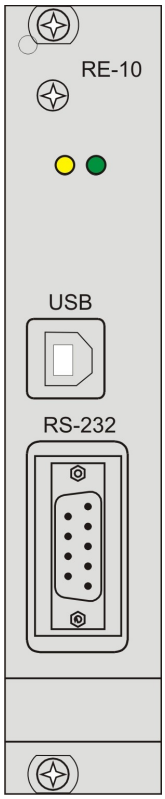
siosifm.dll

Version 1.7

API Description



SIOS Meßtechnik GmbH
Am Vogelherd 46
98693 Ilmenau / GERMANY
Tel.: +49-3677-64470
Fax: +49-3677-64478
Email: info@sios.de
Internet: http://www.sios.de

Updated:
Feb. 2016

SIOS Meßtechnik GmbH

Contents

1 The principle of operation 6
2 Functions for the initialization, opening and closing 10
IfmClose 11
IfmCloseDevice 12
IfmDeviceCount 13
IfmInit 14
IfmMaxDeviceCount 15
IfmOpenCOM 16
IfmOpenDemo 17
IfmOpenUSB 18
IfmSearchUSBDevices 19
IfmUSBDeviceCount 20
IfmUSBDeviceSerial 21
3 Functions for the measurement 23
3.1 Continuous mode 24
IfmGetFilterCoeff 24
IfmGetFilterNotchFrequency 25
IfmGetRecentValues 26
IfmGetValues 27
IfmLengthValue 28
IfmSetPreset 29
IfmGetPreset 30
IfmSetFilter 31
IfmSetFilterCoeff 33
IfmSetFilterNotchFrequency 35
IfmSetMeasurement 36
IfmSetToZero 38
IfmSetTrigger 39
IfmStart 42
IfmStop 43
IfmResetBuffer 44
IfmValueCount 45
IfmAngleValue 46
IfmAngleAvailable 50
3.2 Block mode 51
IfmCancelBlock 52

IfmIsBlockAvailable ... 53
IfmSetBlockMode ... 54
IfmSetBlockModeFilter ... 55
IfmSetBlockModeFilterCoeff ... 56
IfmStartBlock ... 57
4 Functions for the controlling of the interferometers ... 58
IfmGetAGC ... 59
IfmGetRefMirrorVibration ... 60
IfmNewSignalQualityAvailable ... 61
IfmSetAGC ... 63
IfmSetRefMirrorVibration ... 64
IfmSignalQuality ... 65
IfmStatus ... 67
IfmWasBeamBreak ... 69
IfmWasLaserUnstable ... 70
IfmWasLostValues ... 71
5 Functions for the communication with other devices ... 72
5.1 General functions ... 72
IfmI2CRead ... 73
IfmI2CRequestRead ... 74
IfmI2CReadBuffer ... 75
IfmI2CReadValue ... 76
IfmI2CReadReady ... 77
IfmI2CRequestWrite ... 78
IfmI2CStatus ... 79
IfmI2CWrite ... 80
5.2 Functions for the motor control card ... 81
6 Functions for the environment values ... 82
IfmAirPressure ... 85
IfmAirPressureFlags ... 86
IfmAirRefraction ... 88
IfmConversionCoeff ... 89
IfmDeadpathCoeff ... 90
IfmEnableEdlenCorrection ... 91
IfmEnvSensorCount ... 92
IfmHumidity ... 93
IfmHumidityFlags ... 94
IfmIsEdlenEnabled ... 95
IfmGetDeadPath ... 96

IfmNewEnvValuesAvailable ... 97
IfmResetManualEnvironment ... 98
IfmSensorProperty ... 99
IfmSensorValue ... 102
IfmSetAirPressure ... 103
IfmSetConvertionCoeff ... 104
IfmSetDeadPath ... 105
IfmSetHumidity ... 106
IfmSetTemperature ... 107
IfmSetWavelength ... 108
IfmSetWaterVapourPressure ... 109
IfmTemperature ... 110
IfmTemperatureFlags ... 111
IfmVacuumWavelength ... 112
IfmWaterVapourPressure ... 113
IfmWavelength ... 114
7 Extended functions ... 115
IfmAuxValue ... 115
IfmChannels ... 117
IfmDeviceInfo ... 118
IfmDeviceInterface ... 121
IfmDeviceType ... 122
IfmDeviceValid ... 123
IfmDLLVersionString ... 124
IfmFireTrigger ... 125
IfmFirmwareVersion ... 126
IfmGetError ... 127
IfmGetErrorString ... 128
IfmRawValue ... 129
IfmResetDevice ... 130
IfmSetDeviceInfo ... 131
IfmSetOption ... 133
8 Functions for the service ... 134
IfmSaveConfigDevice ... 134
IfmSetMeasurementRawValue ... 135
IfmUpdateDevice ... 136
9 Error codes ... 137

1 The principle of operation

For using the new interferometer interface, introduced with the RE-10 interface card, it is necessary to link the application with “siosifm.dll”. The header files “siosifmdll.h” and “siosifmdef.h” containing the API for the DLL. “siosifmdll.h” contains the function prototypes and includes automatically “siosifmdef.h” which contains most of the definitions. Therefore, for the application it's sufficient to include “siosifmdll.h”.

The RE-10 interface card of the SIOS interferometers can be connected via USB or RS232. The USB-connection builds up a virtual COM-port and can be found in the device manager of Windows among the COM-ports. In Linux the card appears as a modem device with the name `/dev/ttyACM0` or with an other number.

All functions of the API can be arranged in seven parts:

1. functions for initializing, configuration, opening and closing of the SIOS-card/device
2. functions for the measurement
3. functions for the controlling of the interferometer
4. functions for the communication with other card in the SIOS interferometer (via the device internal I2C-bus).
5. functions for the environment values the so called Edlen-correction of measurement values for air refraction
6. extended functions that are used only in special applications
7. service functions

The next chapters of this manual deliver the detailed description of each function.

The following example explains the procedure of a measurement of length values with the output word rate of 1 Hz and with default filter settings. That means, the DLL calculates the settings of the internal signal processing based on the given output word rate so that vibrations of the reference mirror vibrator as well as other possible distortions are suppressed as good as possible.

```
#include <stdio.h>
#include <conio.h>
#include "siosifmdll.h"
int main(int argc, char *argv[])
```

```
{
int error=0;
// before IfmInit; make some settings
// IFM_OPTION_DEBUGFILES let the DLL create files with information for
//debug purposes
//IfmSetOption(IFM_OPTION_DEBUGFILES, true);
// first initialize the DLL
error=IfmInit();
if(error){
  printf("Error %d during IfmInit\n",error);
  return(1);
}
// Search for connected devices; returns the number of via USB
//connected devices
int cnt;
cnt=IfmSearchUSBDevices();
if(cnt<=0){
  printf("A SIOS interferometer could not be found\n");
  IfmClose();
  return(0);
}
int devNo;
// The first device will be opened;
// IfmUSBDeviceSerial and IfmOpenUSB takes a parameter for selecting
//the device from 0 to cnt-1
printf("Open device with serial number %s\n",IfmUSBDeviceSerial(0));
devNo=IfmOpenUSB(0);
if(devNo<0){
  printf("Error during opening the device.\n");
  IfmClose();
  return(0);
}


// IfmOpenUSB has returned a number (devNo) which describes the device
// in further calls to the DLL.
// Now the measurement should be configured. The following
//configuration are required:
// the length values of the first channel with 1 Hz output
// word rate. For the removing of the the vibration of the reference
// mirror from the signal, the default filter will be set with  the
// flag IFM_MEAS_FILTER_DEFAULT
// in some applications the flag IFM_MEAS_FILTER_NONE may be required
// for unfiltered output
error=IfmSetMeasurement(devNo,IFM_MEAS_ONECHANNEL|IFM_MEAS_LENGTH|
IFM_MEAS_FILTER_DEFAULT,1);
// error numbers are always negative
if(error<0){
  printf("Error during opening the device.\n");
  IfmClose();
  return(0);
}
// Set the length values to zero; assuming the measurement mirror is
at the reference/zero position
error=IfmSetToZero(devNo,0x0F);
printf("Printing length data until a key is pressed\n\n");
// begin with the output of dates
error=IfmStart(devNo);
if(error<0){
```

```
    printf("Error during start output.\n");
    IfmClose();
    return(0);
  }
  while(!kbhit()){
   // are new values available?
    if(IfmValueCount(devNo)){
      // put the value in an internal buffer for access via
      //IfmLengthValue
      IfmGetValues(devNo);
      // get the value together with environmental values
      printf("%lf %lf °C %lf   Pa\n", IfmLengthValue(devNo,0),
      IfmTemperature(devNo,0), IfmAirPressure(devNo,0));
    }
  }
  //deinitialize
  // stop the output of data
  IfmStop(devNo);
  // close the device; devNo will be no longer valid
  IfmCloseDevice(devNo);
  // close the DLL
  IfmClose();
  }
```

IfmInit initializes the DLL and starts an internal thread for processing all commands. This function must be called before any other function (with exception of *IfmSetOption*) can be used. The available devices are opened with one of the *IfmOpen*XXX functions. These functions return a number greater or equal zero which acts like a device handle for further calls to the API. If the opening function returns a negative value this is an error number.

IfmSetMeasurement sets the measurement conditions. It is sufficient for most applications to call only this function. But there are other functions which also influence the measurement, like *IfmSetTrigger* or the family of functions for the manual filter settings. *IfmSetMeasurement* must always be called after these functions because it transmits all settings to the device.

The device can sample more channels parallel. To access to the multiple data at different times the appropriate set of data must be frozen internally in a separate buffer. This is done by *IfmGetValues* (or *IfmGetRecentValues*). *IfmGetValues* doesn't return a measurement value itself but it is always required before accessing to the measurement values with *IfmLengthValue*, *IfmRawValue* or *IfmAuxValue.*

A little bit extended example can be found among the sample programs which are available with the DLL.

Most of the API-functions return immediately but there are some exceptions:

- *IfmI2CRead, IfmI2CWrite, IfmSetRefMirrorVibration* and *IfmSetAGC* transmit data via the

internal I2C-bus of the interferometer and wait until the transmission is completed (or a timeout occurs)

- *IfmClose* tries to shut down all devices in a controlled manner. It waits until all devices have sent out the data in their output buffers. Therefore *IfmClose* blocks all other processes until all devices are shut down (or a timeout occurs).

Also, even if a function returns immediately the appropriate action can be delayed. For example, *IfmSetMeasurement* and *IfmStart* are always returning without any delay, but the start of the measurement is delayed until the device has transmitted all information which are required (it's configuration, environmental data, reference mirror status).

Most of the functions return 0 if the action was successfully or a negative number in case of an error. Some functions return on success a non negative number (like the device count) but also a negative error number if something has failed. Only rare functions return boolean values: a zero for *false* and a 1 for *true*. An example is *IfmDeviceValid.*

2 Functions for the initialization, opening and closing

The DLL must be initialized by calling *IfmInit* before other functions can be used. Normally *IfmInit* starts an internal thread which handles the device communication. *IfmClose* is the counterpart of *IfmInit.* It stops the internal thread and gives free all allocated resources.

Each device must be opened by calling *IfmOpenCOM* for a COM-Port or *IfmOpenUSB* for an USB-interface. If the device is not longer in use, the interface have to be closed (before calling *IfmClose* and exiting). *IfmCloseDevice* Should be called for every open device.

IfmOpenCOM and *IfmOpenUSB* return an unique number (further called *devNumber*) which selects the device in other calls to the library. It's very similar to the handle usage in the Windows API.

IfmClose

Syntax

void IfmClose()

Description

This function deinitializes the library, stops the internal thread and gives free all allocated resources. Depending on open devices or on the time span since the last device has been closed *IfmClose* waits for a controlled shut down of the devices. Therefore, *IfmClose* may block for a maximum of 2 seconds (typically some milliseconds).

Input parameters

none

Output parameters

none

See also an example in the section *IfmOpenCOM.*

IfmCloseDevice

Syntax

void IfmCloseDevice(int devNumber)

Description

This function deinitializes a device which has been opened by *IfmOpenCOM* or *IfmOpenUSB.* The *devNumber* becomes invalid after calling this function. The device cannot be used later.

Each call of *IfmOpenCOM* or *IfmOpenUSB* needs a call of *IfmCloseDevice* when the device is not longer in use.

The device is shut down in a controlled manner so that all buffered commands are transmitted. *IfmCloseDevice* blocks until this controlled shut down is done, which lasts typically 150 milliseconds.

To prevent it, *IfmSetOption*(IFM_OPTION_BLOCKONCLOSE,false) changes the behaviour. IfmCloseDevice will return immediately but the device is internal still alive. Normally this doesn't affect the user but has some side effects. For example, in a successive call to *IfmCloseDevice* and *IfmOpenUSB* the opening of the same physical interface causes an error because the resource is still in use.

Input parameters

devNumber	Unique ID for the device, returned by *IfmOpenCOM* or *IfmOpenUSB*

Output parameters

none

See also an example in the section *IfmOpenCOM.*

IfmDeviceCount

Syntax

int IfmDeviceCount()

Description

The function *IfmDeviceCount* returns the number of opened devices.

