrino Device Driver

Isolated 2x CAN Bus

Version 1.1.x

User Manual

Issue 1.1.0

November 2017

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

TDRV010-SW-95

QNX - Neutrino Device Driver

Isolated 2x CAN Bus

Supported Modules:
TPMC810
TPMC310

Issue	Description	Date
1.0.0	First Issue	March 19, 2009
1.0.1	Update Contact Information	December 21, 2010
1.0.2	Chapter for Driver Installation with Momentics added	September 8, 2016
1.1.0	Support for QNX7 added	November 20, 2017

Table of Contents

1 Introduction

The TDRV010-SW-95 QNX-Neutrino device driver allows the operation of the supported CAN Bus devices on QNX-Neutrino operating systems.

The TDRV010 device driver is basically implemented as a 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.

The TDRV010-SW-95 device driver supports the following features:

- Transmission and receive of Standard and Extended Identifiers
- Standard bit rates from 50 kbit up to 1 Mbit and user defined bit rates
- Message acceptance filtering
- Single-Shot transmission
- Listen only mode
- Message self-reception
- Programmable error warning limit

The TDRV010-SW-95 device driver supports the modules listed below:

TPMC310	Isolated 2 x CAN Bus	(PMC, Conduction Cooled)
TPMC810	Isolated 2 x CAN Bus	(PMC)

In this document all supported modules and devices will be called TDRV010. Specials for certain devices will be advised.

To get more information about the features and use of TDRV010 devices it is recommended to read the manuals listed below.

TPMC310, TPMC810 User Manual
SJA1000 CAN Controller Manual

2 Installation

Following files are located in the directory TDRV010-SW-95 on the distribution media:

TDRV010-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV010-SW-95-1.1.0.pdf	This manual in PDF format
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TDRV010-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv010':

/driver/tdrv010.c	Driver source code
/driver/tdrv010.h	Definitions and data structures for driver and application
/driver/tdrv010def.h	Device driver include
/driver/ sja1000.h	Philips SJA1000 CAN controller definitions
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/driver/nto/*	Build path
/example/example.c	Example application
/example/nto/*	Build path

2.1 Building Executables on Native Systems

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar –xzvf TDRV010-SW-95-SRC.tar.gz`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tdrv010*.

It is absolutely important to extract the TDRV010 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

2.1.1 Build the Device Driver

Change to the /usr/src/tdrv010/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tdrv010) will be installed in the /bin directory.

2.1.2 Build the Example Application

Change to the /usr/src/tdrv010/example directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (***tdrv010exa***) will be installed in the /bin directory.

2.2 Building Executables with Momentics IDE (5.0)

This chapter gives just a simple description how to build the drivers with the Momentics IDE (5.0), for more detailed information please refer to the appropriate documentation.

For installation unpack the tar-archive into the desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tdrv010*.

2.2.1 Build the Device Driver

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV010)
- Select the path “tdrv010\driver” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)

Now the device driver can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binary of tdrv010 device driver. (e.g. “tdrv010 – [x86/le]”)

2.2.2 Build the Example Application

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV010-Example)
- Select the path “tdrv010\example” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)

Now the example can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binary of tdrv010 example application. (e.g. “tdrv010exa – [x86/le]”)

2.2.3 Integrate the Device Driver Files to a QNX-Image

To add the device driver file and the example application file to a QNX-Image, just a few steps are necessary.

Copy the desired binary files of the device driver and example project into “sbin” beneath the “install”-path of the target project using the Momentics-IDE.

Add the filenames of the added files into the build-file (e.g. “x86-generic.build”) in “images”. For example the filenames (e.g. tdrv010, tdrv010exa) can be inserted behind the serial driver names (insert each filename in a separate line).

After a rebuild of the QNX-Image, the driver files will be available on the disk and can be used after booting.

2.3 Building Executables with Momentics IDE (7.0)

This chapter gives just a simple description how to build the drivers with the Momentics IDE (7.0), for more detailed information please refer to the appropriate documentation.

