

# TDRV019-SW-65

## Windows Device Driver

ADC, DAC and Digital I/O

Version 1.0.x

## User Manual

Issue 1.0.0

October 2019

---

**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](mailto:info@tews.com) [www.tews.com](http://www.tews.com)

## TDRV019-SW-65

Windows Device Driver

ADC, DAC and Digital I/O

Supported Modules:

TPMC532  
TPMC533  
TPMC542

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.

©2019 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	October 10, 2019

# Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>4</b>
<b>2</b>	<b>INSTALLATION.....</b>	<b>5</b>
<b>3</b>	<b>API DOCUMENTATION .....</b>	<b>6</b>
	<b>3.1 Usage of Function Interface.....</b>	<b>6</b>
	<b>3.2 General Functions.....</b>	<b>7</b>
	3.2.1 tdrv019Open.....	7
	3.2.2 tdrv019Close.....	9
	3.2.3 tdrv019GetPciInfo.....	11
	3.2.4 tdrv019GetBoardInfo.....	13
	3.2.5 tdrv019GetBoardTemperature.....	16
	<b>3.3 ADC Functions .....</b>	<b>18</b>
	3.3.1 tdrv019AdcSetCorrectionValues.....	18
	3.3.2 tdrv019AdcGetCorrectionValues.....	20
	3.3.3 tdrv019AdcConfigInput.....	22
	3.3.4 tdrv019AdcConfigSequencer.....	25
	3.3.5 tdrv019AdcManualSingleValue.....	27
	3.3.6 tdrv019AdcManualSingleSample.....	29
	3.3.7 tdrv019AdcSequencerSampleblock.....	31
	3.3.8 tdrv019AdcFifoFlush.....	34
	3.3.9 tdrv019AdcSequencerStart.....	36
	3.3.10 tdrv019AdcSequencerStop.....	38
	3.3.11 tdrv019AdcSequencerTimeoutSet.....	40
	<b>3.4 DAC Functions .....</b>	<b>42</b>
	3.4.1 tdrv019DacSetCorrectionValues.....	42
	3.4.2 tdrv019DacGetCorrectionValues.....	44
	3.4.3 tdrv019DacConfigOutput.....	47
	3.4.4 tdrv019DacConfigOutputRange.....	50
	3.4.5 tdrv019DacConfigSequencer.....	53
	3.4.6 tdrv019DacManualSingleValue.....	55
	3.4.7 tdrv019DacManualSingleSample.....	57
	3.4.8 tdrv019DacSequencerSampleblock.....	59
	3.4.9 tdrv019DacFifoFlush.....	62
	3.4.10 tdrv019DacSequencerStart.....	64
	3.4.11 tdrv019DacSequencerStop.....	66
	<b>3.5 Digital I/O Functions .....</b>	<b>68</b>
	3.5.1 tdrv019DioRead.....	68
	3.5.2 tdrv019DioWrite.....	70
	3.5.3 tdrv019DioWriteMasked.....	72
	3.5.4 tdrv019DioSetOutputLine.....	74
	3.5.5 tdrv019DioClearOutputLine.....	76
	3.5.6 tdrv019DioConfigureDirection.....	78
	3.5.7 tdrv019DioDebounceConfig.....	80
	3.5.8 tdrv019DioPullResistorConfig.....	82
	<b>3.6 Interrupt Functions .....</b>	<b>84</b>
	3.6.1 tdrv019InterruptWait.....	84
	3.6.2 tdrv019InterruptRegisterCallbackThread.....	86
	3.6.3 tdrv019InterruptUnregisterCallback.....	90

# 1 Introduction

The TDRV019-SW-65 Windows device driver is a kernel mode driver which allows the operation of supported hardware modules on an Intel or Intel-compatible Windows operating system.

The TDRV019-SW-65 device driver supports the following features:

- Configuration of ADC and DAC
- Read Single ADC Samples
- Write Single DAC Samples
- Read ADC Samples using DMA
- Write DAC Samples using DMA
- Configure Frame and Data Synchronization Signals
- Configure and use digital I/O signals

The TDRV019-SW-65 device driver supports the modules listed below:

TPMC532	16x/8x ADC, 8x/4x DAC and 14x Digital I/O	PMC
TPMC533	32x ADC, 16x/0x DAC and 8x Digital I/O	PMC
TPMC542	16x/8x DAC and 20x Digital I/O	PMC

**In this document all supported modules and devices will be called TDRV019. Specials for certain devices will be advised.**

To get more information about the features and use of TDRV019 devices it is recommended to read the manuals listed below.

Corresponding Hardware User Manual
Installation-Guide on Distribution Media

---

## **2 Installation**

Following files are located in the device driver directory TDRV019-SW-65 on the distribution media:

driver\*	Directory containing driver files and API libraries
example\tdrv019exa.c	Example application source
api\tdrv019api.h	Application Programming Interface header
TDRV019-SW-65-1.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history
tdrv019example.exe	Example application (executable for quick start)

Copy all required Header and API files into your desired project directory.

For device driver installation execute the application “Setup” on the distribution media. For more information refer to the Installation-Guide, which is also part of the distribution media.

---

## **3 API Documentation**

### **3.1 Usage of Function Interface**

To use the standard API-function interface in non-managed applications, the tdrv019api.dll must be used.

## 3.2 General Functions

### 3.2.1 tdrv019Open

#### NAME

tdrv019Open – Opens a Device

#### SYNOPSIS

```
TDRV019_HANDLE tdrv019Open
(
    char      *DeviceName
)
```

#### DESCRIPTION

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

#### PARAMETERS

*DeviceName*

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

#### EXAMPLE

```
#include "tdrv019api.h"

TDRV019_HANDLE    hdl;

/*
** open file descriptor to device
*/
hdl = tdrv019Open("\\\\.\\TDRV019_1");
if (hdl == NULL)
{
    /* handle open error */
}
```

## **RETURNS**

A device handle, or NULL if the function fails. To get extended error information, call ***GetLastError***.

## **ERROR CODES**

All error codes are standard error codes set by the I/O system.



## 3.2.2 tdrv019Close

### NAME

tdrv019Close – Closes a Device

### SYNOPSIS

```
TDRV019_STATUS tdrv019Close
(
    TDRV019_HANDLE    hdl
)
```

### DESCRIPTION

This function closes previously opened devices.

### PARAMETERS

*hdl*

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

### EXAMPLE

```
#include "tdrv019api.h"

TDRV019_HANDLE hdl;
TDRV019_STATUS result;

/*
** close file descriptor to device
*/
result = tdrv019Close(hdl);
if (result != TDRV019_OK)
{
    /* handle close error */
}
```

## **RETURNS**

On success TDRV019\_OK, or an appropriate error code.

## **ERROR CODES**

All error codes are standard error codes set by the I/O system.

### 3.2.3 tdrv019GetPciInfo

#### NAME

tdrv019GetPciInfo – get information of the module PCI header

#### SYNOPSIS

```
TDRV019_STATUS tdrv019GetPciInfo
(
    TDRV019_HANDLE          hdl,
    TDRV019_PCIINFO_BUF    *pPciInfoBuf
)
```

#### DESCRIPTION

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

#### PARAMETERS

*hdl*

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

*pPciInfoBuf*

This argument is a pointer to the structure TDRV019\_PCIINFO\_BUF that receives information of the module PCI header.

```
typedef struct
{
    unsigned short    vendorId;
    unsigned short    deviceId;
    unsigned short    subSystemId;
    unsigned short    subSystemVendorId;
    int               pciBusNo;
    int               pciDevNo;
    int               pciFuncNo;
} TDRV019_PCIINFO_BUF;
```

*vendorId*

PCI module vendor ID

*deviceId*

PCI module device ID

*subSystemId*  
PCI module sub system ID

*subSystemVendorId*  
PCI module sub system vendor ID

*pciBusNo*  
Number of the PCI bus, where the module resides.

