The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV009-SW-42

VxWorks Device Driver

High Speed Synch/Asynch Serial Interface

Version 4.1.x

User Manual

Issue 4.1.0

November 2023

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-42

VxWorks Device Driver

High Speed Synch/Asynch Serial Interface

Supported Modules:

TPMC363
TPMC863
TAMC863
TCP863
TPCE863
TMPE863

Issue	Description	Date
1.0.0	First Issue	February 5, 2007
1.1.0	Clock Multiplier changed (X4 instead of X8)	April 4, 2007
1.2.0	Interrupt Wait Ioctl function added	June 20, 2007
1.2.1	Support for TAMC863 added	April 8, 2008
1.2.2	Return value of write() function corrected, functionality of the Valid field in TDRV009_DATA_BUFFER clarified.	February 13, 2009
2.0.0	Support for VxBus and API description added	March 2, 2010
2.0.1	Legacy vs. VxBus Driver modified	March 26, 2010
3.0.0	API modified to support 64bit Description of Legacy I/O functions removed	April 4, 2012
4.0.0	VxWorks 7 Support added	February 23, 2017
4.1.0	New supported hardware added, API functions for hardware identification added	November 3, 2023

Table of Contents

1 Introduction

The TDRV009-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x releases and mandatory for VxWorks SMP systems.

The driver provides an application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TDRV009-SW-42 device driver supports the following features:

- setup and configure a serial channel
- send data buffers
- receive data buffers (buffer based / character based)

The TDRV009-SW-42 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
TAMC863	4 Channel High Speed Synch/Asynch Serial Interface	Advanced Mezzanine Card
TCP863	4 Channel High Speed Synch/Asynch Serial Interface	CompactPCI
TPCE863	4 Channel High Speed Synch/Asynch Serial Interface	Standard PCI-Express
TMPE863	3 Channel High Speed Synch/Asynch Serial Interface	PCIe Mini Card

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.

TEWS TECHNOLOGIES VxWorks Device Drivers - Installation Guide
TPMC863 (or compatible) User manual

2 API Documentation

2.1 General Functions

2.1.1 tdrv009Open

NAME

tdrv009Open() – open a device.

SYNOPSIS

```
TDRV009_DEV tdrv009Open
(
      char          *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV009 channel device is named “/tdrv009/0”, the second channel device is named “/tdrv009/1” and so on.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;

/*
** open file descriptor to device
*/
hdl = tdrv009Open(“/tdrv009/0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

2.1.2 tdrv009Close

NAME

tdrv009Close() – close a device.

SYNOPSIS

```
TDRV009_STATUS tdrv009Close
(
      TDRV009_HANDLE hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE hdl;
TDRV009_STATUS          result;

/*
** close file descriptor to device
*/
result = tdrv009Close(hdl);
if (result != TDRV009_OK)
{
     /* handle close error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The specified device handle is invalid

2.1.3 tdrv009GetPciInfo

NAME

tdrv009GetPciInfo – get information of the module PCI header

SYNOPSIS

```
TDRV009_STATUS tdrv009GetPciInfo
(
    TDRV009_HANDLE          hdl,
    TDRV009_PCIINFO_BUF     *pPciInfoBuf
)
```

DESCRIPTION

This function returns information of the module PCI header in the provided data buffer.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pPciInfoBuf

This argument is a pointer to the structure TDRV009_PCIINFO_BUF that receives information of the module PCI header.

```
typedef struct
{
    unsigned short      vendorId;
    unsigned short      deviceId;
    unsigned short      subSystemId;
    unsigned short      subSystemVendorId;
    int                 pciBusNo;
    int                 pciDevNo;
    int                 pciFuncNo;
} TDRV009_PCIINFO_BUF;
```

vendorId

PCI module vendor ID

deviceId

PCI module device ID

subSystemId

PCI module sub system ID

subSystemVendorId

PCI module sub system vendor ID

pciBusNo

Number of the PCI bus, where the module resides.

pciDevNo

PCI device number

pciFuncNo

PCI function number

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_PCIINFO_BUF     pciInfoBuf

/*
** get module PCI information
*/
result = tdrv009GetPciInfo( hdl, &pciInfoBuf );

if (result != TDRV009_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The specified device handle is invalid

2.1.4 tdrv009GetBoardName

NAME

tdrv009GetBoardName – get Name of the board

SYNOPSIS

```
TDRV009_STATUS tdrv009GetBoardName
(
    TDRV009_HANDLE      hdl,
    char                *pBoardName,
    int                 len
)
```

