The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV010-SW-65

Windows Device Driver

Isolated 2 x CAN Bus

Version 2.0.x

User Manual

Issue 2.0.0

January 2011

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

TDRV010-SW-65

Windows Device Driver

Isolated 2 x CAN Bus

Supported Modules:
TPMC310
TPMC810

Issue	Description	Date
1.0.0	First Issue	January 1, 2005
1.0.1	IOCTL_TDRV010_READ function description added	February 14, 2006
1.0.2	New Address TEWS LLC	February 28, 2007
1.0.3	General Revision, Return value of CloseHandle() corrected	April 7, 2008
1.0.4	Files moved to subdirectory	June 23, 2008
2.0.0	Windows 7, 64-bit support added, API added, Address TEWS LLC removed	January 11, 2011

Table of Contents

1 Introduction

The TDRV010-SW-65 Windows device driver is a kernel mode driver which allows the operation of the supported hardware module on an Intel or Intel-compatible Windows operating system. Supported Windows versions are:

- Windows 2000
- Windows XP
- Windows XP Embedded
- Windows 7 (32bit and 64bit)

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 TDRV010-SW-65 device driver supports the following features:

- Read received messages from input FIFO
- Send a message
- Set channel Bus On/Off
- Configure Listen-Only mode On/Off
- Configure Selftest mode On/Off
- Extended and Standard Identifiers
- Configure Bitrate
- Configure Receive Mask
- Flush receive FIFO
- Read CAN status
- PLD Functions (external CAN Controller Reset, Silent Mode and Interrupt Control)(TPMC310 only)

The TDRV010-SW-65 device driver supports the modules listed below:

TPMC310	Isolated 2 x CAN Bus (Conduction Cooled) (64 pin connector for Back-IO)	(PMC)
TPMC810	Isolated 2 x CAN Bus (2x SUBD 9 pin connectors for Front-Panel I/O) (64 pin connector for Back-I/O)	(PMC)

In this document all supported modules and devices will be called TDRV010. Specials for certain devices will be advised.

To get more information about the features and use of TDRV010 devices it is recommended to read the manuals listed below.

TPMC310/TPMC810 User manual
TPMC310/TPMC810 Engineering Manual
SJA1000 Product Specification

2 Installation

Following files are located in directory TDRV010-SW-65 on the distribution media:

i386\	Directory containing driver files for 32bit Windows versions
amd64\	Directory containing driver files for 64bit Windows versions
installer_32bit.exe	Installation tool for 32bit systems (Windows XP or later)
installer_64bit.exe	Installation tool for 64bit systems (Windows XP or later)
tdrv010.inf	Windows installation script
tdrv010.h	Header file with IOCTL codes and structure definitions
api\tdrv010api.h	API include file
api\tdrv010api.c	API source file
example\tdrv010exa.c	Example application
TDRV010-SW-65-2.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TDRV010 Device Driver on a Windows 2000 / XP operating system.

After installing the hardware 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 TDRV010 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 found module of the TDRV010 product family.
7. Copy needed files (tdrv010.h, API files) to desired target directory.

After successful installation a device is created for each module found (TDRV010_1, TDRV010_2 ...).

2.1.2 Windows 7

This section describes how to install the TDRV010-SW-65 Device Driver on a Windows 7 (32bit or 64bit) operating system.

Depending on the operating system type, execute the installer binaries for either 32bit or 64bit systems. This will install all required driver files using an installation wizard.

Copy needed files (tdrv010.h, API files) to desired target directory.

After successful installation a device is created for each module found (TDRV010_1, TDRV010_2 ...).

2.2 Confirming Driver Installation

To confirm that the driver has been properly loaded, perform the following steps:

1. Open the Windows Device Manager:

   a. For Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***" and click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.

   b. For Windows 7, open the "***Control Panel***" from "***My Computer***" and then click the "***Device Manager***" entry.

2. Click the "**+**" in front of "***Embedded I/O***".
   The driver "***TEWS TECHNOLOGIES - TDRV010 Extended CAN (TPMC…)***" should appear for each installed device.

3 Driver Configuration

3.1 Message FIFO Configuration

After Installation of the TDRV010 Device Driver the number of sequential CAN Messages is limited to 100 without the need to read from the certain device.

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

To change the number of queue entries the following value must be modified.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tdrv010\Parameters\FIFODepth

Valid values are in range between 1 … 1024

4 API Documentation

4.1 General Functions

4.1.1 tdrv010Open()

NAME

tdrv010Open() – opens a device.

SYNOPSIS

```
TDRV010_HANDLE tdrv010Open
(
     char *DeviceName
);
```

DESCRIPTION

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

Make sure that this function is called with enough privileges, especially for Windows 7.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device.

EXAMPLE

```
#include “tdrv010api.h”
TDRV010_HANDLE FileDescriptor;

/*
** open file descriptor to device
*/
FileDescriptor = tdrv010Open(“\\\\.\\TDRV010_1” );
if (FileDescriptor == INVALID_HANDLE_VALUE)
{
     /* handle open error */
}
```

RETURNS

A device descriptor number, or INVALID_HANDLE_VALUE if the function fails. To get extended error information, call ***GetLastError***.

ERROR CODES

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

4.1.2 tdrv010Close()

NAME

tdrv010Close() – closes a device.

SYNOPSIS

```
int tdrv010Close
(
     TDRV010_HANDLE     FileDescriptor
);
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

FileDescriptor

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

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

/*
** close file descriptor to device
*/
result = tdrv010Close( FileDescriptor );
if (result < 0)
{
     /* handle close error */
}
```

RETURNS

Zero, or a negative error code.

ERROR CODES

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

4.2 Device Access Functions

4.2.1 tdrv010Read()

NAME

tdrv010Read – read a CAN message from device

SYNOPSIS

```
int tdrv010Read
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        ULONG               timeout,
        ULONG               *pIdentifier,
        UCHAR               *pIOFlags,
        UCHAR               *pStatus,
        int                 *pLength,
        UCHAR               *pData
);
```

DESCRIPTION

This function reads a CAN message from a specified device. If no message has been received, the function will wait until a message is received, or the function will timeout after the specified time.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

timeout

This argument specifies the maximum time (in seconds) the function is willing to wait for an incoming message. The time is specified in seconds.

pIdentifier

This parameter points to buffer where the message identifier of the received message will be stored to.

pIOFlags

This parameter points to buffer where the I/O flag of the received message will be stored to. The following flags may be set:

Value	Description
TDRV010_EXTENDED	Set if the received message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the received message is a remote transmission request (RTR) frame.

pStatus

This parameter points to buffer where the status information about overrun condition, either in the CAN controller or intermediate software FIFO, will be stored to.

