The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV009-SW-25

Integrity Device Driver

High Speed Synch/Asynch Serial Interface

Version 1.0.x

User Manual

Issue 1.0.0

May 2019

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

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-25

Integrity Device Driver

High Speed Synch/Asynch Serial Interface

Supported Modules:

TPMC363
TPMC863
TCP863
TAMC863
TPCE863

Issue	Description	Date
1.0.0	First Issue	May 27, 2019

Table of Contents

1 INTRODUCTION 4
2 INSTALLATION 5
2.1 Driver Installation 5
2.2 TDRV009 Applications 5
3 API DOCUMENTATION 6
3.1 General Functions 6
3.1.1 tdrv009Open 6
3.1.2 tdrv009Close 8
3.2 Device Access Functions 10
3.2.1 tdrv009Write 10
3.2.2 tdrv009Read 12
3.2.3 tdrv009SetOperationMode 14
3.2.4 tdrv009GetOperationMode 22
3.2.5 tdrv009SetBaudrate 29
3.2.6 tdrv009SetReceiverState 31
3.2.7 tdrv009ClearRxBuffer 33
3.2.8 tdrv009RtsSet 35
3.2.9 tdrv009RtsClear 37
3.2.10 tdrv009CtsGet 39
3.2.11 tdrv009DtrSet 41
3.2.12 tdrv009DtrClear 43
3.2.13 tdrv009DsrGet 45
3.2.14 tdrv009SetExternalXtal 47
3.2.15 tdrv009SccRegisterRead 49
3.2.16 tdrv009SccRegisterWrite 51
3.2.17 tdrv009GlobalRegisterRead 53
3.2.18 tdrv009GlobalRegisterWrite 55
3.2.19 tdrv009EepromRead 57
3.2.20 tdrv009EepromWrite 59
4 APPENDIX 61
4.1 Example Application 61

1 Introduction

The TDRV009-SW-25 Integrity device driver software allows the operation of the supported PMC conforming to the Integrity I/O system specification. The software is designed and tested with Integrity 11.4.4.

The TDRV009-SW-25 device driver supports the following features:

- setup and configure serial channels
- send and receive data buffers (character oriented)
- read and write onboard registers directly
- read and write access to onboard EEPROM
- control handshake lines

The TDRV009-SW-25 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	Advanced Mezzanine Card
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 of the supported modules listed below.

TPMC863 (or compatible) User Manual
Greenhills MULTI and INTEGRITY Documentation

2 Installation

The following files are located on the distribution media:

Directory path TDRV009-SW-25:

tdrv009.c	Device driver source
tdrv009def.h	Driver include file
commCtrl.h	Driver include file
tdrv009.h	Include file for driver and application
tdrv009api.c	Application interface, simplifies device access
tdrv009api.h	Include file for API and applications
example/tdrv009exa.c	Example application
TDRV009-SW-25-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Driver Installation

Copy the TDRV009 driver files into a desired driver or project path. The driver source file tdrv009.c must be included into the kernel project and the BSP paths must be added to the include search paths of the file. Set Options… ➔ Project → Include Directories, then double click and add the new paths:

$(__OS_DIR)/bsp
$(__OS_DIR)/system
$(__OS_DIR)/modules/ghs/bspsrc
$(__OS_DIR)/modules/ghs/bspsrc/support
$(__OS_DIR)/modules/ghs/bspsrc/driver/busspace

Afterwards the project must be rebuilt. The driver will be started automatically after booting the image and the driver will be requested if a matching device is detected in the system.

For further information building a kernel, please refer to Greenhills MULTI and INTEGRITY Documentation.

2.2 TDRV009 Applications

Copy the TDRV009 API files (tdrv009api.c, tdrv009api.h, and tdrv009.h) into a desired application path, and include tdrv009api.c into the application project.

The application source file must include tdrv009api.h. If these steps are done, the TDRV009 API can be used and the devices will be accessible.

3 API Documentation

3.1 General Functions

3.1.1 tdrv009Open

NAME

tdrv009Open() – open a device.