For installation unpack the tar-archive into the desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tdrv010*.

2.3.1 Build the Device Driver

Create a new project ("Makefile Project with Existing Code") in your workspace:
- Select a "Project Name" (e.g. TDRV010)
- Select the path "tdrv010\driver" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")

Now we have to specify the name of the driver executable and additional libraries needed for the driver. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:
- NAME = tdrv010
- LIBS = pci

Now the device driver can be built by "Building the Project".

After successful completion the IDE shows a "Binaries"-path containing the built binaries of tdrv010 device driver of the enabled configurations (e.g. "tdrv010 – [x86/le]" and "tdrv010 – [x86_64/le]").

2.3.2 Build the Example Application

Create a new project ("Makefile Project with Existing Code") in your workspace:
- Select a "Project Name" (e.g. TDRV010-Example)
- Select the path "tdrv010\example" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")

Now we have to specify the name of the driver example executable. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:
- NAME = tdrv010exa

Now the example can be built by "Building the Project".

After successful completion the IDE shows a "Binaries"-path containing the built binaries of tdrv010 example application of the enabled configurations. (e.g. "tdrv010exa – [x86/le]" and "tdrv010exa – [x86_64/le]")

2.3.3 Integrate the Device Driver Files to a QNX-Image

To add the device driver file and the example application file to a QNX-Image, just a few steps are necessary.

Copy the desired binary files of the device driver and example project into “sbin” beneath the “install”-path of the target project using the Momentics-IDE.

Add the filenames of the added files into the build-file (e.g. “x86-generic.build”) in “images”. For example the filenames (e.g. tdrv010, tdrv010exa) can be inserted behind the serial driver names (insert each filename in a separate line).

After a rebuild of the QNX-Image, the driver files will be available on the disk and can be used after booting.

2.4 Start the Driver Process

To start the TDRV010 device driver, you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tdrv010 [-v] &
```

The TDRV010 Resource Manager creates one device for each CAN channel, and registers the created devices in the Neutrinos pathname space under following names.

```
/dev/tdrv010_0
/dev/tdrv010_1
…
/dev/tdrv010_x
```

The reference between the created device names and the physical devices depends on the search order of the PCI bus driver. The TDRV010 searches for supported devices in the following order: TPMC810, TPMC310.

Example: A system with 1x TPMC810-10, and 2x TPMC310-10 will create the following devices:

Module	Local Channel	Device Name
TPMC810-10	1	/dev/tdrv010_0
TPMC810-10	2	/dev/tdrv010_1
TPMC310-10 (1st)	1	/dev/tdrv010_2
TPMC310-10 (1st)	2	/dev/tdrv010_3
TPMC310-10 (2nd)	1	/dev/tdrv010_4
TPMC310-10 (2nd)	2	/dev/tdrv010_5

The pathname must be used in the application program to open a path to the desired TDRV010 device.

```
fd = open("/dev/tdrv010_0", O_RDWR);
```

For debugging, you can start the TDRV010 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TDRV010 configuration and command execution on the terminal window.