DESCRIPTION

This function returns the name of the hardware board to distinguish between the supported hardware modules and devices.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pBoardName

This argument returns a null-terminated ASCII string describing the specific hardware module. The following boarding naming scheme is implemented:

Value	Description
“TPMC863”	TPMC863 Device
“TMPE863”	TMPE863 Device

len

This argument specifies the maximum length available for storing the board name.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
char                    BoardName[40];

/*
** get board name
*/
result = tdrv009GetBoardName( hdl, &BoardName, 40 );

if (result == TDRV009_OK)
{
     printf(“Board Name: %s\n”, BoardName);
} else {
     /* handle error */
}
```

RETURN VALUE

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The specified device handle is invalid

2.2 Device Access Functions

2.2.1 tdrv009Read

NAME

tdrv009Read – Read data from device

SYNOPSIS

```
int tdrv009Read
(
        TDRV009_HANDLE          hdl,
        unsigned char           *pData,
        int                     nBytes
)
```

DESCRIPTION

This function reads data from the device. The data is returned on a byte basis, no frame information is returned. The function returns immediately after copying either the available or the requested amount of data.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pData

This argument points to a user supplied buffer. The data will be copied into this buffer.

nBytes

This parameter specifies the maximum number of bytes to be read (buffer size).

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
int                retval;
unsigned char      buffer[100];

/*-----------------------------------------
  Read up to 100 bytes from TDRV009 channel
  -----------------------------------------*/

retval = tdrv009Read( hdl, buffer, 100 );
if (retval != ERROR)
{
     printf(“%d bytes read\n”, retval);
     printf(“data = %s\n”, buffer);
}
else
{
     /* Handle Error */
}
```

RETURNS

On success, a positive number of properly read bytes is returned. In the case of an error, ERROR is returned and an error code will be stored in *errno*.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified.

2.2.2 tdrv009FrameRead

NAME

tdrv009FrameRead – Read data from a specified device

SYNOPSIS

```
TDRV009_STATUS tdrv009FrameRead
(
    TDRV009_HANDLE       hdl,
    TDRV009_DATA_BUFFER  *pDataBuffer
)
```

DESCRIPTION

This function reads one data buffer from the internal receive buffer. The function will return immediately, if data is available. If no data is available, the function will wait until data arrives, or the specified timeout occurs. The returned data buffer will be available until the next call to this I/O control function.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pDataBuffer

This parameter is a pointer to a *TDRV009_DATA_BUFFER* structure describing the data to read.

```
typedef struct
{
   unsigned char   *pData;
   unsigned int    Length;
   unsigned int    Valid;
   int             Overflow;
   int             Timeout;
} TDRV009_DATA_BUFFER;
```

pData

Pointer to the receive data section. This pointer directly references the driver's internal receive buffer. It points to the beginning of the corresponding data frame. Do not free the associated memory section!

Length

Number of valid data bytes for this data buffer.

Valid

This value specifies if the corresponding data buffer contains valid data, or if the read data buffer also contains a “Frame End”. The given values can be binary OR’ed. Possible values are:

Value	Description
TDRV009_RXBUF_DATAVALID	Current data block contains valid (unread) data.
TDRV009_RXBUF_FRAMEEND	Current data block is the last one of a received data frame.

Overflow

This value specifies if a buffer overrun has happened. The automatically stopped receiver is enabled again after the read access.

Timeout

This value specifies the maximum time to wait for incoming data. The timeout is specified in system ticks.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_DATA_BUFFER     DataBuf;

/*-------------------------------------------------
  Read a data buffer. Wait max. 120 ticks.
  -------------------------------------------------*/
DataBuf.Timeout = 120;

result = tdrv009FrameRead(hdl, &DataBuf);
if (result == TDRV009_OK)
{
     /* function succeeded */
     printf( “%d data bytes received.\n”, DataBuf.Length );
     if (DataBuf.Overflow)
     {
          printf( “Data was lost due to a buffer overflow.\n” );
     }
} else {
     /* handle the write error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_BUSY	The channel is currently busy with another write operation.
TDRV009_ERR_NOMEM	Not enough memory to queue this transmission.
TDRV009_ERR_IO	Failed to initialize the transmitter.