Value	Description
TDRV010_SUCCESS	No messages lost
TDRV010_FIFO_OVERRUN	One or more messages have been overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TDRV010_MSGOBJ_OVERRUN	One or more messages have been overwritten in the CAN controller message FIFO because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

pLength

This parameter points to buffer where the data length of the received message data in bytes will be stored to. The returned length will always be 0…8.

pData

This parameter points to a buffer where the received data bytes will stored to. This buffer must have a length of at least 8 byte. Data[0] receives the first data byte, Data[1] receives the second data byte and so on. The number of valid bytes is specified by *pLength*.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result, i;
ULONG              identifier;
UCHAR              IOFlags;
UCHAR              msgStatus;
int                dataLen;
UCHAR              dataBuf[8];

…
```

```
…

/*
** Read a CAN message from channel 2
**   - timeout after 10 seconds
*/
result = tdrv010Read (  FileDescriptor,
                        2,                  /* channel */
                        10,                 /* timeout */
                        &identifier,
                        &IOFlags,
                        &msgStatus,
                        &dataLen,
                        dataBuf);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
     printf("%s %s Identifier = %ld\n",
            (IOFlags & TDRV010_EXTENDED) ? "Extd" : "Stnd",
            (IOFlags & TDRV010_REMOTE_FRAME) ? "Remote Msg - " : "Msg - ",
            identifier);
     printf("%d data bytes received\n", dataLen);
     for( i = 0; i < dataLen; i++ )
     {
          printf("%02X ", dataBuf[i]);
     }
     printf("\n")
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small, or a buffer pointer is NULL.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_FILE_OFFLINE	The channel must be in BUS ON state to receive messages.
ERROR_NETWORK_BUSY	There is already another job waiting for message reception on this channel.
ERROR_OPERATION_ABORTED	The job has been canceled by Windows.
ERROR_SEM_TIMEOUT	The specified timeout time has expired without receiving a message.

Other returned error codes are system error conditions.

4.2.2 tdrv010ReadNoWait()

NAME

tdrv010ReadNoWait – read a CAN message from device (return immediately)

SYNOPSIS

```
int tdrv010ReadNoWait
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        ULONG               *pIdentifier,
        UCHAR               *pIOFlags,
        UCHAR               *pStatus,
        int                 *pLength,
        UCHAR               *pData
);
```

DESCRIPTION

This function reads a CAN message from a specified device. This function will return immediately, either if a message is available or not. If no message has been received the function will return an error.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

pIdentifier

This parameter points to buffer where the message identifier of the received message will be stored to.

pIOFlags

This parameter points to buffer where the I/O flag of the received message will be stored to. The following flags may be set:

Value	Description
TDRV010_EXTENDED	Set if the received message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the received message is a remote transmission request (RTR) frame.

pStatus

This parameter points to buffer where the status information about overrun condition, either in the CAN controller or intermediate software FIFO, will be stored to.

Value	Description
TDRV010_SUCCESS	No messages lost
TDRV010_FIFO_OVERRUN	One or more messages was overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TDRV010_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message FIFO because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

pLength

This parameter points to buffer where the data length of the received message data in bytes will be stored to. The returned length will always be 0…8.

pData

This parameter points to a buffer where the received data bytes will stored to. This buffer must have a length of at least 8 byte. Data[0] receives the first data byte, Data[1] receives the second data byte and so on. The number of valid bytes is specified by *pLength*.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result, i;
ULONG              identifier;
UCHAR              IOFlags;
UCHAR              msgStatus;
int                dataLen;
UCHAR              dataBuf[8];

…
```

```
…
/*
** Read a CAN message from channel 2
*/
result = tdrv010ReadNoWait(  FileDescriptor,
                             2,                  /* channel */
                             &identifier,
                             &IOFlags,
                             &msgStatus,
                             &dataLen,
                             dataBuf);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
     printf("%s %s Identifier = %ld\n",
            (IOFlags & TDRV010_EXTENDED) ? "Extd" : "Stnd",
            (IOFlags & TDRV010_REMOTE_FRAME) ? "Remote Msg - " : "Msg - ",
            identifier);
     printf("%d data bytes received\n", dataLen);
     for( i = 0; i < dataLen; i++ )
     {
          printf("%02X ", dataBuf[i]);
     }
     printf("\n")
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small, or a buffer pointer is NULL.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_FILE_OFFLINE	The channel must be in BUS ON state to receive messages.
ERROR_MR_MID_NOT_FOUND	There is no message in the FIFO buffer.
ERROR_NETWORK_BUSY	There is already another job waiting for message reception on this channel.
ERROR_OPERATION_ABORTED	The job has been canceled by Windows.

Other returned error codes are system error conditions.

4.2.3 tdrv010Write()

NAME

tdrv010Write – send a CAN message to device

SYNOPSIS

```
int tdrv010Write
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        ULONG               timeout,
        ULONG               identifier,
        UCHAR               IOFlags,
        int                 length,
        UCHAR               *pData
);
```

DESCRIPTION

This function sends a CAN message to a specified device and waits until it is send. If the message cannot be sent within a specified timeout time, the function will return and indicate an error.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

timeout

This argument specifies the maximum time (in seconds) the function is willing to wait for a successful sending (acknowledge) for the sent message.

identifier

This argument specifies the message identifier of the message.

IOFlags

This parameter specifies the I/O flags for the message. The following flags may be set:

Value	Description
TDRV010_EXTENDED	Set if the transmit message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the transmit message is a remote transmission request (RTR) frame.
TDRV010_SINGLE_SHOT	There will be no retry sending the message
TDRV010_SELF_RECEPTION	The message will be able to be received on the sending device.

length

This parameter specifies the data length of the message data in bytes. The length of the message can be 0…8 byte.

pData

This parameter points to a buffer where the message data must be stored. The number of used bytes must be specified in *length*.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;
UCHAR              dataBuf[8] = “123456”;

…
```

…

```
/*
** Send a CAN message on channel 2
**   - extended identifier (42)
**   - timout after 5 seconds
*/
errVal = tdrv010Write ( FileDescriptor,
                        2,            /* channel */
                        5,            /* timeout */
                        42,
                        TDRV010_EXTENDED,
                        6,
                        dataBuf);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small, or a buffer pointer is NULL.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_FILE_OFFLINE	The channel must be in BUS ON state to receive messages.
ERROR_NETWORK_BUSY	There is already another job waiting for message reception on this channel.
ERROR_OPERATION_ABORTED	The job has been canceled by Windows.
ERROR_SEM_TIMEOUT	The specified timeout time has expired without receiving a message.

Other returned error codes are system error conditions.