*pciDevNo*  
PCI device number

*pciFuncNo*  
PCI function number

## EXAMPLE

```
#include "tdrv019api.h"

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
TDRV019_PCIINFO_BUF pciInfoBuf

/*
** get module PCI information
*/
result = tdrv019GetPciInfo( hdl, &pciInfoBuf );

if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

## 3.2.4 tdrv019GetBoardInfo

### NAME

tdrv019GetBoardInfo – get information about the module

### SYNOPSIS

```
TDRV019_STATUS tdrv019GetBoardInfo
(
    TDRV019_HANDLE          hdl,
    TDRV019_BOARDINFO_BUF  *pBoardInfoBuf
)
```

### DESCRIPTION

This function returns information about the module.

### PARAMETERS

*hdl*

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

*pBoardInfoBuf*

This argument is a pointer to the structure TDRV019\_BOARDINFO\_BUF that receives board information.

```
typedef struct
{
    char          ModuleName[20];
    unsigned int  ModuleVariant;
    unsigned int  NumAdcChannels;
    unsigned int  NumDacChannels;
    unsigned int  NumDioLines;
    unsigned int  FirmwareVersion;
    unsigned int  AdcFifoFillLevel;
    unsigned int  DacFifoFillLevel;
    unsigned int  SerialNumber;
} TDRV019_BOARDINFO_BUF;
```

*ModuleName*

This value returns the module name as a null-terminated string. For TPMC532-10, the returned value is "TPMC532-10".

*ModuleVariant*

This value returns the module variant. For TPMC532-10R, the returned value is 10.

*NumAdcChannels*

This value returns the number of available ADC channels.

*NumDacChannels*

This value returns the number of available DAC channels.

*NumDioLines*

This value returns the number of available Digital I/O Lines.

*FirmwareVersion*

This value returns the firmware version of the hardware module.

*AdcFifoFillLevel*

This value returns the fill level of the ADC FIFO. This value can be used for debugging purposes.

*DacFifoFillLevel*

This value returns the fill level of the DAC FIFO. This value can be used for debugging purposes.

*SerialNumber*

This value returns the serial number of the hardware module.

## EXAMPLE

```
#include "tdrv019api.h"

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
TDRV019_BOARDINFO_BUF boardInfo;

...
```

```
...

/*
** get module information
*/
result = tdrv019GetBoardInfo(hdl, &boardInfo);
if (result == TDRV019_OK)
{
    printf("Hardware Module:  %s\n", boardInfo.ModuleName);
    printf("Firmware Version: %08X\n", boardInfo.FirmwareVersion);
    printf("Serial Number:    %d\n", boardInfo.SerialNumber);
}
else
{
    /* handle error */
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.2.5 tdrv019GetBoardTemperature

#### NAME

tdrv019GetBoardTemperature – Read a value from the onboard temperature sensor

#### SYNOPSIS

```
TDRV019_STATUS tdrv019GetBoardTemperature  
(  
    TDRV019_HANDLE    hdl,  
    double             *pTemperatureValue  
)
```

#### DESCRIPTION

This function reads the current temperature from the onboard temperature sensor.

#### PARAMETERS

*hdl*

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

*pTemperatureValue*

This parameter specifies a pointer to a double floating point buffer, where the temperature value is returned in °C.



## EXAMPLE

```
#include "tdrv019api.h"

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
double            TemperatureValue;

/*
** Read current board temperature
*/
result = tdrv019GetBoardTemperature( hdl, &TemperatureValue );
if (result == TDRV019_OK)
{
    Printf("Board Temperature: %5.2f °C\n", TemperatureValue);
}
else
{
    /* handle error */
}
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	Invalid parameter

## 3.3 ADC Functions

### 3.3.1 tdrv019AdcSetCorrectionValues

#### NAME

tdrv019AdcSetCorrectionValues – Set the correction data for a specific ADC channel

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSetCorrectionValues
(
    TDRV019_HANDLE    hdl,
    int                Channel,
    int                GainCorr,
    int                OffsetCorr
)
```

#### DESCRIPTION

This function enables the internal data correction for a specific ADC channel, using the specified correction values. Factory calibration data are used automatically after configuring the input range. The correction is enabled by default.

#### PARAMETERS

##### *hdl*

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

##### *Channel*

This argument specifies the channel. Valid values are 1 to the number of available channels.

##### *GainCorr*

This argument specifies the gain correction value which shall be used for the internal data correction. The value is specified in  $\frac{1}{4}$  LSB (refer to the calculation formula in the corresponding hardware user manual).

##### *OffsetCorr*

This argument specifies the offset correction value which shall be used for the internal data correction. The value is specified in  $\frac{1}{4}$  LSB (refer to the calculation formula in the corresponding hardware user manual).

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
int               GainCorr;
int               OffsetCorr;

/*
** Set Correction values for ADC channel 1
*/
GainCorr          = 12;
OffsetCorr        = 34;

result = tdrv019AdcSetCorrectionValues(hdl, 1, GainCorr, OffsetCorr);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

### 3.3.2 tdrv019AdcGetCorrectionValues

#### NAME

tdrv019AdcGetCorrectionValues – Read the Factory Calibration data for a specific ADC channel

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcGetCorrectionValues
(
    TDRV019_HANDLE    hdl,
    int                Channel,
    int                Range,
    int                *pGainCorr,
    int                *pOffsetCorr
)
```

#### DESCRIPTION

This function reads the Factory Calibration data of the specified ADC channel for the specified input range.

#### PARAMETERS

*hdl*

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

*Channel*

This argument specifies the channel. Valid values are 1 to the number of available channels.

*Range*

This argument specifies the input range. Valid values are:

Value	Description
TDRV019_INPUTRANGE_CURRENT	Currently selected input range
TDRV019_INPUTRANGE_BIPOL5V	Input Range: +/- 5 Volt
TDRV019_INPUTRANGE_BIPOL10V	Input Range: +/- 10 Volt

*pGainCorr*

This argument specifies an int pointer where the Factory Calibration value for Gain Correction will be returned.

*pOffsetCorr*

This argument specifies an int pointer where the Factory Calibration value for Offset Correction will be returned.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
int               GainCorr, OffsetCorr;

/*
** Read Factory Calibration data for ADC channel 1, input range +/- 5V
*/
result = tdrv019AdcGetCorrectionValues(hdl,
                                       1,
                                       TDRV019_INPUTRANGE_BIPOL5V,
                                       &GainCorr,
                                       &OffsetCorr);

if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

### 3.3.3 tdrv019AdcConfigInput

#### NAME

tdrv019AdcConfigInput – Configure ADC Input

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcConfigInput
(
    TDRV019_HANDLE    hdl,
    unsigned int      AdcChannels,
    int               InputRange,
    int               SampleMode,
    int               OversamplingRatio
)
```

#### DESCRIPTION

This function configures the ADC Input Range, the Sample Mode and the Oversampling Ratio for all channels of an ADC chip.

#### PARAMETERS

*hdl*

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

*AdcChannels*

This argument specifies a bitmask of the affected ADC channels. All 8 channels of one ADC chip must be configured identically. Following values are possible and may be OR'ed:

Value	Description
TDRV019_ADCCHANNELS_1_8	ADC channels 1 to 8 (ADC Chip #1)
TDRV019_ADCCHANNELS_9_16	ADC channels 9 to 16 (ADC Chip #2)
TDRV019_ADCCHANNELS_17_24	ADC channels 17 to 24 (ADC Chip #3)
TDRV019_ADCCHANNELS_25_32	ADC channels 25 to 32 (ADC Chip #4)

*InputRange*

Specifies the ADC Input Range. Following values are possible:

Value	Description
TDRV019_INPUTRANGE_BIPOL5V	Input Range: +/- 5 Volt
TDRV019_INPUTRANGE_BIPOL10V	Input Range: +/- 10 Volt

### SampleMode

This value specifies the ADC Sample Mode. Possible values are:

Value	Description
TDRV019_SAMPLEMODE_MANUAL	Sampling is started manually
TDRV019_SAMPLEMODE_SEQUENCER	Sampling is done using the Sequencer Mode.