SYNOPSIS

```
TDRV009_HANDLE 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.

PARAMETER

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 “tdrv009api.h”

TDRV009_HANDLE     hdl;

/*
** open file descriptor to device
*/
hdl = tdrv009Open(“tdrv009_0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURN VALUE

A device descriptor pointer or NULL if the function fails.

3.1.2 tdrv009Close

NAME

tdrv009Close() – close a device.

SYNOPSIS

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

DESCRIPTION

This function closes previously opened devices.

PARAMETER

hdl

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

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;

/*
** close the device
*/
result = tdrv009Close(hdl);
if (result != TDRV009_OK)
{
    /* handle close 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

3.2 Device Access Functions

3.2.1 tdrv009Write

NAME

tdrv009Write – Write data from a buffer to a specified device

SYNOPSIS

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

DESCRIPTION

This function attempts to write a data buffer to the specified TDRV009 channel. The user specifies a character buffer pointed to by *pData*. The argument *nBytes* specifies the length of the buffer. The function performs a blocking write operation, i.e. the function returns after the data has been transferred from the transmit-queue into the hardware FIFO.

PARAMETERS

hdl

This value specifies the device handle to the hardware channel 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 transferred to the device.

nBytes

This parameter specifies the number of bytes to write.

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE  hdl;
unsigned char   Data[100];
int             result;

/*
** Send data on TDRV009 channel
*/
sprintf( (char*)Data, "Hello World" );

result = tdrv009Write (
              hdl,
              Data,
              strlen((char*)Data)
         );
if (result <= 0)
{
     /* handle error */
}
```

RETURNS

On success, the number of transferred bytes is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.2 tdrv009Read

NAME

tdrv009Read – Read data from device

SYNOPSIS

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

DESCRIPTION

This function attempts to read an input buffer from a TDRV009 channel. The argument *nBytes* specifies the maximum length of the buffer. Available data (up to *nBytes* bytes) is copied into the user's buffer pointed to by *pData*. The function performs a non-blocking read operation, i.e. the function returns immediately even if no data is available.

PARAMETERS

hdl

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

pData

This argument points to a user supplied buffer where the received data will be stored.

nBytes

This parameter specifies the number of bytes to read.

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE  hdl;
int             result;
unsigned char   Data[100];

/*
** Receive up to 100 data bytes on TDRV009 channel
*/
result = tdrv009Read (
               hdl,
               Data,
               100
         );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

On success, the number of transferred bytes is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.3 tdrv009SetOperationMode

NAME

tdrv009SetOperationMode – Configure Channel Operation Mode

SYNOPSIS

```
TDRV009_STATUS tdrv009SetOperationMode
(
    TDRV009_HANDLE              hdl,
    TDRV009_OPERATION_MODE      *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 16-times 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 "tdrv009api.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)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. A parameter inside the structure is invalid.

Other returned error codes are system error conditions.

3.2.4 tdrv009GetOperationMode

NAME

tdrv009GetOperationMode – Return Channel's current Operation Mode Configuration

SYNOPSIS