2.2.3 tdrv009Write

NAME

tdrv009Write – Write data from a buffer to a specified device

SYNOPSIS

```
TDRV009_STATUS tdrv009Write
(
    TDRV009_HANDLE      hdl,
    unsigned char       *pData,
    int                 nBytes
)
```

DESCRIPTION

This function can be used to write data to the device. The function returns immediately to the caller after queuing the data into the transmit descriptor list. Make sure that the supplied data buffer persists until the transmission is completed. To check if the data buffer is completely processed, namely the data has been transferred to the hardware FIFO, use the API function tdrv009WriteBufDone (see below).

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pData

This argument points to a user supplied buffer. The data of the buffer will be written to the device. The data buffer must be physically contiguous and accessible by the DMA controller.

nBytes

This parameter specifies the maximum number of bytes to write.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;
unsigned char      *pData;

/*------------------------------
  Write data to a TDRV009 device
  ------------------------------*/
pData = (char*)cacheDmaMalloc( 12 );
sprintf( (char*)pData, “Hello World” );

result = tdrv009Write(hdl, buffer, 12);
if (result == TDRV009_OK)
{
     printf(“Data queued for transmission.\n”);
}
else
{
     /* handle the write error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified.

2.2.4 tdrv009FrameWrite

NAME

tdrv009FrameWrite – Write data from a buffer to a specified device

SYNOPSIS

```
TDRV009_STATUS tdrv009FrameWrite
(
    TDRV009_HANDLE              hdl,
    TDRV009_TX_DATA_BUFFER      *pDataBuffer
)
```

DESCRIPTION

This function transmits one data buffer. The function will return immediately after handing over the buffer to the driver. It is possible to wait until the supplied data buffer is transferred to the hardware FIFO. This can be done either by waiting on the specified semaphore, or by polling the status parameter.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pDataBuffer

This parameter is a pointer to a *TDRV009_TX_DATA_BUFFER* structure describing the data to be sent.

```
typedef struct
{
   unsigned char    *pData;
   unsigned int     Length;
   unsigned int     Status;
   SEM_ID           WaitSema;
} TDRV009_TX_DATA_BUFFER;
```

pData

Pointer to the data section containing the data to be transmitted. This buffer must be physically coherent. Make sure that the data buffer is not modified during transmission.

Length

Number of data bytes for this data buffer.

Status

This value contains the current status of this transmit data packet. Possible values are:

Value	Description
TDRV009_TXDATA_BUSY	Data buffer is currently in use. Do not modify it.
TDRV009_TXDATA_COMPLETED	Data buffer is completed. The data is at least transferred into the hardware FIFO.

WaitSema

This value holds a semaphore created by the user. This semaphore will be signaled after the data buffer is processed.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
int                     retval;
unsigned char           *pData;
TDRV009_TX_DATA_BUFFER  TxDataBuf;

/*-------------------------------------------------------
  Send a data buffer. Wait max. 120 ticks for completion
  -------------------------------------------------------*/
pData = (unsigned char*)cacheDmaMalloc( 20*sizeof(unsigned char) );
sprintf((char*)pData, “Hello World!”);

TxDataBuf.Status   = 0;
TxDataBuf.WaitSema = semBCreate(SEM_Q_FIFO, SEM_EMPTY);
TxDataBuf.pData    = pData;
TxDataBuf.Length   = strlen((char*)pData);

result = tdrv009FrameWrite(hdl, &TxDataBuf);
if (result == TDRV009_OK)
{
     /*
     ** function succeeded, wait for completion of the data packet.
     */
     retval = semTake( TxDataBuf.WaitSema, 120 );
```

```
        if (retval == OK)
        {
            printf( “data buffer processed\n” );
        }
        else
        {
            printf( “data buffer still busy\n” );
        }
    }
    else
    {
        /* handle the write error */
    }
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_BUSY	The channel is currently busy with another write operation.
TDRV009_ERR_NOMEM	Not enough memory to queue this transmission.
TDRV009_ERR_IO	Failed to initialize the transmitter.

2.2.5 tdrv009WriteBufDone

NAME

tdrv009WriteBufDone – Write data from a buffer to a specified device

SYNOPSIS

```
TDRV009_STATUS tdrv009WriteBufDone
(
    TDRV009_HANDLE      hdl,
    unsigned char       *pData
)
```

DESCRIPTION

This function checks the current state of a transmit data buffer, which was previously handed over to the driver using the tdrv009Write() function. If the buffer has already been processed (i.e. data was transferred to the hardware FIFO), the function returns OK. If the buffer has not been processed, the function will return ERROR with an appropriate error code.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pData

This argument points to a user supplied buffer, which has previously been queued for transmission using function tdrv009Write().