### OversamplingRatio

This value specifies the ADC Oversampling Ratio. Possible values are :

Value	Description
TDRV019_OVERSAMPLING_NONE	No oversampling is used
TDRV019_OVERSAMPLING_X2	2-times oversampling
TDRV019_OVERSAMPLING_X4	4-times oversampling
TDRV019_OVERSAMPLING_X8	8-times oversampling
TDRV019_OVERSAMPLING_X16	16-times oversampling
TDRV019_OVERSAMPLING_X32	32-times oversampling
TDRV019_OVERSAMPLING_X64	64-times oversampling

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE hdl;
TDRV019_STATUS result;

/*
** Configure ADC 1-16 for +/-10V, Sample with Sequencer, 16times
** Oversampling
*/
result = tdrv019AdcConfigInput( hdl,
                                TDRV019_ADCCHANNELS_1_8 | TDRV019_ADCCHANNELS_9_16,
                                TDRV019_INPUTRANGE_BIPOL10V,
                                TDRV019_SAMPLEMODE_SEQUENCER,
                                TDRV019_OVERSAMPLING_X16 );

if (result != TDRV019_OK)
{
    /* handle error */
}
```

---

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_CHANNEL	The requested channels are not supported by the hardware module.
TDRV019_ERR_INVALID	At least one of the specified parameters is invalid.



### 3.3.4 tdrv019AdcConfigSequencer

#### NAME

tdrv019AdcConfigSequencer – Configure the ADC Sequencer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcConfigSequencer
(
    TDRV019_HANDLE    hdl,
    int               SequencerMode,
    int               ConversionClockSource,
    int               NumConversions
)
```

#### DESCRIPTION

This function configures the ADC Sequencer.

#### PARAMETERS

*hdl*

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

*SequencerMode*

This value specifies the Input Mode of the sequencer. Valid values are:

Value	Description
TDRV019_SEQMODE_NORMAL	Sampling is started with the next Convert signal
TDRV019_SEQMODE_FRAME	Sampling is started with the next Frame signal

*ConversionClockSource*

This value specifies the clock source used for conversion. The corresponding clock configuration must be done separately. Valid values are:

Value	Description
TDRV019_CONVCLKSRC_CLOCK1	Convert signal based on Conversion Clock 1
TDRV019_CONVCLKSRC_CLOCK2	Convert signal based on Conversion Clock 2

*NumConversions*

This value specifies the number of conversions per data set. Specify '0' for continuous conversion.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Configure ADC Sequencer:
** Use Frame Mode, 50 samples per data set, Conversion Clock 1
*/
result = tdrv019AdcConfigSequencer( hdl,
                                     TDRV019_SEQMODE_FRAME,
                                     TDRV019_CONVCLKSRC_CLOCK1,
                                     50 );
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	Invalid parameter.

### 3.3.5 tdrv019AdcManualSingleValue

#### NAME

tdrv019AdcManualSingleValue – Read a single ADC channel

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcManualSingleValue
(
    TDRV019_HANDLE    hdl,
    int                Channel,
    int                Mode,
    int                *pAdcValue
)
```

#### DESCRIPTION

This function reads an ADC value of a specific channel. Depending on the specified flags, the function initiates the conversion and waits for it to finish.

#### PARAMETERS

##### *hdl*

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

##### *Channel*

This value specifies the desired ADC channel. Valid values are 1 up to the number of available channels. The number of supported channels depends on the used module and variant.

##### *Mode*

This value specifies the data acquisition mode. The following modes are possible:

Value	Description
TDRV019_ADCMODE_CONVREAD	The ADC conversion is started, and the data register is read after the conversion has finished.
TDRV019_ADCMODE_READCONV	The ADC data register is read, returning the data of a previous conversion. A new conversion is started afterwards (for later readout).
TDRV019_ADCMODE_READ	The ADC data register is read without starting a conversion. Useful in sequencer mode.

##### *pAdcValue*

This parameter specifies a pointer to an int buffer, where the ADC value is returned. Calculating the resulting voltage has to be done by the application.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
int                 AdcValue;
int                 Mode;

/*
** Start an ADC conversion on Channel 1, and wait for data
*/
Mode = TDRV019_ADCMODE_CONVERTREAD;

result = tdrv019AdcManualSingleValue( hdl, 1, Mode, &AdcValue );
if (result == TDRV019_OK)
{
    /*
    ** calculate the voltage out of the AdcValue,
    ** based on the ADC configuration.
    */
}
else
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	Invalid parameter
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

### 3.3.6 tdrv019AdcManualSingleSample

#### NAME

tdrv019AdcManualSingleSample – Read a single ADC Sample from all available channels

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcManualSingleSample
(
    TDRV019_HANDLE    hdl,
    unsigned int      ChannelMask,
    int               Mode,
    int               *pAdcBuf
)
```

#### DESCRIPTION

This function reads ADC values of specified channels. Depending on the specified flags, the function initiates the conversion and waits for it to finish.

#### PARAMETERS

*hdl*

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

*ChannelMask*

This value specifies the desired ADC channels using a bitmask. Bit 0 refers to channel 1, bit 1 refers to channel 2 and so on. The number of supported channels depends on the used module and variant.

*Mode*

This value specifies the data acquisition mode. The following modes are possible:

Value	Description
TDRV019_ADCMODE_CONVREAD	The ADC conversion is started, and the data register is read after the conversion has finished.
TDRV019_ADCMODE_READCONV	The ADC data register is read, returning the data of a previous conversion. A new conversion is started afterwards (for later readout).
TDRV019_ADCMODE_READ	The ADC data register is read without starting a conversion. Useful in sequencer mode.

*pAdcBuf*

This parameter specifies a pointer to an int data array, where the ADC values are returned. The buffer must be large enough to receive up to 32 ADC values.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
unsigned int         ChannelMask;
int                  AdcBuf[32];
int                  Mode;

/*
** Read ADC Channels 1, 2 and 16.
*/
ChannelMask = ((1 << 15) | (1 << 1) | (1 << 0));
Mode = TDRV019_ADCMODE_CONVREAD;

result = tdrv019AdcManualSingleSample( hdl, ChannelMask, Mode, AdcBuf );
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid parameter.
TDRV019_ERR_CHANNEL	At least one of the requested channels is not supported by the hardware module.

