The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV009-SW-95

QNX Neutrino Device Driver

High Speed Sync/Async Serial Interface

Version 1.1.x

User Manual

Issue 1.1.0

February 2021

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

TDRV009-SW-95

QNX Neutrino Device Driver

High Speed Sync/Async Serial Interface

Supported Modules:
TPMC363
TPMC863
TCP863
TAMC863
TPCE863

Issue	Description	Date
1.0.0	First Issue	June 15, 2007
1.0.1	General Revision	February 3, 2012
1.0.2	Example (DCMD_TDRV009_SET_TX_TIMEOUT) corrected	April 23, 2013
1.1.0	Support for QNX7 added	February 25, 2021

Table of Contents

1 Introduction

The TDRV009-SW-95 QNX-Neutrino device driver allows the operation of the TPMC863 product family on QNX-Neutrino operating systems.

The TDRV009 device driver is basically implemented as a user installable Resource Manager. The standard file (I/O) functions (open, close, read, write and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TDRV009-SW-95 device driver includes the following functions:

- Setup and configure channels
- Send and Receive Data Buffers
- Switch on or off a channel's receiver
- Read and write onboard registers directly
- Wait for incoming SCC interrupts

The TDRV009-SW-95 supports the modules listed below:

TPMC863	4 Channel High Speed Synch/Asynch Serial Interface	PMC
TPMC363	4 Channel High Speed Synch/Asynch Serial Interface	PMC, Conduction Cooled
TCP863	4 Channel High Speed Synch/Asynch Serial Interface	CompactPCI
TAMC863	4 Channel High Speed Synch/Asynch Serial Interface	AMC
TPCE863	4 Channel High Speed Synch/Asynch Serial Interface	Standard PCI-Express

In this document all supported modules and devices will be called TDRV009. Specials for a certain device will be advised.

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

TPMC863 (or compatible) User manual

2 Installation

Following files are located in the directory TDRV009-SW-95 on the distribution media:

TDRV009-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV009-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 TDRV009-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv009':

/driver/tdrv009.c	Driver source code
/driver/tdrv009.h	Definitions and data structures for driver and application
/driver/tdrv009def.h	Device driver include
/driver/commCtrl.h	Include file for controller chip
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/example/tdrv009exa.c	Example application

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar –xzvf TDRV009-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 *tdrv009*.

It is absolutely important to extract the TDRV009 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

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 TDRV009-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 *tdrv009*.

It is absolutely important to extract the TDRV0009 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/tdrv009/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tdrv009) will be installed in the /bin directory.

2.1.2 Build the Example Application

Change to the /usr/src/tdrv009/example directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (***tdrv009exa***) 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 *tdrv009*.

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. TDRV009)
- Select the path “tdrv009\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 tdrv009 device driver. (e.g. “tdrv009 – [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. TDRV009-Example)
- Select the path “tdrv009\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 tdrv009 example application. (e.g. “tdrv009exa – [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. tdrv009, tdrv009exa) 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 *tdrv009*.

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. TDRV009)
- Select the path "tdrv009\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 = tdrv009
- 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 tdrv009 device driver of the enabled configurations (e.g. "tdrv009 – [x86/le]" and "tdrv009 – [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. TDRV009-Example)
- Select the path "tdrv009\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 = tdrv009exa

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

After successful completion the IDE shows a "Binaries"-path containing the built binaries of tdrv009 example application of the enabled configurations. (e.g. "tdrv009exa – [x86/le]" and "tdrv009exa – [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. tdrv009, tdrv009exa) 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 TDRV009 device driver, you have to enter the process name with optional parameters from the command shell or in the startup script.

Possible parameters are:

-v

For debugging purposes you can start the TDRV009 Resource Manager with the –v option. The Resource Manager will print versatile information about TDRV009 configuration and command execution on the terminal window.

-r <seconds>

Specifies the receive timeout in seconds, after a pending receive job will be canceled. Supply -1 as parameter and the timeout functionality will be disabled and the receive function will block until data is available. Default value is 5 seconds.

-t <seconds>

Specifies the transmit timeout in seconds, after a pending receive job will be canceled. Supply -1 as parameter and the timeout functionality will be disabled and the transmit function will block until the transfer is completed. Default value is 5 seconds.

-m <MaxMsgSize>

Specifies the maximum message size which can be transferred between driver and user application. Default value is 4096 bytes.

-b <TotalRxBufferSize>

Specifies the total size of the internal receive buffer. This buffer is organized as a circular buffer consisting of multiple single buffers, called Packets, to hold the received data. The number of packets is calculated out of the total buffer size and the size of one packet (see next parameter).

-p <RxPacketSize>

Specifies the size of a single receive packet. This size should be large enough to hold the largest buffer received by the application to prevent a message from splitting into multiple packets.

Example:

The following startup call will start the TDRV009 device driver in verbose mode with 100 receive packets 256 bytes in size:

```
# tdrv009 –v -b 25600 –p 256 &
```

After the TDRV009 Resource Manager is started, it creates and registers a device for each found serial channel. The devices are named `/dev/tdrv009_`**x**, where **x** is the number of the channel served by this device.

```
/dev/tdrv009_0, /dev/tdrv009_1, /dev/tdrv009_2, /dev/tdrv009_3,
/dev/tdrv009_4, /dev/tdrv009_5, /dev/tdrv009_6, /dev/tdrv009_7, ...
```

This pathname must be used in the application program to open a path to the desired TDRV009 channel device.

```
fd = open(“/dev/tdrv009_0”, O_RDWR);
```

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 TDRV009 named by pathname. The flags argument controls how the file is to be opened (must be O_RDWR for TDRV009 devices).