Input parameters

No input parameters.

Output parameters

Required count of open devices.

IfmInit

Syntax

int IfmInit()

Description

This function initializes the DLL and starts an internal thread for the communication with the connected devices.

IfmInit must be called before any other function (except *IfmSetOption*) of the library can be used.

When the library is not longer in use, *IfmClose* should be called.

Input parameters

none

Output parameters

0 if the function has success, an error number otherwise.

See also an example on the *IfmOpenCOM.*

IfmMaxDeviceCount

Syntax

int IfmMaxDeviceCount()

Description

The function *IfmMaxDeviceCount* returns the number of the maximum allowed devices.

Input parameters

No input parameters.

Output parameters

Requested count of maximum allowed devices

IfmOpenCOM

Syntax

Windows:

int IfmOpenCOM(int comNumber)

Linux:

int IfmOpenCOM(const char* deviceName)

Description

This function opens a device which is connected by a serial RS232-port.

Please note, the USB connection of the RE-10 card is build internally over a virtual RS232-port. Therefore, an USB-connection of the RE-10 can be opened by *IfmOpenUSB* as well as by *IfmOpenCOM*.

Input parameters

comNumber	The number of the RS232-interface, for example 3 for COM3.
deviceName	The Linux name of the device, for example /dev/ttyACM0

Output parameters

The function returns an unique ID, the *devNumber*, which must be used to access to the device in future calls to the library. The *devNumber* is always a non-negative number. In case of an error, an error number is returned. Error numbers are always negative. See also the part *Error codes.*

IfmOpenDemo

Syntax

int IfmOpenDemo(int channels)

Description

not yet implemented

IfmOpenUSB

Syntax

int IfmOpenUSB(int uniqueId)

Description

This function opens a device which is connected via the USB-interface for communication.

Input parameters

uniqueId	The ID which describes the device. Normally, it's a number between 0 and the amount of connected devices – 1. “0” means, that the device should be opened which was found first on the USB bus. See *IfmSearchUSBDevices* for more information.

Output parameters

The function returns an unique ID, the *devNumber*, which must be used to access the device by the future calls to the library. The *devNumber* is always a non negative number.

In case of an error an error number is returned. Error numbers are always negative.

IfmSearchUSBDevices

Syntax

int IfmSearchUSBDevices()

Description

The function *IfmSearchUSBDevices* looks for devices (this time only RE-10 cards) which are connected to the PC via the USB-Bus and returns the number of connected devices. These devices can be opened by *IfmOpenUSB.* For distinguishing between different devices the serial number can be accessed by the function *IfmUSBDeviceSerial. IfmUSBDeviceSerial* and *IfmOpenUSB* need an unique ID to select the desired device. This ID is the running number between zero and the device count minus one.

Input parameters

No input parameters.

Output parameters

The function returns the devices count. Zero will be returned, if no device can be found. In case of an error a negative error number is returned.

IfmUSBDeviceCount

Syntax

int IfmUSBDeviceCount()

Description

The function *IfmUSBDeviceCount* returns the number of devices on the USB-bus found at the last search by *IfmSearchUSBDevices.*

Input parameters

No input parameters.

Output parameters

Number of devices on the USB-bus.

IfmUSBDeviceSerial

Syntax

int IfmUSBDeviceSerial(int uniqueId)

Description

The function *IfmUSBDeviceSerial* returns the USB-serial number of the requested device. Usually it will be used together with the function *IfmSearchUSBDevices*, that finds the count of the devices on the USB-bus.

Input parameters

uniqueId	The order number in the list of devices, that was created by the function *IfmSearchUSBDevices.* 0 <= *uniqueId* < *deviceCount*

Output parameters

Serial number of the corresponding devices as an integer value. (Even if USB-serials can be a character string, the serial number of a RE10-cards is always a number and it is converted to an integer).

Example

```
comboBoxUSBDev->clear();
int deviceCount=IfmSearchUSBDevices();
if(deviceCount>0){
      for(int i=0;i<deviceCount;i++){
      char s[20];
      sprintf(s,"%6.6d",IfmUSBDeviceSerial(i));
      comboBoxUSBDev->addItem(s,i);
      }
}
```

Explanation: An existing combo box *comboBoxUSBDev* is used for the displaying of available devices. At the begin of the code this list is cleared. After the call of *IfmSearchUSBDevices* the the count of the existing devices on the USB-bus is known. The serial number of this devices delivers

the function *IfmUSBDeviceSerial.* For each device an entry with the serial number in the combo box is created.

3 Functions for the measurement

For accessing the interferometer values two different modes are available. The continuous mode delivers data in a continuous data stream nearly in time with the measurement (plus the delay due to the transport to the PC). With USB-connection the RE-10 can reach data rates approximately up to 20 kHz. The block mode has to be used for faster data rates. In block mode are the data stored in an internal buffer before transmitting to the PC so that the measurement and the transport of the data are decoupled. It allows faster data rates (up to 12.5 MHz with the RE-10) but the maximum data count is limited by the length of the internal buffer (65535 samples in case of the RE-10).

The incoming data are stored on the PC in a FIFO buffer. *IfmGetValues* reads the data from one time sample (length values of all channels and other data according to the configuration of the measurement) from the FIFO and provides them to an internal buffer. These values can be accessed by *IfmLengthValue, IfmRawValue* or *IfmAuxValue. IfmGetValues* empties the FIFO buffer from the bottom (the values incoming first are read out first).

IfmGetRecentValues acts like *IfmGetValues* but reads the data from top of the FIFO without removing it from the buffer. It allows the reading of the newest values without removing values from the buffer, so that *IfmGetValues* will work at the same time (for example, use *IfmGetValues* to read out the values for storing them into a file and use *IfmGetRecentValues* for displaying the most recent data on the screen).

The measurement starts with *IfmStart*. Before that, the measurement options should be set with *IfmSetMeasurement* and if required with IfmSetFilter and/or *IfmSetTrigger*. Otherwise the measurement is configured with the default values from the device (which may vary).

Please note, that IfmSetMeasurement must always be called after *IfmSetFilter (*and related functions) and *IfmSetTrigger*, because *IfmSetMeasurement* transmits the settings to the device.

3.1 Continuous mode

IfmGetFilterCoeff

Syntax

double IfmGetFilterCoeff(int devNumber, int channel)

Description

The function returns the coefficient of the FIR-filter for the corresponding device and channel.

This function should only be used by advanced users. Please see the RE-10 signal processing guide for more information.

See also *IfmSetMeasurement, IfmSetFilterCoeff.*

Input parameters

devNumber Unique ID for the device

channel Channel number

Output parameters

requested FIR-filter coefficient

or

0 in case of an error

IfmGetFilterNotchFrequency

Syntax

double IfmGetFilterNotchFrequency(int devNumber, int channel)

Description

This function returns the notch frequency, which was set by *IfmSetFilterNotchFrequency* in case of the user settings, otherwise it is a default value calculated by the dll on the basis of the measuring conditions.

The function delivers a valid value AFTER *IfmSetMeasurement* **was called.**

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

Notched frequency in Hz

IfmGetRecentValues

Syntax

int IfmGetRecentValues(int devNumber, int index)

Description

The function *IfmGetRecentValues* is used to read out the measurement values at the $index_{readout}$ from the input puffer.

$$index_{readout} = count - index - 1$$

The $count$ of the measuring values can be requested by the function *IfmValueCount.*

The index “0” reads the most recent value.

The values are provided in an internal buffer for accessing via *IfmLengthValue*, *IfmRawValue* or *IfmAuxValue*

In contrast to *IfmGetValues* this function does not influence the amount of available values which is returned by *IfmValueCount*.

Input parameters

devNumber	Unique ID for the device
index	Index, 0 means the most recent value

Output parameters

1 (true) on success, 0 (false) an error has occurred

Example

```
if(IfmGetRecentValues(devNo,0))
  last_measured_distance=IfmLengthValue(devNo,0);
```

Explanation: The most recent value will be provided for accessing via *IfmLengthValue*

IfmGetValues

Syntax

int IfmGetValues(int devNumber)

Description

This function is used to request of the measuring values from the input buffer. The available values will be read out according to FIFO (first in first out) principle. Usually this function will be used together with the functions *IfmLengthValue, IfmRawValue* or *IfmAuxValue.*

Input parameters

devNumber Unique ID for the device

Output parameters

1 (true): on success (otherwise 0 for false)

Example

```
int count=IfmValueCount(devNo);
if(count>0){
   for(int i=0;i<count;i++){
      IfmGetValues(devNo);
      if(file)fprintf(file,"%f %f %f %f\n",
         IfmLengthValue(devNo,0),
         IfmLengthValue(devNo,1),
         IfmLengthValue(devNo,2),
         IfmLengthValue(devNo,3));
   }
}
```

Explanation: Four channels are available in the Unique ID for the device *devNo*. The program tests, whether the device has sent measurement values. If IfmValueCount returns the *value*>0, the values will be read out from the input puffer (function *IfmGetValues*) and the length measuring value will be written to an ASCII-file.
IfmGetValues decrements the count of available values (*IfmValueCount*).

IfmLengthValue

Syntax

double IfmLengthValue(int devNumber,int channel)

Description

The function IfmLengthValue is to read out the length measuring values, that was extracted from a data field of the input buffer. It is intended to use directly after the function *IfmGetValues* or *IfmGetRecentValues*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

measuring value in nanometres

Example

```
if (IfmValueCount(devNo)>0)
{
   IfmGetRecentValues(devNo,0 );
   double value_channel_0=    IfmLengthValue(devNo,0);
   double value_channel_1=    IfmLengthValue(devNo,1);
}
```

Explanation: if the measuring values are available (*IfmValueCount*(devNo)>0) , the last length values for channel 0 and 1 will be read out.

IfmSetPreset

Syntax

int IfmSetPreset(int devNumber, int channel, double presetValue)

Description

Normally IfmSetToZero defines the reference point for the measurement and sets the length value to 0. There are possible applications where it seems more practical to set the position at the reference point to another value, the so called preset value. IfmSetPreset sets this preset value. With the next IfmSetToZero the preset becomes active. This means:

- IfmSetPreset must be called before IfmSetToZero, and
- that the preset value becomes active not before IfmSetToZero is called

In principle this behaviour can also be emulated in the end user software. But the handling in the siosifm library has the advantage that the preset value is exactly set at the time of the definition of the reference point. It prevents jumps in the data.

The preset value influences only the length values of IfmLengthValue. IfmAngleValue and IfmRawValue are not touched.

Input parameters

devNumber	Unique ID for the device
channel	Channel number
presetValue	The value, IfmLengthValue should return at the reference position in nm

Output parameters

0 in success, otherwise an error number

IfmGetPreset

Syntax

double IfmGetPreset(int devNumber, int channel)

Description

With IfmSetPreset a so called preset value can be set, what sets the reference point, defined with IfmSetToZero to a length value other than zero.

The preset value becomes active when IfmSetToZero is called and the zeroing procedure (which may take some ms) is complete.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The current preset value in nm. In case of an error, it returns always 0.

Example

```
IfmSetPreset(devNo,0,3000000);  // set 3 millimeter preset
double a=IfmGetPreset(devNo,0); // will return 0
IfmSetToZero(devNo,0x0F);       // set all channels to zero and loads
                                // the preset
a=IfmGetPreset(devNo,0);        // will likely also return 0, because
                                // zeroing procedure is not complete
Sleep(50);
a=IfmGetPreset(devNo,0);        // will return 3000000
```

IfmSetFilter

Syntax

int IfmSetFilter(int devNumber, unsigned int filterFlags,int avg1, int avg2)

Description

The function is used to set the filter options.

This function is for experts only!

The filtering process demonstrates following figure:

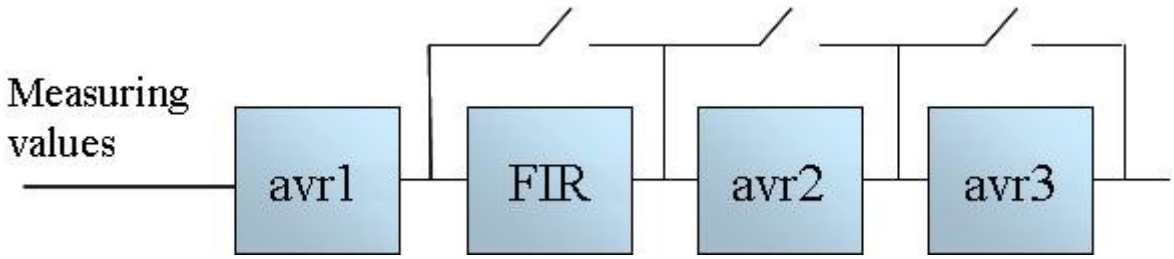


Figure 1: Structure of the signal processing box

avr1, *avr2* and *avr3* are reducing averaging filter. The number of the measuring values for *avr1* and *avr2* will be built as

$$N_1 = 2^{avr1}$$

and

$$N_2 = 2^{avr2}$$

avr3 consists of three stages of averaging filters with 5, 6 and 9 samples.

