The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV009-SW-65

Windows 2000/XP Device Driver

High Speed Sync/Async Serial Interface

Version 1.0.x

User Manual

Issue 1.0.2

June 2008

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com

Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com

Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TDRV009-SW-65

Windows 2000/XP Device Driver

High Speed Sync/Async Serial Interface

Supported Modules:
TPMC863
TPMC363
TCP863

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

Issue	Description	Date
1.0.0	First Issue	May 21, 2007
1.0.1	Description of missing ioctl functions added (RTS/CTS/DTR/DSR)	June 19, 2007
1.0.2	Files moved to subdirectory	June 23, 2008

Table of Contents

1 INTRODUCTION....................4
2 INSTALLATION....................5
2.1 Software Installation....................5
2.1.1 Windows 2000 / XP....................5
2.1.2 Confirming Windows 2000 / XP Installation....................5
3 DRIVER CONFIGURATION....................6
3.1 Receive Buffer Configuration....................6
4 DEVICE DRIVER PROGRAMMING....................7
4.1 Files and I/O Functions....................7
4.1.1 Opening a Device....................7
4.1.2 Closing a Device....................9
4.1.3 Device I/O Control Functions....................10
4.1.3.1 IOCTL_TDRV009_READ....................12
4.1.3.2 IOCTL_TDRV009_WRITE....................14
4.1.3.3 IOCTL_TDRV009_SET_OPERATION_MODE....................16
4.1.3.4 IOCTL_TDRV009_GET_OPERATION_MODE....................23
4.1.3.5 IOCTL_TDRV009_SET_BAUDRATE....................29
4.1.3.6 IOCTL_TDRV009_RINGBUF_REGISTER....................30
4.1.3.7 IOCTL_TDRV009_RINGBUF_UNREGISTER....................36
4.1.3.8 IOCTL_TDRV009_SET_RECEIVER_STATE....................37
4.1.3.9 IOCTL_TDRV009_CLEAR_RX_BUFFER....................38
4.1.3.10 IOCTL_TDRV009_SET_EXT_XTAL....................39
4.1.3.11 IOCTL_TDRV009_SCC_REG_WRITE....................40
4.1.3.12 IOCTL_TDRV009_SCC_REG_READ....................42
4.1.3.13 IOCTL_TDRV009_GLOB_REG_WRITE....................44
4.1.3.14 IOCTL_TDRV009_GLOB_REG_READ....................46
4.1.3.15 IOCTL_TDRV009_EEPROM_WRITE....................48
4.1.3.16 IOCTL_TDRV009_EEPROM_READ....................50
4.1.3.17 IOCTL_TDRV009_WAITFORINTERRUPT....................52
4.1.3.18 IOCTL_TDRV009_EVENT_REGISTER....................54
4.1.3.19 IOCTL_TDRV009_EVENT_UNREGISTER....................56
4.1.3.20 IOCTL_TDRV009_RTS_SET....................58
4.1.3.21 IOCTL_TDRV009_RTS_CLEAR....................59
4.1.3.22 IOCTL_TDRV009_CTS_GET....................60
4.1.3.23 IOCTL_TDRV009_DTR_SET....................61
4.1.3.24 IOCTL_TDRV009_DTR_CLEAR....................62
4.1.3.25 IOCTL_TDRV009_DSR_GET....................63

1 Introduction

The TDRV009-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TPMC863 product family on an Intel or Intel-compatible x86 Windows 2000, Windows XP, Windows XP embedded operating system.

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

The TDRV009-SW-65 device driver supports the following features:

- Setup and configure a channel
- Send and receive Data Buffers
- Register and unregister a Receive-Ringbuffer
- Switch on or off a channel's receiver
- Read and write onboard registers directly
- Wait for Receive Events
- Wait for interrupt events

The TDRV009-SW-65 device driver supports the modules listed below:

TPMC863	4 Channel Interface	(PMC)
TPMC363	4 Channel Interface	(PMC, Conduction Cooled)
TCP863	4 Channel Interface	(CompactPCI)

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

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

TPMC863 Product Family User manual

TPMC863 Product Family Engineering Manual

2 Installation

Following files are located in directory TDRV009-SW-65 on the distribution media:

tdrv009bus.sys	Windows 2000/XP driver binary (Channel Enumerator)
tdrv009bus.inf	Windows 2000/XP installation script (Channel Enumerator)
tdrv009.sys	Windows 2000/XP driver binary (Channel Driver)
tdrv009.inf	Windows 2000/XP installation script (Channel Driver)
tdrv009.h	Header file with IOCTL code and structure definitions
commCtrl.h	Header file with controller specific definitions
TDRV009-SW-65-1.0.2.pdf	This document
example/tdrv009exa.c	Example application
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TDRV009 Device Driver on a Windows 2000 / XP operating system.

After installing the TDRV009 module(s) and boot-up your system, Windows 2000 / XP setup will show a "***New hardware found***" dialog box.

(1) The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
Click "***Next***" button to continue.

(2) In the following dialog box, choose "***Search for a suitable driver for my device***".
Click "***Next***" button to continue.

(3) In Drive A, insert the TDRV009 driver disk; select "***Disk Drive***" in the dialog box.
Click "***Next***" button to continue.

(4) Now the driver wizard should find a suitable device driver on the diskette.
Click "***Next***" button to continue.

(5) Complete the upgrade device driver and click "***Finish***" to take all the changes effect.

(6) Repeat the steps above for each single channel device created by the busdriver.

After successful installation the TDRV009 device driver will start immediately and create devices (TDRV009_1, TDRV009_2 ...) for all recognized TDRV009 channels.

2.1.2 Confirming Windows 2000 / XP Installation

To confirm that the driver has been properly loaded in Windows 2000 / XP, perform the following steps:

(1) From Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***".

(2) Click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.