EXAMPLE

```
int fd;

fd = open("/dev/tdrv009_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 read

NAME

read – read from a device

SYNOPSIS

#include <sys/uio.h>
#include <unistd.h>

size_t read(int filedes, void* buf, size_t nbytes)

DESCRIPTION

The read() function attempts to read *nbyte* bytes from the file associated with the open file descriptor, *filedes*, into the buffer pointed to by *buf*.

EXAMPLE

```
int fd;
char* pData;
ssize_t BytesReceived;

pData = (char*)malloc( 1024 * sizeof(char) );
BytesReceived = read( fd, pData, 1024 );
if (BytesReceived > 0)
{
     printf( " Bytes received: %d, Data: '%s' \n", BytesReceived, pData );
}
```

RETURNS

In the case of an error, a value of –1 is returned, the global variable *errno* contains the detailed error code. Otherwise the number of bytes received is returned.

ERRORS

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

SEE ALSO

Library Reference - read()

3.4 write

NAME

write() – write to a device

SYNOPSIS

#include <sys/uio.h>
#include <unistd.h>

size_t write(int filedes, void* buf, size_t nbytes)

DESCRIPTION

The write() function attempts to write *nbyte* bytes to the file associated with the open file descriptor, *filedes*, from the buffer pointed to by *buf*.

EXAMPLE

```
int fd;
char* pData;
ssize_t BytesTransmitted;

pData = (char*)malloc( 20 * sizeof(char) );
sprintf( pData, "Hello World!\n" );
BytesTransmitted = write( fd, pData, strlen(pData) );
if (BytesTransmitted != strlen(pData))
{
     printf( "Transfer not successful. \n" );
}
```

RETURNS

In the case of an error, a value of –1 is returned, the global variable *errno* contains the detailed error code. Otherwise the number of bytes transmitted is returned.

ERRORS

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

SEE ALSO

Library Reference - write()

3.5 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 TDRV009 driver and should be set to NULL.

The following devctl command codes are defined in *tdrv009.h*:

Value	Description
DCMD_TDRV009_SET_OPERATION_MODE	Setup a channel's operation mode
DCMD_TDRV009_GET_OPERATION_MODE	Retrieve a channel's current configuration
DCMD_TDRV009_SET_BAUDRATE	Setup baudrate of a channel
DCMD_TDRV009_SET_RX_TIMEOUT	Setup receive timeout
DCMD_TDRV009_SET_TX_TIMEOUT	Setup transmit timeout
DCMD_TDRV009_SET_RECEIVER_STATE	Set operation state of the channel’s receiver
DCMD_TDRV009_CLEAR_RX_BUFFER	Clear the internal receive buffer
DCMD_TDRV009_SET_EXT_XTAL	Specify the externally supplied frequency
DCMD_TDRV009_SCC_REG_WRITE	Directly write to an SCC register
DCMD_TDRV009_SCC_REG_READ	Directly read from an SCC register
DCMD_TDRV009_REG_WRITE	Directly write to a register
DCMD_TDRV009_EEPROMWRITE	Write value to EEPROM

DCMD_TDRV009_EEPROMREAD	Read value from EEPROM
DCMD_TDRV009_REG_READ	Directly read from a register
DCMD_TDRV009_WAITFORINTERRUPT	Wait for specific channel interrupt
DCMD_TDRV009_RTS_SET	Assert RTS handshake line
DCMD_TDRV009_RTS_CLEAR	De-Assert RTS handshake line
DCMD_TDRV009_CTS_GET	Read state of CTS
DCMD_TDRV009_DTR_SET	Set DTR signal line (only channel 3)
DCMD_TDRV009_DTR_CLEAR	Clear DTR signal line (only channel 3)
DCMD_TDRV009_DSR_GET	Read state of DSR (only channel 3)

See behind for more detailed information on each control code.

To use these TDRV009 specific control codes, the header file tdrv009.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. Note, the TDRV009 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.5.1 DCMD_TDRV009_SET_OPERATION_MODE

NAME

DCMD_TDRV009_SET_OPERATION_MODE – Setup a channel's operation mode

DESCRIPTION

This I/O control function sets the desired operation mode for a channel. A pointer to the callers message buffer (TDRV009_OPERATION_MODE_STRUCT) and the size of this structure are passed by the parameters data_ptr and n_bytes to the device. It is necessary to completely initialize the structure. This can be done by calling the I/O control function DCMD_TDRV009_GET_OPERATION_MODE described below.