If the values *avr1* or *avr2* is equal 0, will be the corresponding filter deactivated. Additionally there

are the possibility to deactivate the filter stage 1 (FIR), filter stage 2 (*avr2*) and the filter stage 3 (*avr3*) via the *filterFlags* of this function.
For the description of the FIR-filter see function *IfmSetFilterCoeff*.
It is important to use this function before the function *IfmSetMeasurement* because the *IfmSetMeasurement* function sets the filter settings in the device together with all other measurement settings.

Please note also the the *measurementFlags* in the function *IfmSetMeasurement.*To set up the filter manually the measurementFlags must contain IFM_FILTER_USER.

Input parameters

devNumber	Unique ID for the device
filterFlags	Following parameters can be set:
IFM_FILTER_STAGE1	Set the filter stage 1 (*FIR*) on
IFM_FILTER_STAGE2	Set the filter stage 2 (*avr2*) on
IFM_FILTER_STAGE3	Set the filter stage 3 (*avr3*) on

Output parameters

0 on success, an negative error number if an error has occurred

Example

```
IfmSetFilter(0, IFM_FILTER_STAGE2|IFM_FILTER_STAGE3,2,3)
```

Explanation: In Unique ID for the device 0 set on the filter *avr1*,*avr2* and *avr3*. The number of measuring values for the averaging and reducing: averaging filter 1: $N_1 = 2^2 = 4$; averaging filter 2 : $N_2 = 2^3 = 8$. The whole stage3 is also switched on: 5*6*9=270. The reducing factor of the whole signal processing box is $N_r = 4 \cdot 8 \cdot 270 = 8640$ what means that, for example, an input sample rate of 100.000 Hz will result in an output word rate of 100000/8640= 11,574.

IfmSetFilterCoeff

Syntax

int IfmSetFilterCoeff(int devNumer, int channel, double coeff)

Description

The function is used to set the FIR-filter coefficients.

This function is for experts only!

This function should be applied for every of available channel. The FIR-filter contains 3 coefficients from which the second coefficient can be configured. The other two are always set to one *koef*=[1 *coeff* 1]. It means, the FIR filter has only one zero point in his frequency response and, therefore, forms a notch filter.

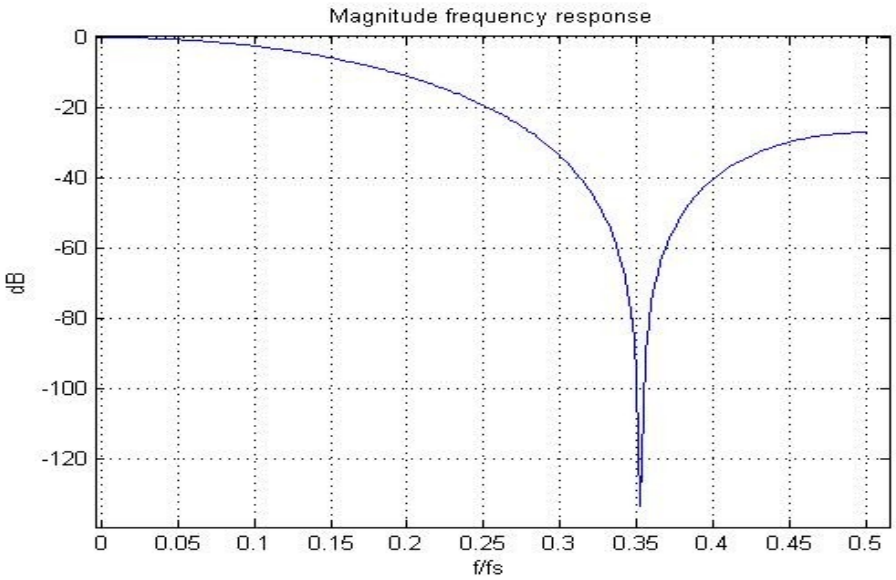


Figure 2: Magnitude frequency response of a FIR-filter (an example)

The notch frequency can be influenced by setting the second coefficient through *IfmSetFilterCoeff* or *IfmSetFilterNotchFrequency.*The function *IfmSetMeasurement* should be used with the flag IFM_MEAS_FILTER_USER for the acceptance of this setting.

If the *measurementFlags* in *IfmSetMeasurement* contains IFM_MEAS_FILTER_DEFAULT the filter coefficients (as well as the other signal processing) is configured, to notch out reference mirror vibration frequency.

Input parameters

devNumber	Unique ID for the device
channel	Channel number
coeff	coefficient

Output parameters (error numbers)

0 on success, an negative error number if an error has occurred

IfmSetFilterNotchFrequency

Syntax

int IfmSetFilterNotchFrequency(int devNumber, int channel, double freq);

Description

This function is a simplified implementation for the setting of the FIR notch filter (see also *IfmSetFilterCoeff)*.

This function is for experts only!

It should be applied for every of available channel. The function *IfmSetMeasurement* should be used with the flag IFM_MEAS_FILTER_USER for the acceptance of this setting. If the *measurementFlags* in *IfmSetMeasurement* contains IFM_MEAS_FILTER_DEFAULT, the filter coefficients (as well as the other signal processing) will be configured, to notch out reference mirror vibration frequency.

The input parameter *freq* is the wished notch frequency in Hz. The dll calculates the FIR-filter coefficients. A precondition for the successful execution is the following term:

$$\frac{f_i}{4} \leq freq \leq \frac{f_i}{2}$$

f_i is the frequency on the input of FIR-filter.

See also the filter stages description with the functions IfmSetFilter and IfmSetFilterCoeff.

Input parameters

devNumber	Unique ID for the device
channel	Channel number
freq	notched frequency in Hz

Output parameters

0 on success, an negative error number if an error has occurred

IfmSetMeasurement

Syntax

int IfmSetMeasurement(int devNumber,unsigned int measurementFlags, double outputWordRate)

Description

The function *IfmSetMeasurement* is used to set the measurement parameters. This function should be called directly **before** the measurement starts (see *IfmStart*). If this function is not called the settings which are saved in the flash of the interferometer are used.

Input parameters

devNumber	Unique ID for the device
measurementFlags	Following parameter can be used:
IFM_MEAS_ONECHANNEL	Channel number 1 is on
IFM_MEAS_TWOCHANNEL	Channels number 1-2 are on
IFM_MEAS_THREECHANNEL	Channels number 1-3 are on
IFM_MEAS_FOURCHANNEL	Channels number 1-4 are on
IFM_MEAS_CH1	Channel number 1 is on
IFM_MEAS_CH2	Channel number 2 is on
IFM_MEAS_CH3	Channel number 3 is on
IFM_MEAS_CH4	Channel number 4 is on
IFM_MEAS_LENGTH	Full length information
IFM_MEAS_SINCOS	SIN / COS values from the input are transmitted
IFM_MEAS_CIRCLE	Amplitude of the Lissajous figure
IFM_MEAS_PATTERN	Test pattern (value 0x5555)*
IFM_MEAS_VAL_COUNTER	Internal value counter*
IFM_MEAS_FILTER_DEFAULT	Default filter will be applied
IFM_MEAS_FILTER_OFF	No filter will be used
IFM_MEAS_FILTER_USER	User filter settings will be used

* for test purpose

Please note: Requesting different types of data as well as multiple channels will reduce the available output rate and the effective block length in block mode!

Output parameters

0 on success, an negative error number if an error has occurred

Example 1

```
IfmSetTrigger(devNumber,IFM_TRIGGER_OFF);
int measFlags=  IFM_MEAS_TWOCHANNEL|
IFM_MEAS_LENGTHIFM_MEAS_FILTER_OFF;
IfmSetMeasurement(devNumber, measFlags , 1000);
IfmStart(devNumber);
```

Explanation: The channels one and two will be activated; the measurement value type that should be transmitted is the length value; there are no filtering and triggering; output word rate is 1 kHz. The measurement starts directly after *IfmStart(devNumber)*. The device samples the signal with 1 kHz, saves the sampled values into a puffer and sends the unprocessed measurement values. The measurement will stop, if *IfmStop(devNumber)* is sent.

Example 2

```
IfmSetTrigger(devNumber, IFM_TRIGGER_STARTSTOP_PROC
|IFM_EXTTRIG_RISING_EDGE);
int IfmSetFilter(devNumber,IFM_FILTER_STAGE2|IFM_FILTER_STAGE3, 2, 3)
int measFlags= IFM_MEAS_ONECHANNEL|IFM_MEAS_LENGTH|
IFM_MEAS_FILTER_DEFAULT;
IfmSetMeasurement(devNumber, measFlags , 10);
IfmStart(devNumber);
```

Explanation: In the device the channel one will be activated with following settings:

measurement value type: length values;

user settings for the filtering;

output word rate: 10 Hz.

After the call of the function *IfmStart*, the device waits for the rising edge on the trigger input. If the appropriate level change on the event input is detected, the measurement will start. The device is set up to sample with the internal highest possible sample rate and to notch the reference mirror vibration. The internal signal processing reduces the sampling rate to the required output word rate of 10 Hz. The processed value will be sent to the PC with the output word rate of 10 Hz. The measurement stops after the next rising edge on the trigger input or by calling *IfmStop*.

Hint: **If the measurementFlags are set to 0, the measurement is started with the device**

internal configured channels, requesting length values and sets the default filter. If you are not sure, set measurementFlags to 0 should work in most conditions.

IfmSetToZero

Syntax

int IfmSetToZero(int devNumber, int channelMask)

Description

The function sets the internal counter of the interferometer to zero and defines the reference point for the displacement measurement. The parameter *channelMask* defines which channel should be cleared.

At the same time the dead path of the channel, set by *IfmSetDeadPath* will be taken over into the environmental correction.

Input parameters

devNumber	Unique ID for the device
channelMask	Mask which defines, which channels should be affected (use the constants IFM_CHANNEL1 to IFM_CHANNEL4)

Output parameters

0 on success, an negative error number if an error has occurred

IfmSetTrigger

Syntax

int IfmSetTrigger(int devNumber, unsigned int triggerMode)

Description

The function is used to set the trigger conditions.

This settings have to be made **before** calling *IfmSetMeasurement.*

The RE-10 card works together with the trigger card TR-10 which provides three inputs:

1. the trigger input

   Normally this input is used for the typical trigger options. The trigger input is internally connected to the logic hardware so that it reacts very fast with a short delay of typical some nanoseconds (for details see the data sheet of the TR-10 or the manual of the interferometer).This trigger input is used for the start/stop of the processed and unprocessed values.

2. the event input

   This input is connected to the microprocessor of the RE-10. It's usage is potentially more flexible and especially suitable in conjunction with processed values (filtered values). But due to the limited reaction speed of the microprocessor the timing is less accurate then the timing of the trigger input. This input is for triggering of processed (filtered) values.

3. the clock input

   Normally the AD-converters of the RE-10 run with 50 MHz and the logic continuously counts the interferences. So the length in the interferometer is always up-to-date. The measurement values are taken from the counter logic with the requested sample rate (or output word rate which is the sample rate reduced by the filter in the processing stage). Instead of using a fixed sample rate the clock input can be used to define at which point in time a sample should be taken from the counter logic.

The usage of the three trigger inputs can be combined.

There are differences between triggering unprocessed and processed values. Unprocessed values mean the values that comes directly from the count logic. Processed values are filtered with the

current filter settings (which can also be switched of so that processed and unprocessed are the same). If filtering is on the filters usually reduces the sampling rate. So more unprocessed values are required to create one processed value. If unprocessed values are triggered and the filtering is on, please note that more than one trigger pulse is necessary for outputting one single sample. If this is desired, trigger processed values or switch the filter off.