(3) Click the "**+**" in front of "***Other Devices***".
The driver "***TEWS TECHNOLOGIES – TDRV009 (<ModuleName>)***" should appear.

3 Driver Configuration

3.1 Receive Buffer Configuration

To configure the size of the internal receive buffer, adjust the following registry keys according to the description in the corresponding manual section:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tdrv009\RxBufferSize
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tdrv009\RxPacketSize
```

Default values are:

RxBufferSize	RxPacketSize
20000	256

If the default values are not suitable the configuration can be changed by modifying the registry, for instance with regedt32.

The changes take effect after a new start of the driver.

4 Device Driver Programming

The TDRV009-SW-65 Windows WDM device driver is a kernel mode device driver using Direct I/O.

All of these standard Win32 functions are described in detail in the Windows Platform SDK Documentation (Windows base services / Hardware / Device Input and Output).

For details refer to the Win32 Programmers Reference of your used programming tools (C++, Visual Basic etc.)

4.1 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TDRV009 device driver. Only the required parameters are described in detail.

4.1.1 Opening a Device

Before you can perform any I/O the TDRV009 channel device must be opened by invoking the CreateFile function. CreateFile returns a handle that can be used to access the TDRV009 channel device.

```
HANDLE CreateFile(
        LPCTSTR    lpFileName,
        DWORD      dwDesiredAccess,
        DWORD      dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD      dwCreationDistribution,
        DWORD      dwFlagsAndAttributes,
        HANDLE     hTemplateFile
)
```

Parameters

lpFileName

Points to a null-terminated string, which specifies the name of the TDRV009 channel to open. The lpFileName string should be of the form \\.\TDRV009_x to open the device x. The ending x is a one-based number. The first channel device found by the driver is \\.\TDRV009_1, the second \\.\TDRV009_2 and so on.

dwDesiredAccess

Specifies the type of access to the TDRV009. For the TDRV009 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE)

dwShareMode

Set of bit flags that specify how the object can be shared. Set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TDRV009 channel devices.

dwCreationDistribution

Specifies which action to take on files that exist, and which action to take when files do not exist. TDRV009 channel devices must be always opened **OPEN_EXISTING**.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to 0 (no overlapped I/O).

hTemplateFile

This value must be NULL for TDRV009 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TDRV009 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call *GetLastError*.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TDRV009_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TDRV009 device always open existing
     0,                 // no overlapped I/O
     NULL
);

if (hDevice == INVALID_HANDLE_VALUE) {
     ErrorHandler( "Could not open device" ); // process error
}
```

See Also

CloseHandle(), Win32 documentation CreateFile()

4.1.2 Closing a Device

The CloseHandle function closes an open TDRV009 handle.

BOOL CloseHandle(
    HANDLE *hDevice*
)

Parameters

hDevice

Identifies an open TDRV009 handle.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Example

```
HANDLE hDevice;
if( !CloseHandle( hDevice ) ) {
     ErrorHandler("Could not close device" ); // process error
}
```

See Also

CreateFile (), Win32 documentation CloseHandle ()

4.1.3 Device I/O Control Functions

The DeviceIoControl function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```
BOOL DeviceIoControl(
        HANDLE      hDevice,
        DWORD       dwIoControlCode,
        LPVOID      lpInBuffer,
        DWORD       nInBufferSize,
        LPVOID      lpOutBuffer,
        DWORD       nOutBufferSize,
        LPDWORD lpBytesReturned,
        LPOVERLAPPED  lpOverlapped
)
```

Parameters

hDevice

Handle to the TDRV009 channel that is to perform the operation.

dwIoControlCode

Specifies the control code for the operation. This value identifies the specific operation to be performed. The following values are defined in tdrv009.h:

Value	**Meaning**
IOCTL_TDRV009_READ	read one data packet from internal buffer
IOCTL_TDRV009_WRITE	write one data packet
IOCTL_TDRV009_SET_OPERATION_MODE	set a channel's operation mode
IOCTL_TDRV009_GET_OPERATION_MODE	get a channel's current operation mode
IOCTL_TDRV009_SET_BAUDRATE	set baudrate without other changes
IOCTL_TDRV009_RINGBUF_REGISTER	register a ringbuffer for receive
IOCTL_TDRV009_RINGBUF_UNREGISTER	unregister the ringbuffer
IOCTL_TDRV009_SET_RECEIVER_STATE	set operation state of the channel's receiver
IOCTL_TDRV009_CLEAR_RX_BUFFER	clear the internal receive buffer
IOCTL_TDRV009_SET_EXT_XTAL	specify an externally supplied frequency
IOCTL_TDRV009_SCC_REG_WRITE	directly write to an SCC register
IOCTL_TDRV009_SCC_REG_READ	directly read from an SCC register
IOCTL_TDRV009_GLOB_REG_WRITE	directly write to a GLOBAL register
IOCTL_TDRV009_GLOB_REG_READ	directly read from a GLOBAL register
IOCTL_TDRV009_EEPROM_WRITE	Write value to onboard EEPROM
IOCTL_TDRV009_EEPROM_READ	Read value from onboard EEPROM
IOCTL_TDRV009_WAITFORINTERRUPT	Wait for specific channel interrupt
IOCTL_TDRV009_EVENT_REGISTER	register event which is signaled by the driver
IOCTL_TDRV009_EVENT_UNREGISTER	unregister formerly registered event

IOCTL_TDRV009_RTS_SET	Assert RTS handshake line
IOCTL_TDRV009_RTS_CLEAR	De-Assert RTS handshake line
IOCTL_TDRV009_CTS_GET	Read state of CTS
IOCTL_TDRV009_DTR_SET	Set DTR signal line (only channel 3)
IOCTL_TDRV009_DTR_CLEAR	Clear DTR signal line (only channel 3)
IOCTL_TDRV009_DSR_GET	Read state of DSR (only channel 3)

See behind for more detailed information on each control code.

lpInBuffer

Pointer to a buffer that contains the data required to perform the operation.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by lpInBuffer.

lpOutBuffer

Pointer to a buffer that receives the operation's output data.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by lpOutBuffer.

lpBytesReturned

Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by lpOutBuffer. A valid pointer is required.

lpOverlapped

Pointer to an Overlapped structure. This value must be set to NULL (no overlapped I/O).

To use these TDRV009 specific control codes the header file tdrv009.h must be included.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Please note that the TDRV009 device driver returns always standard Win32 error codes on failure. Please refer to the Windows Platform SDK Documentation for a detailed description of returned error codes.

See Also

Win32 documentation DeviceIoControl ()

4.1.3.1 IOCTL_TDRV009_READ

This TDRV009 control function reads a data frame from the internal receive buffer. A pointer to the callers receive buffer (*TDRV009_RX_BUFFER*) is passed by the parameters *lpInBuffer* and *lpOutBuffer* to the driver.

This control function returns immediately, even if there is no data currently available. The caller has to verify the structure member *Valid* to determine if the buffer contains data. All structure members are declared as ULONG to guarantee the correct 4byte-alignment for DMA operations.

The received data is copied into the user-supplied buffer by the driver. For high-speed transfers better use a ringbuffer (refer to chapter 4.1.3.6).