```
tdrv010 –v &
```

Make sure that only one instance of the device driver process is started.

2.5 Receive Queue Configuration

Received CAN messages will be stored in a FIFO buffer. The depth of the FIFO can be adapted by changing the following symbol in tdrv010def.h.

TDRV010_RX_FIFO_SIZE

Defines the depth of the message FIFO buffer (default = 100). Valid numbers are in range between 1 and MAXINT.

3 Device Input/Output 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 TDRV010 device named by pathname. The flags argument controls how the file is to be opened. TDRV010 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open(“/dev/tdrv010_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 TDRV010 driver and should be set to NULL.

The following devctl command codes are defined in *tdrv010.h*:

Value	Description
DCMD_TDRV010_READ	Read a CAN message from the specified queue
DCMD_TDRV010_READ_NOWAIT	Read CAN message from the specified queue and return immediately if queue is empty
DCMD_TDRV010_WRITE	Write message to the CAN bus
DCMD_TDRV010_BITTIMING	Setup a new bit timing
DCMD_TDRV010_SETFILTER	Setup acceptance filter
DCMD_TDRV010_BUSON	Enter the bus on state
DCMD_TDRV010_BUSOFF	Enter the bus off state
DCMD_TDRV010_FLUSH	Flush one or all receive queues

DCMD_TDRV010_CANSTATUS	Returns CAN controller status information
DCMD_TDRV010_ENABLE_SELFTEST	Enable self-test mode
DCMD_TDRV010_DISABLE_SELFTEST	Disable self-test mode
DCMD_TDRV010_ENABLE_LISTENONLY	Enable listen only mode
DCMD_TDRV010_DISABLE_LISTENONLY	Disable listen only mode
DCMD_TDRV010_SETLIMIT	Set new error warning limit
DCMD_TDRV010_TRANSCEIVER_OPERATING	Set transceivers to operating state (TPMC310 only)
DCMD_TDRV010_TRANSCEIVER_SILENT	Set transceivers to silent state (TPMC310 only)

See behind for more detailed information on each control code.

To use these TDRV010 specific control codes the header file tdrv010.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

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

Other function dependent error codes will be described for each devctl code separately. Note, the TDRV010 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TDRV010_READ(_NOWAIT)

NAME

DCMD_TDRV010_READ(_NOWAIT) – Read a CAN message

DESCRIPTION

The read function reads a CAN message from the driver receive queue. A pointer to the callers message buffer (TDRV010_MSG_BUF) and the size of this structure are passed by the parameters data_ptr and n_bytes to the device.

```
typedef struct {
        unsigned int            Identifier;
        unsigned char           IOFlags;
        unsigned char           MsgLen;
        unsigned char           Data[8];
        int                     Timeout;
        unsigned char           Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

Identifier

Receives the message identifier of the read CAN message.

IOFlags

Receives CAN message attributes as a set of bit flags. The following attribute flags are possible:

Value	Description
TDRV010_EXTENDED	Set if the received message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the received message is a remote transmission request (RTR) frame.

MsgLen

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

Data[8]

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

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.

Status

Receives status information about overrun conditions either in the CAN controller or intermediate software FIFO.

Value	Description
TDRV010_SUCCESS	No messages lost
TDRV010_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.
TDRV010_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message FIFO because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_MSG_BUF    MsgBuf;

MsgBuf.Timeout = 10;    /* seconds */

result = devctl(   fd,
                   DCMD_TDRV010_READ,
                   &MsgBuf,
                   sizeof(MsgBuf),
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.
ENOMEM	No memory available to allocated 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 (only DCMD_TDRV010_READ_NOWAIT).
ETIMEDOUT	The allowed time to finish the read request is elapsed.