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;
unsigned char      *pData;

/*-------------------------------------------------
  send TX data buffer
  -------------------------------------------------*/
pData = (char*)cacheDmaMalloc(…);
…
result = tdrv009Write(hdl, pData, …);
…

/*-------------------------------------------------
  check state of a TX data buffer
  -------------------------------------------------*/
result = tdrv009WriteBufDone(hdl, buffer);
if (result == TDRV009_OK)
{
     printf( “Data buffer completed.\n” );
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified.

2.2.6 tdrv009SetOperationMode

NAME

tdrv009SetOperationMode – Configure channel operation mode

SYNOPSIS

```
TDRV009_STATUS tdrv009SetOperationMode
(
    TDRV009_HANDLE                  hdl,
    TDRV009_OPERATION_MODE_STRUCT   *pOperationMode
)
```

DESCRIPTION

This function configures the channel's operation mode.

A call to this function must be done prior to any communication operation, because after driver startup, the channel's transceivers are disabled.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pOperationMode

This argument points to a TDRV009_OPERATION_MODE_STRUCT structure. It is necessary to completely initialize the structure. This can be done by calling the API function tdrv009GetOperationMode 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 clock 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”

TDRV009_HANDLE                  hdl;
TDRV009_STATUS                  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 = tdrv009SetOperationMode(hdl, &OperationMode);
if (result != TDRV009_OK)
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid parameter specified. Either the supplied buffer pointer is NULL, or a parameter inside the structure is invalid.

2.2.7 tdrv009GetOperationMode

NAME

tdrv009GetOperationMode – Return channel's current operation mode configuration

SYNOPSIS

```
TDRV009_STATUS tdrv009SetOperationMode
(
    TDRV009_HANDLE                  hdl,
    TDRV009_OPERATION_MODE_STRUCT   *pOperationMode
)
```

DESCRIPTION

This function function returns the channel's current operation mode.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pOperationMode

This argument points to a TDRV009_OPERATION_MODE_STRUCT structure.

```
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 clock source. 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 clock 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”

TDRV009_HANDLE                  hdl;
TDRV009_STATUS                  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 = tdrv009SetOperationMode(hdl, &OperationMode);
if (result != TDRV009_OK)
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid parameter specified. Either the supplied buffer pointer is NULL, or a parameter inside the structure is invalid.

2.2.8 tdrv009SetBaudrate

NAME

tdrv009SetBaudrate – Configure transmission rate

SYNOPSIS

```
TDRV009_STATUS tdrv009SetBaudrate
(
    TDRV009_HANDLE      hdl,
    int                 Baudrate
)
```

DESCRIPTION

This function sets up the transmission rate for the specific channel. This is done without changing the configuration set by tdrv009SetOperationMode. 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. This function specifies the desired frequency which should be generated by the Baud Rate Generator (BRG).

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Baudrate

This parameter specifies the baudrate which should be generated by the Baud Rate Generator. Be sure that the baudrate can be derived from the previously selected clock source.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Set baudrate to 14400bps
  ------------------------------*/
result = tdrv009SetBaudrate(hdl, 14400);
if (result == TDRV009_OK)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid parameter specified. The desired baudrate cannot be derived from the selected clock source

2.2.9 tdrv009SetReceiverState

NAME

tdrv009SetReceiverState – Enable or disable receiver

SYNOPSIS

```
TDRV009_STATUS tdrv009SetReceiverState
(
    TDRV009_HANDLE      hdl,
    int                 ReceiverState
)
```

DESCRIPTION

This function sets the channel's receiver either to active or inactive. This function must be called in user-ringbuffer-mode, after a buffer-overflow has happened and free receive buffers are available again.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

ReceiverState

This parameter defines the new state of the receiver. Possible values are:

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

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Enable the receiver
  ------------------------------*/
result = tdrv009SetReceiverState(hdl, TDRV009_RCVR_ON);
if (result == TDRV009_OK)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid parameter specified.

2.2.10 tdrv009ClearRxBuffer

NAME

tdrv009ClearRxBuffer – Discard all received data

SYNOPSIS

```
TDRV009_STATUS tdrv009ClearRxBuffer
(
    TDRV009_HANDLE      hdl
)
```

DESCRIPTION

This function removes all received data from the channel's receive buffer, and flushes the hardware FIFO as well.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include "tdrv009.h"

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Clear receive buffer
  ------------------------------*/
result = tdrv009ClearRxBuffer(hdl);
if (result == TDRV009_OK)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_IO	Error during reset or init of the receiver's hardware DMA engine.

2.2.11 tdrv009RtsSet

NAME

tdrv009RtsSet – Assert RTS signal

SYNOPSIS