Input parameters

devNumber	Unique ID for the device
triggerMode	Following parameters can be set:
IFM_TRIGGER_OFF	no trigger
IFM_TRIGGER_START	start the measuring after a valid (falling or rising) edge on the *trigger* input (for unprocessed values)
IFM_TRIGGER_START_STOP	start and stop of measuring after a valid (falling or rising) edge on the *trigger* input (for unprocessed values)
IFM_TRIGGER_CLOCK	The clock input gives the sample clock
IFM_TRIGGER_EVENT	outputting one (processed) measuring value after every valid (falling or rising) edge on the *event* input
IFM_TRIGGER_STARTSTOP_PROC	start and stop of measuring after a valid (falling or rising) edge on the *trigger* input (for processed values, can be combined with IFM_TRIGGER_EVENT)
IFM_TRIGGER_STARTSTOP_RISING_EDGE	the rising slope is active at the trigger input (otherwise the falling slope)
IFM_TRIGGER_EVENT_RISING_EDGE	the rising slope is active at the event input (otherwise the falling slope)
IFM_TRIGGER_CLOCK_RISING_EDGE	the the rising slope is active on the clock input

Output parameters (error numbers)

0 on success, an negative error number if an error has occurred

Example

```
int error=IfmSetTrigger(devNumber,IFM_TRIGGER_STARTSTOP_PROC|
IFM_TRIGGER_EVENT);
```

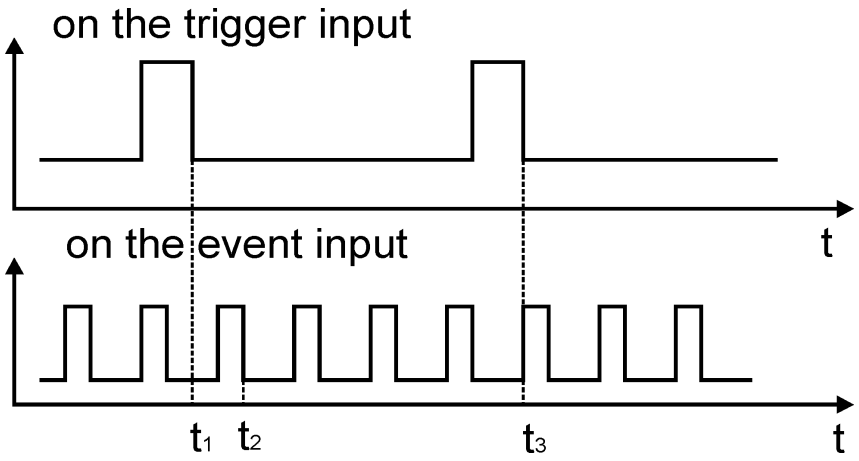


Figure 3: Measuring by the using of the trigger and event inputs of the trigger card (for the processed values)

Explanation: After the measurement is enabled (see the functions *IfmSetMeasurement* and *IfmStart*), the device observes the *trigger* input for the falling edge. If the edge appears (time point *t1* on the Figure 3), the device will observe the *event* input for the falling edge. If it occurs (time point *t2* on the Figure 3), the device will start the measurement with the current conditions for signal sampling and processing. The processed value will be sent to the PC. After them the device is waiting for the next falling edge on the *event* input and so on. As soon as the second falling edge was registered on the trigger input (time point *t3* on the Figure 3), will be the measurement stopped. By the conditions shown on the Figure 3, the device output four measurement values.

IfmStart

Syntax

int IfmStart(int devNumber)

Description

The function gives the measurement free. It should be used after the function *IfmSetMeasurement.* Otherwise the device takes the settings, that has been saved in the flash as default settings. See functions *IfmSaveConfigDevice* and *IfmSetMeasurement* for the detailed information.

Input parameters

devNumber Unique ID for the device

Output parameters

0 on success, an negative error number if an error has occurred

lfmStop

Syntax

int lfmStop(int devNumber)

Description

The function stops the measurement. It's the counterpart to *lfmStart*.

Input parameters

devNumber Unique ID for the device

Output parameters

0 on success, an negative error number if an error has occurred

IfmResetBuffer

Syntax

int IfmClearBuffers(int devNumber)

Description

The function the erasing of the PC-input buffer

Input parameters

devNumber Unique ID for the device

Output parameters

0 on success, an negative error number if an error has occurred

lfmValueCount

Syntax

int lfmValueCount(int devNumber)

Description

The function delivers the count of the samples which are available in the input buffer. The count is incremented by incoming data from the device and decremented by *lfmGetValues.*

See also *lfmGetValues* for more information.

Input parameters

devNumber Unique ID for the device

Output parameters

Count of values which can be read out from the input buffer by *lfmGetValues.*

IfmAngleValue

Syntax

double IfmAngleValue(int devNumber,int channel1,int channel2,int unit)

Description

Two and three beam interferometer are able to measure changes in pitch and yaw angle of the measurement mirror. This is done via the length value difference of two channels whose beams are directed in parallel to the same measurement mirror. The angle difference is calculated as follows:

$$angle = \arctan\left(\frac{channel1 - channel2}{base\ distance}\right)$$

The base distance of the two beams must be known. Normally it is factory calibrated and stored in the device configuration of the RE-10 interface card. Under these conditions the angle can be read out with IfmAngleValue. As with IfmLengthValue IfmAngleValue is intended to use directly after the function *IfmGetValues* or *IfmGetRecentValues* which freeze the values in an internal buffer.

Input parameters

devNumber	Unique ID for the device
channel1	Channel number for the first channel, use the constants IFM_CHANNEL1 to IFM_CHANNEL4
channel2	Channel number for the second channel, use the constants IFM_CHANNEL1 to IFM_CHANNEL4
unit	0: the angle is returned in mrads 1: the angle is returned in seconds

Output parameters

angle between the two channels in the requested unit. If a calculation of the angle was not possible (due to missing length values for at least one channel or due to missing configuration of the base distance) zero is returned.

Note: The channels must be given using the IFM_CHANNELx constants which code the channel as a specific bit. An integer number cannot be used. The sign of the angle can be inverted by permuting the parameter *channel1* and *channel2.*

There are some possibilities to check, if the angle calculation is possible. The easiest way is to use IfmAngleAvailable to check, if the configuration of the interferometer permits the angle measurement between two channels.

More information can be achieved with IfmDeviceInfo. The configuration of the RE-10 allows to define 2 pairs of channels for calculating specific angles. On three beam devices the rectangular geometry of the beams allows typically to calculate angles between channel 2 and channel 1 (yaw angle, note the order) as well as between channel 2 and channel 3 (pitch angle). But the configuration of the sensor heads are potentially flexible so that deviations from this typical setup are possible.

The following example shows the usage of IfmAngleValue as well as the achieving of the configuration details, which are not .

Example

```
// ask for the configuration of the angle calculation
int k1=-1,k2=-1,k,j;
double distance;

// the first pair of channels in the configuration
k=IfmDeviceInfo(devNo,IFM_DEVINFO_LINKEDCHANNELS1);

float
*f=(float*)IfmDeviceInfo(devNo,IFM_DEVINFO_BASEDISTANCE1_POINTER);

distance=0;
// be carefull, older DLL versions don't know the parameter
// and would return 0
if(f)distance=*f;

// check, which channels are in thr channel mask for this pair
for(j=0;j<4;j++){
    if(k&(1<<j))k1==-1?k1=j:k2=j;
}

printf("\nChannel connection 1: %d - %d\n"
   "base distance %f\n\n",k1,k2,distance);

// the same for channel 2
k=IfmDeviceInfo(devNo,IFM_DEVINFO_LINKEDCHANNELS2);
f=(float*)IfmDeviceInfo(devNo,IFM_DEVINFO_BASEDISTANCE2_POINTER);
distance=0;
```

```
if(f)distance=*f;

k1=-1;k2=-1;
for(j=0;j<4;j++){
    if(k&(1<<j))k1==-11?k1=j:k2=j;
}
printf("Channel connection 2: %d - %d\n2
  "base distance %f\n\n",k1,k2,distance);

// normally between the channels 2 and 1 the yaw angle is calculated
if(IfmAngleAvailable(devNo,IFM_CHANNEL1|IFM_CHANNEL2))
  printf("Angle calculation between channels 2 and 1 is possible.\n");
else
  printf("Angle calculation between channels 2 and 1 is "
  "NOT possible.\n");

// normally between the channels 2 and 3 the pitch angle is calculated
if(IfmAngleAvailable(devNo,IFM_CHANNEL3|IFM_CHANNEL2))
  printf("Angle calculation between channels 2 and 3 is possible.\n");
else printf("Angle calculation between channels 2 and 3 is"
  "NOT possible.\n");

// Set the length values to zero; assuming the measurement mirror
//  is at the reference/zero position
error=IfmSetToZero(devNo,0x0F);

printf("Printing length data until a key is pressed\n\n");

// begin with the output of data
error=IfmStart(devNo);
if(error<0){
    printf("Error during start output.\n");
    IfmClose();
    return(0);
}

while(!kbhit()){

    // are new values available?
    if(IfmValueCount(devNo)){
    // put the value in an internal buffer for access via
    //IfmLengthValue and IfmAngleValue
    // this is necessary to access the same syncronuously
    //sampled values (e.g. different channels) at different times
    // IfmValueCount is decremented

       IfmGetValues(devNo);
       // get the values together with the angles
       printf("%8.0lf µm %8.0lf nm %8.0lf nm - yaw (2-1) %.3f"
            " pitch (2-3) %.3f      \r",
            IfmLengthValue(devNo,0)/1000,IfmLengthValue(devNo,1)/1000,
            IfmLengthValue(devNo,2)/1000,
            IfmAngleValue(devNo,IFM_CHANNEL2,IFM_CHANNEL1,1),
            IfmAngleValue(devNo,IFM_CHANNEL2,IFM_CHANNEL3,1));
     }
}
```

IfmDeviceInfo(devNo,IFM_DEVINFO_LINKEDCHANNELS1) returns a byte which codes the channels used for the first angle. The channels are expressed with the IFM_CHANNELx constants

as bits in the returned value.

f=(float*)IfmDeviceInfo(devNo,IFM_DEVINFO_BASEDISTANCE1_POINTER) returns a pointer to a float value (4-Byte width single precision float) in which the base distance for the first pair of channels are stored. The base distance is the distance of the beam balance points which is (internally) necessary to calculate the angle from the length distance between the two beams. Please note, a pointer is returned (because it's not possible to return a float or double) which implies :

- it must be typecasted to use it, the function is declared as returning an integer (which has the same length in a 32-bit environment as a pointer)
- it must be checked if the value is not null before using it, especially older library version don't know about the requested info and will return null.

But these advanced informations are not necessary to use the IfmAngleValue. It's always save to call it. If an angle calculation is not possible IfmAngleValue returns 0. An easy way to check if the device contains the proper configuration for the required angles IfmAngleAvailable can be used as shown in the example.

IfmAngleAvailable

Syntax

int IfmAngleAvailable(int devNumber,int channels)

Description

The function

See also IfmAngleValue for more information.

Input parameters

devNumber	Unique ID for the device
channels	mask of two IFM_CHANNELx constants, to test if an angle between these channels can be calculated

Output parameters

1, if the configuration allows the calculation of an angle between the channels, given in the channels parameter, 0 otherwise

See IfmAngleValue for an example.

3.2 Block mode

With continuous measurement and transmission an output word rate of approximately 22 kHz can be reached. The limiting factor is the data transmission over the USB bus as well as the computation power of the microprocessor. But the interferometer with the RE-10 is able to sample the length values with a sample rate up to 12.5 MHz. To use the higher sample rates the values must be stored in the device with the full sample rate, processed and transmitted with the lower transmission rate. The RE-10 has a FIFO (first-in-first out) buffer with a length of 65535 samples (by the configuration:1 channel, only length values) which is used to decouple the measurement from processing and transmitting. To use it, the SIOSIFM.DLL implements a special block mode.

The block mode may have different measurement settings as the continuous mode so a switching between these two modes is very easy.

IfmSetBlockMode enables the block mode and sets the configuration (similar to *IfmSetMeasurement* for the continuous mode). *IfmStartBlock* starts the measurement. The measurement stops automatically after the requested amount of data has been collected. The maximum amount is limited by the internal FIFO (see *IfmStartBlock*). With *IfmIsBlockAvailable* can be asked if the requested amount of data are already transmitted to the PC and, therefore, are ready for the reading out. The reading out of the measurement values is done in the same way as in the continuous mode.

The typical applications for the block mode are vibration analysis (which is mostly done in blocks to calculate for example the FFT) and the mostly triggered measurement of single short term events.

The following example gets 10 blocks with an sample rate of 12.500 MHz.

```
IfmSetBlockMode(devNumber,IFM_MEAS_CHANNEL1|
IFM_MEAS_FILTER_OFF,IFM_TRIGGER_OFF,12500000);

int i;
for(i=0;i<10;i++){
   IfmStartBlock(devNumber,65535);
   //wait for block
   while(!IfmBlockAvailable(devNumber));
   int j;
   // read out the measurement values
   for(j=0;j<65535;j++){
      IfmGetValues(devNumber);
      fprintf("%f\n",IfmGetLengthValue(devNumber,0);
   }
}
```

lfmCancelBlock

Syntax

int lfmCancelBlock(int devNumber)

Description

This function stops the measurement of a data block which was initiated by *lfmStartBlock.* It also clears an armed trigger condition.

Input parameters

devNumber Unique ID for the device

Output parameters

It returns 0 on success otherwise an error number.

IfmIsBlockAvailable

Syntax

int IfmIsBlockAvailable(int devNumber)

Description

This function is used to poll if a required block is available.

Input parameters

devNumber Unique ID for the device

Output parameters

1 of the block is available, 0 otherwise

A negative return value may indicate an error.

lfmSetBlockMode

Syntax

int lfmSetBlockMode(int devNumber,int measurementFlags,int triggerMode, int outputWordRate)

Description

This function sets the block mode and defines the measurement . A running continuous measurement is stopped. The meaning of the input parameters is very similar to the appropriate parameters of the continuous mode, see *lfmSetMeasurement*.

Input parameters

devNumber	Unique ID for the device
measurementFlags	see *lfmSetMeasurement*
triggerMode	see *lfmSetTrigger*
outputWordRate	resulting sample rate (real sample rate divided by the reducing of filters, if used)

Output parameters

0 on success, an negative error number if an error has occurred

IfmSetBlockModeFilter

Syntax

int IfmSetBlockModeFilter(int devNumber, unsigned int filterFlags,int avg1, int avg2)

Description

This function is similar to *IfmSetFilter* but works for the block mode.