```
typedef struct {
        ULONG       NumberOfBytes;
        ULONG       Valid;
        ULONG       Overflow;
        UCHAR       pData[1];     /* dynamically expandable */
} TDRV009_RX_BUFFER;
```

NumberOfBytes

Returns the amount of valid bytes inside the buffer.

Valid

This OR'ed value describes if the returned buffer contains valid data. Additionally, the FrameEnd status is returned. This value consists of the following OR'ed values (defined in *tdrv009.h*):

Value	Description
TDRV009_RXBUF_DATAVALID	The data buffer contains valid data.
TDRV009_RXBUF_FRAMEEND	The data buffer contains a FrameEnd mark.

Overflow

This value marks an internal buffer overflow.

pData

The received values are copied into this buffer. It must be large enough to hold all data.

Example

```
#include “tdrv009.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TDRV009_RX_BUFFER   *pRxBuf;

/*
** read one buffer with up to 100 data bytes
*/
BufferSize = 100*sizeof(unsigned char) + sizeof(TDRV009_RX_BUFFER);
```

```
pRxBuf = (TDRV009_RX_BUFFER*)malloc( BufferSize );

success = DeviceIoControl (
     hDevice,                 // TDRV009 handle
     IOCTL_TDRV009_READ,      // control code
     NULL,                    // input buffer
     0,
     pRxBuf,                  // output buffer
     BufferSize,
     &NumBytes,               // number of bytes transferred
     NULL
);


if( success ) {
     // Process data
     if (  (pRxBuf->Valid & TDRV009_RXBUF_DATAVALID) &&
           (pRxBuf->NumberOfBytes > 0) ) {
          printf( “Received %d valid bytes.\n”, pRxBuf->NumberOfBytes );
     }
} else {
     // Process DeviceIoControl() error
}

free( pRxBuf );
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the output buffer is invalid.
ERROR_INSUFFICIENT_BUFFER	The size of the supplied buffer is too small for the available amount of data.
ERROR_ACCESS_DENIED	A ringbuffer is registered, no read operation possible

All other returned error codes are system error conditions.

4.1.3.2 IOCTL_TDRV009_WRITE

This TDRV009 control function sends a buffer of data. A pointer to the caller's data buffer (*TDRV009_TX_BUFFER*) must be passed by both parameters *lpInBuffer* and *lpOutBuffer* to the driver.

It is necessary to supply the pointer to the data buffer on both parameters because of the memory mapping used by Windows2000/XP.

The transfer of large physically scattered buffers (bigger than one memory page) is supported.

```
typedef struct {
        ULONG       NumberOfBytes;
        UCHAR       pData[1];     /* dynamically expandable */
} TDRV009_TX_BUFFER;
```

NumberOfBytes

Number of bytes that should be written to the corresponding channel

pData

Dynamically enlargeable data buffer

Example

```
#include “tdrv009.h”

HANDLE             hDevice;
BOOLEAN            success;
ULONG              NumBytes;
TDRV009_TX_BUFFER  *pTxBuf;

//
// allocate some memory for up to 20 data bytes
//
pTxBuf = (TDRV009_TX_BUFFER*)malloc( sizeof(TDRV009_TX_BUFFER) + 20 );
memset( pTxBuf, 0, sizeof(TDRV009_TX_BUFFER) + 20 );
sprintf( pTxBuf->pData, “Hello World!”);
pTxBuf->NumberOfBytes = strlen( pTxBuf->pData );

success = DeviceIoControl (
     hDevice,                         // TDRV009 handle
     IOCTL_TDRV009_WRITE,             // control code
     pTxBuf,                          // data buffer
     sizeof(TP862_TX_BUFFER) + 20,    // size to be mapped
     pTxBuf,                          // data buffer
     sizeof(TP862_TX_BUFFER) + 20,    // size to be mapped
     &NumBytes,                       // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is invalid.
ERROR_INSUFFICIENT_BUFFER	NumberOfBytes is larger than the specified memory area.
ERROR_NO_SYSTEM_RESOURCES	Not enough resources available while building ScatterGather list. DMA error.

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.1.3.3 IOCTL_TDRV009_SET_OPERATION_MODE

This TDRV001 control function sets the desired operation mode for a channel. The parameter *lpInBuffer* passes a pointer to a TDRV009_OPERATION_MODE_STRUCT buffer to the device driver. After setting the desired values a channel reset is performed, so all received data is lost. The *lpOutBuffer* parameter is not used for this function.

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

CommType

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

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

TransceiverMode

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

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

Oversampling

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

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

BrgSource

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

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

TxClkSource

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

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

TxClkOutput

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

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

RxClkSource

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

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

ClockMultiplier

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

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

Baudrate

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

ClockInversion

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

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

Encoding

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

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

Parity

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

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

Stopbits

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

Databits

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

UseTermChar

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

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

TermChar

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

HwHs

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

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

Crc

This parameter is a structure describing the CRC checking configuration.

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

Type

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

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

RxChecking

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

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

TxGeneration

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

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

ResetValue

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

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

Example