### 3.3.7 tdrv019AdcSequencerSampleblock

#### NAME

tdrv019AdcSequencerSampleblock – Read a sample block of ADC data

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSequencerSampleblock
(
    TDRV019_HANDLE    hdl,
    signed short      *pAdcBuf,
    size_t            numBytes,
    size_t            *validBytes,
    unsigned int      *pStatus
)
```

#### DESCRIPTION

This function reads a number of ADC samples. It returns either a complete data set, or fills the buffer with valid data. This function blocks until data is available or the configured timeout occurred (see function `tdrv019AdcSequencerTimeoutSet`). In case of a FIFO overflow condition, the function returns with the remaining data. To enable the data sampling again after an overflow condition, execute function `tdrv019AdcSequencerStart`.

#### PARAMETERS

*hdl*

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

*pAdcBuf*

This value specifies a pointer to a signed short data buffer, where the sample block will be stored.

Depending on the number of selected channels, the data will be stored as follows:

16 Channels:

Index \ Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#9	Ch#10	Ch#11	Ch#12	Ch#13	Ch#14	Ch#15	Ch#16
0x20	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x30	Ch#9	Ch#10	Ch#11	Ch#12	Ch#13	Ch#14	Ch#15	...

**8 Channels:**

Index Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	...

*numBytes*

This value specifies the size of the data buffer in bytes. This value should be at least the size of a data set (if used), and must be a multiple of samples (Number of Sequencer Channels x 16bit).

*validBytes*

This parameter specifies a pointer where the number of valid data bytes is returned. If this value does not match numBytes, a data set has been stored or a FIFO error has occurred.

*pStatus*

This parameter specifies a pointer for the status of the corresponding data buffer. Possible values are:

Value	Description
TDRV019_SEQSTATUS_FIFOERROR	A FIFO overflow has occurred. The data buffer contains the remaining valid ADC data.
TDRV019_SEQSTATUS_NOTRUNNING	The ADC Sequencer is not running.

**EXAMPLE**

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
signed short      *pAdcBlock;
size_t            AdcBlockSize;
size_t            validBytes;
unsigned int      Status;

...
```



```

...

/*
** Read ADC Sample Block, wait up to 1 second
*/

/* allocate memory */
AdcBlockSize = ...;
pAdcBlock = (signed short*)malloc( AdcBlockSize );

result = tdrv019AdcSequencerSampleblock(hdl,
                                         pAdcBlock,
                                         AdcBlockSize,
                                         &validBytes,
                                         &Status);

if (result != TDRV019_OK)
{
    /* handle error */
}
free( pAdcBlock );

```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified TDRV019_HANDLE is invalid.
TDRV019_ERR_INVAL	Invalid parameter.
TDRV019_ERR_BUSY	There is already a function waiting for a sample block.
TDRV019_ERR_TIMEOUT	Timeout occurred.
TDRV019_ERR_FIFOERROR	FIFO Overflow. Data acquisition has been stopped.

### 3.3.8 tdrv019AdcFifoFlush

#### NAME

tdrv019AdcFifoFlush – Clears the ADC FIFO

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcDmaFlush  
(  
    TDRV019_HANDLE    hdl  
)
```

#### DESCRIPTION

This function clears the ADC FIFO. All previously received data is discarded. Using this function should be used after a FIFO overflow.

#### PARAMETERS

*hdl*

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

#### EXAMPLE

```
#include <tdrv019api.h>  
  
TDRV019_HANDLE    hdl;  
TDRV019_STATUS    result;  
  
/*  
** Flush ADC Data  
*/  
result = tdrv019AdcFifoFlush(hdl);  
if (result != TDRV019_OK)  
{  
    /* handle error */  
}
```

---

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

### 3.3.9 tdrv019AdcSequencerStart

#### NAME

tdrv019AdcSequencerStart – Starts the ADC Sequencer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSequencerStart
(
    TDRV019_HANDLE    hdl
)
```

#### DESCRIPTION

This function starts the ADC Sequencer. The sequencer must be configured before executing this function. ADC data is stored in the internal FIFO, and can be read using function `tdrv019AdcSequencerSampleblock`.

#### PARAMETERS

*hdl*

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

#### EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Start ADC Sequencer
*/
result = tdrv019AdcSequencerStart(hdl);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

### 3.3.10 tdrv019AdcSequencerStop

#### NAME

tdrv019AdcSequencerStop – Stops the ADC Sequencer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSequencerStop  
(  
    TDRV019_HANDLE    hdl  
)
```

#### DESCRIPTION

This function stops the ADC Sequencer.

#### PARAMETERS

*hdl*

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

#### EXAMPLE

```
#include <tdrv019api.h>  
  
TDRV019_HANDLE    hdl;  
TDRV019_STATUS    result;  
  
/*  
** Stop ADC Sequencer  
*/  
result = tdrv019AdcSequencerStop(hdl);  
if (result != TDRV019_OK)  
{  
    /* handle error */  
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

### 3.3.11 tdrv019AdcSequencerTimeoutSet

#### NAME

tdrv019AdcSequencerTimeoutSet – Configure the ADC Sequencer Timeout

#### SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSequencerTimeoutSet
(
    TDRV019_HANDLE    hdl,
    int               timeout_ms
)
```

#### DESCRIPTION

This function configures the timeout used for an ADC Sequencer read request.

#### PARAMETERS

*hdl*

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

*timeout*

This value specifies the timeout in milliseconds the function will wait for ADC data to arrive. The granularity depends on the operating system. To wait indefinitely, specify -1 as timeout parameter.

#### EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Configure ADC Sequencer Timeout for 5 seconds
*/
result = tdrv019AdcSequencerTimeoutSet(hdl, 5000);
if (result != TDRV019_OK)
{
    /* handle error */
}
```



## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

## 3.4 DAC Functions

### 3.4.1 tdrv019DacSetCorrectionValues

#### NAME

tdrv019DacSetCorrectionValues – Set the correction data for a specific DAC channel

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacSetCorrectionValues
(
    TDRV019_HANDLE    hdl,
    int                Channel,
    int                GainCorr,
    int                OffsetCorr
)
```

#### DESCRIPTION

This function enables the internal data correction for a specific DAC channel, using the specified correction values. Factory calibration data are used automatically after configuring the output range. The correction is enabled by default.

#### PARAMETERS

##### *hdl*

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

##### *Channel*

This argument specifies the channel. Valid values are 1 to the number of available channels.

##### *GainCorr*

This argument specifies the gain correction value which shall be used for the internal data correction. The value is specified in  $\frac{1}{4}$  LSB (refer to the calculation formula in the corresponding hardware user manual).

##### *OffsetCorr*

This argument specifies the offset correction value which shall be used for the internal data correction. The value is specified in  $\frac{1}{4}$  LSB (refer to the calculation formula in the corresponding hardware user manual).

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
int               GainCorr;
int               OffsetCorr;

/*
** Set Correction values for DAC channel 1
*/
GainCorr          = 12;
OffsetCorr        = 34;

result = tdrv019DacSetCorrectionValues(hdl, 1, GainCorr, OffsetCorr);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

### 3.4.2 tdrv019DacGetCorrectionValues

#### NAME

tdrv019DacGetCorrectionValues – Read the Factory Calibration data for a specific DAC channel

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacCorrectionDisable  
(  
    TDRV019_HANDLE    hdl,  
    int               Channel,  
    int               Range,  
    int               *pGainCorr,  
    int               *pOffsetCorr  
)
```

#### DESCRIPTION

This function reads the Factory Calibration data of the specified DAC channel for the specified output range. The supported output ranges depend on the hardware module.

#### PARAMETERS

*hdl*

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

*Channel*

This argument specifies the channel. Valid values are 1 to the number of available channels.

### Range

This argument specifies the output range. Valid values are:

Value	Description
TDRV019_OUTPUTRANGE_CURRENT	Currently selected output range
TDRV019_OUTPUTRANGE_UNIPOL5V	Output Range: 0-5 Volt
TDRV019_OUTPUTRANGE_UNIPOL6V	Output Range: 0-6 Volt
TDRV019_OUTPUTRANGE_UNIPOL10V	Output Range: 0-10 Volt
TDRV019_OUTPUTRANGE_UNIPOL10P8V	Output Range: 0-10.8 Volt
TDRV019_OUTPUTRANGE_UNIPOL12V	Output Range: 0-12 Volt
TDRV019_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV019_OUTPUTRANGE_BIPOL6V	Output Range: +/- 6 Volt
TDRV019_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV019_OUTPUTRANGE_BIPOL10P8V	Output Range: +/- 10.8 Volt
TDRV019_OUTPUTRANGE_BIPOL12V	Output Range: +/- 12 Volt

### *pGainCorr*

This argument specifies an int pointer where the Factory Calibration value for Gain Correction will be returned.