4.2.4 tdrv010WriteNoWait()

NAME

tdrv010WriteNoWait – send a CAN message to device (return immediately)

SYNOPSIS

```
int tdrv010WriteNoWait
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        ULONG               identifier,
        UCHAR               IOFlags,
        int                 length,
        UCHAR               *pData
);
```

DESCRIPTION

This function starts sending a CAN message to a specified device and returns. The function will return immediately and will not wait until the message is send.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

identifier

This argument specifies the message identifier of the message.

IOFlags

This parameter specifies the I/O flags for the message. The following flags may be set:

Value	Description
TDRV010_EXTENDED	Set if the transmit message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the transmit message is a remote transmission request (RTR) frame.
TDRV010_SINGLE_SHOT	There will be no retry sending the message
TDRV010_SELF_RECEPTION	The message will be able to be received on the sending device.

length

This parameter specifies the data length of the message data in bytes. The length of the message can be 0…8 byte.

pData

This parameter points to a buffer where the message data must be stored to. The number of used bytes must be specified in *length*.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;
UCHAR              dataBuf[8] = “123456”;

/*
** Send a CAN message on channel 1
**   - standard identifier (42)
*/
errVal = tdrv010WriteNoWait( FileDescriptor,
                             1,            /* channel */
                             42,
                             TDRV010_STANDARD,
                             6,
                             dataBuf);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small, or a buffer pointer is NULL.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_FILE_OFFLINE	The channel must be in BUS ON state to receive messages.
ERROR_NETWORK_BUSY	There is already another job waiting for message reception on this channel.
ERROR_OPERATION_ABORTED	The job has been canceled by Windows.

Other returned error codes are system error conditions.

4.2.5 tdrv010SetBitTiming()

NAME

tdrv010SetBitTiming – configure the bit-timing of the specified device

SYNOPSIS

```
int tdrv010SetBitTiming
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        USHORT              timingValue,
        BOOLEAN             useThreeSamples
);
```

DESCRIPTION

This function configures a new bit-timing for the specified CAN channel.

This function must be executed in BUS OFF state.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to configure. Allowed values are 1 for channel 1 and 2 for channel 2.

timingValue

This argument specifies the new value for the bit timing register 0 (bit 0...7) and for the bit timing register 1 (bit 8...15). Possible transfer rates are between 60 Kbit per second and 1 Mbit per second. The include file 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_100KBIT … TDRV010_1MBIT).
For other transfer rates please follow the instructions of the *SJA1000 Product Specification*, which is also part of the corresponding hardware engineering documentation.

useThreeSampes

If this argument is TRUE (1) the CAN bus is sampled three times per bit time instead of once.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

/*
** Configure channel 2 for 250Kbit/s
*/
errVal = tdrv010SetBitTiming (   FileDescriptor,
                                 2,                  /* channel */
                                 TDRV010_250KBIT,
                                 FALSE);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.6 tdrv010SetFilter()

NAME

tdrv010SetFilter – configure the acceptance filter

SYNOPSIS

```
int tdrv010SetFilter
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        BOOLEAN             singleFilter,
        ULONG               acceptanceCode,
        ULONG               acceptanceMask
);
```

DESCRIPTION

This function configures the accaptance filtering of the specified CAN channel.

The acceptance filter compares the received identifier with the acceptance filter and decides whether a message should be accepted or not. If a message passes the acceptance filter it is stored in the receive FIFO.

The acceptance filter is defined by the acceptance code registers and the acceptance mask registers. The bit patterns of messages to be received are defined in the acceptance code register.

The corresponding acceptance mask registers allow defining certain bit positions to be "don't care" (a 1 at a bit position means "don't care").

A detailed description of the acceptance filter and possible filter modes can be found in the SJA1000 Product Specification Manual.

This function must be executed in BUS OFF state.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to configure. Allowed values are 1 for channel 1 and 2 for channel 2.

singleFilter

Set TRUE (1) for single filter mode, set FALSE (0) for dual filter mode.

acceptanceCode

The contents of this parameter will be written to acceptance code register of the controller

acceptanceMask

The contents of this parameter will be written to the acceptance mask register of the controller.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

/*
** Configure channel 2 to accept all incoming messages
*/
errVal = tdrv010SetFilter (  FileDescriptor,
                             2,                  /* channel */
                             FALSE,
                             0x00000000,
                             0xFFFFFFFF);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.7 tdrv010Start()

NAME

tdrv010Start – sets the CAN channel to bus on mode

SYNOPSIS

```
int tdrv010Start
(
      TDRV010_HANDLE     FileDescriptor,
      UCHAR              canChan
);
```

DESCRIPTION

This function starts the specified CAN channel and set it to bus on mode.

After an abnormal rate of occurrences of errors on the CAN bus or after driver startup, the CAN controller enters the Bus OFF state. This function resets the "reset mode" bit in the mode register. The CAN controller begins the BUS OFF recovery sequence and resets the transmit and receive error counters. If the CAN controller counts 128 packets of 11 consecutive recessive bits on the CAN bus, the Bus Off state is exited.

Before the driver is able to communicate over the CAN bus after driver startup, this control function must be executed.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to start. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Start up channel 2
*/
errVal = tdrv010Start ( FileDescriptor,
                        2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_IO_DEVICE	Error during Bus On sequence.

Other returned error codes are system error conditions.

4.2.8 tdrv010Stop()

NAME

tdrv010Stop – sets the CAN channel to bus off mode

SYNOPSIS

```
int tdrv010Stop
(
      TDRV010_HANDLE     FileDescriptor,
      UCHAR              canChan
);
```

DESCRIPTION

This function stops the specified CAN channel and set it to bus off mode.

After execution of this function the CAN controller is completely removed from the CAN bus and cannot communicate until the function tdrv010Start() is executed.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to stop. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Stop up channel 2
*/
errVal = tdrv010Stop (  FileDescriptor,
                        2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_IO_DEVICE	Error during Bus Off sequence.

Other returned error codes are system error conditions.

4.2.9 tdrv010FlushReceiveFifo()

NAME

tdrv010FlushReceiveFifo – flush receive buffer of CAN channel

SYNOPSIS

```
int tdrv010FlushReceiveFifo
(
      TDRV010_HANDLE     FileDescriptor,
      UCHAR              canChan
);
```

DESCRIPTION

This function flushes the receive buffer of the specified channel, and removes all previously received messages from the FIFO.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to stop. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

…

```
/*
** Flush receive FIFO of channel 2
*/
errVal = tdrv010FlushReceiveFifo(FileDescriptor,
                                 2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.

Other returned error codes are system error conditions.

4.2.10 tdrv010GetControllerStatus()

NAME

tdrv010GetControllerStatus – returns the CAN controller state

SYNOPSIS

```
int tdrv010GetControllerStatus
(
    TDRV010_HANDLE    FileDescriptor,
    UCHAR             canChan,
    TDRV010_STATUS    *pCANStatus
);
```

DESCRIPTION

This function returns the actual contents of several CAN controller registers for diagnostic purposes in an application supplied buffer (*TDRV010_STATUS*).

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

pCANStatus

This argument points to a buffer (*TDRV010_STATUS*) where the current status will be filled in.