```
typedef struct
{
   TDRV009_COMM_TYPE            CommType;
   TDRV009_TRANSCEIVER_MODE     TransceiverMode;
   TDRV009_ENABLE_DISABLE       Oversampling;
   TDRV009_BRGSOURCE            BrgSource;
   TDRV009_TXCSOURCE            TxClkSource;
   unsigned int                 TxClkOutput;
   TDRV009_RXCSOURCE            RxClkSource;
   TDRV009_CLKMULTIPLIER        ClockMultiplier;
   unsigned int                 Baudrate;
   unsigned char                ClockInversion;
   unsigned char                Encoding;
   TDRV009_PARITY               Parity;
   int                          Stopbits;
   int                          Databits;
   TDRV009_ENABLE_DISABLE       UseTermChar;
   char                         TermChar;
   TDRV009_ENABLE_DISABLE       HwHs;
   TDRV009_CRC                  Crc;
} TDRV009_OPERATION_MODE_STRUCT;
```

CommType

This parameter describes the general communication type for the specific channel. Possible values are:

Value	Description
TDRV009_COMMTYPE_ASYNC	Asynchronous communication
TDRV009_COMMTYPE_HDLC_ADDR0	Standard HDLC communication without address recognition. Used for synchronous communication.
TDRV009_COMMTYPE_HDLC_TRANSP	Extended Transparent mode. No protocol processing, channel works as simple bit collector.

TransceiverMode

This parameter describes the transceiver mode of the programmable multi-protocol transceivers. Possible values are:

Value	Description
TDRV009_TRNSCVR_NOT_USED	Default V.11
TDRV009_TRNSCVR_RS530A	EIA-530A (V.11 / V.10)
TDRV009_TRNSCVR_RS530	EIA-530 (V.11), also suitable for RS422
TDRV009_TRNSCVR_X21	X.21 (V.11)
TDRV009_TRNSCVR_V35	V.35 (V.35 / V.28)
TDRV009_TRNSCVR_RS449	EIA-449 (V.11)
TDRV009_TRNSCVR_V36	V.36 (V.11)
TDRV009_TRNSCVR_RS232	EIA-232 (V.28)
TDRV009_TRNSCVR_V28	V.28 (V.28)
TDRV009_TRNSCVR_NO_CABLE	High impedance

Oversampling

This parameter enables or disables 16times oversampling, used for asynchronous communication. For communication with standard UARTs it is recommended to enable this feature. Valid values are:

Value	Description
TDRV009_DISABLED	The 16 times oversampling is not used.
TDRV009_ENABLED	The 16 times oversampling is used.

BrgSource

This parameter specifies the frequency source used as input to the BRG (Baud Rate Generator). Valid values are:

Value	Description
TDRV009_BRGSRC_XTAL1	XTAL1 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL2	XTAL2 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL3	XTAL3 oscillator is used for BRG input
TDRV009_BRGSRC_RXCEXTERN	External clock at RxC input used for BRG input
TDRV009_BRGSRC_TXCEXTERN	External clock at TxC input used for BRG input

TxClkSource

This parameter specifies the frequency source used as input to the transmit engine. Valid values are:

Value	Description
TDRV009_TXCSRC_BRG	Baud Rate Generator output used for Tx clock
TDRV009_TXCSRC_BRGDIV16	BRG output divided by 16 used for Tx clock
TDRV009_TXCSRC_RXCEXTERN	External clock at RxC input used for Tx clock
TDRV009_TXCSRC_TXCEXTERN	External clock at TxC input used for Tx clock
TDRV009_TXCSRC_DPLL	DPLL output used for Tx clock

TxClkOutput

This parameter specifies which output lines are used to output the transmit clock, e.g. for synchronous communication. The given values can be binary OR'ed. Valid values are:

Value	Description
TDRV009_TXCOUT_TXC	Transmit clock available at TxC output line
TDRV009_TXCOUT_RTS	Transmit clock available at RTS output line

RxClkSource

This parameter specifies the frequency source used as input to the receive engine. Valid values are:

Value	Description
TDRV009_RXCSRC_BRG	Baud Rate Generator output used for Rx clock
TDRV009_RXCSRC_RXCEXTERN	External clock at RxC input used for Rx clock
TDRV009_RXCSRC_DPLL	DPLL output used for Rx clock

ClockMultiplier

This parameter specifies the multiplier used for BRG clock input. Valid values are:

Value	Description
TDRV009_CLKMULT_X1	Clock multiplier disabled
TDRV009_CLKMULT_X4	Selected input clock is multiplied by 4

Baudrate

This parameter specifies the desired frequency to be generated by the Baud Rate Generator (BRG), which can be used as clock input signal. The value is derived from the selected clocksource. Please note that only specific values depending on the selected oscillator are valid. This frequency is internally multiplied by 16, if oversampling shall be used.

ClockInversion

This parameter specifies the inversion of the transmit and/or the receive clock. This value can be binary OR'ed. Possible values are:

Value	Description
TDRV009_CLKINV_NONE	no clock inversion
TDRV009_CLKINV_TXC	transmit clock is inverted
TDRV009_CLKINV_RXC	receive clock is inverted

Encoding

This parameter specifies the data encoding used for communication. Valid values are:

Value	Description
TDRV009_ENC_NRZ	NRZ data encoding
TDRV009_ENC_NRZI	NRZI data encoding
TDRV009_ENC_FM0	FM0 data encoding
TDRV009_ENC_FM1	FM1 data encoding
TDRV009_ENC_MANCHESTER	Manchester data encoding

Parity

This parameter specifies the parity bit generation used for asynchronous communication. Valid values are:

Value	Description
TDRV009_PAR_DISABLED	No parity generation is used.
TDRV009_PAR_EVEN	EVEN parity bit
TDRV009_PAR_ODD	ODD parity bit
TDRV009_PAR_SPACE	SPACE parity bit (always insert ‘0’)
TDRV009_PAR_MARK	MARK parity bit (always insert ‘1’)

Stopbits

This parameter specifies the number of stop bits to use for asynchronous communication. Possible values are 1 or 2.

Databits

This parameter specifies the number of data bits to use for asynchronous communication. Possible values are 5 to 8.

UseTermChar

This parameter enables or disables the usage of a termination character for asynchronous communication. Valid values are:

Value	Description
TDRV009_DISABLED	A termination character is not used.
TDRV009_ENABLED	A termination character is used.

TermChar

This parameter specifies the termination character. After receiving this termination character, the communication controller will forward the received data packet immediately to the host system and use a new data packet for further received data. Any 8bit value may be used for this parameter.

HwHs

This parameter enables or disables the hardware handshaking mechanism using RTS/CTS. Valid values are:

Value	Description
TDRV009_DISABLED	Hardware handshaking is not used.
TDRV009_ENABLED	Hardware handshaking is used.

Crc

This parameter is a structure describing the CRC checking configuration.

```
typedef struct
{
   TDRV009_CRC_TYPE           Type;
   TDRV009_ENABLE_DISABLE     RxChecking;
   TDRV009_ENABLE_DISABLE     TxGeneration;
   TDRV009_CRC_RESET          ResetValue;
} TDRV009_CRC;
```

Type

This parameter describes the CRC type to be used. Possible values are:

Value	Description
TDRV009_CRC_16	16bit CRC algorithm is used for checksum
TDRV009_CRC_32	32bit CRC algorithm is used for checksum

RxChecking

This parameter enables or disables the receive CRC checking. Possible values are:

Value	Description
TDRV009_DISABLED	CRC checking will not be used
TDRV009_ENABLED	CRC checking will be used

TxGeneration

This parameter enables or disables the transmit CRC generation. Possible values are:

Value	Description
TDRV009_DISABLED	A CRC checksum will be generated
TDRV009_ENABLED	A CRC checksum will not be generated

ResetValue

This parameter describes the reset value for the CRC algorithm. Possible values are:

Value	Description
TDRV009_CRC_RST_FFFF	CRC reset value will be 0xFFFF
TDRV009_CRC_RST_0000	CRC reset value will be 0x0000

EXAMPLE