```
TDRV009_STATUS tdrv009RtsSet
(
     TDRV009_HANDLE          hdl
)
```

DESCRIPTION

This function asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Assert RTS
  ------------------------------*/
result = tdrv009RtsSet(hdl);
if (result == TDRV009_OK)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_PERM	The channel is in hardware handshake mode, so this function is not allowed.
TDRV009_ERR_NOTSUP	This function is not supported by the specific channel.

2.2.12 tdrv009RtsClear

NAME

tdrv009RtsClear – De-assert RTS signal

SYNOPSIS

```
TDRV009_STATUS tdrv009RtsClear
(
    TDRV009_HANDLE      hdl
)
```

DESCRIPTION

This 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.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  De-assert RTS
  ------------------------------*/
result = tdrv009RtsClear(hdl);
if (result == TDRV009_OK)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_PERM	The channel is in hardware handshake mode, so this function is not allowed.
TDRV009_ERR_NOTSUP	This function is not supported by the specific channel.

2.2.13 tdrv009CtsGet

NAME

tdrv009CtsGet – Return status of CTS signal

SYNOPSIS

```
TDRV009_STATUS tdrv009CtsGet
(
    TDRV009_HANDLE      hdl,
    unsigned int        *pCtsState
)
```

DESCRIPTION

This function returns the current state of the CTS handshake signal line of the specific channel.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pCtsState

This parameter points to an unsigned int buffer where the status of the CTS signal will be stored. Depending on the state of CTS, either 0 (inactive) or 1 (active) is returned.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;
unsigned int       CtsStatus;

/*------------------------------
  Read CTS state
  ------------------------------*/
result = tdrv009CtsGet(hdl, &CtsStatus);
if (result == TDRV009_OK)
{
     /* function succeeded */
     printf( “CTS = %d\n”, CtsStatus );
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	The supplied buffer pointer is NULL.
TDRV009_ERR_NOTSUP	This function is not supported by the specific channel.

2.2.14 tdrv009DtrSet

NAME

tdrv009DtrSet – Assert DTR signal

SYNOPSIS

```
TDRV009_STATUS tdrv009DtrSet
(
     TDRV009_HANDLE          hdl
)
```

DESCRIPTION

This function sets the DTR signal line to HIGH. This function is only available for the 4th channel of a TDRV009 module.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Set DTR to HIGH (only valid for channel 3)
  ------------------------------*/
result = tdrv009DtrSet(hdl);
if (result == TDRV009_OK)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_NOTSUP	This function is not supported by the specific channel.

2.2.15 tdrv009DtrClear

NAME

tdrv009DtrClear – De-assert DTR signal

SYNOPSIS

```
TDRV009_STATUS tdrv009DtrClear
(
    TDRV009_HANDLE      hdl
)
```

DESCRIPTION

This function sets the DTR signal line to LOW. This function is only available for the 4th channel of a TDRV009 module.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include "tdrv009.h"

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Set DTR to LOW (only valid for channel 3)
  ------------------------------*/
result = tdrv009DtrClear(hdl);
if (result == TDRV009_OK)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_NOTSUP	This function is not supported by the specific channel.

2.2.16 tdrv009DsrGet

NAME

tdrv009DsrGet – Return status of DSR signal

SYNOPSIS

```
TDRV009_STATUS tdrv009DsrGet
(
    TDRV009_HANDLE      hdl,
    unsigned int        *pDsrState
)
```

DESCRIPTION

This function returns the current state of the DSR signal line of the specific channel. This function is only available for the 4th channel of a TDRV009 module

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pDsrState

This parameter points to an unsigned int buffer where the status of the DSR signal will be stored. Depending on the state of DSR, either 0 (inactive) or 1 (active) is returned.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;
unsigned int       DsrStatus;

/*------------------------------
  Read DSR state
  ------------------------------*/
result = tdrv009DsrGet(hdl, &DsrStatus);
if (result == TDRV009_OK)
{
     /* function succeeded */
     printf( “DSR = %d\n”, DsrStatus );
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	The supplied buffer pointer is NULL.

2.2.17 tdrv009SetExternalXtal

NAME

tdrv009SetExternalXtal – Configure externally supplied oscillator frequency

SYNOPSIS