```
typedef struct
{
    UCHAR    channel;
    UCHAR    ArbitrationLostCapture;
    UCHAR    ErrorCodeCapture;
    UCHAR    TxErrorCounter;
    UCHAR    RxErrorCounter;
    UCHAR    ErrorWarningLimit;
    UCHAR    StatusRegister;
    UCHAR    ModeRegister;
    UCHAR    RxMessageCounterMax;
} TDRV010_STATUS, *PTDRV010_STATUS;
```

channel

This parameter is not used by this function. Set to same value as *canChan*.

ArbitrationLostCapture

Contents of the arbitration lost capture register. This register contains information about the bit position of losing arbitration.

ErrorCodeCapture

Contents of the error code capture register. This register contains information about the type and location of errors on the bus.

TxErrorCounter

Contents of the TX error counter register. This register contains the current value of the transmit error counter.

RxErrorCounter

Contents of the RX error counter register. This register contains the current value of the receive error counter.

ErrorWarningLimit

Contents of the error warning limit register.

StatusRegister

Contents of the status register.

ModeRegister

Contents of the mode register.

RxMessageCounterMax

Contains the peak value of messages in the software receive FIFO. This internal counter value will be reset to 0 after reading.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;
TDRV010_STATUS     statusBuf;

…
```

```
…

/*
** Read a CAN controller status of channel 2
*/
result = tdrv010GetControllerStatus ( FileDescriptor,
                                       channel,
                                       &statusBuf);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
     printf (  "\nArbitration lost capture register  = 0x%02X\n",
               statusBuf.ArbitrationLostCapture);
     printf (  "Error code capture register          = 0x%02X\n",
               statusBuf.ErrorCodeCapture);
     printf (  "TX error counter register            = 0x%02X\n",
               statusBuf.TxErrorCounter);
     printf (  "RX error counter register            = 0x%02X\n",
               statusBuf.RxErrorCounter);
     printf (  "Error warning limit register         = 0x%02X\n",
               statusBuf.ErrorWarningLimit);
     printf (  "Status register                      = 0x%02X\n",
               statusBuf.StatusRegister);
     printf (  "Mode register                        = 0x%02X\n",
               statusBuf.ModeRegister);
     printf (  "Peak value RX message counter        = %d\n",
               statusBuf.RxMessageCounterMax);
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small, or a buffer pointer is NULL.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.

Other returned error codes are system error conditions.

4.2.11 tdrv010SelftestEnable()

NAME

tdrv010SelftestEnable – enable the self test facility of CAN channel

SYNOPSIS

```
int tdrv010SelftestEnable
(
      TDRV010_HANDLE     FileDescriptor,
      UCHAR              canChan
);
```

DESCRIPTION

This function enables the self test facility of the SJA1000 CAN controller for the specified channel.

In this mode a full node test is possible without any other active node on the bus using the self reception facility. The CAN controller will perform a successful transmission even if there is no acknowledge received.

Also in self test mode the normal functionality is given, that means the CAN controller is able to receive messages from other nodes and can transmit message to other nodes if any connected.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Enable self test for channel 2
*/
errVal = tdrv010SelftestEnable ( FileDescriptor,
                                 2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.12 tdrv010SelftestDisable()

NAME

tdrv010SelftestDisable – disable the self test facility of CAN channel

SYNOPSIS

```
int tdrv010SelftestDisable
(
      TDRV010_HANDLE     FileDescriptor,
      UCHAR              canChan
);
```

DESCRIPTION

This function disables the self test facility of the SJA1000 CAN controller for the specified channel.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
...

/*
** Disable self test for channel 2
*/
errVal = tdrv010SelftestDisable (FileDescriptor,
                                 2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.13 tdrv010ListenOnlyEnable()

NAME

tdrv010ListenOnlyEnable – enable the listen only facility of CAN channel

SYNOPSIS

```
int tdrv010ListenOnlyEnable
(
      TDRV010_HANDLE    FileDescriptor,
      UCHAR             canChan
);
```

DESCRIPTION

This function enables the listen only facility of the SJA1000 CAN controller for the specified channel.

In this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully. Message transmission is not possible. All other functions can be used like in normal mode.

This mode can be used for software driver bit rate detection and 'hot-plugging'.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Enable listen only for channel 2
*/
errVal = tdrv010ListenOnlyEnable(FileDescriptor,
                                 2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.14 tdrv010ListenOnlyDisable()

NAME

tdrv010ListenOnlyDisable – disable the listen only facility of CAN channel

SYNOPSIS

```
int tdrv010ListenOnlyDisable
(
      TDRV010_HANDLE     FileDescriptor,
      UCHAR              canChan
);
```

DESCRIPTION

This function disables the listen only facility of the SJA1000 CAN controller for the specified channel.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Disable listen only for channel 2
*/
errVal = tdrv010ListenOnlyDisable (   FileDescriptor,
                                      2);           /* channel */
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.15 tdrv010SetLimit()

NAME

tdrv010SetLimit – configure error warning limit of the specified device

SYNOPSIS

```
int tdrv010SetLimit
(
        TDRV010_HANDLE      FileDescriptor,
        UCHAR               canChan,
        UCHAR               errorLimit
);
```

DESCRIPTION

This function sets a new error warning limit in the corresponding CAN controller register of the specified CAN channel.

This function must be executed in BUS OFF state.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to configure. Allowed values are 1 for channel 1 and 2 for channel 2.

errorLimit

This parameter specifies the new warning limit. Allowed values are between 0 and 255.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

/*
** Set error warning limit to 100 for channel 2
*/
errVal = tdrv010SetLimit (   FileDescriptor,
                             2,            /* channel */
                             100);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

Other returned error codes are system error conditions.

4.2.16 tdrv010CanReset()

NAME

tdrv010CanReset – set CAN channel to reset / operating mode

SYNOPSIS

```
int tdrv010CanReset
(
      TDRV010_HANDLE    FileDescriptor,
      UCHAR             canChan,
      BOOLEAN           canReset
);
```

DESCRIPTION

This function sets the specified CAN controller in reset or operating mode by controlling the external controller reset pin of the specified CAN channel.

This function is only available for TPMC310.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to configure. Allowed values are 1 for channel 1 and 2 for channel 2.

canReset

This parameter specifies if controller shall be set in reset (TRUE) or in operating mode (FALSE).

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Set channel 1 to operating mode and channel 2 into reset
*/
errVal = tdrv010CanReset (   FileDescriptor,
                             1,            /* channel */
                             FALSE);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}

errVal = tdrv010CanReset (   FileDescriptor,
                             2,            /* channel */
                             TRUE);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_IO_DEVICE	Unable to reinitialize the certain CAN controller.
ERROR_INVALID_FUNCTION	This function is only supported by TPMC310 modules

Other returned error codes are system error conditions.

4.2.17 tdrv010CanSel()

NAME

tdrv010CanSel – set CAN channel to silent mode

SYNOPSIS