```
#include “tdrv009.h”

int                                fd;
int                                result;
TDRV009_OPERATION_MODE_STRUCT      OperationMode;

/*-------------------------------------------------
  Configure channel for Async / RS232 / 115200bps
  -------------------------------------------------*/
OperationMode.CommType             = TDRV009_COMMTYPE_ASYNC;
OperationMode.TransceiverMode      = TDRV009_TRNSCVR _RS232;
OperationMode.Oversampling         = TDRV009_ENABLED;
OperationMode.BrgSource            = TDRV009_BRGSRC_XTAL1;
OperationMode.TxClkSource          = TDRV009_TXCSRC_BRG;
OperationMode.TxClkOutput          = 0;
OperationMode.RxClkSource          = TDRV009_RXCSRC_BRG;
OperationMode.ClockMultiplier      = TDRV009_CLKMULT_X1;
OperationMode.Baudrate             = 115200;
OperationMode.ClockInversion       = TDRV009_CLKINV_NONE;
OperationMode.Encoding             = TDRV009_ENC_NRZ;
OperationMode.Parity               = TDRV009_PAR_DISABLED;
OperationMode.Stopbits             = 1;
OperationMode.Databits             = 8;
OperationMode.UseTermChar          = TDRV009_DISABLED;
OperationMode.TermChar             = 0;
OperationMode.HwHs                 = TDRV009_DISABLED;
OperationMode.Crc.Type             = TDRV009_CRC_16;
OperationMode.Crc.RxChecking       = TDRV009_DISABLED;
OperationMode.Crc.TxGeneration     = TDRV009_DISABLED;
OperationMode.Crc.ResetValue       = TDRV009_CRC_RST_FFFF;

result = devctl(   fd,
                   DCMD_TDRV009_SET_OPERATION_MODE,
                   &OperationMode,
                   sizeof(TDRV009_OPERATION_MODE_STRUCT),
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	Invalid argument. At least one of the supplied arguments is invalid.

All other returned errors are system error conditions.

3.5.2 DCMD_TDRV009_GET_OPERATION_MODE

NAME

DCMD_TDRV009_GET_OPERATION_MODE – Retrieve a channel's current operation mode

DESCRIPTION

This I/O control function reads the current channel configuration and returns the value in the buffer supplied by the application. A pointer to the callers message buffer (TDRV009_OPERATION_MODE_STRUCT) and the size of this structure are passed by the parameters data_ptr and n_bytes to the device.

```
typedef struct
{
   TDRV009_COMM_TYPE            CommType;
   TDRV009_TRANSCEIVER_MODE     TransceiverMode;
   TDRV009_ENABLE_DISABLE       Oversampling;
   TDRV009_BRGSOURCE            BrgSource;
   TDRV009_TXCSOURCE            TxClkSource;
   unsigned int                 TxClkOutput;
   TDRV009_RXCSOURCE            RxClkSource;
   TDRV009_CLKMULTIPLIER        ClockMultiplier;
   unsigned int                 Baudrate;
   unsigned char                ClockInversion;
   unsigned char                Encoding;
   TDRV009_PARITY               Parity;
   int                          Stopbits;
   int                          Databits;
   TDRV009_ENABLE_DISABLE       UseTermChar;
   char                         TermChar;
   TDRV009_ENABLE_DISABLE       HwHs;
   TDRV009_CRC                  Crc;
} TDRV009_OPERATION_MODE_STRUCT;
```

CommType

This parameter describes the general communication type for the specific channel. Possible values are:

Value	Description
TDRV009_COMMTYPE_ASYNC	Asynchronous communication
TDRV009_COMMTYPE_HDLC_ADDR0	Standard HDLC communication without address recognition. Used for synchronous communication.
TDRV009_COMMTYPE_HDLC_TRANSP	Extended Transparent mode. No protocol processing, channel works as simple bit collector.

TransceiverMode

This parameter describes the transceiver mode of the programmable multi-protocol transceivers. Possible values are:

Value	Description
TDRV009_TRNSCVR _NOT_USED	Default V.11
TDRV009_TRNSCVR _RS530A	EIA-530A (V.11 / V.10)
TDRV009_TRNSCVR _RS530	EIA-530 (V.11), also suitable for RS422
TDRV009_TRNSCVR _X21	X.21 (V.11)
TDRV009_TRNSCVR _V35	V.35 (V.35 / V.28)
TDRV009_TRNSCVR _RS449	EIA-449 (V.11)
TDRV009_TRNSCVR _V36	V.36 (V.11)
TDRV009_TRNSCVR _RS232	EIA-232 (V.28)
TDRV009_TRNSCVR _V28	V.28 (V.28)
TDRV009_TRNSCVR _NO_CABLE	High impedance

Oversampling

This parameter enables or disables 16times oversampling, used for asynchronous communication. For communication with standard UARTs it is recommended to enable this feature. Valid values are:

Value	Description
TDRV009_DISABLED	The 16 times oversampling is not used.
TDRV009_ENABLED	The 16 times oversampling is used.