```
#include “tdrv009.h”

HANDLE                              hDevice;
BOOLEAN                             success;
ULONG                               NumBytes;
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;

success = DeviceIoControl (
     hDevice,                                // TDRV009 handle
     IOCTL_TDRV009_SET_OPERATION_MODE,       // control code
     &OperationMode,                         // parameter buffer
     sizeof(TDRV009_OPERATION_MODE_STRUCT),
     NULL,
     0,
     &NumBytes,                              // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	A supplied parameter within the structure is invalid

All other returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.1.3.4 IOCTL_TDRV009_GET_OPERATION_MODE

This TDRV009 control function reads the current channel configuration and returns the value in an application supplied buffer (TDRV009_OPERATION_MODE_STRUCT). The parameter *lpOutBuffer* passes a pointer to this buffer to the device driver. The *lpInBuffer* parameter is not used for this function.

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

CommType

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

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

TransceiverMode

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

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

Oversampling

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

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

BrgSource

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

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

TxClkSource

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

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

TxClkOutput

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

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

RxClkSource

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

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

ClockMultiplier

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

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

Baudrate

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

ClockInversion

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

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

Encoding

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

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

Parity

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

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

Stopbits

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

Databits

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

UseTermChar

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

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

TermChar

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

HwHs

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

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

Crc

This parameter is a structure describing the CRC checking configuration.

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

Type

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

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

RxChecking

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

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

TxGeneration

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

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

ResetValue

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

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

Example

```
#include “tdrv009.h”

HANDLE                              hDevice;
BOOLEAN                             success;
ULONG                               NumBytes;
TDRV009_OPERATION_MODE_STRUCT       OperationMode;

/*------------------------------------------------
  Retrieve current channel configuration
  ------------------------------------------------*/
success = DeviceIoControl (
     hDevice,                                // TDRV009 handle
     IOCTL_TDRV009_GET_OPERATION_MODE,       // control code
     NULL,                                   // parameter buffer
     0,
     &OperationMode,
     sizeof(TDRV009_OPERATION_MODE_STRUCT),
     &NumBytes,                              // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
} else {
     printf(“Baudrate = %d\n”, OperationMode.Baudrate);
}
```

Error Codes

ERROR_INVALID_USER_BUFFER   The size of the input or output buffer is too small.

All other returned error codes are system error conditions.

4.1.3.5 IOCTL_TDRV009_SET_BAUDRATE

This TDRV001 control function sets up the transmission rate for the specific channel. This is done without all the other configuration stuff performed by IOCTL_TDRV009_SET_OPERATION_MODE. No channel-reset is performed either. If async oversampling is enabled, the desired baudrate is internally multiplied by 16. It is important that this result can be derived from the selected clocksource. The parameter *lpInBuffer* passes a pointer to an ULONG value containing the desired baudrate to the device driver. The *lpOutBuffer* parameter is not used for this function.

For pre-defined baudrate-values see file “*tdrv009.h*”.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
ULONG     Baudrate;

Baudrate = 14400;  // 14400 bps
success = DeviceIoControl (
     hDevice,                      // TDRV009 handle
     IOCTL_TDRV009_SET_BAUDRATE,  // control code
     &Baudrate,                    // input buffer
     sizeof(ULONG),
     NULL,                         // output buffer
     0,
     &NumBytes,                    // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The desired baudrate is invalid.

All other returned error codes are system error conditions.

4.1.3.6 IOCTL_TDRV009_RINGBUF_REGISTER

This TDRV001 control function registers a ringbuffer structure for received data directly accessible from user-space. The parameters *lpInBuffer* and *lpOutBuffer* passes pointers to the ringbuffer-structure (*TDRV009_RINGBUFFER*) to the device driver. Only call this function in OVERLAPPED file mode, otherwise the application will wait indefinitely for return.

```
typedef struct
{
        ULONG       BufferSize;
        ULONG       PacketSize;
        ULONG       NumberOfEntries;
        ULONG       get;
        ULONG       put;
        ULONG       Overflow;
} RINGBUFFER_HEADER;
```

BufferSize

Total size of memory available for data (user-defined)

PacketSize

Typical size of one data packet (user-defined), adjusted by driver

NumberOfEntries

Nmber of entries available in the ringbuffer

get

Index where new data can be read

put

Index where new data is filled in by the driver

Overflow

TRUE if a buffer overflow has happened. The receiver must be enabled again.

```
typedef struct
{
        ULONG       NumberOfBytes;
        ULONG       Size;
        ULONG       Offset;
        ULONG       DmaAddress;
        ULONG       Valid;
} RINGBUFFER_ENTRY;
```

NumberOfBytes

This value specifies the number of valid bytes inside the corresponding buffer.

Size

This value specifies the total size of the corresponding buffer.

Offset

This value specifies the offset relative to the beginning of DataSection for the corresponding data buffer.

DmaAddress

This value specifies the physical address used by the DMA controller for the corresponding buffer.