3.3.2 DCMD_TDRV010_WRITE

NAME

DCMD_TDRV010_WRITE - Write a CAN message

DESCRIPTION

This devctl function writes a message to the CAN bus. A pointer to the callers message buffer (*TDRV010_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 a message the caller becomes blocked until all previous pending requests are serviced or a timeout occurs.

```
typedef struct {
        unsigned int            Identifier;
        unsigned char           IOFlags;
        unsigned char           MsgLen;
        unsigned char           Data[8];
        int                     Timeout;
        unsigned char           Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

Identifier

Contains the message identifier of the CAN message to write.

IOFLags

Contains a set of bit flags, which define message attributes and controls the write operation. To set more than one bit flag the predefined macros must be binary OR'ed.

Value	Description
TDRV010_EXTENDED	Transmit an extended message frame. If this macro isn't set or the "dummy" macro TDRV010_STANDARD is set a standard frame will be transmitted.
TDRV010_REMOTE_FRAME	A remote transmission request (RTR bit is set) will be transmitted.
TDRV010_SINGLE_SHOT	No re-transmission will be performed if an error occurred or the arbitration will be lost during transmission (single-shot transmission).
TDRV010_SELF_RECEPTION	The message will be transmitted and simultaneously received if the acceptance filter is set to the corresponding identifier.

MsgLen

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

Data[8]

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

Timeout

Specifies the amount of time (in seconds) the caller is willing to wait for execution of write.

Status

Unused for this control function.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_MSG_BUF    MsgBuf;

MsgBuf.Identifier  = 1234;
MsgBuf.Timeout     = 2;      /* sec*/
MsgBuf.IOFlags     = TDRV010_EXTENDED;
MsgBuf.MsgLen      = 2;
MsgBuf.Data[0]     = 0xaa;
MsgBuf.Data[1]     = 0x55;

result = devctl(   fd,
                   DCMD_TDRV010_WRITE,
                   &MsgBuf,
                   sizeof(MsgBuf),
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.
ENOMEM	No memory available to allocated resources to handle the read 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 is elapsed.

3.3.3 DCMD_TDRV010_BITTIMING

NAME

DCMD_TDRV010_BITTIMING – Setup new bit timing

DESCRIPTION

This devctl function modifies the bit timing register of the CAN controller to setup a new CAN bus transfer speed. A pointer to the callers parameter buffer (*TDRV010_TIMING*) 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;
}TDRV010_TIMING, *PTDRV010_TIMING;
```

TimingValue

This parameter holds the new value for the bit timing register 0 (bit 0...7) and for the bit timing register 1 (bit 8...15). Possible transfer rates are between 50 Kbit per second and 1 Mbit per second. The include file 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_50KBIT ... TDRV010_1MBIT).
For other transfer rates please follow the instructions of the *SJA1000 Product Specification*.

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 rate to make the CAN bus more immune against noise spikes.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_TIMING     BitTimingParam;


BitTimingParam.TimingValue   = TDRV010_100KBIT;
BitTimingParam.ThreeSamples  = FALSE;

result = devctl(   fd,
                   DCMD_TDRV010_BITTIMING,
                   &BitTimingParam,
                   sizeof(BitTimingParam),
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

SEE ALSO

tdrv010.h for predefined bus timing constants.

SJA1000 Product Specification Manual – 6.5.1/2 BUS TIMING REGISTER.

3.3.4 DCMD_TDRV010_SETFILTER

NAME

DCMD_TDRV010_SETFILTER - Setup acceptance filter

DESCRIPTION

This devctl function modifies the acceptance filter of the specified CAN controller device.

The acceptance filter compares the received identifier with the acceptance filter and decides whether a message should be accepted or not. If a message passes the acceptance filter it is stored in the receive FIFO.

The acceptance filter is defined by the acceptance code registers and the acceptance mask registers. The bit patterns of messages to be received are defined in the acceptance code register.

The corresponding acceptance mask registers allow defining certain bit positions to be "don't care" (a 1 at a bit position means "don't care").