```
TDRV009_STATUS tdrv009GetOperationMode
(
    TDRV009_HANDLE              hdl,
    TDRV009_OPERATION_MODE      *pOperationMode
)
```

DESCRIPTION

This function reads the channel's current operation mode configuration.

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 16-times 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 "tdrv009api.h"

TDRV009_HANDLE                  hdl;
TDRV009_STATUS                  result;
TDRV009_OPERATION_MODE_STRUCT   OperationMode;

/*-------------------------------------------------
  Read Channel Operation Mode
  -------------------------------------------------*/
result = tdrv009GetOperationMode(hdl, &OperationMode);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.5 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Set baudrate to 14400bps
  ------------------------------*/
result = tdrv009SetBaudrate(hdl, 14400);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The desired baudrate cannot be derived from the selected clock source.

Other returned error codes are system error conditions.

3.2.6 tdrv009SetReceiverState

NAME

tdrv009SetReceiverState – Enable/Disable the receiver

SYNOPSIS

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

DESCRIPTION

This function sets the channel's receiver either to active or inactive.

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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Enable the receiver
  ------------------------------*/
result = tdrv009SetReceiverState(hdl, TDRV009_RCVR_ON);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified.

Other returned error codes are system error conditions.

3.2.7 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Clear receive buffer
  ------------------------------*/
result = tdrv009ClearRxBuffer( hdl );
if (result == TDRV009_OK)
{
    /* OK */
} else {
    /* handle 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 TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.8 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Assert RTS
  ------------------------------*/
result = tdrv009RtsSet( hdl );
if (result == TDRV009_OK)
{
    /* OK */
} else {
    /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	The channel is in handshake mode, so this function is not allowed.

Other returned error codes are system error conditions.

3.2.9 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  De-Assert RTS
  ------------------------------*/
result = tdrv009RtsClear( hdl );
if (result == TDRV009_OK)
{
    /* OK */
} else {
    /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	The channel is in handshake mode, so this function is not allowed.

Other returned error codes are system error conditions.

3.2.10 tdrv009CtsGet

NAME

tdrv009CtsGet – Return status of CTS signal

SYNOPSIS

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

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.

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 "tdrv009api.h"

TDRV009_HANDLE   hdl;
TDRV009_STATUS   result;
unsigned int     CtsStatus;

/*------------------------------
  Read CTS state
  ------------------------------*/
result = tdrv009CtsGet(hdl, &CtsStatus);
if (result == TDRV009_OK)
{
     /* OK */
     printf( "CTS = %d\n", CtsStatus );
} else {
     /* handle 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 TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.11 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Set DTR to HIGH (only valid for channel 3)
  ------------------------------*/
result = tdrv009DtrSet(hdl);
if (result == TDRV009_OK)
{
    /* OK */
} else {
    /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	This function is not supported by the specific channel.

Other returned error codes are system error conditions.

3.2.12 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Set DTR to LOW (only valid for channel 3)
  ------------------------------*/
result = tdrv009DtrClear(hdl);
if (result == TDRV009_OK)
{
    /* OK */
} else {
    /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	This function is not supported by the specific channel.

Other returned error codes are system error conditions.

3.2.13 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 "tdrv009api.h"

TDRV009_HANDLE   hdl;
TDRV009_STATUS   result;
unsigned int     DsrStatus;

/*------------------------------
  Read DSR state
  ------------------------------*/
result = tdrv009DsrGet(hdl, &DsrStatus);
if (result == TDRV009_OK)
{
     /* OK */
     printf( "DSR = %d\n", DsrStatus);
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	This function is not supported by the specific channel.

Other returned error codes are system error conditions.

3.2.14 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 "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Specify 1MHz as external clock frequency
  ------------------------------*/
result = tdrv009SetExternalXtal(hdl, 1000000);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.15 tdrv009SccRegisterRead

NAME

tdrv009SccRegisterRead – Read from Controller's SCC Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009SccRegisterRead
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
)
```

DESCRIPTION

This function reads one 32bit word from the communication controller's channel-specific SCC 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 channel-specific SCC register space, relative to the start of the channel's SCC register area. Please refer to the hardware user manual for further information.

Value

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

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Read a 32bit value (Status Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0004;
result = tdrv009SccRegisterRead ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     printf( "Value = 0x%X\n", AddrBuf.Value );
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.16 tdrv009SccRegisterWrite

NAME

tdrv009SccRegisterWrite – Write to Controller's SCC Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009SccRegisterWrite
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
)
```

DESCRIPTION

This function writes one 32bit word to the communication controller's channel-specific SCC 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 channel-specific SCC register space, relative to the start of the channel's SCC register area. Please refer to the hardware user manual for further information.

Value

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

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Write a 32bit value (Termination Character Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0048;
AddrBuf.Value  = (1 << 15) | 0x42;
result = tdrv009SccRegisterWrite ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.17 tdrv009GlobalRegisterRead

NAME

tdrv009GlobalRegisterRead – Read from Controller's Global Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009GlobalRegisterRead
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
)
```

DESCRIPTION

This function reads one 32bit word from the communication controller's Global 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 global register space. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication global register space.

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Read a 32bit value (Version Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x00F0;
result = tdrv009GlobalRegisterRead ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     printf( "Value = 0x%X\n", AddrBuf.Value );
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.18 tdrv009GlobalRegisterWrite

NAME

tdrv009GlobalRegisterWrite – Write to controller's Global Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009GlobalRegisterWrite
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
)
```

DESCRIPTION

This function writes one 32bit word to the communication controller's Global 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 global register space. Please refer to the hardware user manual for further information.

Value

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

EXAMPLE

```
#include "tdrv009api.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 = tdrv009GlobalRegisterWrite ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.19 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 short    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 "tdrv009api.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 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.20 tdrv009EepromWrite

NAME

tdrv009EepromWrite – Write from EEPROM

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 short    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 "tdrv009api.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)
{
     /* OK */
} else {
     /* handle 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 TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

4 Appendix

4.1 Example Application

The example application shall give an overview about the use of the TDRV009 devices and how to use the TDRV009 API.

The example application is designed as an interactive console application, so make sure to properly redirect the standard input and standard output for the example application's address space. If using a Dynamic Download Build e.g. in a telnet shell, use the following command:

```
# run -filtered <example_filename> -args <example_address_space>
```