### *pOffsetCorr*

This argument specifies an int pointer where the Factory Calibration value for Offset Correction will be returned.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
int               GainCorr, OffsetCorr;

/*
** Read Factory Calibration data for DAC channel 1, output range +/- 5V
*/
result = tdrv019DacGetCorrectionValues(hdl,
                                       1,
                                       TDRV019_OUTPUTRANGE_BIPOL5V,
                                       &GainCorr,
                                       &OffsetCorr);

if (result != TDRV019_OK)
{
    /* handle error */
}
```

---

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

### 3.4.3 tdrv019DacConfigOutput

#### NAME

tdrv019DacConfigOutput – Configure DAC Output

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacConfigOutput
(
    TDRV019_HANDLE    hdl,
    unsigned int      DacChannels,
    int               OutputRange,
    int               SampleMode
)
```

#### DESCRIPTION

This function configures the DAC Output Range and the Sample Mode for all channels of a DAC chip. The number of DAC channels and the supported output ranges depend on the hardware module.

#### PARAMETERS

*hdl*

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

*DacChannels*

This argument specifies a bitmask of the affected DAC channels. All 4 DAC channels of one DAC chip must be configured identically. Following values are possible and may be OR'ed:

Value	Description
TDRV019_DACCHANNELS_1_4	DAC channels 1 to 4 (DAC Chip #1)
TDRV019_DACCHANNELS_5_8	DAC channels 5 to 8 (DAC Chip #2)
TDRV019_DACCHANNELS_9_12	DAC channels 9 to 12 (DAC Chip #3)
TDRV019_DACCHANNELS_13_16	DAC channels 13 to 16 (DAC Chip #4)
TDRV019_DACCHANNELS_17_20	DAC channels 17 to 20 (ADC Chip #5)
TDRV019_DACCHANNELS_21_24	DAC channels 21 to 24 (ADC Chip #6)
TDRV019_DACCHANNELS_25_28	DAC channels 25 to 28 (ADC Chip #7)
TDRV019_DACCHANNELS_29_32	DAC channels 29 to 32 (ADC Chip #8)

### OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV019_OUTPUTRANGE_UNIPOL5V	Output Range: 0-5 Volt
TDRV019_OUTPUTRANGE_UNIPOL6V	Output Range: 0-6 Volt
TDRV019_OUTPUTRANGE_UNIPOL10V	Output Range: 0-10 Volt
TDRV019_OUTPUTRANGE_UNIPOL10P8V	Output Range: 0-10.8 Volt
TDRV019_OUTPUTRANGE_UNIPOL12V	Output Range: 0-12 Volt
TDRV019_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV019_OUTPUTRANGE_BIPOL6V	Output Range: +/- 6 Volt
TDRV019_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV019_OUTPUTRANGE_BIPOL10P8V	Output Range: +/- 10.8 Volt
TDRV019_OUTPUTRANGE_BIPOL12V	Output Range: +/- 12 Volt
TDRV019_OUTPUTRANGE_0MA_20MA	Output Range: 0mA to 20mA Current
TDRV019_OUTPUTRANGE_0MA_24MA	Output Range: 0mA to 24mA Current
TDRV019_OUTPUTRANGE_4MA_20MA	Output Range: 4mA to 20mA Current

### SampleMode

This value specifies the DAC Sample Mode. Possible values are:

Value	Description
TDRV019_SAMPLEMODE_MANUAL	Sampling is started manually
TDRV019_SAMPLEMODE_SEQUENCER	Sampling is done using the Sequencer Mode

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Configure the DAC for +/-5V, Sample with Sequencer
*/
result = tdrv019DacConfigOutput( hdl,
                                TDRV019_OUTPUTRANGE_BIPOL5V,
                                TDRV019_SAMPLEMODE_SEQUENCER );

if (result != TDRV019_OK)
{
    /* handle error */
}
```



---

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	At least one of the specified parameters is invalid.
TDRV019_ERR_NOTSUP	Channel or output range not supported.

### 3.4.4 tdrv019DacConfigOutputRange

#### NAME

tdrv019DacConfigOutputRange – Configure DAC Output Range

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacConfigOutputRange  
(  
    TDRV019_HANDLE    hdl,  
    int               Channel,  
    int               OutputRange  
)
```

#### DESCRIPTION

This function configures the DAC Output Range of an individual DAC channel. The number of DAC channels and the supported output ranges depend on the hardware module.

#### PARAMETERS

*hdl*

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

*Channel*

This argument specifies the desired DAC channel. Possible values are 1 to the available number of channels.

## OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV019_OUTPUTRANGE_UNIPOL5V	Output Range: 0-5 Volt
TDRV019_OUTPUTRANGE_UNIPOL6V	Output Range: 0-6 Volt
TDRV019_OUTPUTRANGE_UNIPOL10V	Output Range: 0-10 Volt
TDRV019_OUTPUTRANGE_UNIPOL10P8V	Output Range: 0-10.8 Volt
TDRV019_OUTPUTRANGE_UNIPOL12V	Output Range: 0-12 Volt
TDRV019_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV019_OUTPUTRANGE_BIPOL6V	Output Range: +/- 6 Volt
TDRV019_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV019_OUTPUTRANGE_BIPOL10P8V	Output Range: +/- 10.8 Volt
TDRV019_OUTPUTRANGE_BIPOL12V	Output Range: +/- 12 Volt
TDRV019_OUTPUTRANGE_0MA_20MA	Output Range: 0mA to 20mA Current
TDRV019_OUTPUTRANGE_0MA_24MA	Output Range: 0mA to 24mA Current
TDRV019_OUTPUTRANGE_4MA_20MA	Output Range: 4mA to 20mA Current

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Configure the output range of DAC channel 10 to 0-10V
*/
result = tdrv019DacConfigOutputRange( hdl,
                                       10,
                                       TDRV019_OUTPUTRANGE_UNIPOL10V );

if (result != TPDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

---

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	At least one of the specified parameters is invalid.
TDRV019_ERR_NOTSUP	Channel or output range not supported.

### 3.4.5 tdrv019DacConfigSequencer

#### NAME

tdrv019DacConfigSequencer – Configure the DAC Sequencer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacConfigSequencer
(
    TDRV019_HANDLE    hdl,
    int               SequencerMode,
    int               ConversionClockSource,
    int               NumConversions
)
```

#### DESCRIPTION

This function configures the DAC Sequencer.

#### PARAMETERS

*hdl*

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

*SequencerMode*

This value specifies the Input Mode of the sequencer. Valid values are:

Value	Description
TDRV019_SEQMODE_NORMAL	Sampling is started with the next Convert signal
TDRV019_SEQMODE_FRAME	Sampling is started with the next Frame signal

*ConversionClockSource*

This value specifies the clock source used for conversion. The corresponding clock configuration must be done separately. Valid values are:

Value	Description
TDRV019_CONVCLKSRC_CLOCK1	Convert signal based on Conversion Clock 1
TDRV019_CONVCLKSRC_CLOCK2	Convert signal based on Conversion Clock 2

*NumConversions*

This value specifies the number of conversions per data set.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Configure DAC Sequencer:
** Use Frame Mode, 50 samples per data set, Conversion Clock 2
*/
result = tdrv019DacConfigSequencer( hdl,
                                     TDRV019_SEQMODE_FRAME,
                                     TDRV019_CONVCLKSRC_CLOCK2,
                                     50 );
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	Invalid parameter.

### 3.4.6 tdrv019DacManualSingleValue

#### NAME

tdrv019DacManualSingleValue – Write a single DAC channel

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacManualSingleValue
(
    TDRV019_HANDLE    hdl,
    int               Channel,
    int               Flags,
    int               DacValue
)
```

#### DESCRIPTION

This function writes a DAC output value to a specific channel. Depending on the specified flags, the function initiates loading of the DAC output, resulting in the actual output of all DAC channels.

This function is only available if the DAC channel is configured for Manual Sampling mode.