A pointer to the callers parameter buffer (*TDRV010_FILTER*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct {
        int             SingleFilter;
        unsigned int    AcceptanceCode;
        unsigned int    AcceptanceMask;
} TDRV010_FILTER, *PTDRV010_FILTER;
```

SingleFilter

Set TRUE (1) for single filter mode.
Set FALSE (0) for dual filter mode.

AcceptanceCode

The content of this parameter will be written to acceptance code register of the controller.

AcceptanceMask

The content of this parameter will be written to the acceptance mask register of the controller.

A detailed description of the acceptance filter and possible filter modes can be found in the SJA1000 Product Specification Manual.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_FILTER     AcceptFilter;


/* Not relevant because all bits are "don't care" */
AcceptFilter.AcceptanceCode  = 0x0;

/* Mark all bit position don't care */
AcceptFilter.AcceptanceMask  = 0xffffffff;

/* Single Filter Mode */
AcceptFilter.SingleFilter    = 1; // TRUE

result = devctl(   fd,
                   DCMD_TDRV010_SETFILTER,
                   & AcceptFilter,
                   sizeof(AcceptFilter),
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the acceptance filter.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER*

3.3.5 DCMD_TDRV010_BUSON

NAME

DCMD_TDRV010_BUSON - Enter the bus ON state

DESCRIPTION

This devctl function sets the specified CAN controller into the bus ON state.

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 "reset mode" bit in the mode register. The CAN controller begins the Bus OFF recovery sequence and resets the 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 “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_BUSON,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
ECONNREFUSED	Unable to enter the Bus ON mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD).*

3.3.6 DCMD_TDRV010_BUSOFF

NAME

DCMD_TDRV010_BUSOFF - Enter the bus OFF state

DESCRIPTION

This devctl function sets the specified CAN controller into the bus OFF state.

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function DCMD_TDRV010_BUSON is executed.

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_BUSOFF,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EIO	Unable to enter the bus OFF mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD).*

3.3.7 DCMD_TDRV010_FLUSH

NAME

DCMD_TDRV010_FLUSH - Flush the received message FIFO

DESCRIPTION

This devctl function flushes the FIFO buffer of received CAN messages.

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_FLUSH,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

3.3.8 DCMD_TDRV010_CANSTATUS

NAME

DCMD_TDRV010_CANSTATUS - Returns CAN controller status information

DESCRIPTION

This devctl function returns the actual contents of several CAN controller registers for diagnostic purposes.

A pointer to the callers status buffer (*TDRV010_STATUS*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct {
        unsigned char       ArbitrationLostCapture;
        unsigned char       ErrorCodeCapture;
        unsigned char       TxErrorCounter;
        unsigned char       RxErrorCounter;
        unsigned char       ErrorWarningLimit;
        unsigned char       StatusRegister;
        unsigned char       ModeRegister;
        unsigned char       RxMessageCounterMax;
} TDRV010_STATUS, *PTDRV010_STATUS;
```

ArbitrationLostCapture

Contents of the arbitration lost capture register. This register contains information about the bit position of losing arbitration.

ErrorCodeCapture

Contents of the error code capture register. This register contains information about the type and location of errors on the bus.

TxErrorCounter

Contents of the TX error counter register. This register contains the current value of the transmit error counter.

RxErrorCounter

Contents of the TX error counter register. This register contains the current value of the receive error counter.

ErrorWarningLimit

Contents of the error warning limit register.

StatusRegister

Contents of the status register.

ModeRegister

Contents of the mode register.

RxMessageCounterMax

Contains the peak value of messages in the receive FIFO. This internal counter value will be reset to 0 after reading.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_STATUS     CanStatus;


result = devctl(   fd,
                   DCMD_TDRV010_CANSTATUS,
                   &CanStatus,
                   sizeof(CanStatus),
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

SEE ALSO

SJA1000 Product Specification Manual

3.3.9 DCMD_TDRV010_ENABLE_SELFTEST

NAME

DCMD_TDRV010_ENABLE_SELFTEST - Enable self-test mode

DESCRIPTION

This devctl function enables the self-test facility of the SJA1000 CAN controller.

In this mode a full node test is possible without any other active node on the bus using the self-reception facility. The CAN controller will perform a successful transmission even if there is no acknowledge received.

Also in self-test mode the normal functionality is given, that means the CAN controller is able to receive messages from other nodes and can transmit message to other nodes if any connected.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_ENABLE_SELFTEST,
                   NULL,
                   0,
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.10 DCMD_TDRV010_DISABLE_SELFTEST

NAME

DCMD_TDRV010_DISABLE_SELFTEST - Disable self-test mode

DESCRIPTION

This devctl function disables the self-test facility of the SJA1000 CAN controller, which has been enabled before with the devctl command DCMD_TDRV010_ENABLE_SELFTEST.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_DISABLE_SELFTEST,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.11 DCMD_TDRV010_ENABLE_LISTENONLY

NAME

DCMD_TDRV010_ENABLE_LISTENONLY - Enable listen only mode

DESCRIPTION

This devctl function enables the listen only facility of the SJA1000 CAN controller.