```
int tdrv010CanSel
(
      TDRV010_HANDLE    FileDescriptor,
      UCHAR             canChan,
      BOOLEAN           canSel
);
```

DESCRIPTION

This function sets the specified CAN channel in silent or operating mode.

This function is only available for TPMC310.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to configure. Allowed values are 1 for channel 1 and 2 for channel 2.

canSel

This parameter specifies if controller shall be in silent mode (TRUE) or in operating mode (FALSE).

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Set channel 1 to silent mode
*/
errVal = tdrv010CanSel( FileDescriptor,
                        1,            /* channel */
                        TRUE);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_INVALID_FUNCTION	This function is only supported by TPMC310 modules

Other returned error codes are system error conditions.

4.2.18 tdrv010CanInt()

NAME

tdrv010CanInt – enable or disable the global interrupt of the CAN controller

SYNOPSIS

```
int tdrv010CanInt
(
      TDRV010_HANDLE    FileDescriptor,
      UCHAR             canChan,
      BOOLEAN           canInt
);
```

DESCRIPTION

This function enables or disables the global interrupt of the specified CAN controller.

This function is only available for TPMC310.

PARAMETERS

FileDescriptor

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

canChan

This argument specifies the channel to configure. Allowed values are 1 for channel 1 and 2 for channel 2.

canInt

This parameter specifies if the global interrupt of the CAN controller shall be enabled (TRUE) or disabled (FALSE).

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     FileDescriptor;
int                result;

…
```

```
…

/*
** Enable global interrupt on channel 2
*/
errVal = tdrv010CanInt( FileDescriptor,
                        2,            /* channel */
                        TRUE);
if (result != 0)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

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

ERROR CODES

Error code	Description
ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_INVALID_FUNCTION	This function is only supported by TPMC310 modules

Other returned error codes are system error conditions.

5 Device Driver Programming

The TDRV010-SW-65 Windows device driver is a kernel mode device driver.

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.

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.)

5.1 Files and I/O Functions

The following section does not contain a full description of the Win32 functions for interaction with the TDRV010 device driver. Only the required parameters are described in detail.

5.1.1 Opening a Device

Before you can perform any I/O the *TDRV010* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TDRV010* device.

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

PARAMETERS

lpFileName

This parameter points to a null-terminated string, which specifies the name of the TDRV010 to open. The *lpFileName* string should be of the form **\\.\TDRV010_*x*** to open the device *x*. The ending x is a one-based number. The first device found by the driver is \\.\TDRV010_1, the second \\.\TDRV010_2 and so on.

dwDesiredAccess

This parameter specifies the type of access to the TDRV010.
For the TDRV010 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

This argument is a pointer to a security structure. Set to NULL for TDRV010 devices.

dwCreationDistribution

Specifies the action to take on existing files, and which action to take when files do not exist. TDRV010 devices must be always opened **OPEN_EXISTING**.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to FILE_FLAG_OVERLAPPED for TDRV010 devices.

hTemplateFile

This value must be NULL for TDRV010 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TDRV010 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call ***GetLastError***.

EXAMPLE

```
HANDLE    hDevice;

hDevice = CreateFile(
     "\\\\.\\TDRV010_1",
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,                    // no security attrs
     OPEN_EXISTING,           // TDRV010 device always open existing
     FILE_FLAG_OVERLAPPED,    // overlapped I/O
     NULL
);

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

SEE ALSO

CloseHandle(), Win32 documentation CreateFile()

5.1.2 Closing a Device

The **CloseHandle** function closes an open TDRV010 handle.

BOOL CloseHandle(
HANDLE *hDevice;*
)

PARAMETERS

hDevice

Identifies an open TDRV010 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 ()

5.1.3 TDRV010 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,            // handle to device of interest
    DWORD           dwIoControlCode,    // control code of operation to perform
    LPVOID          lpInBuffer,         // pointer to buffer to supply input data
    DWORD           nInBufferSize,      // size of input buffer
    LPVOID          lpOutBuffer,        // pointer to buffer to receive output data
    DWORD           nOutBufferSize,     // size of output buffer
    LPDWORD         lpBytesReturned,    // pointer to variable to receive output byte count
    LPOVERLAPPED    lpOverlapped        // pointer to overlapped structure for asynchronous
                                        // operation
)
```

PARAMETERS

hDevice

Handle to the TDRV010 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 *tdrv010.h*:

Value	Meaning
IOCTL_TDRV010_READ	Read CAN message from FIFO
IOCTL_TDRV010_WRITE	Send CAN-Message
IOCTL_TDRV010_BITTIMING	Setup Bit-timing
IOCTL_TDRV010_SETFILTER	Setup Acceptance Filter
IOCTL_TDRV010_BUSON	Enter Bus ON mode
IOCTL_TDRV010_BUSOFF	Enter Bus OFF mode
IOCTL_TDRV010_FLUSH	Flush Receive FIFO
IOCTL_TDRV010_CANSTATUS	Read CAN status from Controller
IOCTL_TDRV010_ENABLE_SELFTEST	Enable Selftest Mode
IOCTL_TDRV010_DISABLE_SELFTEST	Disable Selftest Mode
IOCTL_TDRV010_ENABLE_LISTENONLY	Enable Listen Only Mode
IOCTL_TDRV010_DISABLE_LISTENONLY	Disable Listen Only Mode
IOCTL_TDRV010_SETLIMIT	Set Warning Limit
IOCTL_TDRV010_CAN_RESET	Setup CAN Controller reset/operating mode
IOCTL_TDRV010_CAN_SEL	Setup CAN Controller silent/operating mode
IOCTL_TDRV010_CAN_INT	Enable/disable CAN Controller interrupts

See below for more detailed information on each control code.

To use these TDRV010 specific control codes the header file tdrv010.h must be included in the application

lpInBuffer

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

nInBufferSize

Specifies the size of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

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

nOutBufferSize

Specifies the size of the buffer in bytes 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. Refer to the Ioctl specific manual section how this parameter must be set.

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***.

SEE ALSO

Win32 documentation DeviceIoControl()

5.1.3.1 IOCTL_TDRV010_READ

This control function reads a CAN message from the receive FIFO and returns it in an application supplied buffer (*TDRV010_MSG_BUF*). The parameters *lpInBuffer and lpOutBuffer* pass pointers to the buffer to the device driver.