#### PARAMETERS

##### *hdl*

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

##### *Channel*

This value specifies the desired DAC channel. Valid values are 1 to the number of supported channels.

##### *Flags*

This value specifies flags regarding the data acquisition. The following flags are possible:

Value	Description
TDRV019_DACFLAG_LOAD	If the DACs are configured for Manual Sampling Mode, output loading is initiated. If flag is not set, the function updates the data register value without affecting the actual output.

##### *DacValue*

This parameter specifies the new DAC value which shall be written to the specified channel.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
int               DacValue;
int               Flags;

/*
** Write to DAC Channel 4, initiate actual output of all DAC channels
*/
Flags    = TDRV019_DACFLAG_LOAD;
DacValue = 0x7FFF;

result = tdrv019DacManualSingleValue( hdl, 4, Flags, DacValue );
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid parameter.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.



### 3.4.7 tdrv019DacManualSingleSample

#### NAME

tdrv019DacManualSingleSample – Writes a single DAC Sample to specified channels

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacManualSingleSample
(
    TDRV019_HANDLE    hdl,
    unsigned int      ChannelMask,
    int                Flags,
    int                *pDacBuf
)
```

#### DESCRIPTION

This function writes DAC values to the specified channels. Depending on the specified flags, the function initiates the output loading.

This function is only available if the desired DAC channels are configured for Manual Sampling mode.

#### PARAMETERS

##### *hdl*

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

##### *ChannelMask*

This value specifies the desired DAC channels using a bitmask. Bit 0 refers to channel 1, bit 1 refers to channel 2 and so on. The number of supported channels depends on the used module and variant.

##### *Flags*

This value specifies flags regarding the data acquisition. The following flags are possible:

Value	Description
TDRV019_DACFLAG_LOAD	If the DACs are configured for Manual Sampling Mode, output loading is initiated. If flag is not set, the function updates the data register values without affecting the actual output.

##### *pDacBuf*

This parameter specifies a pointer to an int data array, where the DAC values are stored. The buffer must be large enough to hold up to 32 DAC values.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
unsigned int         ChannelMask;
int                 DacBuf[32];
int                 Flags;

/*
** Write DAC Channels 1, 2 and 4. Update the outputs.
*/
ChannelMask = ((1 << 3) | (1 << 1) | (1 << 0));
Flags = TDRV019_DACFLAG_LOAD;
DacBuf[0] = ...;
DacBuf[1] = ...;
DacBuf[3] = ...;

result = tdrv019DacManualSingleSample( hdl, ChannelMask, Flags, DacBuf );
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	Invalid parameter.
TDRV019_ERR_CHANNEL	At least one of the requested channels is not supported by the hardware module.

### 3.4.8 tdrv019DacSequencerSampleblock

#### NAME

tdrv019DacSequencerSampleblock – Write a sample block of DAC data

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacSequencerSampleblock
(
    TDRV019_HANDLE    hdl,
    signed short      *pDacBuf,
    size_t             numBytes
)
```

#### DESCRIPTION

This function writes a sample block of DAC data to the configured sequencer outputs. The data is transferred into the driver's FIFO and into the hardware FIFO for output.

#### PARAMETERS

*hdl*

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

*pDacBuf*

This value specifies a pointer to a signed short data buffer containing the DAC data. Depending on the number of selected channels, the data must be stored as follows:

16 Channels:

Index \ Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#9	Ch#10	Ch#11	Ch#12	Ch#13	Ch#14	Ch#15	Ch#16
0x20	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	...

12 Channels:

Index \ Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#9	Ch#10	Ch#11	Ch#12	Ch#1	Ch#2	Ch#3	...

#### 8 Channels:

Index \ Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	...

#### 4 Channels:

Index \ Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#1	Ch#2	Ch#3	Ch#4
0x10	Ch#1	Ch#2	Ch#3	Ch#4	Ch#1	Ch#2	Ch#3	...

#### *numBytes*

This value specifies the size of the data buffer in bytes. This value must be a multiple of complete channel sets (samples).

### EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
signed short      *pDacBuf;
size_t            DacBlockSize;

/*
** Write DAC sample block
*/

/* allocate memory */
DacBlockSize = ...;
pDacBuf = (signed short*)malloc( DacBlockSize );
/* fill data memory */
...
result = tdrv019DacSequencerSampleblock(hdl, pDacBuf, DacBlockSize);
if (result != TDRV019_OK)
{
    /* handle error */
}
free( pDacBuf );
```

### RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

---

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVALID	Invalid parameter.
TDRV019_ERR_NOMEM	Error allocating memory

### 3.4.9 tdrv019DacFifoFlush

#### NAME

tdrv019DacFifoFlush – Clears the DAC FIFO

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacFifoFlush
(
    TDRV019_HANDLE    hdl
)
```

#### DESCRIPTION

This function clears the DAC FIFO. All previously loaded data is discarded. Using this function should be used after a FIFO underflow.

#### PARAMETERS

*hdl*

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

#### EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Flush DAC Data
*/
result = tdrv019DacFifoFlush(hdl);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

### 3.4.10 tdrv019DacSequencerStart

#### NAME

tdrv019DacSequencerStart – Starts the DAC Sequencer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacSequencerStart  
(  
    TDRV019_HANDLE    hdl  
)
```

#### DESCRIPTION

This function starts the DAC Sequencer. The sequencer must be configured before executing this function. DAC data has to be written using function `tdrv019DacSequencerSampleblock`.

#### PARAMETERS

*hdl*

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

#### EXAMPLE

```
#include <tdrv019api.h>  
  
TDRV019_HANDLE    hdl;  
TDRV019_STATUS    result;  
  
/*  
** Start DAC Sequencer  
*/  
result = tdrv019DacSequencerStart(hdl);  
if (result != TDRV019_OK)  
{  
    /* handle error */  
}
```



---

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

### 3.4.11 tdrv019DacSequencerStop

#### NAME

tdrv019DacSequencerStop – Stops the DAC Sequencer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DacSequencerStop
(
    TDRV019_HANDLE    hdl
)
```

#### DESCRIPTION

This function stops the DAC Sequencer. No data is written to the DAC outputs anymore, even if data is available within the internal FIFOs.

#### PARAMETERS

*hdl*

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

#### EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Stop DAC Sequencer
*/
result = tdrv019DacSequencerStop(hdl);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

---

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

---

## 3.5 Digital I/O Functions

### 3.5.1 tdrv019DioRead

#### NAME

tdrv019DioRead – read current input value of the I/O lines

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioRead
(
    TDRV019_HANDLE    hdl,
    unsigned int      *input
)
```

#### DESCRIPTION

This function reads the current input value of the I/O lines.

#### PARAMETER

*hdl*

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

*input*

This argument points to a buffer where the current value of the I/O lines will be returned. Bit 0 returns the value of the first I/O line, bit 1 the value of the second I/O line, and so on. Only available I/O lines will return valid values. The number of available I/O lines depends on the used module type and variant.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
unsigned int       input;

/*
** read current state of I/O lines
*/
result = tdrv019Read(hdl, &input);
if (result != TDRV019_OK)
{
    /* handle error */
}
else
{
    printf("INPUT: 0x%08X\n", input);
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid
TDRV019_ERR_INVAL	A NULL pointer is referenced for an input value

## 3.5.2 tdrv019DioWrite

### NAME

tdrv019DioWrite – set output value

### SYNOPSIS

```
TDRV019_STATUS tdrv019DioWrite  
(  
    TDRV019_HANDLE    hdl,  
    unsigned int      output  
)
```

### DESCRIPTION

This function sets the output value.

**The specified value will only appear on the I/O lines which are configured for output.**

### PARAMETER

*hdl*

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

*output*

This argument specifies the output value for I/O. Bit 0 specifies the value of the first I/O line, bit 1 the value of the second I/O line, and so on. The number of available I/O lines depends on the used module type and variant.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Set output value (set first eighth I/O lines), clear all others
*/
result = tdrv019DioWrite(hdl, 0x000000FF);
if (result != TDRV019_OK)
{
    /* error handling */
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid

### 3.5.3 tdrv019DioWriteMasked

#### NAME

tdrv019DioWriteMasked – set output value for specified I/O lines

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioWriteMasked
(
    TDRV019_HANDLE    hdl,
    unsigned int      output,
    unsigned int      mask
)
```

#### DESCRIPTION

This function sets the output value for specified I/O lines. The mask specifies which I/O bits shall be set to the specified output value and which shall keep the current value.