```
TDRV009_STATUS tdrv009SetExternalXtal
(
    TDRV009_HANDLE      hdl,
    int                 XtalFrequency
)
```

DESCRIPTION

This function specifies the frequency of an externally provided clock. 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.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

XtalFrequency

This parameter specifies the clock frequency in Hz.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*------------------------------
  Specify 1MHz as external clock frequency
  ------------------------------*/
result = tdrv009SetExternalXtal(hdl, 1000000);
if (result == TDRV009_OK)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid parameter specified. The specified frequency must be larger than 0.

2.2.18 tdrv009RegisterRead

NAME

tdrv009RegisterRead – Read from controller's register space

SYNOPSIS

```
TDRV009_STATUS tdrv009RegisterRead
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
)
```

DESCRIPTION

This function reads one 32bit word from the communication controller's register space.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRegisterBuffer

This Parameter is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
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”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

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

result = tdrv009RegisterRead(hdl, &AddrBuf);
if (result == TDRV009_OK)
{
     printf( “Value = 0x%X\n”, AddrBuf.Value );
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_ACCESS	The specified offset is invalid.

2.2.19 tdrv009RegisterWrite

NAME

tdrv009RegisterWrite – Write to controller's register space

SYNOPSIS

```
TDRV009_STATUS tdrv009RegisterWrite
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
)
```

DESCRIPTION

This function writes one 32bit word to the communication controller's register space.

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

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRegisterBuffer

This Parameter is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
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.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

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

result = tdrv009RegisterWrite(hdl, &AddrBuf);
if (result != TDRV009_OK)
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_ACCESS	The specified offset is invalid.

2.2.20 tdrv009EepromRead

NAME

tdrv009EepromRead – Read from EEPROM

SYNOPSIS

```
TDRV009_STATUS tdrv009EepromRead
(
    TDRV009_HANDLE          hdl,
    TDRV009_EEPROM_BUFFER   *pEepromBuffer
)
```

DESCRIPTION

This function reads one 16bit word from the onboard EEPROM.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pEepromBuffer

This Parameter is a pointer to a *TDRV009_EEPROM_BUFFER* structure.

```
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

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_EEPROM_BUFFER   EepromBuf;

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

result = tdrv009EepromRead(hdl, &EepromBuf);
if (result == TDRV009_OK)
{
     printf( “Value = 0x%X\n”, EepromBuf.Value );
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_ACCESS	Specified offset is invalid, and may not be accessed.

2.2.21 tdrv009EepromWrite

NAME

tdrv009EepromWrite – Write to controller's register space

SYNOPSIS

```
TDRV009_STATUS tdrv009EepromWrite
(
    TDRV009_HANDLE          hdl,
    TDRV009_EEPROM_BUFFER   *pEepromBuffer
)
```

DESCRIPTION

This 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.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pEepromBuffer

This Parameter is a pointer to a *TDRV009_EEPROM_BUFFER* structure.

```
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

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_EEPROM_BUFFER   EepromBuf;

/*-------------------------------------------------
  Write a 16bit value into the EEPROM, offset 60h
  -------------------------------------------------*/
EepromBuf.Offset = 0x60;
EepromBuf.Value  = 0x1234;

result = tdrv009EepromWrite(hdl, &EepromBuf);
if (result != TDRV009_OK)
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_ACCESS	The specified offset address is invalid, or read-only.

2.2.22 tdrv009RingbufferRegister

NAME

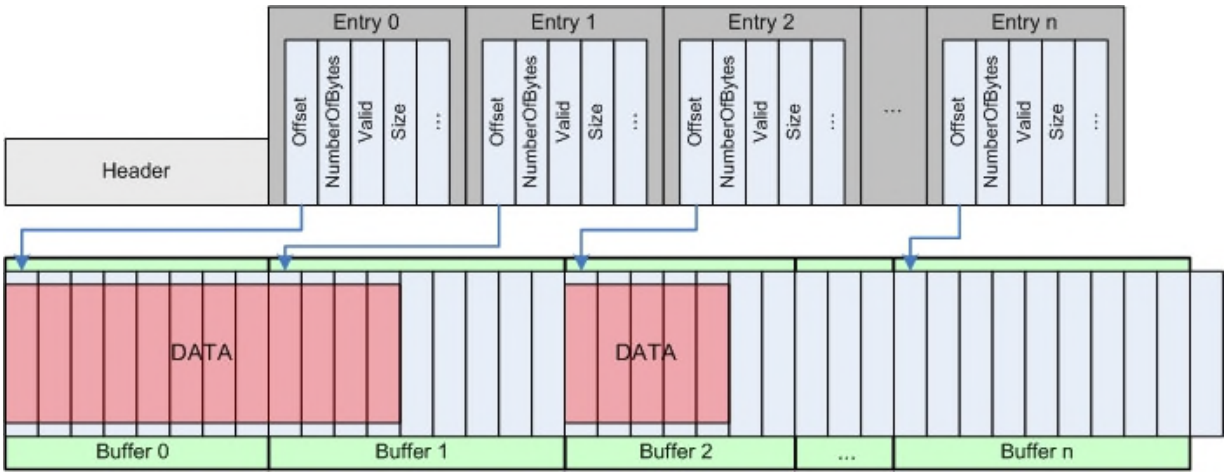
tdrv009RingbufferRegister – Register user-supplied ringbuffer

SYNOPSIS