BrgSource

This parameter specifies the frequency source used as input to the BRG (Baud Rate Generator). Valid values are:

Value	Description
TDRV009_BRGSRC_XTAL1	XTAL1 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL2	XTAL2 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL3	XTAL3 oscillator is used for BRG input
TDRV009_BRGSRC_RXCEXTERN	External clock at RxC input used for BRG input
TDRV009_BRGSRC_TXCEXTERN	External clock at TxC input used for BRG input

TxClkSource

This parameter specifies the frequency source used as input to the transmit engine. Valid values are:

Value	Description
TDRV009_TXCSRC_BRG	Baud Rate Generator output used for Tx clock
TDRV009_TXCSRC_BRGDIV16	BRG output divided by 16 used for Tx clock
TDRV009_TXCSRC_RXCEXTERN	External clock at RxC input used for Tx clock
TDRV009_TXCSRC_TXCEXTERN	External clock at TxC input used for Tx clock
TDRV009_TXCSRC_DPLL	DPLL output used for Tx clock

TxClkOutput

This parameter specifies which output lines are used to output the transmit clock, e.g. for synchronous communication. The given values can be binary OR'ed. Valid values are:

Value	Description
TDRV009_TXCOUT_TXC	Transmit clock available at TxC output line
TDRV009_TXCOUT_RTS	Transmit clock available at RTS output line

RxClkSource

This parameter specifies the frequency source used as input to the receive engine. Valid values are:

Value	Description
TDRV009_RXCSRC_BRG	Baud Rate Generator output used for Rx clock
TDRV009_RXCSRC_RXCEXTERN	External clock at RxC input used for Rx clock
TDRV009_RXCSRC_DPLL	DPLL output used for Rx clock

ClockMultiplier

This parameter specifies the multiplier used for BRG clock input. Valid values are:

Value	Description
TDRV009_CLKMULT_X1	Clock multiplier disabled
TDRV009_CLKMULT_X4	Selected input clock is multiplied by 4

Baudrate

This parameter specifies the desired frequency to be generated by the BRG (Baud Rate Generator), which can be used as clock input signal. The value is derived from the selected clocksource. Please note that only specific values depending on the selected oscillator are valid. This frequency is internally multiplied by 16, if oversampling shall be used.

ClockInversion

This parameter specifies the inversion of the transmit and/or the receive clock. This value can be binary OR'ed. Possible values are:

Value	Description
TDRV009_CLKINV_NONE	no clock inversion
TDRV009_CLKINV_TXC	transmit clock is inverted
TDRV009_CLKINV_RXC	receive clock is inverted

Encoding

This parameter specifies the data encoding used for communication. Valid values are:

Value	Description
TDRV009_ENC_NRZ	NRZ data encoding
TDRV009_ENC_NRZI	NRZI data encoding
TDRV009_ENC_FM0	FM0 data encoding
TDRV009_ENC_FM1	FM1 data encoding
TDRV009_ENC_MANCHESTER	Manchester data encoding

Parity

This parameter specifies the parity bit generation used for asynchronous communication. Valid values are:

Value	Description
TDRV009_PAR_DISABLED	No parity generation is used.
TDRV009_PAR_EVEN	EVEN parity bit
TDRV009_PAR_ODD	ODD parity bit
TDRV009_PAR_SPACE	SPACE parity bit (always insert '0')
TDRV009_PAR_MARK	MARK parity bit (always insert '1')

Stopbits

This parameter specifies the number of stop bits to use for asynchronous communication. Possible values are 1 or 2.

Databits

This parameter specifies the number of data bits to use for asynchronous communication. Possible values are 5 to 8.

UseTermChar

This parameter enables or disables the usage of a termination character for asynchronous communication. Valid values are:

Value	Description
TDRV009_DISABLED	A termination character is not used.
TDRV009_ENABLED	A termination character is used.

TermChar

This parameter specifies the termination character. After receiving this termination character, the communication controller will forward the received data packet immediately to the host system and use a new data packet for further received data. Any 8bit value may be used for this parameter.

HwHs

This parameter enables or disables the hardware handshaking mechanism using RTS/CTS. Valid values are:

Value	Description
TDRV009_DISABLED	Hardware handshaking is not used.
TDRV009_ENABLED	Hardware handshaking is used.

Crc

This parameter is a structure describing the CRC checking configuration.

```
typedef struct
{
   TDRV009_CRC_TYPE          Type;
   TDRV009_ENABLE_DISABLE    RxChecking;
   TDRV009_ENABLE_DISABLE    TxGeneration;
   TDRV009_CRC_RESET         ResetValue;
} TDRV009_CRC;
```

Type

This parameter describes the CRC type to be used. Possible values are:

Value	Description
TDRV009_CRC_16	16bit CRC algorithm is used for checksum
TDRV009_CRC_32	32bit CRC algorithm is used for checksum

RxChecking

This parameter enables or disables the receive CRC checking. Possible values are:

Value	Description
TDRV009_DISABLED	CRC checking will not be used
TDRV009_ENABLED	CRC checking will be used

TxGeneration

This parameter enables or disables the transmit CRC generation. Possible values are:

Value	Description
TDRV009_DISABLED	A CRC checksum will be generated
TDRV009_ENABLED	A CRC checksum will not be generated

ResetValue

This parameter describes the reset value for the CRC algorithm. Possible values are:

Value	Description
TDRV009_CRC_RST_FFFF	CRC reset value will be 0xFFFF
TDRV009_CRC_RST_0000	CRC reset value will be 0x0000

EXAMPLE

```
#include “tdrv009.h”

int                                fd;
int                                result;
TDRV009_OPERATION_MODE_STRUCT      OperationMode;

/*
** retrieve current operation mode
*/
result = devctl(   fd,
                   DCMD_TDRV009_GET_OPERATION_MODE,
                   &OperationMode,
                   sizeof(TDRV009_OPERATION_MODE_STRUCT),
                   NULL);

if (result == EOK) {
     printf( "Current baudrate = %d \n", OperationMode.Baudrate );
} else {
     /* process devctl() error */
}
```

ERRORS

All returned errors are system error conditions.