Input parameters

see *IfmSetFilter*

Output parameters

0 on success, an negative error number if an error has occurred

IfmSetBlockModeFilterCoeff

Syntax

int IfmSetBlockModeFilterCoeff(int devNumer, int channel, double coeff)

Description

This function sets the FIR filter coefficient for the build in filter in the firmware of the RE-10 for the block mode. See *IfmSetFilterCoeff* for more information.

IfmStartBlock

Syntax

int IfmStartBlock(int devNumber, int blockLen)

Description

This function requests a block of the length *blockLen.* Before calling it, the DLL has to be switched to the block mode by *IfmSetBlockMode*. The block mode persists until the next *IfmStart*, so after *IfmSetBlockMode* *IfmStartBlock* can be called multiple times.

The maximum block length depends on the amount of channels and data types. For one channel with only length values, the maximum value of *blockLen* is 65535.

A second channel reduces the available *blockLen* to 32768 samples, if additional SIN/COS-values are requested (as required for an Lissajous display for signal quality) the *blockLen* is also reduced by the factor two.

Input parameters

devNumber	Unique ID for the device
blockLen	length of the block in samples

Output parameters

0 on success, an negative error number in case of an error

4 Functions for the controlling of the interferometers

The SIOS devices can contain up to four channel. Every channel has got a separate sensor and control system. In the *siosifm.dll* there are a lot of functions, which are used for the monitoring of the states for the available channels of connected devices and for the setting of the options for the service purposes.

For the inspection of signal conditions are the functions *IfmNewSignalQualityAvailable, IfmSignalQuality* scheduled.

The functions *IfmGetAGC, IfmGetRefMirrorVibration, IfmSetRefMirrorVibration, IfmSetAGC* are for the controlling of the mirror vibration and of the controller state. Each interferometer channel has a variable gain controller which can adjust the input amplifier gain to the amplitude of the signals from the sensor head (and also the offsets). This function is called AGC and can be switched on and off for each channel. The gain control is only possible if there are signal changes over at least one interference (lambda/2 = 316nm). To have such changes also if the sensor head is at rest or moves only very slow, most of the SIOS sensor head have build in a so called modulator or “reference mirror vibrator”. This is a piezo plate which excites the reference arm of the sensor to swing with a frequency between 500 and 1000Hz. With low output rates this reference arm vibrations are filtered out very well so the modulator can be left on. With high speed measurements the modulator usually should be switched off after the device is warmed up.
IfmSetRefMirrorVibration is used to control the modulator.

Please note, for the ***adjusting*** of the laser signal the mirror vibration modulator and AGC should be *set on.*

The functions *IfmWasBeamBreak, IfmWasLaserUnstable, IfmWasLostValues* deliver the information about possible error sources in the interferometer.

IfmGetAGC

Syntax

int IfmGetAGC(int devNumber,int channel)

Description

This function delivers the state of the controller for the according device and channel.

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)

Output parameters

zero for the state “controller is off”

or

1 for the state “controller is on”

IfmGetRefMirrorVibration

Syntax

int IfmGetRefMirrorVibration(int devNumber,int channel)

Description

This function delivers the state of the mirror vibration for the according device and channel.

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)

Output parameters

zero for the state “vibration is off”

or

1 for the state “vibration is on”

IfmNewSignalQualityAvailable

Syntax

int IfmNewSignalQualityAvailable(int devNumber)

Description

The device sends to the PC the service informations about the quality of the laser signals. The period can be vary. Typical value is 500 ms. With the function *IfmNewSignalQualityAvailable* can be tested whether the new quality signal values are arrived. See also *IfmSignalQuality* for detail information about the quality signals.

Input parameters

devNumber Unique ID for the device

Output parameters

0 : signal quality parameters are not available

1 : signal quality parameters are available

Example

```
typedef struct
{
   int A1;
   int O1;
   int A2;
   int O2;
} QualityParameterType;

..........

void DisplaySignalQuality(int channelCount)
{
     float x,y;

     if(IfmNewSignalQualityAvailable(devNo))
     {
```

```
for (int channel=0; channel<channelCount; channel++)
   {
            QualityParameterType qParam1;
            qParam1.A1=IfmSignalQuality(devNo,channel,IFM_SIGNALQ_A1);
            qParam1.A2=IfmSignalQuality(devNo,channel,IFM_SIGNALQ_A2);
            qParam1.O1=IfmSignalQuality(devNo,channel,IFM_SIGNALQ_O1);
            qParam1.O2=IfmSignalQuality(devNo,channel,IFM_SIGNALQ_O2);
            showSignalQuality(channel, qParam1);
        }
}
```

Explanation:

The function *DisplaySignalQuality* tests, whether the new signal quality values are available. If it is true, the new parameters for the required channels will be read out and passed to the function *showSignalQuality.*

IfmSetAGC

Syntax

int IfmSetAGC(int devNumber,int channel, int on)

Description

This function sets the state of the controller for the according device and channel. The switching on of the controller has got a sense only if the the mirror vibration (modulator) is on too. See also *IfmSetRefMirrorVibration, IfmGetRefMirrorVibration, IfmGetAGC.*

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)
on	=0 means “the set the controller off”
	=1 means “the set the controller on”

Output parameters

0 on success, an negative error number in case of an error

IfmSetRefMirrorVibration

Syntax

int IfmSetRefMirrorVibration(int devNumber,int channel, int on)

Description

The modulator on the interferometers generates a low-amplitude sinusoidal motion of the mirror on their reference arm that will be needed at times, particularly when aligning them (Cf. the instruction manual for the interferometer involved for instructions on aligning it).Their modulators may be left switched on during many types of measurements, and will have to be switched off only if unfiltered measurement data is to be recorded, since measurement errors might occur if they are left switched on.

This function sets the state of the mirror vibration for the according device and channel.

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)
on	=0 means “the set the modulator off”
	=1 means “the set the modulator on”

Output parameters

0 on success, an negative error number in case of an error

IfmSignalQuality

Syntax

int IfmSignalQuality(int devNumber,int channel, int select)

Description

The function delivers the required value dependent on the input parameters.

The sampling period can be different. The typical value is 500 ms.

Please note, for the ***adjusting*** of the laser signal the mirror vibration modulator and amplifier gain controller should be set *on* (See *IfmSetRefMirrorVibration* and *IfmSetAGC*)

Input parameters

devNumber Unique ID for the device

channel channel number (from 0 up to 3)

select possible selection (defined in siosifmdef.h):

Name	Description	Meaning
IFM_SIGNALQ_A1	Amplitude of the sine-signal	Amplifier gain: 0: the best signal quality 255: the worst signal quality
IFM_SIGNALQ_O1	Offset of the sine-signal*	Offset position: 127 (=50%): the best position 0, 255: the worst positions
IFM_SIGNALQ_A2	Amplitude of the cosine-signal	Amplifier gain: 0: the best signal quality 255: the worst signal quality
IFM_SIGNALQ_O2	Offset of the cosine-signal*	Offset position: 127 (=50%): the best position 0, 255: the worst positions
IFM_SIGNALQ_SUM	Overall signal	Amplitude signal quality in (0..100)% in both channels together. The best value: 100%
IFM_SIGNALQ_FREQ	Frequency of the reference mirror vibrator	Frequency in Hz

* A difference more than 40% between the sine-offset and cosine-offset signals indicates problems in the optic adjustment. For the monitoring of the Lissajous figure please use the according software “SignalMonitor.exe” or an oscilloscope. Please use the SIOS user's guide for the device or contact a SIOS support for the solving of the problems.

See also *IfmNewSignalQualityAvailable*

Output parameters

Required value

Example

See *IfmNewSignalQualityAvailable*

IfmStatus

Syntax

int IfmStatus(int devNumber,int channel)

Description

This function delivers the information about the device status as well as about the possible measuring error sources.

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)

Output parameters

The flags, which can be set (defined in “siosifmdef.h”):

The lower 16 bit are channel depended

IFM_STATUS_BEAMBREAK_QUADRANT	miscount detected: the interferometer counter has detected an invalid jump over more than one quadrants
IFM_STATUS_BEAMBREAK_LEVEL	the signal amplitude is lower than a given threshold so that miscounts are likely
IFM_STATUS_LASER_STABLE	the laser(s) is(are) stable (only in systems with stabilized lasers)
IFM_STATUS_LASER_WAS_UNSTABLE	since last *IfmSetToZero* the laser was at least one time unstable

The upper 16 bit are independent from the channel

IFM_STATUS_BUFFER_OVERFLOW_DEV	the FIFO in the interferometer had an overflow; data loss by to large sample rate has occurred

IFM_STATUS_BUFFER_OVERFLOW_DLL	the measurement value buffer in the DLL had an overflow; data loss by infrequent call of *IfmGetValues*
IFM_STATUS_BLOCKMODE	the device is in block mode
IFM_STATUS_BAD_REQUEST	the status request could not be answered, perhaps due to bad *deviceNumber* or invalid *channel*

Some bits in the status bit field indicate that the measurement in invalid and are reset only during *IfmSetToZero.* But they could be cleared also by the user using the *IfmDeviceInfo* command with the parameter IFM_DEVINFO_RESETSTATUS.

See also *IfmWasBeamBreak, IfmWasLaserUnstable, IfmWasLostValues*

IfmWasBeamBreak

Syntax

int IfmWasBeamBreak(int devNumber,int channel)

Description

This function checks whether the laser beam was broken in the according device and sensor channel. It can cause a failed measurement.

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)

Output parameters

zero for the state “the beam was not broken”

or

1 for the state “the beam was broken”

IfmWasLaserUnstable

Syntax

int IfmWasLeaserUnstable(int devNumber,int channel)

Description

This function checks whether the laser was unstable in the according device and sensor channel. An unstable laser is an error source for a measurement. This option is relevant only for the stabilized laser.

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)

Output parameters

zero for the state “the laser was stable”

or

1 for the state “the laser was unstable”

IfmWasLostValues

Syntax

int IfmWasLostValues(int devNumber)

Description

This function checks whether the measuring values ware lost due to buffer overflow in the PC or device. For the getting of exact information see *IfmStatus.*

Input parameters

devNumber	Unique ID for the device
channel	channel number (from 0 up to 3)

Output parameters

zero for the state “no measuring values were lost”

or

1 for the state “the measuring values were lost”

5 Functions for the communication with other devices

Inside the interferometer most of the cards are connected via an I2C-bus. This section describes functions for the communication with these cards via this bus. Basically each card provides a memory area which can be written and read via I2C interface. The meaning of the different cells in the I2C-memory area of the different cards is described in the appropriate documentation of the card. Because of the meaning can also change with the firmware version of the cards their usage requires a good documentation and a possible frequently update of the software.

This functions builds also the base API for a lot of other functions (for example motor control, configuration etc.).

Don't use it without a suitable documentation about the target memory

5.1 General functions

General functions are low-level functions for the access to the devices on the I2C-bus by read/write operations or for the access to the input buffer after read-operation.

The functions are available in two versions: blocking and non blocking.

IfmI2CRead and *IfmI2CWrite* block until the operation is terminated which may take typically up to 200 ms. We recommend to use these function because their usage is easy.

IfmI2CRequestRead and *IfmI2CRequestWrite* return immediately but the availability of the results must be polled. It's more complicated but the functions doesn't block and prevent freezing of the user interface. We recommend these function for sophisticated users.

IfmI2CRead

Syntax

int IfmI2CRead(int devNumber, int i2cAddr, int ramAddr, int count)

Description

This function requests a block of data with a length of maximum 160 bytes from the I2C memory of a card in the interferometer.

The data will be written into an internal buffer on the PC. The read operation will take an unpredictable time, typically between 100 and 200 milliseconds, while the function is waiting.

The internal buffer can be accessed via the the *IfmI2CReadBuffer* function.
It can occur that the reading operation cannot be done, for example due to high traffic on the I2C bus. It is important to check the return value before reading out the buffer.

Input parameters

devNumber	Unique ID for the device
i2cAddr	the address of the card inside the interferometer
ramAddr	the address inside the accessible memory of the card
count	the amount of bytes that should be read

Output parameters

The function returns en error number and a zero in case of success.

Example

```
// read out a block of 4 bytes from the card with the I2C-Address 0x70
int error=IfmI2CRead(devNumber,0x70,0,4);
if(!error){
  unsigned char *b=IfmI2CReadBuffer(devNumber);
  printf("I2C read: %2.2X  %2.2X %2.2X %2.2X\n",b[0],b[1],b[2],b[3]);
}
```

IfmI2CRequestRead

Syntax

int IfmI2CRequestRead(int devNumber, int i2cAddr, int ramAddr, int count)

Description

This function requests a block of data from the I2C memory of a card of the interferometer.

The data will be written into an internal buffer on the PC. The read operation will take an unpredictable time, typically between 100 and 200 milliseconds.

If the data are written into the internal buffer successfully, it can be polled with the *IfmI2CReadReady* function.
The internal buffer can be accessed via *IfmI2CReadBuffer* function.
It can occur that the reading operation cannot be done, for example due to high traffic on the I2C bus. So it should be set up a time out of approximately 200ms, after which the read request should be considered as unsuccessfully, if the requested info is not available.