Valid

This OR'ed value describes whether or not the corresponding data buffer contains valid data. Additionally, the FrameEnd status is returned. This value consists of the following OR'ed values:

Value	**Description**
TDRV009_RXBUF_DATAVALID	The data buffer contains valid data.
TDRV009_RXBUF_FRAMEEND	The data buffer contains a FrameEnd mark.

```
typedef struct
{
        RINGBUFFER_HEADER       Header;
        UCHAR                   DataSection[40];    // dynamically expandable
} TDRV009_RINGBUFFER;
```

Header

Header information for actual ringbuffer (see description of RINGBUFFER_HEADER above).

DataSection

This value points to a user-defined section for entries and data buffers. The size is dynamically expandable and must be provided inside the header.

Ringbuffer Concept

The used ringbuffer is highly configurable by the user. Not only the total size but also the typical size of one packet can be specified. One big problem for DMA drivers is that the hardware performing the direct memory access needs physically consistent memory. The user allocates a large buffer which is virtually consistent but physically scattered. The driver has to split it into physically consistent memory parts taking the desired typical packet size into mind too. Each single buffer is accessible via a corresponding entry that holds specific information like offset, size and the number of contained valid data. To assist the user working with these entries some macros are supplied which are described later. The ringbuffer concept is explained in detail in the following illustration.

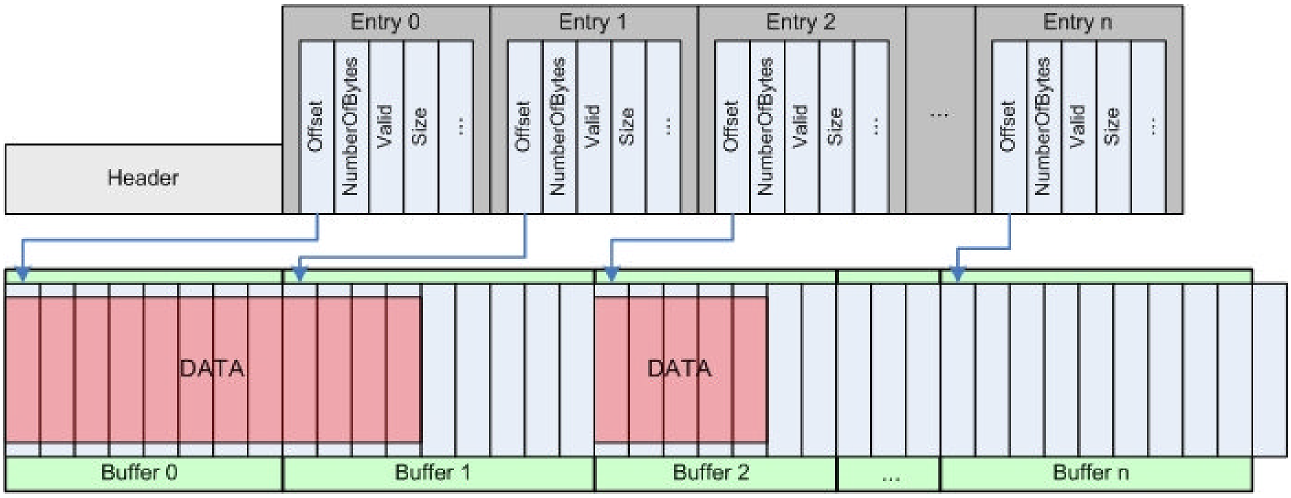

Figure: Ringbuffer Concept

Assistant Macros and Functions for Bufferhandling

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.

```
CALCULATE_RINGBUFFER_SIZE( BufferSize, PacketSize )
```

calculates the total size necessary for allocation of the complete ringbuffer

```
GET_BUFFER( pRingBuffer, index )
```

returns a pointer to the corresponding data buffer

```
GET_VALID( pRingBuffer, index )
```

returns the value of the member *Valid*.

```
IS_VALID( pRingBuffer, index )
```

TRUE if the corresponding buffer contains valid data, otherwise FALSE

```
IS_FRAMEEND( pRingBuffer, index )
```

TRUE if the corresponding buffer contains a FrameEnd flag, otherwise FALSE

```
SET_VALID( pRingBuffer, index, value )
```

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

```
CLEAR_BUFFER( pRingBuffer, index )
```

clears the corresponding data buffer

```
GET_POS( pRingBuffer )
```

returns the current get position where new data can be read

```
OVERFLOW( pRingBuffer )
```

TRUE if a buffer overflow has happened, otherwise false

```
UCHAR* GetNewBuffer( TDRV009_RINGBUFFER* pRingBuffer, ULONG* 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*.

Example

```
#include “tdrv009.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
ULONG                   BufferSize, PacketSize, TotalSize;
TDRV009_RINGBUFFER*     pRingBuffer;
OVERLAPPED              Overlapped;


//
// init Overlapped structure
//
Overlapped.Offset = 0;
Overlapped.hEvent = 0;

//
// allocate and init ringbuffer
//
BufferSize    = 8000;
PacketSize    = 120;
TotalSize     = CALCULATE_RINGBUFFER_SIZE( BufferSize, PacketSize );
pRingBuffer = (TDRV009_RINGBUFFER*)malloc( TotalSize );
memset( pRingBuffer, 0, TotalSize );
pRingBuffer->Header.BufferSize = BufferSize;
pRingBuffer->Header.PacketSize = PacketSize;

success = DeviceIoControl (
     hDevice,                            // TDRV009 handle
     IOCTL_TDRV009_RINGBUF_REGISTER,     // control code
     pRingBuffer,                        // input buffer
     TotalSize,
     pRingBuffer,                        // output buffer
     TotalSize,
     &NumBytes,                     // number of bytes transferred
     &Overlapped
);


if( !success ) {
     // Process DeviceIoControl() error and free allocated memory
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is too small.
ERROR_ACCESS_DENIED	A ringbuffer is already registered. Unregister it first.
ERROR_NO_SYSTEM_RESOURCES	Not enough resources available while building ScatterGather list. DMA error.

