The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV002-SW-95

QNX Neutrino Device Driver

Multiple Channel Serial Interface

Version 1.3.x

User Manual

Issue 1.3.0

January 2020

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

TDRV002-SW-95

QNX Neutrino Device Driver

Multiple Channel Serial Interface

Supported Modules:

TPMC37x
TPMC46x
TPMC47x
TCP46x
TCP47x
TXMC37x
TXMC46x

Issue	Description	Date
1.0.0	First Issue	September 27, 2005
1.0.1	TPMC467/TCP467 support added, modified file list	February 2, 2007
1.0.2	Modification for devc-devices and QNX6.3.x with SPx	February 23, 2007
1.0.3	Description of installation corrected, “Avoiding Data Loss” added, Address TEWS LLC removed	October 7, 2010
1.1.0	TPMC377, TPMC470, TCP469 and TCP470 support added	May 4, 2011
1.2.0	New Files added to file list, TXMC375 support added devctl-function TDRV002_DCMD_SET_TRANSC added	April 4, 2013
1.3.0	QNX 6.6, QNX 7 Support added (Driver Installation, Start) Support for TPMC378, TCP468, TXMC376, TXMC 463, TXMC465 TXMC466 added	January 21, 2020

Table of Contents

1 Introduction

The TDRV002-SW-95 QNX Neutrino device driver is a full-duplex serial device driver which allows the operation of TDRV002 serial devices on Intel x86 based QNX Neutrino operating systems.

The TDRV002-SW-95 device driver is based on a version of the standard QNX 8250 serial communication manager. Due to this way of implementation the driver interface and function is compatible to the standard QNX serial device manager.

All standard utility programs for configuration (e.g. stty) and maintaining terminal interfaces can be used in the same manner.

Additional supported features:

- Receive and transmit FIFO trigger levels are configurable during driver start up
- Configuration of programmable interface. (If hardware supports programmable interface)

The TDRV002-SW-95 device driver supports the modules listed below:

Module	Serial Interfaces	Programmable Interfaces	FIFO-Size (Bytes)	Isolated	Form Factor	Conduction Cooled
TPMC371	8		64		PMC	●
TPMC372	4		64		PMC	●
TPMC375	8	●	64		PMC	●
TPMC376	4	●	64		PMC	●
TPMC377	4	●	64	●	PMC	●
TPMC378	8		64	●	PMC	●
TPMC460	16		64		PMC	
TPMC461	8		64		PMC	
TPMC462	4		64		PMC	
TPMC463	4		64		PMC	
TPMC465	8	●	64		PMC	
TPMC466	4	●	64		PMC	
TPMC467	4	●	64		PMC	
TPMC470	4	●	64	●	PMC	
TCP460	16		64		cPCI	
TCP461	8		64		cPCI	
TCP462	4		64		cPCI	
TCP463	4		64		cPCI	
TCP465	8	●	64		cPCI	
TCP466	4	●	64		cPCI	

Module	Serial Interfaces	Programmable Interfaces	FIFO-Size (Bytes)	Isolated	Form Factor	Conduction Cooled
TCP467	4	●	64		cPCI	
TCP468	4		64		cPCI	
TCP469	8	●	64	●	cPCI	
TCP470	4	●	64	●	cPCI	
TXMC375	8	●	256		XMC	●
TXMC376	4	●	256		XMC	●
TXMC463	4		256		XMC	
TXMC465	8	●	256		XMC	
TXMC466	4	●	256		XMC	

In this document all supported modules and devices will be called TDRV002. Specials for certain devices will be advised.

2 Installation

The directory TDRV002-SW-95 on the distribution media contains the following files:

TDRV002-SW-95-1.3.0.pdf	This manual in PDF format
Release.txt	Release information
ChangeLog.txt	Release history
TDRV002-SW-95-SRC.tar.gz	Driver source archive for QNX versions up to 6.5
TDRV002-SW-95-QNX66.zip	Driver source archive for QNX6.6
TDRV002-SW-95-QNX7.zip	Driver source archive for QNX7

2.1 Building Executables on Native Systems

In order to perform an installation on a native system, extract all files of the archive TDRV002-SW-95-SRC.tar.gz to the /usr/src directory. The command 'tar -xzvf TDRV002-SW-95-SRC.tar.gz' will extract the files into the local directory. After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called tdrv002.

It is absolute important to create the tdrv002 project directory in the /usr/src directory, otherwise the automatic build with make will fail.

For building the device driver it is necessary that the QNX serial DDK is installed. (Installer: "/QNX Realtime Platform/Software Development/Device Driver Kits/ Character (Serial) DDK targeting x86").