Input parameters

devNumber	Unique ID for the device
i2cAddr	the address of the card inside the interferometer
ramAddr	the address inside the accessible memory of the card
count	the amount of bytes that should be read

Output parameters

The function returns an error number or zero on success.

Example

See *IfmI2CReadBuffer*

IfmI2CReadBuffer

Syntax

unsigned char *IfmI2CReadBuffer(int devNumber)

Description

The function gives access to the read buffer after the read operation from a I2C-device (*see IfmI2CRead, IfmI2CRequestRead*).

Input parameters

devNumber Unique ID for the device

Output parameters

Pointer to a unsigned char field with the length of the requested memory block.

IfmI2CReadValue

Syntax

unsigned char *IfmI2CReadValue(int devNumber, int index)

Description

The function gives access to the read buffer after the read operation from a I2C-device (*see IfmI2CRead, IfmI2CRequestRead*).

Input parameters

devNumber Unique ID for the device

Output parameters

Unsigned char from the *index* of requested memory block

Example

```
// read out a block of 4 bytes from the card with the I2C-Address 0x70
int error=IfmI2CRead(devNumber,0x70,0,4);
if(!error){
  printf("I2C read: %2.2X  %2.2X %2.2X %2.2X\n",
     IfmI2CReadValue(devNumber,0),
     IfmI2CReadValue(devNumber,1),
     IfmI2CReadValue(devNumber,2),
     IfmI2CReadValue(devNumber,3));
}
```

IfmI2CReadReady

Syntax

int IfmI2CReadReady(int devNumber)

Description

Test if a request to read (see *IfmI2CRequestRead)* was successfully and the data can be read out (see *IfmI2CReadBuffer)*.

Input parameters

devNumber Unique ID for the device

Output parameters

1: the read operation was successfully

0: the read operation is yet in progress or was not successfully

IfmI2CRequestWrite

Syntax

int IfmI2CRequestWrite(int devNumber, int i2cAddr, int ramAddr, int count, unsigned char* buffer)

Description

This function does exactly the same as *IfmI2CWrite* but it doesn't wait until the writing operation has been executed.

IfmI2CStatus can be used to check if and when the writing operation was successfully executed.

IfmI2CStatus

Syntax

int IfmI2CStatus(int devNumber)

Description

This function returns the status of the last I2C-bus command.

0	The I2C bus is free for usage and the last operation was successful.
IFM_ERROR_I2C_INUSE	is returned while the I2C subsystem waits for the completion of a read or write command. With new requests is should be wait until this status has changed.
IFM_ERROR_I2C_TIMEOUT	indicates an error. The last command could not be processed during the given time. It is possible but not likely that the command was successfully after the timeout period.
IFM_ERROR_I2C_WRITE	is returned when the last write command has failed.
IFM_ERROR_I2C_READ	is returned when the last read command has failed.

Please refer also to the list of error codes in the chapter *Error codes.*

IfmI2CWrite

Syntax

int IfmI2CWrite(int devNumber, int i2cAddr, int ramAddr, int count, unsigned char* buffer)

Description

This function writes a block of data with a maximum size of 160 bytes to the I2C memory of a selected card. It will take some milliseconds to complete the operation.

Input parameters

devNumber	Unique ID for the device
i2cAddr	the address of the card to be written
ramAddr	the address inside the accessible memory of the card
count	the amount of bytes that should be written,
buffer	the pointer to the data that should be written

Output parameters

The function returns an error number or zero on success.

Example

```
int WriteValueToI2CDevice(){
  unsigned char b[4];
  b[0]= 0xFF;
  b[1]= 0xFA;
  b[2]= 0x01;
  b[3]= 0xC1;
  return  IfmI2CWrite(devNumber, 0x38, 0xAA, 4, b);
}
```

Explanation: At first in the function *WriteValueToI2CDevice* will be initialized a buffer with 4 bytes. This buffer will be sent to a device with the I2C-address 0x38. In this device the data will be written to the internal memory address 0xAA, 0xAB, 0xAC,0xAD.

5.2 Functions for the motor control card

not yet implemented

6 Functions for the environment values

A SIOS-interferometer can contain up to 4 cards (UW10, UW11 and so on) for measuring of environment values like temperature, humidity and air pressure. Every card can have up to 6 sensors, usually 4 tempeatures, air pressure and humidity. In case of the wireless temperaure module there can be another 15 temperature sensors. The environment values will be periodically requested by the RE-10 or R-06 card and automatically forwarded to the PC. These values will be used for the correction of the environment influences to the measurement values (Edlen-correction) or can be used for user-specific purposes.

The values of the sensors can be read by *IfmSensorValue. IfmSensorProperty* says which type of sensor it concerns. This function informs also about the interferometer channels to which the sensor belongs to. For iterating overt the environmental sensors the overall sensor count may be usefully: See *IfmEnvSensorCount.*

The environment values which belong to the Edlen correction of a measurement channel can be read by the functions: *IfmAirPressure, IfmHumidity, IfmTemperature.* Additional values are calculated for the correction process (*IfmWaterVapourPressure, IfmWavelength).* The (device dependent) vacuum wavelength is a basis parameter for the interferometric length measurement and is stored in the memory of the the device.

When the SIOS-device doesn't have any environment cards or if no sensor is attached default values for temperature (20°C), air pressure (101300 Pa) and humidity (50%) are used because it is assumed that the measurement is made in air and a refraction correction (Edlen correction) is necessary. For applications in vacuum the Edlen correction can be switched off by *IfmEnableEdlenCorrection.*

If the user wants to use his own climate measurement, the environment values can be set (*IfmSetAirPressure, IfmSetHumidity, IfmSetTemperature* as well as *IfmSetWaterVapourPressure*). In this case the DLL ignores the according measured values from the interferometer.

The calculations rules on basis of Edlen-principle are represented in following equations*.

For the refraction correction of the influences of the temperature *t* and pressure *p* will be calculated as follows:

*see G. Boensch and E. Potulski *Fit of Edlen's formulae to measured values of the refractive index of air*

$$(n-1)_{tp}=\frac{2.877555143880165\cdot 10^{-9}\cdot p}{93214.60}\cdot\frac{1+10^{-8}\cdot(0.5953-0.009876\cdot t)\cdot p}{1+0.0036610\cdot t}$$

where *t* is current temperature in °C, *p* current air pressure in Pa.

As next will be the influences of humidity as partial pressure of water vapour (Magnus formula)

$$p_{\varphi}=\frac{h}{100}\cdot 6.11213\cdot\exp\left(\frac{17.5043\cdot t}{241.2+t}\right)$$

h is the current humidity in %.

The last step for the correction of the environment influences is the correction of wave length:

$$\lambda=\frac{\lambda_0}{1+n_{tp}-p_{\varphi}\cdot 3.7\cdot 10^{-10}}$$

λ_0 is the wave length in vacuum. Usually this parameter is stored in the device configuration.

There is also a possibility to read/write of the scaling factor by the functions *IfmConversionCoeff* and *IfmSetConversionCoeff.* This factor is used for conversion of the interferometer counter values to the length values.

On every measuring value is additionally applied the dead path correction. Dead path is the space between the interferometer's sensor head (*Pos.1* at Figure 4) and the zero point for the measurements (*Pos. 2* at Figure 4) to be performed. Since changes in ambient conditions change the refractive index of ambient air, and thus change the laser wavelength, entire stretch between its sensor head and the moving mirror must be taken into account when computing corrections to the laser wavelength.

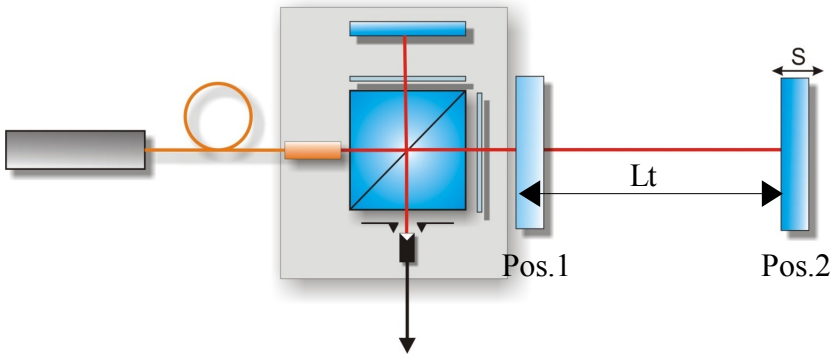


Figure 4: Dead path correction

By zero sitting on the *Pos.1* is the *Lt*=0.

The calculation rule for the dead path correction describes the following equation:

$$L_{corr} = -L_t \cdot (1 - \frac{n_0}{n}) \quad \text{, nm}$$

n_0 is the air refraction at the time where *IfmSetToZero* was called.

n is the actual air refraction.

The setting of the dead path value is provided by the function *IfmSetDeadPath* . By *IfmDeadpathCoeff* can be read this setting . Like prementioned this function should be applied directly **before** the function *IfmSetToZero.*

The finally equation for the calculating of the length values in nanometers describes the following term:

$$L = -L_{corr} + k \cdot x_{adc} \quad \text{, nm}$$

k is the scaling factor, which depends on the environmental values, the gain of the selected filters and a device type specific factor.

x_{adc} Is a measured counter value.

IfmAirPressure

Syntax

double IfmAirPressure(int devNumber, int channel)

Description

This function returns the currently used air pressure value for the given *channel.* If it not set manually, the measured air pressure is returned. If no air pressure could be measured (no environment card or no sensor connected) a default value of 101300 Pa is returned.

See also: *IfmSetAirPressure*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The air pressure value in Pa for required device and channel.

IfmAirPressureFlags

Syntax

int IfmAirPressureFlags(int devNumber, int channel)

Description

This function returns the information about the sensor mask, source and state of the measured air pressure value.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The following flags can be set:

IFM_ENVIRFLAG_SENSORMASK	in this byte is coded the sensor number (from 0 up to 23) from which sensor comes the measured value
IFM_ENVIRFLAG_MEASURED	the value was measured; when this flag is not set, it's a default value
IFM_ENVIRFLAG_CURRENT	the value was measured with the last data set (in a typical configuration it's not older than 4 seconds)
IFM_ENVIRFLAG_MANUAL	value was given manually by *IfmSetAirPressure* function

This flags are in the “siosifmdef.h” defined.

Example

```
void GetAirPressureValue(int devNumber,int channel){
     int flags=IfmAirPressureFlags(devNumber,channel);
     printf("The air pressure sensor is on the position %d\n",
            (flags & IFM_ENVIRFLAG_SENSORMASK ));
```

```
    if (flags & IFM_ENVIRFLAG_MEASURED)
      printf("Value is measured\n");
    else if (flags & IFM_ENVIRFLAG_MANUAL)
      printf("Value is set manual\n");
    else printf("Default value\n");

    double value=IfmAirPressure(int devNumber, int channel);
    printf("Current air pressure value: %f\n", value);
}
```

Explanation:

At the beginning the procedure *GetAirPressureValue* calls the function *IfmAirPressureFlags.* The delivered information will be decoded and printed. Then the actual air pressure value will be required by *IfmAirPressure* and printed.

lfmAirRefraction

Syntax

double lfmAirRefraction(int devNumber, int channel)

Description

This function returns the actual calculated air refraction index for the given channel.

See also the description of the environment correction: *Functions for the environment values*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The air refraction index for requested device and channel.

lfmConversionCoeff

Syntax

double lfmConversionCoeff(int devNumber, int channel)

Description

This function returns the currently used conversion factor. It's the multiplier to calculate the length in nanometres from the interferometer counts and it depends on the environmental values, the gain of the selected filters and a device type specific factor. The conversation factor is calculated automatically by the DLL but the user can set it also manually. See also: *lfmSetConversionCoeff* and two last the equations in the part *Functions for the environment values*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

Current conversion coefficient of the requested device and channel.

lfmDeadpathCoeff

Syntax

double lfmDeadpathCoeff(int devNumber, int channel)

Description

This function returns the currently used dead path coefficient. It's the summand to dead path correction of the environment corrected length in nanometres.

See also: *lfmSetDeadPath* and two last equations in the part *Functions for the environment values.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

Current dead path coefficient of the requested device and channel.

lfmEnableEdlenCorrection

Syntax

void lfmEnableEdlenCorrection(int devNumber, int channel, int on)

Description

Via this function can be enabled or disabled the correction for the refraction index of the air and the true wavelength of the laser light as the reference for the length measurement (the so called Edlen-correction). If the parameter *on* is set to 0 , the correction will not be applied and the user gets the uncorrected measurement values like for a measurement in vacuum. Otherwise the correction will be made. See also: *lfmIsEdlenEnabled.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number
on	1=enable, 0=disable

Output parameters

none

IfmEnvSensorCount

Syntax

unsigned int IfmEnvSensorCount(int devNumber)

Description

This function returns the count of the sensors which are currently configured for use with the interferometer. The configuration is stored in the device. Normally this value is required for iterating over the sensors with *IfmSensorProperty* and *IfmSensorValue.*

The maximum count is 24 (4 cards with maximum 6 sensors each).