```
TDRV009_STATUS tdrv009RingbufferRegister
(
    TDRV009_HANDLE          hdl,
    TDRV009_RINGBUFFER      *pRingBuffer
)
```

DESCRIPTION

This advanced function provides a user-supplied ringbuffer to the driver using a pointer to a dynamic TDRV009_RINGBUFFER structure. The buffer consists of a header and a dynamically expandable data section. The driver formats the allocated buffer automatically according to the specification. The user must ensure that the buffer persists as long as it is registered to the driver. The buffer must be physically coherent, because direct memory access is used. Please refer to the following figure for further information.



PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRingBuffer

This Parameter is a pointer to a *TDRV009_RINGBUFFER* structure.

```
typedef struct
{
   TDRV009RINGBUFFER_HEADER   Header;
   unsigned char              DataSection[40];  /* dynamically expandable buffer     */
} TDRV009_RINGBUFFER;
```

Header

Contains administrative information used for buffer management.

DataSection

Inside the data section the real data buffers and their corresponding entry information is stored.

The TDRV009_RINGBUFFER_HEADER structure has the following layout:

```
typedef struct
{
   unsigned int    BufferSize;      /* total size of memory available for data          */
   unsigned int    PacketSize;      /* size of one data packet, adjusted by driver      */
   unsigned int    NumberOfEntries; /* number of entries                                */
   unsigned int    get;             /* index where data can be read                     */
   unsigned int    put;             /* current index where new data is filled in        */
   unsigned int    Overflow;        /* TRUE if an overflow happened. Receiver is disabled. */
} TDRV009_RINGBUFFER_HEADER;
```

BufferSize

Memory size available for the complete data buffers.

PacketSize

Memory size available for one data packet. The complete data buffer is divided into several packets matching the specified PacketSize.

NumberOfEntries

Number of available single buffers. This value is calculated out of BufferSize and PacketSize. Read Only.

get

Indicates the index where new data can be read out of the ringbuffer.

put

Indicates the index where the driver fills in new arrived data. Read Only.

Overflow

Indicates if a buffer overflow has happened. After a buffer overflow the receiving channel is disabled and must be enabled again by hand. Read Only.

The TDRV009_RINGBUFFER_ENTRY structure has the following layout:

```
typedef struct
{
   unsigned int     NumberOfBytes;     /* number of valid bytes for the current buffer index      */
   unsigned int     Size;              /* total size of the current buffer                        */
   unsigned int     Offset;            /* offset into the data section where the buffer starts    */
   unsigned int     DmaAddress;      /* physical address for the dma controller                   */
   unsigned int     Valid;             /* TRUE if the current buffer contains valid data          */
} TDRV009_RINGBUFFER_ENTRY;
```

NumberOfBytes

This value indicates the number of valid bytes inside the corresponding buffer. Read Only.

Size

This value indicates the total size of the corresponding data buffer. Read Only.

Offset

This value specifies the offset relative to the beginning of the data section, where the corresponding data buffer starts. Read Only.

DmaAddress

This value states the physical address of the data buffer used by the DMA controller. Read Only.

Valid

This value indicates if the corresponding data buffer contains valid data. This is the only value that should be modified by the user.

To ensure the correct functionality of the ringbuffer mode, do not change any of the administrative read-only information values.

ASSISTANT MACROS AND FUNCTIONS FOR BUFFER HANDLING

To help the user work with this ringbuffer concept some assistant macros and functions were defined in "*tdrv009.h*". They are explained in the following.

`TDRV009_CALCULATE_RINGBUFFER_SIZE( BufferSize, PacketSize )`

calculates the total size necessary for allocation of the complete ringbuffer

`TDRV009_GET_BUFFER( pRingBuffer, index )`

returns a pointer to the corresponding data buffer

`TDRV009_IS_VALID( pRingBuffer, index )`

TRUE if the corresponding buffer contains valid data, otherwise FALSE

`TDRV009_SET_VALID( pRingBuffer, index, value )`

sets the Valid-flag for the specified entry to *value*

`TDRV009_CLEAR_BUFFER( pRingBuffer, index )`

clears the corresponding data buffer

`TDRV009_GET_POS( pRingBuffer )`

returns the current get-position where new data can be read

`TDRV009_OVERFLOW( pRingBuffer )`

TRUE if a buffer overflow has happened, otherwise false