In this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully. Message transmission is not possible. All other functions can be used like in normal mode.

This mode can be used for software driver bit rate detection and 'hot-plugging'.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;
result = devctl(   fd,
                   DCMD_TDRV010_ENABLE_LISTENONLY,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.12 DCMD_TDRV010_DISABLE_LISTENONLY

NAME

DCMD_TDRV010_DISABLE_LISTENONLY - Disable listen only mode

DESCRIPTION

This devctl function disables the listen only facility of the SJA1000 CAN controller, which has been enabled before with the devctl command DCMD_TDRV010_ENABLE_LISTENONLY.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_DISABLE_LISTENONLY,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.13 DCMD_TDRV010_SETLIMIT

NAME

DCMD_TDRV010_SETLIMIT - Disable listen only mode

DESCRIPTION

This devctl function sets a new error warning limit in the corresponding CAN controller register. The default value (after hardware reset) is 96.

The new error warning limit will be set in an unsigned char variable. A pointer to this variable is passed by the parameters *data_ptr* to the driver. The size of this variable is passed by the parameter *n_bytes* to the driver.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TDRV010_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int           fd;
int           result;
unsigned char ErrorLimit

ErrorLimit = 20;
result = devctl(   fd,
                   DCMD_TDRV010_SETLIMIT,
                   &ErrorLimit,
                   sizeof(ErrorLimit),
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.14 DCMD_TDRV010_TRANSCEIVER_OPERATING

NAME

DCMD_TDRV010_TRANSCEIVER_OPERATING – Switch transceiver into Operating Mode

DESCRIPTION

This devctl function switches the onboard transceivers for the specific channel into Operating Mode.

This function is only supported by TPMC310 modules.

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_TRANSCEIVER_OPERATING,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
ENOTSUP	Function is not supported by hardware.

SEE ALSO

TPMC310 Hardware User Manual

3.3.15 DCMD_TDRV010_TRANSCEIVER_SILENT

NAME

DCMD_TDRV010_TRANSCEIVER_SILENT – Switch transceiver into Silent Mode

DESCRIPTION

This devctl function switches the onboard transceivers for the specific channel into Silent Mode.

This function is only supported by TPMC310 modules.

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

result = devctl(   fd,
                   DCMD_TDRV010_TRANSCEIVER_SILENT,
                   NULL,
                   0,
                   NULL);

if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
ENOTSUP	Function is not supported by hardware.

SEE ALSO

TPMC310 Hardware User Manual

4 Step by Step Driver Initialization

The following code example illustrates all necessary steps to initialize a CAN device for communication, assuming the device is in BUSOFF state.

```
/*
**  ( 1.) Setup CAN bus bit timing
*/
BitTimingParam.TimingValue   = TDRV010_100KBIT;
BitTimingParam.ThreeSamples  = 0;      /* FALSE */

result = devctl(   fd,
                   DCMD_TDRV010_BITTIMING,
                   &BitTimingParam,
                   sizeof(BitTimingParam),
                   NULL);

/*
**  ( 2.) Setup acceptance filter masks
*/
AcceptFilter.AcceptanceCode  = 0x0;
AcceptFilter.AcceptanceMask  = 0xFFFFFFFF;
AcceptFilter.SingleFilter    = 1;

result = devctl(   fd,
                   DCMD_TDRV010_SETFILTER,
                   & AcceptFilter,
                   sizeof(AcceptFilter),
                   NULL);

/*
**  ( 3.) Enter Bus On State
*/
result = devctl(   fd,
                   DCMD_TDRV010_BUSON,
                   NULL,
                   0,
                   NULL);
```

Now you should be able to send and receive CAN messages with appropriate calls to DCMD_TDRV010_WRITE and DCMD_TDRV010_READ functions.