For Serial DDKs not using the pm library, please use common.mk-nopm instead of common.mk build file. In detail, for QNX system releases before 6.3.0 copy common.mk-nopm to common.mk and start the build process.

For Serial DDKs not using the ps library, please use common.mk-nops instead of common.mk build file. In detail, for QNX system releases 6.3.x without a Service Pack copy common.mk-nops to common.mk and start the build process.

2.1.1 Build the Device Driver

- Change to the /usr/src/tdrv002/driver directory

- Execute the Makefile

```
# make install
```

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

2.1.2 Build the Example Applications

- Change to the example directory (e.g. /usr/src/tdrv002/example)

- Execute the Makefile:

```
# make install
```

After successful completion the example binary (e.g. *tdrv002exa*) 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.

Extract the content of the TDRV002-SW-95-QNX66.zip archive from the distribution media to a desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tdrv002*.

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. TDRV002-driver)
- Select the path “tdrv002\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 tdrv002 device driver. (e.g. “devc-tdrv002 – [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. TDRV002-example)
- Select the path “tdrv002\example” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)
- If necessary, extend the include path, add a link to tdrv002.h
  or copy tdrv002.h into the example path

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

After successful completion the IDE shows a “Binaries”-path containing the built binary of tdrv002 example application. (e.g. “tdrv002exa – [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. tdrv002, tdrv002exa) 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.

Extract the content of the TDRV002-SW-95-QNX7.zip archive from the distribution media to a desired working directory.

If necessary an additional path below of nto-path is needed for the desired target architecture. Therefore simply copy an existing path and rename it corresponding to the target architecture (e.g. “x86_64”).

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tdrv002*.

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. TDRV002-driver)
- Select the path “tdrv002\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 = devc-tdrv002
- LIBS = pci io-char

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 devc-tdrv002 device driver of the enabled configurations (e.g. “devc-tdrv002 – [x86/le]” and “devc-tdrv002 – [x86_64/le]”).

2.3.2 Build the Example Applications

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV002-example)
- Select the path “tdrv002\example” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)
- If necessary, extend the include path, add a link to tdrv002.h
  or copy tdrv002.h into the example path

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 = tdrv002exa

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

After successful completion the IDE shows a “Binaries”-path containing the built binaries of tdrv002 example application of the enabled configurations. (e.g. “tdrv002exa – [x86/le]” and “tdrv002exa – [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 pr oject 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_64-generic.build”) in “images”. For example the filenames (e.g. devc-tdrv002, tdrv002exa) 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 TDRV002 device driver respective the TDRV002 serial communications manager you have to enter the process name with optional parameter from the command shell or in the startup script.

```
# devc-tdrv002 [options] &
```

OPTIONS

-b number	Initial baud rate (default 9600).
-C size	The size of the canonical buffer in bytes (default 1024).
-E	Start in raw mode (the default). Software flow control is disabled by default.
-e	Start in edit mode (default raw). Software flow control is enabled by default.
-F	Disable hardware flow control (default to hardware flow control enabled).
-f	Enable hardware flow control (default).
-I number	The size of the interrupt input buffer in bytes (default 8192).
-O number	The size of the interrupt output buffer in bytes (default 8192).
-S|s	Disable / enable software flow control. The default depends on the mode: in raw mode (*-E*, the default), its disabled; in edited mode (*-e*), it's enabled. The order in which you specify the *–E* or *–e*, and *–S* or *–s* options matters:

Options	Mode	Software flow control
-e	Edited	Enabled
-S –e	Edited	Enabled
-e –S	Edited	Disabled
-E	Raw	Disabled
-s –E	Raw	Disabled
-E –s	Raw	Enabled

-u number	Append number to the device name prefix (/dev/ser). The default is 3, which mean the first TDRV002 device is /dev/ser3; additional devices are given increasing numbers.
-v	Print out debug information.
-L filename	Specifies the filename of the configuration file which defines the setup for FIFO trigger levels and programmable interfaces. A detailed description of the configuration file can be found in the chapter *Configuration File*.

Most of the options above are standard options for serial communications manager. Please refer also to related QNX documentation if necessary.

DESCRIPTION

The devc-tdrv002 manager is based on a version of the standard QNX devc-ser8250 serial communications manager and can support any number of serial ports and TDRV002 modules.

The devc-tdrv002 manager searches the entire PCI-bus for TDRV002 devices and creates devices for each serial channel. The first device created depends on the **–u** option. If the **–u** option is omitted the first TDRV002 serial device is */dev/ser3*. If a TPMC461 (8 channel) and a TPMC462 (4 channel) are used, the devices */dev/ser3, /dev/ser4, …/dev/ser10* will be created for the TPMC461, */dev/ser11 … /dev/ser14* will be created for the TPMC462.