3.5.3 DCMD_TDRV009_SET_BAUDRATE

NAME

DCMD_TDRV009_SET_BAUDRATE - Setup baudrate of a channel (without other changes)

DESCRIPTION

This I/O control function sets up the transmission rate for the specific channel. This is done without all the other configuration stuff contained in SET_OPERATION_MODE. No channel-reset is performed either. If async oversampling is enabled, the desired baudrate is internally multiplied by 16. It is important that this result can be derived from the selected clocksource. A pointer to the caller's message buffer (*unsigned int*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

EXAMPLE

```
#include "tdrv009.h"

int           fd;
int           result;
unsigned int  Baudrate;

Baudrate = 14400;  /* 14400 bps */

result = devctl(   fd,
                   DCMD_TDRV009_SET_BAUDRATE,
                   &Baudrate,
                   sizeof(unsigned int),
                   NULL);

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

ERRORS

Error Code	Description
EINVAL	Invalid argument. The supplied baudrate is not valid.

All other returned errors are system error conditions.

3.5.4 DCMD_TDRV009_SET_RX_TIMEOUT

NAME

DCMD_TDRV009_SET_RX_TIMEOUT – Setup receive timeout

DESCRIPTION

This I/O control function sets up the receive timeout. A pending read operation is cancelled after this time expires. A pointer to the caller's parameter buffer (*unsigned int*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

The time is specified in seconds. If no timeout should be used, specifiy -1 as value for this control function.

EXAMPLE

```
#include “tdrv009.h”

int                fd;
int                result;
unsigned int   RxTimeout;

/*
** Setup Receive Timeout to 2 seconds
*/
RxTimeout = 2;

result = devctl(   fd,
                   DCMD_TDRV009_SET_RX_TIMEOUT,
                   &RxTimeout,
                   sizeof(unsigned int),
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

All returned errors are system error conditions.

3.5.5 DCMD_TDRV009_SET_TX_TIMEOUT

NAME

DCMD_TDRV009_SET_TX_TIMEOUT – Setup transmit timeout

DESCRIPTION

This I/O control function sets up the transmit timeout. A pending write operation is cancelled after this time expires. A pointer to the caller's parameter buffer (*unsigned int*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

The time is specified in seconds. If no timeout should be used, specifiy -1 as value for this control function.

EXAMPLE

```
#include “tdrv009.h”

int                fd;
int                result;
unsigned int       TxTimeout;

/*
** Disable Transmit Timeout
*/
TxTimeout = -1;

result = devctl(   fd,
                   DCMD_TDRV009_SET_TX_TIMEOUT,
                   &TxTimeout,
                   sizeof(unsigned int),
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

All returned errors are system error conditions.

3.5.6 DCMD_TDRV009_SET_RECEIVER_STATE

NAME

DCMD_TDRV009_SET_RECEIVER_STATE - Set operation state of the channel's receiver

DESCRIPTION

This I/O control function sets the state of the receiver module.

A pointer to the callers parameter buffer (*TDRV009_RECEIVER*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

Possible values for the new receiver-state are:

Value	Description
TDRV009_RCVR_ON	The receiver is enabled.
TDRV009_RCVR_OFF	The receiver is disabled.

EXAMPLE

```
#include "tdrv009.h"

int fd;
int result;
TDRV009_RECEIVER ReceiverState;

/*
**  set receiver to ON
*/
ReceiverState = TDRV009_RCVR_ON;

result = devctl(   fd,
                   DCMD_TDRV009_SET_RECEIVER_STATE,
                   &ReceiverState,
                   sizeof(TDRV009_RECEIVER),
                   NULL);
if (result != EOK) {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	Invalid receiver state specified.

All other returned errors are system error conditions.

3.5.7 DCMD_TDRV009_CLEAR_RX_BUFFER

NAME

DCMD_TDRV009_CLEAR_RX_BUFFER - Clear the internal receive buffer

DESCRIPTION

This I/O control function clears the internal receive-buffer of the corresponding channel. No parameter is needed for this call.

EXAMPLE

```
#include “tdrv009.h”

int fd;
int result;

result = devctl(   fd,
                   DCMD_TDRV009_CLEAR_RX_BUFFER,
                   NULL,
                   0,
                   NULL);

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

ERRORS

All returned errors are system error conditions.

3.5.8 DCMD_TDRV009_SET_EXT_XTAL

NAME

DCMD_TDRV009_SET_EXT_XTAL - Specify the externally supplied frequency

DESCRIPTION

This I/O control function sets the frequency of an externally supplied frequency. This frequency is used for baudrate calculation, and describes the input frequency to the Baud Rate Generator (BRG). The external frequency may be supplied either at input line TxC or RxC.

A pointer to the caller's parameter buffer (*unsigned int*) containing the frequency in Hz and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

EXAMPLE

```
#include "tdrv009.h"

int           fd;
int           result;
unsigned int  ext_xtal_hz;

/*  specify an external frequency of 1 MHz      */
ext_xtal_hz = 1000000;


result = devctl(   fd,
                   DCMD_TDRV009_SET_EXT_XTAL,
                   &ext_xtal_hz,
                   sizeof(unsigned int),
                   NULL);

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

ERRORS

Error Code	Description
EINVAL	Invalid frequency specified. A frequency of 0 is not allowed.

All other returned errors are system error conditions.

3.5.9 DCMD_TDRV009_SCC_REG_WRITE

NAME

DCMD_TDRV009_SCC_REG_WRITE - Directly write to an SCC register

DESCRIPTION

This I/O control function writes one 32bit word to the communication controller's register space, relative to the beginning of the specific channel's SCC register set.

A pointer to the callers status buffer (*TDRV009_ADDR_STRUCT*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Offset;
   unsigned int      Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's channel SCC register space. Please refer to the hardware user manual for further information.

Value

This 32bit word will be written to the communication controller's channel SCC register space.

Modifying register contents may result in communication problems, system crash or other unexpected behavior.