All other returned error codes are system error conditions.

4.1.3.7 IOCTL_TDRV009_RINGBUF_UNREGISTER

This TDRV009 control function unregisters a formerly registered ringbuffer. The driver uses its internal receive buffer structure again.

No additional parameter is required for this function.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                              // TDRV009 handle
     IOCTL_TDRV009_RINGBUF_UNREGISTER,     // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	No ringbuffer is registered at the moment.

4.1.3.8 IOCTL_TDRV009_SET_RECEIVER_STATE

This TDRV009 control function sets the state of the receiver module. The parameter *lpInBuffer* passes a pointer to a ULONG value containing the new receiver state to the device driver. The *lpOutBuffer* parameter is not used for this function.

Possible values are:

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

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
ULONG     ReceiverState;

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

success = DeviceIoControl (
     hDevice,                             // TDRV009 handle
     IOCTL_TDRV009_SET_RECEIVER_STATE,    // control code
     &ReceiverState,                      // parameter buffer
     sizeof(ULONG),
     NULL,
     0,
     &NumBytes,                           // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The specified receiverstate is invalid.

All other returned error codes are system error conditions.

4.1.3.9 IOCTL_TDRV009_CLEAR_RX_BUFFER

This TDRV009 control function clears the internal receive-buffer of the corresponding channel. If a ringbuffer was registered earlier the function will return with an error. Otherwise the internal receive-buffer is cleared. The receiver is stopped for the duration of the clearing process.

No additional parameter is needed for this call.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                         // TDRV009 handle
     IOCTL_TDRV009_CLEAR_RX_BUFFER,   // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                       // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	There is a registered ringbuffer in use.

All other returned error codes are system error conditions.

4.1.3.10 IOCTL_TDRV009_SET_EXT_XTAL

This TDRV009 control function sets the frequency of an externally supplied frequency. This frequency is used for baudrate calculation, and describes the input frequency to the Baud Rate Generator (BRG). The external frequency may be supplied either at input line TxC or RxC. The parameter *lpInBuffer* passes a pointer to a ULONG value containing the new clock frequency to the device driver. The *lpOutBuffer* parameter is not used for this function.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
ULONG     ExtXtal;

//  specify an external frequency of 1 MHz
ExtXtal = 1000000;

success = DeviceIoControl (
     hDevice,                       // TDRV009 handle
     IOCTL_TDRV009_SET_EXT_XTAL,    // control code
     &ExtXtal,                      // input buffer
     sizeof(ULONG),
     NULL,
     0,
     &NumBytes,                     // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The specified frequency is invalid (zero).

All other returned error codes are system error conditions.

4.1.3.11 IOCTL_TDRV009_SCC_REG_WRITE

This TDRV009 control function writes one 32bit word to the communication controller's register space, relative to the beginning of the specific channel's SCC register set. The parameter *lpInBuffer* passes a pointer to the configuration buffer (*TDRV009_ADDR_STRUCT*) to the device driver. A verification of the written data is not performed. The *lpOutBuffer* parameter is not used for this function.

```
typedef struct
{
        ULONG   Offset;
        ULONG   Value;
} TDRV009_ADDR_STRUCT;
```

Offset

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

Value

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

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

Example

```
#include "tdrv009.h"

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_ADDR_STRUCT     AddrBuf;

/*------------------------------------------------
  Write a 32bit value (Termination Character Register)
  ------------------------------------------------*/
AddrBuf.Offset = 0x0048;
AddrBuf.Value  = (1 << 15) | 0x42;
success = DeviceIoControl (
     hDevice,                        // TDRV009 handle
     IOCTL_TDRV009_SCC_REG_WRITE,    // control code
     &AddrBuf,                       // input buffer
     sizeof(TDRV009_ADDR_STRUCT),
     NULL,
     0,
     &NumBytes,                      // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The specified offset is invalid (too large).

All other returned error codes are system error conditions.

4.1.3.12 IOCTL_TDRV009_SCC_REG_READ

This TDRV009 control function reads one 32bit word from the communication controller's register space, relative to the beginning of the specific channel's SCC register set. The parameter *lpOutBuffer* passes a pointer to the configuration buffer (*TDRV009_ADDR_STRUCT*) to the device driver. The *lpIntBuffer* parameter is not used for this function.

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

Offset

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

Value

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

Example

```
#include “tdrv009.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_ADDR_STRUCT     AddrBuf;


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

success = DeviceIoControl (
     hDevice,                         // TDRV009 handle
     IOCTL_TDRV009_SCC_REG_READ,      // control code
     NULL,
     0,
     &AddrBuf,                        // output buffer
     sizeof(TDRV009_ADDR_STRUCT),
     &NumBytes,                       // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
} else {
     printf( “Value = 0x%lX\n”, AddrBuf.Value );
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The specified offset is invalid (too large).

All other returned error codes are system error conditions.

4.1.3.13 IOCTL_TDRV009_GLOB_REG_WRITE

This TDRV009 control function writes one 32bit word to the communication controller's register space, relative to the beginning of the register set. The parameter *lpInBuffer* passes a pointer to the configuration buffer (*TDRV009_ADDR_STRUCT*) to the device driver. A verification of the written data is not performed. The *lpOutBuffer* parameter is not used for this function.