```
unsigned char*
tdrv009GetNewBuffer(TDRV009_RINGBUFFER *pRingBuffer, unsigned int *length);
```

returns a pointer to a buffer containing new data. The *get*-position is set to the next entry, the number of valid bytes is returned in *length*. The previous data buffer is marked as invalid, so the driver may reuse this buffer for data reception.

For further information on how to deal with the ringbuffer, the special assistant macros, and functions, please refer to the provided example application.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_RINGBUFFER*     pRingBuffer;

#define BUFFER_SIZE     5000
#define PACKET_SIZE     100

/*-------------------------------------------------------------
  allocate memory for ringbuffer (physically coherent, nonchached):
     5000 bytes total data space
      100 bytes per packet
  -------------------------------------------------------------*/
pRingBuffer = (TDRV009_RINGBUFFER*)cacheDmaMalloc(
              TDRV009_CALCULATE_RINGBUFFER_SIZE( BUFFER_SIZE, PACKET_SIZE )
                  );
pRingBuffer->Header.BufferSize = BUFFER_SIZE;
pRingBuffer->Header.PacketSize = PACKET_SIZE;

result = tdrv009RingbufferRegister(hdl, pRingBuffer);
if (result != TDRV009_OK)
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_BUSY	A ringbuffer is already registered. Unregister it first.
TDRV009_ERR_NOMEM	Not enough memory available to create necessary receive descriptor list.
TDRV009_ERR_IO	Error during reset or init of the receiver's hardware DMA engine.

2.2.23 tdrv009RingbufferUnregister

NAME

tdrv009RingbufferUnregister – Unregister user-supplied ringbuffer

SYNOPSIS

```
TDRV009_STATUS tdrv009RingbufferUnregister
(
    TDRV009_HANDLE        hdl
)
```

DESCRIPTION

This function unregisters a previously registered user-ringbuffer. A channel-reset is performed and the driver uses its internal ringbuffer to receive data again.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*-----------------------------------
  unregister user-supplied ringbuffer
  -----------------------------------*/
result = tdrv009RingbufferUnregister(hdl);
if (result == TDRV009_OK)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_ACCESS	No user-supplied ringbuffer is used, so this function is not supported at the moment.
TDRV009_ERR_IO	Error during reset or init of the receiver's hardware DMA engine.

2.2.24 tdrv009WaitForInterrupt

NAME

tdrv009WaitForInterrupt – Wait for SCC interrupt event

SYNOPSIS

```
TDRV009_STATUS tdrv009WaitForInterrupt
(
    TDRV009_HANDLE          hdl,
    TDRV009_WAIT_STRUCT     *pWaitBuffer
)
```

DESCRIPTION

This function waits until a specified SCC-interrupt or the timeout occurs.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pWaitBuffer

This parameter is a pointer to a *TDRV009_WAIT_STRUCT* structure.

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

Interrupts

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

Timeout

This parameter specifies the time (in system ticks) to wait for an interrupt. If -1 is specified, the function will block indefinitely.

EXAMPLE

```
#include “tdrv009.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_WAIT_STRUCT     WaitStruct;

/*-------------------------------------------------
  Wait at least 60 system ticks for a
  CTS Status Change (CSC) interrupt
  -------------------------------------------------*/
WaitStruct.Interrupts = (1 << 14);
WaitStruct.Timeout    = 60;

result = tdrv009WaitForInterrupt(hdl, &WaitStruct);
if (result == TDRV009_OK)
{
     printf( “Occurred Interrupt = 0x%X\n”, WaitStruct.Interrupts );
}
else
{
     /* handle the error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The device handle is invalid
TDRV009_ERR_INVAL	Invalid data buffer specified. The supplied buffer pointer is NULL.
TDRV009_ERR_BUSY	Too many simultaneous wait jobs active.
TDRV009_ERR_TIMEOUT	Timeout occurred.