The order of device creation of the devices on different modules depends on the PCI deviceID. (Ascending order as described in chapter 1: TPMCxxx, TCPxxx, TXMCxxx)

Usually the device names /dev/ser1 and /dev/ser2 are assigned to the default PC serial ports, therefore the TDRV002 devices can start with /dev/ser3 (default). If there are additional onboard serial devices you have to start with a higher device number for the TDRV002 devices by defining an appropriate number with the **–u** option (please check also the */dev* directory).

A read request by default returns when at least 1 character is available. To increase efficiency, you can control three parameters to control when a read is satisfied:

Time	Return after a specified amount of time has elapsed (c_cc[VTIME]).
Min	Return when this number of characters is in the input buffer (c_cc[VMIN]).
Char	Return if the forwarding character is in the input buffer (c_cc[VEND]).

These parameters, and others, are set using library routines (see *tcgetattr(), txsetattr(), readcond()* and *TimerTimeout()* in the Library Reference).

The following fields and flags are supported in the *termios* structure.

Field	Supported fields and flags
c_cc	All characters
c_iflag	BRKINT ICRNL IGNBRK IXON
c_oflag	OPOST
c_cflag	CLOCAL CSIZE CSTOPB PARENB PARODD
c_lflag	ECHO ECHOE ECHOK ECHONL ICANON IEXTEN ISIG NOFLSH

EXAMPLES

Start the device driver with default parameters (first created device is */dev/ser3*, 9600 baud, see also options above…):

```
# devc-tdrv002 -F &
```

Start the device driver with default parameters and change baud rate to 38400

```
# devc-tdrv002 -F –b 38400 &
```

Start the device driver with default parameters. The first created device is */dev/ser5*.

```
# devc-tdrv002 -F –u 5 &
```

Start the device driver with default parameters and configuration information from ./tdrv002config.txt.

```
# devc-tdrv002 –L tdrv002config.txt &
```

2.5 Configuration File

This chapter describes the syntax used in the configuration file.

Each line starts with a prefix, a '*#*' specifies a line with comment and a '*$*' specifies a line with configuration data. Leading spaces will be ignored.

Behind the prefix '*#*' all character will be ignored.

Behind the prefix '*$*'only valid characters are allowed. The line must have the following syntax:

`$<mod>/<chan>-<P0><P1><P2><P3><P4><P5><P6><P7>-<Rx>/<Tx>`

<mod>

Selects the module the configuration should be set to. *0* selects the 1st found module, *1* the 2nd and so on. A configuration that should be used for all modules can be specified with '*'.

<chan>

Selects the channel the configuration should be set to. *0* selects the 1st channel of the module, *1* the 2nd, and so on. A configuration that should be used for all channels of a specified module can be specified with '*'.

<P0>

This value specifies the setting of the *RS485/RS232#* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P1>

This value specifies the setting of the *HDPLX* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P2>

This value specifies the setting of the *RENA* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P3>

This value specifies the setting of the *RTERM* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P4>

This value specifies the setting of the *TTERM* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P5>

This value specifies the setting of the *SLEW LIMIT* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P6>

This value specifies the setting of the *SHDN* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<P7>

This value specifies the setting of the *Auto RS485 Operation* configuration. A '*1*' sets this bit and a '*0*' resets the bit. (This value is ignored for non-programmable interfaces)

<Rx>

Specifies the receive FIFO trigger level. The value must be between 1 and 63 for TPMCxxx and TCPxxx and between 1 and 255 for TXMCxxx. (56 is a value that fits into most applications and systems)

<Tx>

Specifies the transmit FIFO trigger level. The value must be between 1 and 63 for TPMCxxx and TCPxxx and between 1 and 255 for TXMCxxx. (8 is a value that fits into most applications and systems)

The configuration entries will always be scanned from the beginning of the file and the 1st matching configuration will be used. This allows the specification of general configurations and some special, that will be used for specifies channels.

Values for configuration parameters <P0>...<P7> are described in detail in the modules hardware user manual.

EXAMPLE

1) All channels shall get the same configuration:

```
# Setup all channel for RS232 with trigger levels of 56 for Rx and 8 for Tx
$*/*-00000000-56/8
```

2) Setup the first module different to the other modules

```
# Setup all channels of the 1st module for RS232, and all other channels for
# RS422 (RS485/RTERM) with trigger levels of 56 for Rx and 8 for Tx
$0/*-00000000-56/8
$*/*-10010000-56/8
```

3) Like 2) but 4th channel of the 1st module should also be RS422 (R485/RTERM)

```
$0/3-10010000-56/8
$0/*-00000000-56/8
$*/*-10010000-56/8
```

2.6 Avoiding Data Loss

If higher baud rates are used, or system load is high, it may be necessary to change the serial configurations. First FIFO trigger levels can be modified and second the size of the SW-buffers can be increased.