```
typedef struct
{
    UCHAR     channel;
    BOOLEAN   noWait;
    ULONG     Identifier;
    UCHAR     IOFlags;
    UCHAR     MsgLen;
    UCHAR     Data[8];
    ULONG     Timeout;
    UCHAR     Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

noWait

This parameter specifies if the function shall wait for a message reception if no message is stored in the receive FIFO. If this flag is set, the function will return immediately.

Identifier

Obtains the message identifier of the read CAN message.

IOFlags

Obtains CAN message attributes as a set of bit flags. The following attribute flags are possible:

Value	Description
TDRV010_EXTENDED	Set if the received message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the received message is a remote transmission request (RTR) frame.

MsgLen

Obtains the number of message data bytes (0...8).

Data[8]

This buffer receives up to 8 data bytes. Data[0] receives message Data 0, Data[1] receives message Data 1 and so on.

Timeout

Specifies the amount of time (in seconds) the caller is willing to wait for execution of read request. This parameter is not used if the *noWait* option is enabled.

Status

Obtains status information about overrun conditions, either in the CAN controller or intermediate software FIFO.

Value	Description
TDRV010_SUCCESS	No messages lost
TDRV010_FIFO_OVERRUN	One or more messages was overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TDRV010_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message FIFO because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_MSG_BUF     msgBuf;
int                 i;

//
// Read message from FIFO (channel 1)
//
msgBuf.channel = 1;
msgBuf.noWait = FALSE;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_READ,
     &msgBuf,                    // contains the input data
     sizeof(msgBuf),             // size of message buffer
     &msgBuf,                    // contains the received message
     sizeof(msgBuf),             // size of message buffer
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

…
```

```
…

if( success ) {
     printf("\nRead Message successful completed!\n");
     printf("%s %s Id = %ld (0x%lx)\n",
          (msgBuf.IOFlags & TDRV010_EXTENDED) ? "Extended" : "Standard",
          (msgBuf.IOFlags & TDRV010_REMOTE_FRAME) ?
               "Remote Frame Message - " : "Message - ",
          msgBuf.Identifier, msgBuf.Identifier);
     printf("%d data bytes received\n", msgBuf.MsgLen);
     for( i = 0; i < msgBuf.MsgLen; i++ )
     {
          printf("%02X ", msgBuf.Data[i]);
     }
     printf("\nMessage status field = %d\n", msgBuf.Status );
}
else
{
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_FILE_OFFLINE	The channel must be in BUS ON state to receive messages.
ERROR_MR_MID_NOT_FOUND	There is no message in the FIFO buffer.
ERROR_NETWORK_BUSY	There is already another job waiting for message reception on this channel.
ERROR_OPERATION_ABORTED	The job has been canceled by Windows.
ERROR_SEM_TIMEOUT	The specified timeout time has expired without receiving a message.

5.1.3.2 IOCTL_TDRV010_WRITE

This TDRV010 control function starts the transmission of a CAN message specified in an application supplied buffer (*TDRV010_MSG_BUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

```
typedef struct
{
    UCHAR     channel;
    BOOLEAN   noWait;
    ULONG     Identifier;
    UCHAR     IOFlags;
    UCHAR     MsgLen;
    UCHAR     Data[8];
    ULONG     Timeout;
    UCHAR     Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

noWait

This parameter specifies if the function shall wait for a message transmission acknowledge or not. If this flag is set, the function will return immediately.

Identifier

Specifies the message identifier of the CAN message.

IOFlags

Specifies CAN message attributes as a set of ORed bit flags. The following attribute flags are possible:

Value	Description
TDRV010_EXTENDED	Set if the transmit message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the transmit message is a remote transmission request (RTR) frame.
TDRV010_SINGLE_SHOT	There will be no retry sending the message
TDRV010_SELF_RECEPTION	The message will be able to be received on the sending device.

MsgLen

Specifies the number of message data bytes (0...8) specified in Data.

Data[8]

This buffer contains up to 8 data bytes. Data[0] will be transmitted as Data 0, Data[1] will be transmitted as Data 1 and so on.

Timeout

Specifies the amount of time (in seconds) the caller is willing to wait for execution of the transmit request. If the *noWait* option is enabled this parameter specifies the time (in seconds) the driver will try to transmit this message.

Status

Parameter unused for this function.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_MSG_BUF     msgBuf;

//
// Transmit message (channel 1)
//
msgBuf.channel     = 1;
msgBuf.noWait      = FALSE;
msgBuf.Identifier  = 1234;
msgBuf.Timeout     = 10;
msgBuf.IOFlags     = TDRV010_EXTENDED | TDRV010_SINGLE_SHOT;
msgBuf.MsgLen      = 2;
msgBuf.Data[0]     = 0xaa;
msgBuf.Data[1]     = 0x55;

success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_WRITE,
     &msgBuf,                    // contains the input data
     sizeof(msgBuf),             // size of message buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

…
```

```
…

if( success ) {
     printf("\nWrite Message successful completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_FILE_OFFLINE	The channel must be in BUS ON state to transmit messages.
ERROR_NETWORK_BUSY	There is already another job waiting for message transmission on this channel.
ERROR_OPERATION_ABORTED	The job has been canceled by Windows.
ERROR_SEM_TIMEOUT	The specified timeout time has expired without successful transmission of the message.
ERROR_IO_DEVICE	An error occurred while starting the transmission.

5.1.3.3 IOCTL_TDRV010_BITTIMING

This TDRV010 control function modifies the bit timing registers for the specified channel. The new timing parameters are specified in an application supplied buffer (*TDRV010_TIMING*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

```
typedef struct
{
    UCHAR     channel;
    USHORT    TimingValue;
    BOOLEAN   ThreeSamples;
} TDRV010_TIMING, *PTDRV010_TIMING;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

TimingValue

This parameter holds the new value for the bit timing register 0 (bit 0...7) and for the bit timing register 1 (bit 8...15). Possible transfer rates are between 60 Kbit per second and 1 Mbit per second. The include file 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_100KBIT … TDRV010_1MBIT).
For other transfer rates please follow the instructions of the *SJA1000 Product Specification*, which is also part of the corresponding hardware engineering documentation.

ThreeSamples

If this parameter is TRUE (1) the CAN bus is sampled three times per bit time instead of one.