**This specified value will only appear on the I/O lines which are configured for output.**

#### PARAMETER

*hdl*

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

*output*

This argument specifies the output value for I/O lines. Bit 0 specifies the value of the first I/O line, bit 1 the value of second I/O line, and so on.

*mask*

This argument specifies the output mask for output lines. Bit 0 specifies the mask for the first I/O line, bit 1 the value for the second I/O line, and so on.

A set bit (1) means the bit shall be set to the value specified by *output*.

A reset bit (0) means that the old output value will not be changed.



## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Set a part of the output value (set/reset first four I/O lines)
*/
result = tdrv019DioWriteMasked(hdl, 0x12345678, 0x0000000F);
if (result != TDRV019_OK)
{
    /* error handling */
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid

### 3.5.4 tdrv019DioSetOutputLine

#### NAME

tdrv019DioSetOutputLine – set a specified output line

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioSetOutputLine  
(  
    TDRV019_HANDLE    hdl,  
    int               outputLine  
)
```

#### DESCRIPTION

This function sets a single bit of the output value.

**This specified value will only appear if the corresponding I/O line is configured for output.**

#### PARAMETER

*hdl*

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

*outputLine*

This argument specifies a data bit that shall be set. Allowed values are 1 up to the number of available I/O lines.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Set I/O line 8
*/
result = tdrv019DioSetOutputLine(hdl, 8);
if (result != TDRV019_OK)
{
    /* error handling */
}
```

## RETURN VALUE

On success, TDRV006\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid
TDRV019_ERR_INVAL	An invalid line number is specified

### 3.5.5 tdrv019DioClearOutputLine

#### NAME

tdrv019DioClearOutputLine – reset a specified I/O line

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioClearOutputLine  
(  
    TDRV019_HANDLE    hdl,  
    int                outputLine  
)
```

#### DESCRIPTION

This function resets a single bit of the output value.

**This specified value will only appear if the corresponding I/O line is configured for output.**

#### PARAMETER

*hdl*

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

*outputLine*

This argument specifies data bit that shall be reset. Allowed values are 1 up to the number of available I/O lines.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** Clear I/O line 4
*/
result = tdrv019DioClearOutputLine(hdl, 4);
if (result != TDRV019_OK)
{
    /* error handling */
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid
TDRV019_ERR_INVAL	An invalid line number is specified

### 3.5.6 tdrv019DioConfigureDirection

#### NAME

tdrv019DioConfigureDirection – set the I/O line direction

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioConfigureDirection
(
    TDRV019_HANDLE    hdl,
    unsigned int      DirectionValue,
    unsigned int      DirectionMask
)
```

#### DESCRIPTION

This function sets the I/O line direction. The value specifies which I/O lines shall be configured for output and which I/O lines should be used for input.

#### PARAMETER

*hdl*

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

*DirectionValue*

This value specifies the direction of the corresponding I/O lines. An active (1) bit will configure the corresponding I/O line to OUTPUT, an unset (0) bit will configure the corresponding I/O line to INPUT. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

*DirectionMask*

This parameter specifies the bitmask. Only active bits (1) will have an effect on the I/O direction, the direction of all other I/O lines will be left unchanged. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*
** configure new I/O direction:
** set lowest 4 I/O lines to OUTPUT, and higher 4 I/O lines to INPUT.
** leave all other I/O lines unchanged.
*/
result = tdrv012ConfigureDirection(hdl,
                                   (0xf0 << 0) | (0x0f << 0),
                                   (0x00 << 0) | (0x0f << 0));
if (result != TDRV019_OK)
{
    /* error handling */
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid

### 3.5.7 tdrv019DioDebounceConfig

#### NAME

tdrv019DioDebounceConfig – Configure digital I/O (input) debouncer

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioDebounceConfig  
(  
    TDRV019_HANDLE    hdl,  
    unsigned short    DebounceTime  
)
```

#### DESCRIPTION

This function configures the digital I/O input debouncing mechanism to avoid detection of invalid signal changes in noisy environments.

#### PARAMETERS

*hdl*

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

*DebounceTime*

Specifies the debounce time. For the filter calculation, refer to the corresponding hardware user manual.



## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*-----
   Enable Debouncer
   -----*/
result = tdrv019DioDebounceConfig(hdl, 10000 );    /* DebounceTime */
if (result == TDRV019_OK)
{
    /* function succeeded */
}
else
{
    /* handle error */
}
}
```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.5.8 tdrv019DioPullResistorConfig

#### NAME

tdrv019DioPullResistorConfig – Configure Pull Resistors for DIO and synchronization signals

#### SYNOPSIS

```
TDRV019_STATUS tdrv019DioPullResistorConfig
(
    TDRV019_HANDLE    hdl,
    unsigned int      PullFront,
    unsigned int      PullRear
)
```

#### DESCRIPTION

This function configures the pull resistors of front I/O and rear I/O.

#### PARAMETERS

*hdl*

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

*PullFront*

Specifies the pull resistor used for front I/O. The following values are possible:

Value	Description
TDRV019_DIOPULL_FLOATING	I/O lines are left floating
TDRV019_DIOPULL_5V	I/O lines are pulled to 5V
TDRV019_DIOPULL_3P3V	I/O lines are pulled to 3.3V
TDRV019_DIOPULL_GND	I/O lines are pulled to GND

*PullRear*

Specifies the pull resistor used for rear I/O. The following values are possible:

Value	Description
TDRV019_DIOPULL_FLOATING	I/O lines are left floating
TDRV019_DIOPULL_5V	I/O lines are pulled to 5V
TDRV019_DIOPULL_3P3V	I/O lines are pulled to 3.3V
TDRV019_DIOPULL_GND	I/O lines are pulled to GND

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;

/*-----
   Configure Front I/O DIO to 5V Pull-Up, Rear I/O Sync to 3.3V
   -----*/
result = tdrv019DioPullResistorConfig(hdl,
                                       TDRV019_DIOPULL_5V,
                                       TDRV019_DIOPULL_3P3V);
if (result == TDRV019_OK)
{
    /* function succeeded */
}
else
{
    /* handle error */
}

```

## RETURN VALUE

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV019_ERR_INVAL	At least one of the specified parameters is invalid

## 3.6 Interrupt Functions

### 3.6.1 tdrv019InterruptWait

#### NAME

tdrv019InterruptWait – Wait for incoming Local Interrupt Source

#### SYNOPSIS