The receive trigger level specifies the number of characters that have to be stored in the FIFO before an interrupt is generated. The remaining space in the FIFO specifies the time before a data overrun will occur and data gets lost. Therefore changing the configuration may be necessary if there is a high interrupt load on the system and the ISR may be delayed. The FIFO trigger level is defined in the configuration file. (See 2.5 Configuration File)

Example 1:

Configuration: receive trigger level is set to 56, 115200-8N1
8 Characters space in FIFO when interrupt occurs (FIFO-size [64] – trigger level [56])
➔ time until the FIFO must be read is at least ~0.69 ms ((8 * 10Bit) / 115200Baud)

Example 2:

Configuration: receive trigger level is set to 16, 115200-8N1
48 Characters space in FIFO when interrupt occurs (FIFO-size [64] – trigger level [16])
➔ time until the FIFO must be read is at least ~4.16 ms ((48 * 10Bit) / 115200Baud)

The example shows calculations for FIFO with a depth of 64 characters, if UARTS with a FIFO depth of 256 characters are used the calculation must be adapted with the corresponding FIFO size.

Changing the FIFO trigger level also changes the interrupt load. Decreasing the Rx FIFO trigger level will increase the number of interrupts!

The second modification, changing sizes of SW-buffers is useful, if the interrupts can be handled in time, but there is still loss of data. The size of the SW-buffers can be specified when the driver is started. (See 2.4 Start the Driver Process)

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system used for the special devctl functions.

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 TDRV002 named by pathname. The flags argument controls how the file is to be opened. TDRV002 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open(“/dev/ser3”, 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 special devctl functions of the TDRV002 driver and should be set to NULL.

The following devctl command codes are defined in *tdrv002.h* and can be used in addition to the standard devctl functions for tty devices.

Value	Description
DCMD_TDRV002_SET_TRANSC	Configure programmable transceiver interface

See behind for more detailed information on each control code.

To use these TDRV002 specific control codes, the header file tdrv002.h must be included by 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.

The TDRV002 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 TDRV002_DCMD_SET_TRANSC

NAME

TDRV002_DCMD_SET_TRANSC – setup programmable transceiver

DESCRIPTION

This special devctl function configures programmable transceivers. The function allows changing the transceiver configuration while the driver process is running. A pointer to the new configuration value (unsigned char) and the size of the value (1 byte) are passed by the parameters *data_ptr* and *n_bytes* to the device.

For a more detailed description of programming the transceivers, please refer to the corresponding hardware User Manual.

The configuration value is an ORed value of the following defines (tdrv002.h):

Definition	Description
TDRV002_CFG_RS485_RS232	If set RS485 interface is selected, else RS232 interface is selected
TDRV002_CFG_HDPLX	If set half-duplex interface is selected, else full-duplex interface is selected
TDRV002_CFG_RENA	If set auto RS485 receiver is enabled, else auto RS485 receiver is disabled
TDRV002_CFG_RTERM	If set receive termination is enabled, else receive termination is disabled
TDRV002_CFG_TTERM	If set transmit termination is enabled, else transmit termination is disabled
TDRV002_CFG_SLEWLIMIT	If set slew limit mode is selected, else slew limit mode is disabled
TDRV002_CFG_SHDN	If set transceiver is shut down, else transceiver works in the configured mode
TDRV002_CFG_AUTO_RS485	If set the UART controller uses auto RS485 mode, else UART controller does not use the auto RS485 mode

There are also some typical transceiver configuration predefined in tdrv002.h, which can be used instead of building an own configuration value.

Definition	Description
TDRV002_INTF_OFF	Shutdown mode / disable interface
TDRV002_INTF_RS232	RS232 mode
TDRV002_INTF_RS422	RS422 (Multidrop / Full Duplex)
TDRV002_INTF_RS485FDM	RS485 Full Duplex (Master)
TDRV002_INTF_RS485FDS	RS485 Full Duplex (Slave)
TDRV002_INTF_RS485HD	RS485 Half Duplex

EXAMPLE

```
#include <tdrv002.h>


int                ttyDev;
int                retVal;
unsigned char      config;

ttyDev = open("/dev/ser3", O_RDWR);
if (ttyDev == -1)
{
       /* device not opened ==> error handling */
}

/* setup RS422 interface (use predefined value) */
config = TDRV002_INTF_RS422;
retVal = devctl( ttyDev,
                 TDRV002_DCMD_SET_TRANSC,
                 &config,
                 sizeof(config),
                 NULL);
if (retVal!= EOK)
{
       /* setting transceiver interface failed ==> error handling */
}

close(ttyDev);
```

ERRORS

There are no special error codes.