This function must be executed in BUS OFF state.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_TIMING      TimingBuf;

//
// Change bitrate (channel 1)
//
TimingBuf.channel       = 1;
TimingBuf.TimingValue   = TDRV010_500KBIT;
TimingBuf.ThreeSamples  = FALSE;

…
```

```
…

success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_BITTIMING,
     &TimingBuf,                 // contains the input data
     sizeof(TimingBuf),          // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);
if( success ) {
     printf("\nChange Bitrate successfully completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the user buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.4 IOCTL_TDRV010_SETFILTER

This TDRV010 control function modifies the acceptance filter of the specified channel. The acceptance filter parameters are specified in an application supplied buffer (*TDRV010_FILTER*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

The acceptance filter compares the received identifier with the acceptance filter and decides whether a message should be accepted or not. If a message passes the acceptance filter it is stored in the receive FIFO.

The acceptance filter is defined by the acceptance code registers and the acceptance mask registers. The bit patterns of messages to be received are defined in the acceptance code register.

The corresponding acceptance mask registers allow defining certain bit positions to be "don't care" (a 1 at a bit position means "don't care").

```
typedef struct
{
    UCHAR     channel;
    BOOLEAN   SingleFilter;
    ULONG     AcceptanceCode;
    ULONG     AcceptanceMask;
} TDRV010_FILTER, *PTDRV010_FILTER;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

SingleFilter

Set TRUE (1) for single filter mode, set FALSE (0) for dual filter mode.

AcceptanceCode

The contents of this parameter will be written to acceptance code register of the controller

AcceptanceMask

The contents of this parameter will be written to the acceptance mask register of the controller.

A detailed description of the acceptance filter and possible filter modes can be found in the SJA1000 Product Specification Manual.

This function must be executed in BUS OFF state.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_FILTER      FilterBuf;

//
// Change acceptance mask (channel 1)
//
FilterBuf.channel           = 1;
FilterBuf.SingleFilter      = TRUE;
FilterBuf.AcceptanceCode    = 0x00000000;
FilterBuf.AcceptanceMask    = 0xffffffff;

success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_SETFILTER,
     &FilterBuf,                 // contains the input data
     sizeof(FilterBuf),          // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

if( success ) {
     printf("\nChange Filter successfully completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.5 IOCTL_TDRV010_BUSON

This TDRV010 control function sets the channel into BUS ON state. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

After an abnormal rate of occurrences of errors on the CAN bus or after driver startup, the CAN controller enters the Bus OFF state. This control function resets the "reset mode" bit in the mode register. The CAN controller begins the BUS OFF recovery sequence and resets the transmit and receive error counters. If the CAN controller counts 128 packets of 11 consecutive recessive bits on the CAN bus, the Bus Off state is exited.

typedef struct
{
    UCHAR   channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

Before the driver is able to communicate over the CAN bus after driver startup, this control function must be executed.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

…
```

```
…

//
// Go Bus on (channel 1)
//
defBuf.channel     = 1;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_BUSON,
     &defBuf,                    // contains the input data
     sizeof(defBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

if( success ) {
     printf("\nChanging to BUS ON completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_IO_DEVICE	Error during Bus On sequence.

5.1.3.6 IOCTL_TDRV010_BUSOFF

This TDRV010 control function sets the channel into BUS OFF state. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function IOCTL_TDRV010_BUSON is executed.

typedef struct
{
    UCHAR   channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

//
// Go Bus off (channel 1)
//
defBuf.channel     = 1;

success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_BUSOFF,
     &defBuf,                    // contains the input data
     sizeof(defBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

…
```

```
…

if( success ) {
     printf("\nChanging to BUS OFF completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_IO_DEVICE	Error during Bus Off sequence.

5.1.3.7 IOCTL_TDRV010_FLUSH

This TDRV010 control function flushes the receive buffer of the specified channel. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

typedef struct
{
    UCHAR   channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

//
// Flush buffer (channel 1)
//
defBuf.channel     = 1;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_FLUSH,
     &defBuf,                    // contains the input data
     sizeof(defBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

…
```

…

```
if( success ) {
     printf("\nFlushing receive buffer completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.

5.1.3.8 IOCTL_TDRV010_CANSTATUS

This TDRV010 control function returns the actual contents of several CAN controller registers for diagnostic purposes in an application supplied buffer (*TDRV010_MSG_BUF*). The parameters *lpInBuffer and lpOutBuffer* pass pointers to the buffer to the device driver.

```
typedef struct
{
    UCHAR    channel;
    UCHAR    ArbitrationLostCapture;
    UCHAR    ErrorCodeCapture;
    UCHAR    TxErrorCounter;
    UCHAR    RxErrorCounter;
    UCHAR    ErrorWarningLimit;
    UCHAR    StatusRegister;
    UCHAR    ModeRegister;
    UCHAR    RxMessageCounterMax;
} TDRV010_STATUS, *PTDRV010_STATUS;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

ArbitrationLostCapture

Contents of the arbitration lost capture register. This register contains information about the bit position of losing arbitration.

ErrorCodeCapture

Contents of the error code capture register. This register contains information about the type and location of errors on the bus.

TxErrorCounter

Contents of the TX error counter register. This register contains the current value of the transmit error counter.

RxErrorCounter

Contents of the RX error counter register. This register contains the current value of the receive error counter.

ErrorWarningLimit

Contents of the error warning limit register.

StatusRegister

Contents of the status register.

ModeRegister

Contents of the mode register.

RxMessageCounterMax

Contains the peak value of messages in the software receive FIFO. This internal counter value will be reset to 0 after reading.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_STATUS      statusBuf;

//
// Read message from FIFO (channel 1)
//
statusBuf.channel = 1;

success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_CANSTATUS,
     &statusBuf,                 // contains the input data
     sizeof(statusBuf),          // size of message buffer
     &statusBuf,                 // contains the received message
     sizeof(statusBuf),          // size of message buffer
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);
if( success ) {
     printf("\nRead Status completed!\n");
}
else
{
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.

5.1.3.9 IOCTL_TDRV010_ENABLE_SELFTEST

This TDRV010 control function enables the self test facility of the SJA1000 CAN controller. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

In this mode a full node test is possible without any other active node on the bus using the self reception facility. The CAN controller will perform a successful transmission even if there is no acknowledge received.

Also in self test mode the normal functionality is given, that means the CAN controller is able to receive messages from other nodes and can transmit message to other nodes if any connected.