```
TDRV019_STATUS tdrv019InterruptWait
(
    TDRV019_HANDLE    hdl,
    unsigned int      InterruptMaskArray[],
    unsigned int      InterruptOccurredArray[],
    int               timeout
)
```

#### DESCRIPTION

This function enables the specified local interrupt sources, and waits for interrupts on the specified interrupt sources. Multiple functions may wait for the same interrupt source to occur.

**The delay between an incoming interrupt and the return of the described function is system-dependent, and is most likely several microseconds.**

#### PARAMETERS

*hdl*

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

*InterruptMaskArray*

This parameter specifies specific interrupt bits to wait for. The interrupt bits of array index 0 correspond to the Interrupt Status Register, array index 1 corresponds to the Error Interrupt Status Register, and array index 2 corresponds to the DIO Interrupt Status Register. Please refer to the hardware user manual for further information on the possible interrupt bits. The function returns if at least one of the specified interrupt sources is detected.

*InterruptOccurredArray*

If at least one of the specified interrupt sources occurs, the value is returned through this pointer. The interrupt bits of array index 0 correspond to the Interrupt Status Register, array index 1 corresponds to the Error Interrupt Status Register, and array index 2 corresponds to the DIO Interrupt Status Register. Please refer to the hardware user manual for further information on the possible interrupt bits.

### *timeout*

This value specifies the timeout in milliseconds the function will wait for the interrupt to arrive. The granularity depends on the operating system. To wait indefinitely, specify -1 as timeout parameter.

## EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
unsigned int         interruptMask;
unsigned int         interruptOccurred;

/*
** Wait at least 5 seconds for incoming interrupts on Frame Trigger
*/
interruptMask = (1 << 28);
result = tdrv019InterruptWait(    hdl,
                                  interruptMask,
                                  &interruptOccurred,
                                  5000);

if (result == TDRV019_OK)
{
    /* Interrupt arrived. */
}
else
{
    /* handle error */
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified TDRV019_HANDLE is invalid.
TDRV019_ERR_TIMEOUT	The specified timeout occurred.

## 3.6.2 tdrv019InterruptRegisterCallbackThread

### NAME

tdrv019InterruptRegisterCallbackThread – Register a User Callback Function for Interrupt Handling

### SYNOPSIS

```
TDRV019_STATUS tdrv019InterruptRegisterCallbackThread
(
    TDRV019_HANDLE    hdl,
    int               threadPriority,
    int               stackSize,
    unsigned int      interruptMaskArray[],
    FUNCINTCALLBACK  callbackFunction,
    void              *funcparam,
    TDRV019_HANDLE    *pCallbackHandle
)
```

### DESCRIPTION

This function registers a user callback function which is executed after detection of the specified interrupt source. It is possible to register multiple callback functions to one or a set (bit mask) of interrupt sources.

The callback function is executed in a thread context, so using TDRV019 device driver functions and system functions is allowed. The callback function should be kept as short as possible. The specified callback function is executed with the occurred interrupt bits and the specified function parameter as function arguments. Additionally, a status value is passed to the callback function, which reflects the result of the involved API functions.

**The delay between an incoming interrupt and the execution of the callback function is system-dependent, and is most likely several microseconds.**

### PARAMETERS

*hdl*

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

### *threadPriority*

This parameter specifies the priority to be used for the callback thread. Possible values are:

Value	Description
TDRV019_PRIORITY_NORMAL	Normal Priority (THREAD_PRIORITY_NORMAL)
TDRV019_PRIORITY_HIGH	High Priority (THREAD_PRIORITY_HIGHEST)
TDRV019_PRIORITY_LOW	Low Priority (THREAD_PRIORITY_LOWEST)

Other values might be possible.

### *stackSize*

This parameter specifies the stack size to be used for the callback thread. The value is specified in bytes.

### *interruptMaskArray*

This parameter specifies specific interrupt bits to wait for. The interrupt bits of array index 0 correspond to the Interrupt Status Register, array index 1 corresponds to the Error Interrupt Status Register, and array index 2 corresponds to the DIO Interrupt Status Register. Please refer to the hardware user manual for further information on the possible interrupt bits. The callback function is executed if at least one of the specified interrupt sources occurred.

### *callbackFunction*

This parameter is a function pointer to the user callback function. The callback function pointer is defined as follows:

```
typedef void(*FUNCINTCALLBACK)( TDRV019_HANDLE  hdl,
                                unsigned int      interruptOccurredArray[],
                                void              *param,
                                TDRV019_STATUS   status )
```

### *hdl*

This parameter specifies a device handle which can be used for hardware access or other API functions by the callback function.

### *interruptOccurredArray*

This parameter contains the occurred interrupts. It is useful if the callback function handles multiple interrupt sources. The interrupt bits of array index 0 correspond to the Interrupt Status Register, array index 1 corresponds to the Error Interrupt Status Register, and array index 2 corresponds to the DIO Interrupt Status Register. Please refer to the hardware user manual for further information on the possible interrupt bits.

### *param*

This parameter is the user-specified *funcparam* value (see below) which has been specified on callback registration. This value can be used to pass a pointer to a specific control structure, to supply the callback function with specific information.

### *status*

This parameter hands over interrupt callback status information. The callback function needs to check this parameter. If the specified interrupt source has occurred properly, and no errors were detected, this parameter is TDRV019\_OK. If this parameter differs from TDRV019\_OK, an internal error has been detected and the callback handling is stopped. The callback function must implement an appropriate error handling.

*funcparam*

This value specifies a user parameter, which will be handed over to the callback function on execution. This parameter can be used to pass a pointer to a specific control structure used by the callback function.

*pCallbackHandle*

This value specifies a pointer to a handle, where the callback handle will be returned. This callback handle must be used to unregister a callback function.

**EXAMPLE**

```
#include <tdrv019api.h>

TDRV019_HANDLE    hdl;
TDRV019_STATUS    result;
unsigned int      interruptMaskArray[3];
USER_DATA_AREA    userDataArea;
TDRV019_HANDLE    callbackHandle;

/* forward declaration of callback functions */
void callback_FRAME(    TDRV019_HANDLE    hdl,
                        unsigned int      interruptOccurredArray[],
                        void              *param,
                        TDRV019_STATUS    status);

/*
** Register callback function for FRAME
** Use a "normal" priority, and 64KB stack.
*/
interruptMaskArray[0] = (1 << 16);
interruptMaskArray[1] = 0;
interruptMaskArray[2] = 0;

result = tdrv019InterruptRegisterCallbackThread(hdl,
                                                TDRV019_PRIORITY_NORMAL,
                                                0x10000,
                                                interruptMaskArray,
                                                callback_FRAME,
                                                &userDataArea,
                                                &callbackHandle);

...
```



```

...
if (result != TDRV019_OK)
{
    /* handle error */
}

...
/*
** Callback Function
*/
void callback_FRAME(    TDRV019_HANDLE    hdl,
                       unsigned int      interruptOccurredArray[],
                       void              *param,
                       TDRV019_STATUS    status)
{
    TDRV019_STATUS      result;
    USER_DATA_AREA     *pUsrData = (USER_DATA_AREA*)param;
    unsigned int        u32value;

    if (status != TDRV019_OK)
    {
        /* handle error status */
    }

    printf("[FRAME Interrupt]\n");

    /* Do something useful */

    /* or handle errors */
    return;
}

```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified TDRV019_HANDLE is invalid.
TDRV019_ERR_INVALID	Function or callback handle pointer is NULL.
TDRV019_ERR_TASK_CREATE	Creation of the callback thread (task) failed.

### 3.6.3 tdrv019InterruptUnregisterCallback

#### NAME

tdrv019InterruptUnregisterCallback – Unregister a User Callback Function

#### SYNOPSIS

```
TDRV019_STATUS tdrv019InterruptUnregisterCallback  
(  
    TDRV019_HANDLE    hdl  
)
```

#### DESCRIPTION

This function unregisters a previously registered user callback thread or ISR function.

#### PARAMETERS

*hdl*

This value specifies the callback handle retrieved by a call to the corresponding register-function.

#### EXAMPLE

```
#include <tdrv019api.h>  
  
TDRV019_HANDLE    callbackHdl;  
TDRV019_STATUS    result;  
  
/*  
** Unregister a callback function  
*/  
result = tdrv019InterruptUnregisterCallback(callbackHdl);  
if (result == TDRV019_OK)  
{  
    /* OK */  
} else {  
    /* handle error */  
}
```

## RETURNS

On success, TDRV019\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified callback handle is invalid.