EXAMPLE

```
#include "tdrv009.h"

int                    fd;
int                    result;
TDRV009_ADDR_STRUCT    AddrBuf;

/*-------------------------------------------------
  Write a 32bit value (Termination Character Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0048;
AddrBuf.Value  = (1 << 15) | 0x42;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_SCC_REG_WRITE,
                   &AddrBuf,
                   sizeof(TDRV009_ADDR_STRUCT),
                   NULL);

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

ERRORS

Error Code	Description
EINVAL	Invalid parameter. The specified offset is out of range.

All other returned errors are system error conditions.

3.5.10 DCMD_TDRV009_SCC_REG_READ

NAME

DCMD_TDRV009_SCC_REG_READ - Directly read from an SCC register

DESCRIPTION

This I/O control function reads one 32bit word from the communication controller's register space, relative to the beginning of the specific channel's SCC register set.

A pointer to the callers status buffer (*TDRV009_ADDR_STRUCT*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Offset;
   unsigned int      Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's channel SCC register space. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication controller's channel SCC register space.

EXAMPLE

```
#include "tdrv009.h"

int                     fd;
int                     result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Read a 32bit value (Status Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0004;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_SCC_REG_READ,
                   &AddrBuf,
                   sizeof(TDRV009_ADDR_STRUCT),
                   NULL);

if (result == EOK) {
     printf( “Value = 0x%0l8X\n”, AddrBuf.Value );
} else {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	Invalid parameter. The specified offset is out of range.

All other returned errors are system error conditions.

3.5.11 DCMD_TDRV009_REG_WRITE

NAME

DCMD_TDRV009_GLOB_REG_WRITE - Directly write to a register

DESCRIPTION

This I/O control function writes one 32bit word to the communication controller's register space, relative to the beginning of the register set.

A pointer to the callers status buffer (*TDRV009_ADDR_STRUCT*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Offset;
   unsigned int      Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's register space. Please refer to the hardware user manual for further information.

Value

This 32bit word will be written to the communication controller's register space.

Modifying register contents may result in communication problems, system crash or other unexpected behavior.

EXAMPLE

```
#include "tdrv009.h"

int                    fd;
int                    result;
TDRV009_ADDR_STRUCT    AddrBuf;

/*-------------------------------------------------
  Write a 32bit value (FIFO Control Register 4)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0034;
AddrBuf.Value  = 0xffffffff;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_REG_WRITE,
                   &AddrBuf,
                   sizeof(TDRV009_ADDR_STRUCT),
                   NULL);

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

ERRORS

Error Code	Description
EINVAL	Invalid parameter. The specified offset is out of range.

All other returned errors are system error conditions.

3.5.12 DCMD_TDRV009_REG_READ

NAME

DCMD_TDRV009_ REG_READ - Directly read from a register

DESCRIPTION

This I/O control function reads one 32bit word from the communication controller's register space, relative to the beginning of the register set.

A pointer to the callers status buffer (*TDRV009_ADDR_STRUCT*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Offset;
   unsigned int      Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's register space. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication controller's register space.

EXAMPLE

```
#include "tdrv009.h"

int                    fd;
int                    result;
TDRV009_ADDR_STRUCT    AddrBuf;

/*-------------------------------------------------
  Read a 32bit value (Version Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x00F0;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_ REG_READ,
                   &AddrBuf,
                   sizeof(TDRV009_ADDR_STRUCT),
                   NULL);

if (result == EOK) {
     printf( “Value = 0x%08lX\n”, AddrBuf.Value );
} else {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	Invalid parameter. The specified offset is out of range.

All other returned errors are system error conditions.

3.5.13 DCMD_TDRV009_EEPROMWRITE

NAME

DCMD_TDRV009_EEPROMWRITE – Write value to EEPROM

DESCRIPTION

This I/O control function writes one 16bit word into the onboard EEPROM. The first part of the EEPROM is reserved for factory usage, write accesses to this area will result in an error.

A pointer to the callers status buffer (*TDRV009_EEPROM_BUFFER*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Offset;
   unsigned int      Value;
} TDRV009_EEPROM_BUFFER;
```

Offset

This parameter specifies a 16bit word offset into the EEPROM.
Following offsets are available:

Offset	Access
00h – 5Fh	R
60h – 7Fh	R / W

Value

This parameter specifies the 16bit word to be written into the EEPROM at the given offset.

EXAMPLE

```
int                     fd;
int                     result;
TDRV009_EEPROM_BUFFER   EepromBuf;

/*-------------------------------------------------
  Write a 16bit value into the EEPROM, offset 0x61
  -------------------------------------------------*/
EepromBuf.Offset = 0x61;
EepromBuf.Value  = 0x1234;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_EEPROMWRITE,
                   &EepromBuf,
                   sizeof(TDRV009_EEPROM_BUFFER),
                   NULL);

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

ERRORS

Error Code	Description
EINVAL	The specified offset address is invalid, or read-only.
EIO	Error during EEPROM access.

All other returned errors are system error conditions.

3.5.14 DCMD_TDRV009_EEPROMREAD

NAME

DCMD_TDRV009_EEPROMREAD – Read value from EEPROM

DESCRIPTION

This I/O control function reads one 16bit word from the onboard EEPROM.

A pointer to the callers status buffer (*TDRV009_EEPROM_BUFFER*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Offset;
   unsigned int      Value;
} TDRV009_EEPROM_BUFFER;
```

Offset

This parameter specifies a 16bit word offset into the EEPROM.
Following offsets are available:

Offset	Access
00h – 5Fh	R
60h – 7Fh	R / W

Value

This parameter returns the 16bit word from the EEPROM at the given offset.

EXAMPLE

```
int                     fd;
int                     result;
TDRV009_EEPROM_BUFFER   EepromBuf;

/*-------------------------------------------------
  Read a 16bit value from the EEPROM, offset 0
  -------------------------------------------------*/
EepromBuf.Offset = 0;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_EEPROMREAD,
                   &EepromBuf,
                   sizeof(TDRV009_EEPROM_BUFFER),
                   NULL);

if (result == EOK) {
     printf( “Value = 0x%X\n”, EepromBuf.Value );
} else {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EINVAL	The specified offset address is invalid.
EIO	Error during EEPROM access.

All other returned errors are system error conditions.

3.5.15 DCMD_TDRV009_WAITFORINTERRUPT

NAME

DCMD_TDRV009_WAITFORINTERRUPT – Wait for specific channel interrupt

DESCRIPTION

This I/O control function waits until a specified SCC-interrupt or the timeout occurs.