This function must be executed in BUS OFF state.

typedef struct
{
    UCHAR   channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

…
```

```
…

//
// Enable Selftest (channel 1)
//
defBuf.channel     = 1;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_ENABLE_SELFTEST,
     &defBuf,                    // contains the input data
     sizeof(defBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);
if( success ) {
     printf("\Enable Selftest mode completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.10 IOCTL_TDRV010_DISABLE_SELFTEST

This TDRV010 control function disables the self test facility of the SJA1000 CAN controller. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

This function must be executed in BUS OFF state.

```
typedef struct
{
     UCHAR      channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

//
// Disable Selftest (channel 1)
//
defBuf.channel     = 1;
success = DeviceIoControl (
     hDevice,                       // TDRV010 handle
     IOCTL_TDRV010_DISABLE_SELFTEST,
     &defBuf,                       // contains the input data
     sizeof(defBuf),                // size of input buffer
     NULL,
     0,
     &NumWritten,                   // unused but required
     NULL                           // no overlapped I/O
);

…
```

```
…

if( success ) {
     printf("\Disable Selftest mode completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.11 IOCTL_TDRV010_ENABLE_LISTENONLY

This TDRV010 control function enables the listen only facility of the SJA1000 CAN controller. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

In this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully. Message transmission is not possible. All other functions can be used like in normal mode.

This mode can be used for software driver bit rate detection and 'hot-plugging'.

This function must be executed in BUS OFF state.

typedef struct
{
    UCHAR   channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

…
```

```
…

//
// Enable Listen only mode (channel 1)
//
defBuf.channel     = 1;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_ENABLE_LISTENONLY,
     &defBuf,                    // contains the input data
     sizeof(defBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

if( success ) {
     printf("\nEnable Listen only completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.12 IOCTL_TDRV010_DISABLE_LISTENONLY

This TDRV010 control function disables the listen only facility of the SJA1000 CAN controller. The channel parameter is specified in an application supplied buffer (*TDRV010_DEFAULTBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

This function must be executed in BUS OFF state.

```
typedef struct
{
      UCHAR      channel;
} TDRV010_DEFAULTBUF, *PTDRV010_DEFAULTBUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

EXAMPLE

```
#include “tdrv010.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumWritten;
TDRV010_DEFAULTBUF      defBuf;

//
// Disable Listen only mode (channel 1)
//
defBuf.channel     = 1;

success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_DISABLE_LISTENONLY,
     &defBuf,                    // contains the input data
     sizeof(defBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

…
```

```
…

if( success ) {
     printf("\nDisable Listen only completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.13 IOCTL_TDRV010_SETLIMIT

This TDRV010 control function sets a new error warning limit in the corresponding CAN controller register of the specified channel. The new limit parameters are specified in an application supplied buffer (*TDRV010_LIMITBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

```
typedef struct
{
    UCHAR    channel;
    UCHAR    limit;
} TDRV010_LIMITBUF, *PTDRV010_LIMITBUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

limit

This parameter specifies the new warning limit. Allowed values are between 0 and 255.

This function must be executed in BUS OFF state.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_LIMITBUF    LimitBuf;

…
```

```
…

//
// Set warning limit (channel 1)
//
LimitBuf.channel       = 1;
LimitBuf.limit         = 20;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_SETLIMIT,
     &LimitBuf,                  // contains the input data
     sizeof(LimitBuf),           // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);
if( success ) {
     printf("\nChange Warning Limit successfully completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_OPEN_FILES	The channel must be in BUS OFF state to execute this function.

5.1.3.14 IOCTL_TDRV010_CAN_RESET

This TDRV010 control function sets the specified CAN controller in reset or operating mode by controlling the external controller reset pin. The new mode is specified in an application supplied buffer (*TDRV010_PLDBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

This function is only supported by TPMC310 modules.

```
typedef struct
{
      UCHAR     channel;
      UCHAR     bitvalue;
} TDRV010_PLDBUF, *PTDRV010_PLDBUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

bitvalue

This parameter specifies the new mode. Allowed values are 0 to set the certain controller in reset mode and 1 to set it to operating mode.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_PLDBUF      PldBuf;

…
```

```
…

//
// Set CAN channel 1 to reset mode
//
PldBuf.channel     = 1;
PldBuf.bitvalue    = 0;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_CAN_RESET,
     &PldBuf,                    // contains the input data
     sizeof(PldBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

if( success ) {
     printf("\Setup reset/operating mode completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_IO_DEVICE	Unable to reinitialize the certain CAN controller.
ERROR_INVALID_FUNCTION	This function is only supported by TPMC310 modules

5.1.3.15 IOCTL_TDRV010_CAN_SEL

This TDRV010 control function sets the specified CAN controller to silent or operating mode. The new mode is specified in an application supplied buffer (*TDRV010_PLDBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

This function is only supported by TPMC310 modules.

```
typedef struct
{
        UCHAR       channel;
        UCHAR       bitvalue;
} TDRV010_PLDBUF, *PTDRV010_PLDBUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

bitvalue

This parameter specifies the new mode. Allowed values are 0 to set the certain controller in silent mode and 1 to set it to operating mode.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_PLDBUF      PldBuf;

…
```

```
…

//
// Set CAN channel 2 to silent mode
//
PldBuf.channel     = 2;
PldBuf.bitvalue    = 0;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_CAN_SEL,
     &PldBuf,                    // contains the input data
     sizeof(PldBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

if( success ) {
     printf("\Setup silent/operating mode completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_INVALID_FUNCTION	This function is only supported by TPMC310 modules

5.1.3.16 IOCTL_TDRV010_CAN_INT

This TDRV010 control function enables or disables the global interrupt of the specified CAN controller. The new mode is specified in an application supplied buffer (*TDRV010_PLDBUF*). The parameter *lpInBuffer* passes a pointer to the buffer to the device driver. The parameter *lpOutBuffer* is not used.

This function is only supported by TPMC310 modules.

```
typedef struct
{
        UCHAR       channel;
        UCHAR       bitvalue;
} TDRV010_PLDBUF, *PTDRV010_PLDBUF;
```

channel

This parameter specifies the channel to use. Allowed values are 1 for channel 1 and 2 for channel 2.

bitvalue

This parameter specifies the new mode. Allowed values are 0 to disable and 1 to enable all supported interrupts of the certain CAN controller.

EXAMPLE

```
#include “tdrv010.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumWritten;
TDRV010_PLDBUF      PldBuf;

…
```

```
…

//
// Enable all interrupts for CAN channel 1
//
PldBuf.channel     = 1;
PldBuf.bitvalue    = 1;
success = DeviceIoControl (
     hDevice,                    // TDRV010 handle
     IOCTL_TDRV010_CAN_INT,
     &PldBuf,                    // contains the input data
     sizeof(PldBuf),             // size of input buffer
     NULL,
     0,
     &NumWritten,                // unused but required
     NULL                        // no overlapped I/O
);

if( success ) {
     printf("\Enable/disable interrupts completed!\n");
}
else {
     ErrorHandler("Device I/O control error");      // process error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER	This error will be returned if the size of the write buffer is too small or the gain parameter is invalid.
ERROR_FILE_NOT_FOUND	Illegal channel number specified.
ERROR_INVALID_FUNCTION	This function is only supported by TPMC310 modules