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

Offset

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

Value

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

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

Example

```
#include “tdrv009.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Write a 32bit value (FIFO Control Register 4)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0034;
AddrBuf.Value  = 0xffffffff;
success = DeviceIoControl (
     hDevice,                           // TDRV009 handle
     IOCTL_TDRV009_GLOB_REG_WRITE,      // control code
     &AddrBuf,                          // input buffer
     sizeof(TDRV009_ADDR_STRUCT),
     NULL,
     0,
     &NumBytes,                         // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The specified offset is invalid (too large).

All other returned error codes are system error conditions.

4.1.3.14 IOCTL_TDRV009_GLOB_REG_READ

This TDRV009 control function reads one 32bit word from the communication controller’s register space, relative to the beginning of the register set. The parameter *lpOutBuffer* passes a pointer to the configuration buffer (*TDRV009_ADDR_STRUCT*) to the device driver. The *lpInBuffer* parameter is not used for this function.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    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”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_ADDR_STRUCT     AddrBuf;


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

success = DeviceIoControl (
     hDevice,                          // TDRV009 handle
     IOCTL_TDRV009_GLOB_REG_READ,      // control code
     NULL,
     0,
     AddrBuf,                          // output buffer
     sizeof(TDRV009_ADDR_STRUCT),
     &NumBytes,                        // number of bytes transferred
     NULL
);
```

```
if( !success ) {
    // Process DeviceIoControl() error
} else {
    printf( “Value = 0x%lX\n”, AddrBuf.Value );
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the input buffer is too small.
ERROR_INVALID_PARAMETER	The specified offset is invalid (too large).

All other returned error codes are system error conditions.

4.1.3.15 IOCTL_TDRV009_EEPROM_WRITE

This TDRV009 control function writes one 16bit word into the onboard EEPROM. The first part of the EEPROM is reserved for factory usage, write accesses to this area will result in an error. The parameter *lpInBuffer* passes a pointer to the user buffer (*TDRV009_EEPROM_BUFFER*) to the device driver. A verification of the written data is not performed. The *lpOutBuffer* parameter is not used for this function.

```
typedef struct {
    ULONG      Offset;
    USHORT     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”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_EEPROM_BUFFER   EepromBuf;

/*-------------------------------------------------
  Write a 16bit value into the EEPROM, offset 0x61
  -------------------------------------------------*/
EepromBuf.Offset = 0x61;
EepromBuf.Value  = 0x1234;
success = DeviceIoControl (
     hDevice,                           // TDRV009 handle
     IOCTL_TDRV009_EEPROM_WRITE,        // control code
     &EepromBuf,                        // input buffer buffer
     sizeof(TDRV009_EEPROM_BUFFER),
     NULL,
     0,
     &NumBytes,                         // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is invalid.
ERROR_INVALID_PARAMETER	The specified offset address is invalid, or read-only.

All other returned error codes are system error conditions.

4.1.3.16 IOCTL_TDRV009_EEPROM_READ

This TDRV009 control function reads one 16bit word from the onboard EEPROM. The parameter *lpInBuffer* passes a pointer to the user buffer (*TDRV009_EEPROM_BUFFER*) to the device driver. The *lpOutBuffer* parameter is not used for this function.

```
typedef struct {
        ULONG      Offset;
        USHORT     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”

HANDLE                 hDevice;
BOOLEAN                success;
ULONG                  NumBytes;
TDRV009_EEPROM_BUFFER  EepromBuf;


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

success = DeviceIoControl (
     hDevice,                           // TDRV009 handle
     IOCTL_TDRV009_EEPROM_WRITE,        // control code
     &EepromBuf,                        // input buffer buffer
     sizeof(TDRV009_EEPROM_BUFFER),
     NULL,
     0,
     &NumBytes,                         // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
} else {
     printf( “Value = 0x%X\n”, EepromBuf.Value );
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is invalid.
ERROR_INVALID_PARAMETER	The specified offset address is invalid, or read-only.

All other returned error codes are system error conditions.

4.1.3.17 IOCTL_TDRV009_WAITFORINTERRUPT

This TDRV009 control function waits until a specified SCC-interrupt or the timeout occurs. The parameters *lpInBuffer* and *lpOutBuffer* pass a pointer to the user buffer (*TDRV009_WAIT_STRUCT*) to the device driver.

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

Interrupts

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

Timeout

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

Example

```
#include “tdrv009.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_WAIT_STRUCT     WaitStruct;

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

success = DeviceIoControl (
     hDevice,                          // TDRV009 handle
     IOCTL_TDRV009_WAITFORINTERRUPT,   // control code
     &WaitStruct,                      // input buffer buffer
     sizeof(TDRV009_WAIT_STRUCT),
     &WaitStruct,                      // output buffer
     sizeof(TDRV009_WAIT_STRUCT),
     &NumBytes,                        // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
} else {
     printf( “Interrupt Event occurred.\n” );
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is invalid.
ERROR_BUSY	Too many simultaneous wait jobs active.

All other returned error codes are system error conditions.

4.1.3.18 IOCTL_TDRV009_EVENT_REGISTER

This TDRV009 control function registers an event which is signaled by the driver for some reason. The event TDRV009_RX_EVENT is signaled every time a new data buffer is received.

The parameter *lpInBuffer* passes a pointer to the registration buffer (*TDRV009_EVENT_STRUCT*) to the device driver. The *lpOutBuffer* parameter is not used for this function.

```
typedef struct
{
        HANDLE   hEvent;
        ULONG    type;
} TDRV009_EVENT_STRUCT;
```

hEvent

This parameter specifies the event which should be signaled by the driver. This event must be created by a call to *CreateEvent()*.

type

This parameter specifies the event type. The only event type possible is TDRV009_RX_EVENT (refer to *tdrv009.h* for definition).

Keep in mind that signaling an event for each received packet means additional overhead, which should be omitted in environments with high packet-rates.