A pointer to the callers status buffer (*TDRV009_WAIT_STRUCT*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
   unsigned int      Interrupts;
   int               Timeout;
} TDRV009_WAIT_STRUCT;
```

Interrupts

This parameter specifies specific interrupt bits to wait for. If one interrupt occurs, the value is returned in this parameter. Please refer to the hardware user manual for further information on the possible interrupt bits.

Timeout

This parameter specifies the time (in seconds) to wait for an interrupt. If 0 is specified, the function will block indefinitely.

EXAMPLE

```
int                     fd;
int                     result;
TDRV009_WAIT_STRUCT     WaitStruct;

/*-------------------------------------------------
  Wait at least 5 seconds for a
  CTS Status Change (CSC) interrupt
  -------------------------------------------------*/
WaitStruct.Interrupts = (1 << 14);
WaitStruct.Timeout    = 5;
```

```
result = devctl(   fd,
                   DCMD_TDRV009_WAITFORINTERRUPT,
                   &WaitStruct,
                   sizeof(TDRV009_WAIT_STRUCT),
                   NULL);

if (result == EOK) {
     printf( “Interrupt Event occurred.\n” );
} else {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EBUSY	Too many simultaneous wait jobs active.
ETIME	Timeout. The interrupt did not occur.

All other returned errors are system error conditions.

3.5.16 DCMD_TDRV009_RTS_SET

NAME

DCMD_TDRV009_RTS_SET - Assert RTS handshake line

DESCRIPTION

This I/O control function asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking. No parameter is needed for this call.

EXAMPLE

```
#include “tdrv009.h”

int           fd;
int           result;

/*-------------------------------------------------------------
  Assert RTS
  -------------------------------------------------------------*/
result = devctl(   fd,
                   DCMD_TDRV009_RTS_SET,
                   NULL,
                   0,
                   NULL);

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

ERRORS

Error Code	Description
EPERM	The channel is in handshake mode, so this function is not allowed.

All other returned errors are system error conditions.

3.5.17 DCMD_TDRV009_RTS_CLEAR

NAME

DCMD_TDRV009_RTS_CLEAR - De-Assert RTS handshake line

DESCRIPTION

This I/O control function de-asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking. No parameter is needed for this call.

EXAMPLE

```
#include “tdrv009.h”

int           fd;
int           result;

/*-------------------------------------------------------------
  De-assert RTS
  -------------------------------------------------------------*/
result = devctl(   fd,
                   DCMD_TDRV009_RTS_CLEAR,
                   NULL,
                   0,
                   NULL);

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

ERRORS

Error Code	Description
EPERM	The channel is in handshake mode, so this function is not allowed.

All other returned errors are system error conditions.

3.5.18 DCMD_TDRV009_CTS_GET

NAME

DCMD_TDRV009_CTS_GET - Read state of CTS

DESCRIPTION

This I/O control function returns the current state of the CTS handshake signal line of the specific channel. A pointer to the caller's message buffer (*unsigned int*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device. Depending on the state of CTS, either 0 (inactive) or 1 (active) is returned.

EXAMPLE

```
#include "tdrv009.h"

int           fd;
int           result;
unsigned int  CtsState;

/*-------------------------------------------------------------
  Read CTS state
  -------------------------------------------------------------*/
result = devctl(   fd,
                   DCMD_TDRV009_CTS_GET,
                   &CtsState,
                   sizeof(unsigned int),
                   NULL);

if (result == EOK) {
     printf( "CTS = %ld\n", CtsState );
} else {
     /* process devctl() error */
}
```

ERRORS

All other returned errors are system error conditions.

3.5.19 DCMD_TDRV009_DTR_SET

NAME

DCMD_TDRV009_DTR_SET - Set DTR signal line (only channel 3)

DESCRIPTION

This I/O control function sets the DTR signal line to HIGH. This function is only available for local module channel 3. No parameter is needed for this call.

EXAMPLE

```
#include “tdrv009.h”

int           fd;
int           result;

/*-------------------------------------------------------------
  Set DTR to HIGH (only valid for channel 3)
  -------------------------------------------------------------*/
result = devctl(   fd,
                   DCMD_TDRV009_DTR_SET,
                   NULL,
                   0,
                   NULL);

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

ERRORS

Error Code	Description
EACCES	This function is not supported by the specific channel.

All other returned errors are system error conditions.

3.5.20 DCMD_TDRV009_DTR_CLEAR

NAME

DCMD_TDRV009_DTR_CLEAR - Clear DTR signal line (only channel 3)

DESCRIPTION

This I/O control function sets the DTR signal line to LOW. This function is only available for local module channel 3. No parameter is needed for this call.

EXAMPLE

```
#include “tdrv009.h”

int           fd;
int           result;

/*-------------------------------------------------------------
  Set DTR to LOW (only valid for channel 3)
  -------------------------------------------------------------*/
result = devctl(   fd,
                   DCMD_TDRV009_DTR_CLEAR,
                   NULL,
                   0,
                   NULL);

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

ERRORS

Error Code	Description
EACCES	This function is not supported by the specific channel.

All other returned errors are system error conditions.

3.5.21 DCMD_TDRV009_DSR_GET

NAME

DCMD_TDRV009_DSR_GET - Read state of DSR (only channel 3)

DESCRIPTION

This I/O control function returns the current state of the DSR signal line of the specific channel. This function is only available for local module channel 3. A pointer to the caller's message buffer (unsigned int) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device. Depending on the state of DSR, either 0 (inactive) or 1 (active) is returned.

EXAMPLE

```
#include "tdrv009.h"

int           fd;
int           result;
unsigned int  DsrState;

/*-------------------------------------------------------------
  Read DSR state
  -------------------------------------------------------------*/
result = devctl(   fd,
                   DCMD_TDRV009_DSR_GET,
                   &DsrState,
                   sizeof(unsigned int),
                   NULL);

if (result == EOK) {
     printf( "DSR = %ld\n", DsrState );
} else {
     /* process devctl() error */
}
```

ERRORS

Error Code	Description
EACCES	This function is not supported by the specific channel.

All other returned errors are system error conditions.