See also *IfmSensorProperty.*

Input parameters

devNumber	Unique ID for the device

Output parameters

if returning value >=0	the count of the sensors in all available environment cards
otherwise	Error code

lfmHumidity

Syntax

double lfmHumidity(int devNumber, int channel)

Description

This function returns the currently used humidity value for the given channel. If it not set manually, the measured humidity is returned. If no humidity could be measured (no environment card or no sensor connected) a default value of 50% is returned.

See also *lfmSetHumidity*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The actual humidity value in % for the requested device and channel.

IfmHumidityFlags

Syntax

int IfmHumidityFlags(int devNumber, int channel)

Description

This function returns the information about the sensor mask, source and state of the measured humidity value.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The following flags are possible:

IFM_ENVIRFLAG_SENSORMASK	in this byte is coded the sensor number (from 0 up to 23) from which sensor comes the measured value
IFM_ENVIRFLAG_MEASURED	the value was measured; when this flag is not set, it's a default value
IFM_ENVIRFLAG_CURRENT	the value was measured with the last data set (in a typical configuration it's not older than 4 seconds)
IFM_ENVIRFLAG_MANUAL	value was given manually by *IfmSetHumidity* function

This flags are in the “siosifmdef.h” defined.

See also a similar example in the part *IfmAirPressureFlags*

IfmIsEdlenEnabled

Syntax

int IfmIsEdlenEnabled(int devNumber,int channel)

Description

This function will return 1, if the Edlen-correction is enabled, otherwise it will return 0.

See also *IfmEnableEdlenCorrection*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

1	if the correction is enabled
0	if the correction is disabled

lfmGetDeadPath

Syntax

int lfmGetDeadPath(int devNumber, int channel)

Description

The function returns the active dead path for the given channel in mm. Because the dead path is set before *lfmSetToZero* and the dead path can be changed in the interferometer (for instance in Laser-tracers), the returned value may be different from the value set by *lfmSetDeadPath*

The dead path is always positive. A negative value is an error number.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The dead pass value if positive or 0
an error code if negative

IfmNewEnvValuesAvailable

Syntax

int IfmNewEnvValuesAvailable(int devNumber)

Description

This function will return 1, if the new measured environment values are available. Then this values can be read by the functions: *IfmAirPressure, IfmHumidity* and *IfmTemperature.*

Input parameters

devNumber	Unique ID for the device

Output parameters

1	if the new values are available
0	otherwise

Example

```
double P;
double H;
double T1,T2;
  if (IfmNewEnvValuesAvailable(devNumber)){
       P=IfmAirPressure(0,0);
       H=IfmHumidity(0,0);
       T1=IfmTemperature(0,0);
       T2=IfmTemperature(0,1);
       .....
  }
```

Explanation:

The device has got two channels (0 and 1). There are only one sensor for air pressure and humidity but individual temperature sensors for each channel.

After testing for new environment values (*IfmNewEnvValuesAvailable*), all environment values will be read out.

IfmResetManualEnvironment

Syntax

void IfmResetManualEnvironment(int devNumber,int channel)

Description

After setting user environment values (*IfmSetTemperature*, *IfmSetAirPressure*, *IfmSetHumidity*, *IfmSetConversionCoeff)* this function switches back the environment values to the values measured by the interferometer. It means, all values (temperature, air pressure, humidity, water vapour as well as coefficients), that were set manually by functions *IfmSetTemperature*, *IfmSetAirPressure*, *IfmSetHumidity*, *IfmSetConversionCoeff* will be refused and the DLL takes the measured values for the calculation of Edlen-correction and the scaling coefficient for the conversion the digits to length values.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

none

IfmSensorProperty

Syntax

unsigned int IfmSensorProperty(int devNumber, int sensor)

Description

This function queries the types of the environment sensors, that are available in the whole device with the unique ID *devNumber.* The device can contain up to 24 environment sensors. The returned value has got a length of 1 byte. The mapping of this byte represents the Figure 5:

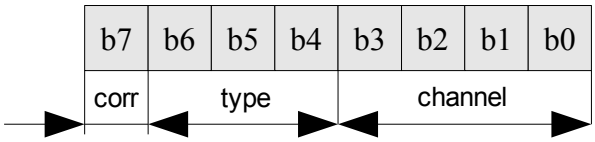


Figure 5: The structure of the returned sensor property

corr is a bit, that indicates, whether the sensor's measuring values will be taken for the Edlen-correction

type are three bits for the sensor type description (see output parameters)

channel are four bits, that shows the channel belonging (see output parameters)

The matched constants are defined in the header “siosifmdef.h”.

Input parameters

devNumber	Unique ID for the device
sensor	Sensor number (0..23)

Output parameters

corr – bit:

0	The value will not be taken for Edlen correction

1	The value will be taken for Edlen correction

type of the sensor:

1	Temperature sensor
2	Humidity sensor
3	Air pressure sensor

Channel-flags:

0	The sensor is not associated with according channel *
1	The sensor is associated with according channel *

* The bit number is equal the channel number. So bit 0 (b0 on the Figure 5) shows belonging to channel 0, bit 1: to channel 1 and so on.

The constants from the header “siosifmdef.h”:

- Type of the Sensor:

identifier:	**meaning**
IFM_ENVIR_SENSOR_TEMP	0x10
IFM_ENVIR_SENSOR_HUMIDITY	0x20
IFM_ENVIR_SENSOR_AIRPRESSURE	0x30

- Channel belonging:

IFM_ENVIR_CHANNEL1	0x01
IFM_ENVIR_CHANNEL2	0x02
IFM_ENVIR_CHANNEL3	0x04
IFM_ENVIR_CHANNEL4	0x08

- Flag to mark the values for Edlen-correction

IFM_ENVIR_EDLEN	0x80

Example

```
int tempSensorCounter=0;
unsigned int tempSensorID[4];
```

```
unsigned int humiditySensorID;
unsigned int airPressureSensorID;

void  searchEnviromentsSensorIDs(){
unsigned int idn=0;
int N=IfmEnvSensorCount(devNr);
if (N>0)
{
  for (int ii=0;ii<N;ii++){
    idn=IfmSensorProperty(devNr, ii);
    if (idn & IFM_SENSOR_MASK)==IFM_ENVIR_SENSOR_TEMP ) {
      tempSensorID[tempSensorCounter]=ii;
      tempSensorCounter++;
      }
    if (idn & IFM_SENSOR_MASK)==IFM_ENVIR_SENSOR_HUMIDITY)
      humiditySensorID=ii;

    if ((idn & IFM_SENSOR_MASK)==IFM_ENVIR_SENSOR_AIRPRESSURE )
      airPressureSensorID=ii;

    }
  }
}
```

Explanation: In a SIOS-Unique ID for the device *devNr* are six environment sensors available: four temperature sensors, one humidity sensor and one air pressure sensor. With the procedure *searchEnviromentsSensorIDs* will be searched the identifiers of these sensors. At the begin will be tested, how many sensors are really available (function *IfmEnvSensorCount*). If this function returns a *value*>0, the identifiers will saved to the according value.

IfmSensorValue

Syntax

double IfmSensorValue(int devNumber, int sensor)

Description

This function returns the measured environment value according to the sensor number. See *IfmSensorProperty* for detail information.

Input parameters

devNumber	Unique ID for the device
sensor	Sensor number

Output parameters

Requested value.

IfmSetAirPressure

Syntax

void IfmSetAirPressure(int devNumber, int channel, double value)

Description

This function is used for the manual setting of the air pressure value.

Attention! The dll takes this value for the Edlen-correction. The measured value (if available) will be ignored, until the user calls IfmResetManualEnvironment or the dll will be reinitialized again.

If the user does not set the manual value and the measured values are not available, the dll takes the default value 101300 Pa. The current value can be read by *IfmAirPressure.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number
value	Air pressure value to be set

Output parameters

none

IfmSetConversionCoeff

Syntax

void IfmSetConversionCoeff(int devNumber, int channel, double value)

Description

This function is used for the manual setting of the coefficient for the conversion of the digits to the length values for the according device and channel. The default value will be ignored, until the user calls *IfmResetManualEnvironment* or the dll will be reinitialized again.

The actual coefficients can be read by *IfmConversionCoeff.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number
value	Coefficient to be set

Output parameters

none

IfmSetDeadPath

Syntax

int IfmSetDeadPath(int devNumber, int channel, int deadPath)

Description

This function is used for the setting of the dead path value. Usually this value should be set directly **before** the command *IfmSetToZero* because this setting will be transferred to the device together with the settings for the zero point.

Input parameters

devNumber	Unique ID for the device
channel	Channel number
deadPath	dead path value in mm

Output parameters

The function returns an error number or zero on success.

IfmSetHumidity

Syntax

void IfmSetHumidity(int devNumber, int channel, double value)

Description

This function is used for the manual setting of the humidity value for the Edlen correction for the according device and channel.

Attention! The dll takes this value for the Edlen-correction. The measured value (if available) will be ignored, until the user calls IfmResetManualEnvironment or the dll will be reinitialized again.

If the user dos not set a manual value and the measured values are not available, the dll will take the default value 50% for the correction. The current value can be read by *IfmHumidity.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number
value	Value to be set

Output parameters

none

IfmSetTemperature

Syntax

void IfmSetTemperature(int devNumber, int channel, double value)

Description

This function is used for the manual setting of the temperature value for the Edlen correction for the according device and channel.

Attention! The dll takes this value for the Edlen-correction. The measured value (if available) will be ignored, until the user calls IfmResetManualEnvironment or the dll will be reinitialized again.

If the user dos not set the manual value and the measured values are not available, the dll will take the default value 20°C for the correction. The current value can be read by *IfmTemperature.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number
value	Value to be set

Output parameters

none

lfmSetWavelength

Syntax

void lfmSetWavelength(int devNumber, int channel, double value)

Description

This function is used for the manual setting of the corrected wave length value for the Edlen correction for the according device and channel.

Attention! The dll takes this value for the Edlen-correction. The measured value (if available) will be ignored, until the user calls lfmResetManualEnvironment or the dll will be reinitialized again.

The wavelength is the result of the Edlen-correction, which is based on the vacuum wavelength and the environmental values temperature, air pressure and humidity. If the wavelength is set manually, the Edlen-correction will be overwritten.

Input parameters

devNumber	Unique ID for the device
channel	Channel number
value	Value to be set

Output parameters

none

IfmSetWaterVapourPressure

Syntax

void IfmSetWaterVapourPressure(int devNumber, int channel, double value)

Description

This function is used for the manual setting of the vapour pressure value for the Edlen correction for the according device and channel.

Attention! The dll takes this value for the Edlen-correction. The by default calculated value will be ignored, until the user calls IfmResetManualEnvironment or the dll will be initialized again.

The actual value can be read by *IfmWaterVapourPressure.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number
value	Value to be set

Output parameters

none

IfmTemperature

Syntax

double IfmTemperature(int devNumber, int channel)

Description

This function returns the currently used temperature value for the given channel. If not set manually, the measured temperature is returned. If no temperature could be measured (no environment card or no sensor connected) a default value of 20 °C is returned.

This function returns only the temperature value of sensors which are dedicated to an interferometer channel for the Edlen correction. The values of other temperature sensors, like sensors for the material temperature, must be requested by *IfmSensorValue,* whereas the sensor place must be known.

See also *IfmSetTemperature.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The requested temperature value in °C for the corresponding device and channel.

IfmTemperatureFlags

Syntax

int IfmTemperatureFlags(int devNumber, int channel)

Description

This function returns the information about the sensor mask, source and state of the measured temperature value.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The following flags are possible:

IFM_ENVIRFLAG_SENSORMASK	in this byte is coded the sensor number (from 0 up to 23) from which sensor comes the measured value
IFM_ENVIRFLAG_MEASURED	the value was measured; when this flag is not set, it's a default value
IFM_ENVIRFLAG_CURRENT	the value was measured with the last data set (in a typical configuration it's not older than 4 seconds)
IFM_ENVIRFLAG_MANUAL	value was given manually by *IfmSetTemperature* function

This flags are in the “siosifmdef.h” defined.

See also a similar example in the part *IfmAirPressureFlags*

lfmVacuumWavelength

Syntax

double lfmVacuumWavelength(int devNumber, int channel)

Description

This function returns currently used vacuum wave length value, that is saved in the configuration of the devices.

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The the requested value for the corresponding device and channel.

IfmWaterVapourPressure

Syntax

double IfmWaterVapourPressure(int devNumber, int channel)

Description

This function returns currently used water vapour pressure value, that was calculated on the basis of the environment values or was set by user with the function *IfmSetWaterVapourPressure.*

Input parameters

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The the requested value for the corresponding device and channel.

lfmWavelength

Syntax

double lfmWavelength(int devNumber, int channel)

Description

This function returns corrected wave length value, that was calculated on the basis of Edlen-correction.

devNumber	Unique ID for the device
channel	Channel number

Output parameters

The the requested value for the corresponding device and channel.

7 Extended functions

IfmAuxValue

Syntax

int IfmAuxValue(int devNumber,int channel,int valueType)

Description

The function returns specific measurement values, like sine, cosine, AD-conversion values as digits, test pattern, from the input buffer. Usually this function will be used in the same manner as *IfmLengthValue* after one of the functions *IfmGetRecentValues* or *IfmGetValues.*

Please note: To get one of the special values the transmission of these values must be requested with the *IfmSetMeasurement* function. Otherwise always zero is returned.