Example

```
#include “tdrv009.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_EVENT_STRUCT    EventStruct;

//
//  init event structure
//
EventStruct.type = TP862_RX_EVENT;
EventStruct.hEvent = CreateEvent(
                   NULL,   // lpEventAttributes
                   TRUE,   // bManualReset
                   FALSE,  // bInitialState
                   NULL    // lpName
                   );

success = DeviceIoControl (
     hDevice,                          // TDRV009 handle
     IOCTL_TDRV009_EVENT_REGISTER,     // control code
     &EventStruct,                     // input buffer buffer
     sizeof(TDRV009_EVENT_STRUCT),
     NULL,
     0,
     &NumBytes,                        // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is invalid.
ERROR_INVALID_PARAMETER	The specified parameters are invalid.
ERROR_ACCESS_DENIED	An event is already registered for this event type.

All other returned error codes are system error conditions.

4.1.3.19 IOCTL_TDRV009_EVENT_UNREGISTER

This TDRV009 control unregisters an event which has been registered by a previous call to IOCTL_TDRV009_EVENT_REGISTER. The parameter *lpInBuffer* passes a pointer to the registration buffer (*TDRV009_EVENT_STRUCT*) to the device driver. The *lpOutBuffer* parameter is not used for this function.

```
typedef struct
{
    HANDLE   hEvent;
    ULONG    type;
} TDRV009_EVENT_STRUCT;
```

hEvent

This parameter specifies the event which should be signaled by the driver. This event must be created by a call to *CreateEvent()*. This parameter is not evaluated by the device driver.

type

This parameter specifies the event type. The only event type possible is TDRV009_RX_EVENT (refer to *tdrv009.h* for definition).

Example

```
#include “tdrv009.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TDRV009_EVENT_STRUCT    EventStruct;

//
//  init event structure
//
EventStruct.type = TP862_RX_EVENT;

success = DeviceIoControl (
     hDevice,                           // TDRV009 handle
     IOCTL_TDRV009_EVENT_UNREGISTER,    // control code
     &EventStruct,                      // input buffer buffer
     sizeof(TDRV009_EVENT_STRUCT),
     NULL,
     0,
     &NumBytes,                         // number of bytes transferred
     NULL
);

if( !success ) {
        // Process DeviceIoControl() error
    }
```

Error Codes

ERROR_INVALID_USER_BUFFER	The size of the buffer is invalid.
ERROR_INVALID_PARAMETER	The specified parameters are invalid.
ERROR_ACCESS_DENIED	No event is registered for this event type.

All other returned error codes are system error conditions.

4.1.3.20 IOCTL_TDRV009_RTS_SET

This TDRV009 control function asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking.

No additional parameter is needed for this call.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                            // TDRV009 handle
     IOCTL_TDRV009_RTS_SET,              // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                          // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	The channel is in handshake mode, so this function is not allowed.

All other returned error codes are system error conditions.

4.1.3.21 IOCTL_TDRV009_RTS_CLEAR

This TDRV009 control function de-asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking.

No additional parameter is needed for this call.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                         // TDRV009 handle
     IOCTL_TDRV009_RTS_CLEAR,         // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                       // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	The channel is in handshake mode, so this function is not allowed.

All other returned error codes are system error conditions.

4.1.3.22 IOCTL_TDRV009_CTS_GET

This TDRV009 control function returns the current state of the CTS handshake signal line of the specific channel.

The parameter *lpOutBuffer* passes a pointer to the user buffer (*unsigned long*) to the device driver. Depending on the state of CTS, either 0 (inactive) or 1 (active) is returned.

The *lpIntBuffer* parameter is not used for this function.

Example

```
#include “tdrv009.h”

HANDLE        hDevice;
BOOLEAN       success;
ULONG         NumBytes;
unsigned long CtsState;

success = DeviceIoControl (
     hDevice,                        // TDRV009 handle
     IOCTL_TDRV009_CTS_GET,          // control code
     NULL,
     0,
     &CtsState,
     sizeof(unsigned long),
     &NumBytes,                      // number of bytes transferred
     NULL
);

if( success ) {
     printf( “CTS = %ld\n”, CtsState );
} else {
     // Process DeviceIoControl() error
}
```

Error Codes

All other returned error codes are system error conditions.

4.1.3.23 IOCTL_TDRV009_DTR_SET

This TDRV009 control function sets the DTR signal line to HIGH. This function is only available for local module channel 3.

No additional parameter is needed for this call.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                           // TDRV009 handle
     IOCTL_TDRV009_DTR_SET,             // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                         // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	This function is not supported by the specific channel.

All other returned error codes are system error conditions.

4.1.3.24 IOCTL_TDRV009_DTR_CLEAR

This TDRV009 control function sets the DTR signal line to LOW. This function is only available for local module channel 3.

No additional parameter is needed for this call.

Example

```
#include “tdrv009.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

success = DeviceIoControl (
     hDevice,                              // TDRV009 handle
     IOCTL_TDRV009_DTR_CLEAR,              // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                            // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	This function is not supported by the specific channel.

All other returned error codes are system error conditions.

4.1.3.25 IOCTL_TDRV009_DSR_GET

This TDRV009 control function returns the current state of the DSR signal line of the specific channel. This function is only available for local module channel 3.

The parameter *lpOutBuffer* passes a pointer to the user buffer (*unsigned long*) to the device driver. Depending on the state of DSR, either 0 (inactive) or 1 (active) is returned.

The *lpIntBuffer* parameter is not used for this function.

Example

```
#include “tdrv009.h”

HANDLE        hDevice;
BOOLEAN       success;
ULONG         NumBytes;
unsigned long DsrState;

success = DeviceIoControl (
     hDevice,                           // TDRV009 handle
     IOCTL_TDRV009_DSR_GET,             // control code
     NULL,
     0,
     &DsrState,
     sizeof(unsigned long),
     &NumBytes,                         // number of bytes transferred
     NULL
);

if( success ) {
     printf( “DSR = %ld\n”, DsrState );
} else {
     // Process DeviceIoControl() error
}
```

Error Codes

ERROR_ACCESS_DENIED	This function is not supported by the specific channel.

All other returned error codes are system error conditions.