Input parameters

devNumber	Unique ID for the device
channel	Channel number
valueType	Type of the requested information. Following types are possible:
IFM_VALUETYPE_SIN	sine will be needed for Lissajous-figure
IFM_VALUETYPE_ADC1	AD-conversion values (from ADC1)
IFM_VALUETYPE_COS	cosine will be needed for Lissajous-figure
IFM_VALUETYPE_ADC2	AD-conversion values (from ADC2)
IFM_VALUETYPE_NORM	magnitude of the laser signal
IFM_VALUETYPE_TESTPATTERN	test pattern (device specific, the values 0x00FF, 0xFF00, 0x5555, 0xAAAA are possible)
IFM_VALUETYPE_CLOCKCOUNT	returns the value of a 50 MHz clock counter as a time stamp for the sample
IFM_VALUETYPE_COUNTER	same as IFM_VALUETYPE_CLOCKCOUNT
IFM_VALUETYPE_SAMPLECOUNT	returns the samplecount, a running number for each sample,

	can be used to detect missing samples
IFM_VALUETYPE_PSD_X	lateral displacement measured with a PSD in 10nm steps x direction
IFM_VALUETYPE_PSD_Y	y direction
IFM_VALUETYPE_PSDRAW_X	PSD value without normalisation, normal range is between -0x2000 and +0x2000 x direction
IFM_VALUETYPE_PSDRAW_Y	y direction
IFM_VALUETYPE_PSD_SUM	energy on the PSD

Output parameters

the requested parameter

Example

```
IfmSetMeasurement(devNo,IFM_MEAS_LENGTH|IFM_MEAS_SINCOS|
IFM_MEAS_FILTER_NONE,10000);
IfmStart(devNo);
...
int count=IfmValueCount(devNo);
if(count>0){
     int x,y;
     for(int i=0;i<count;i++){
     IfmGetRecentValues(devNo,i);
     x=IfmAuxValue(devNo,0,IFM_VALUETYPE_SIN);
     y=IfmAuxValue(devNo,0,IFM_VALUETYPE_COS);
     lissajousWindow->SetPoint(x,y);
   }
}
```

Explanation: At the beginning the program tests, whether the device with unique ID f*devNo* has sent measurement values. If the *IfmValueCount* returns the *value*>0, the values will be read out from the input puffer (function *IfmGetRecentValues*) in according to LIFO (last-in-first-out) principle. The sine and cosine values will be picked up from the DLL-buffer and forwarded to the object lissajousWindow.

IfmChannels

Syntax

int IfmCannels(int devNumber)

Description

The function returns the count in the device available channels.

Input parameters

devNumber	Unique ID for the device

Output parameters

Required information if the answer >=0

or

error code otherwise

IfmDeviceInfo

Syntax

int IfmDeviceInfo(int devNumber, int requestedInfo)

Description

The function requests current device parameters.

The requested parameter is returned as an integer or – if this is not possible or makes no sense – as a pointer to the parameter. Please read the following description to typecast to the right type.

Depending of the device type or the firmware version different information may not be available. In this case an 0 is returned. Please check the return value, especially if a pointer should be returned.

Input parameters

devNumber	Unique ID for the device
requestedInfo	Following parameters can be requested:
IFM_DEVINFO_SRATE	current sample rate (the sample rate before the reducing filters)
IFM_DEVINFO_OUTRATE	output word rate, sample rate after the filter stage, may slightly differ from the requested rate in *IfmSetMeasurement* due to limited divider resolution
IFM_DEVINFO_FILTERFLAGS	filter flags, like set by *IfmSetFilter*
IFM_DEVINFO_MEASUREMENTFLAGS	measurement flags, like set by *IfmSetMeasurement*
IFM_DEVINFO_TRIGGERMODE	trigger mode, like set by *IfmSetTrigger* or or saved in the flash
IFM_DEVINFO_AVG1	filter option for the average 1, like set by *IfmSetFilter*
IFM_DEVINFO_AVG2	filter option for the average 2, like set by *IfmSetFilter*
IFM_DEVINFO_CMDDELAY	delay in ms between the transmission of two commands

IFM_DEVINFO_AVAILABLE	After opening the device the DLL reads out the settings from the device. This parameter signals with a return code of 1, that the configuration is available.
IFM_DEVINFO_SERIALNUMBER	returns the USB-serial number; like *IfmUSBDeviceSerial* but with an open device
IFM_DEVINFO_READY	returns 1 if the device is ready for start the measurement
IFM_DEVINFO_I2CTIMEOUT	returns the timeout in ms for I2C operations
IFM_DEVINFO_VERSIONSTRING	returns the firmware version number (an integer !! between 1 and 255) of the RE-10
IFM_DEVINFO_FPGAVERSION	Version of the programmable hardware (FPGA)
IFM_DEVINFO_OUTRATE_POINTER	Pointer (!) to a double value, containing the real OWR in higher resolution
IFM_DEVINFO_LINKEDCHANNELS1	wich channels are linked to calculate an angle pair 1, see IfmAngleValue
IFM_DEVINFO_LINKEDCHANNELS2	wich channels are linked to calculate an angle pair 2, see IfmAngleValue
IFM_DEVINFO_BASEDISTANCE1_POINTER	base distance in mm of the linked channels, for pair 1 returns a pointer to a double value
IFM_DEVINFO_BASEDISTANCE2_POINTER	base distance in mm of the linked channels, for pair 2 returns a pointer to a double value
IFM_DEVINFO_LAST_I2C_CARDADDR	the card address of the active or last I2C request
IFM_DEVINFO_LAST_I2C_MEMADDR	memory address of the active or last I2C request
IFM_DEVINFO_SRATE_POINTER	returns a pointer to a double value, which holds the exact sample rate
IFM_DEVINFO_PSD_COUNT	count of the available PSD-Sensors (lateral shift sensors)
IFM_DEVINFO_PSD_TYPE	type of PSD implementation 0: no PSD available

1: PSD-04 cards, slow data rate up to 4 Hz
2: special ADU card with higher sample rate

Output parameters

requested parameter, if it is available and the function is correct

or

0 otherwise

Example

```
int deviceDelay= IfmDeviceInfo(0,  IFM_DEVINFO_CMDDELAY);
```

Explanation:

The value *deviceDelay* contains the information about the delay between transmitted commands in device with unique ID 0.

IfmDeviceInterface

Syntax

int IfmDeviceInterface(int devNumber)

Description

The function returns the device interface, which the device is connected to.

Input parameters

devNumber Unique ID for the device

Output parameters

The device types are defined in “siosifmdef.h”

IFM_INTERFACE_NONE	No information about interface type
IFM_INTERFACE_DEMO	The demo-application runs
IFM_INTERFACE_RS232	RS232-interface is configured
IFM_INTERFACE_USB	USB-interface is configured
IFM_INTERFACE_NET	Ethernet-interface is configured

IfmDeviceType

Syntax

int IfmDeviceType(int devNumber)

Description

The function returns the device type.

Input parameters

devNumber Unique ID for the device

Output parameters

The device types are defined in “siosifmdef.h”

IFM_TYPE_NONE	No information about the device type
IFM_TYPE_DEMO	The demo-application runs
IFM_TYPE_RE10	The device type is the RE-10-card
IFM_TYPE_RE06	The device type is the RE-06-card

IfmDeviceValid

Syntax

int IfmDeviceValid(int devNumber)

Description

The function checks whether the device ID *devNumber* is correct.

Input parameters

devNumber	Unique ID for the device

Output parameters

0	the device dos not exist
1	the device is ok

IfmDLLVersionString

Syntax

const char *IfmDLLVersionString()

Description

The function returns a pointer to a char field with the DLL name and version number.

Input parameters

No input parameters

IfmFireTrigger

Syntax

int IfmFireTrigger(int devNumber)

Description

The function returns one measuring value.

Input parameters

devNumber Unique ID for the device

Output parameters

Required information

IfmFirmwareVersion

Syntax

int IfmFirmwareVersion(int devNumber)

Description

The function returns the version number of the firmware for required device.

Input parameters

devNumber Unique ID for the device

Output parameters

Required information

IfmGetError

Syntax

int IfmGetError()

Description

The function returns the first error number after last calling of this function.

See also the part *Error codes* and *IfmGetErrorString.*

Input parameters

No input parameters.

Output parameters

Error number.

IfmGetErrorString

Syntax

const char *IfmGetErrorString(int errorNumber)

Description

The function returns the a pointer to a char field with the description of the error with required *errorNumber.*

See also the part *IfmGetError.*

Input parameters

errorNumber a negative integer value

Output parameters

A pointer to a char field.

IfmRawValue

Syntax

int64 IfmRawValue(int devNumber,int channel)

Description

Normally the measurement values from the interferometer are read out in nm. But in principle the interferometer measures the displacement in counts. This function returns the raw counts before they are converted in nm.

Usually this function will be used in the same manner as *IfmLengthValue* after one of the functions *IfmGetRecentValues* or *IfmGetValues.*

Input parameters

devNumber Unique ID for the device

channel Channel number

Output parameters

0: no values are available

or

Requested value

Example

```
if (IfmValueCount(devNo)>0)
{
  IfmGetRecentValues(devNo,0 );
  int64 value_channel _0=    IfmRawValue(devNo,0);
  int64 value_channel _1=    IfmRawValue(devNo,1);
}
```

Explanation: if the measuring values are available (*IfmValueCount(devNo)*>0) , the last counter values for channel 0 and 1 will be read out.

IfmResetDevice

Syntax

void IfmResetDevice(int devNumber)

Description

The function causes the software reset of the according card.

Alike after hardware reset, the device stops after this command all running processes and breaks all connections. The USB-interface have to be reinitialized and the device have to be reconfigured by user again.

Input parameters

devNumber Unique ID for the device

Output parameters

No output parameters

IfmSetDeviceInfo

Syntax

int IfmSetDeviceInfo(int devNumber, int infoNo, int newValue)

Description

This function sets some device parameters which can be read by *IfmDeviceInfo.* It can be used instead of *IfmSetFilter* and *IfmSetTrigger* and additionally for the setting of other parameters like IFM_DEVINFO_CMDDELAY. For the setting of trigger and filter options this function should be called before *IfmSetMeasurement.*

Input parameters

devNumber	Unique ID for the device
newValue	The value to be setted
infoNo	The parameter id which should be set
newValue	The new value for the parameter

The following parameters can be set: IFM_DEVINFO_OUTRATE, IFM_DEVINFO_FILTERFLAGS, IFM_DEVINFO_MEASUREMENTFLAGS, IFM_DEVINFO_TRIGGERMODE, IFM_DEVINFO_AVG1, IFM_DEVINFO_AVG2, IFM_DEVINFO_CMDDELAY, IFM_DEVINFO_I2CTIMEOUT.

For the meaning of these parameters please refer *IfmDeviceInfo.*

With IFM_DEVINFO_RESETSTATUS the device status can be reset. *newValue* is a mask for the bits that should be cleared.

Output parameters

Zero on success or an error number is returned.

Example

```
int error= IfmSetDeviceInfo(0,  IFM_DEVINFO_CMDDELAY, 200);
```

Explanation:

this function sets the delay=200 ms between transmitted command.

IfmSetOption

Syntax

int IfmSetOption(int option, int param1)

Description

The function modifies some behaviour of the library. It should be called before *IfmInit* to take effect.

Input parameters

option: Following parameters can be requested:

IFM_ OPTION_DEBUGFILES

If *param1* is set to 1 the library creates some text files with debug informations

IFM_OPTION_POLLSELF

Normally the library starts with IfmInit a thread and calls *IfmPoll* frequently in this thread. If *param1* is set to 1, no thread is started and *IfmPoll* must be called by the application. **Use with care!**

IFM_OPTION_BLOCKONCLOSE

Normally a device will be destroyed some times after closing it. If *param1* is set to 1 *IfmCloseDevice* will wait until the device is closed and destroyed before returning.

IFM_OPTION_BEAMBREAK

For IfmWasBeamBreak exists two sources of beam break detection. Both sources are visible in the IfmStatus flags: IFM_STATUS_BEAMBREAK_QUADRANT (0x01) and IFM_STATUS_BEAMBREAK_LEVEL (0x03). This option allows to mask the beambreak sources. For instance, if after IfmSetOption(IFM_OPTION_BEAMBREAK,IFM_STATUS_BEAMBREAK_LEVEL) only the “Level criteria” of the beam break detection is used for IfmWasBeamBreak.

IFM_OPTION_ZEROPHASE

The raw value (see IfmRawValue) consists of a counter part (counts the mimima in the interference pattern) and a phase part (the interpolated position inside an interface pattern). For special applications after calling IfmSetOption(IFM_OPTION_ZEROPHASE,0) the phase part (lower 16 bit of the raw value) is untouched by IfmSetToZero and only the counter part is cleared.

Output parameters

Zero on success or an error number is returned.

8 Functions for the service

IfmSaveConfigDevice

Syntax

void IfmSaveConfigDevice(int devNumber)

Description

The function saves the current measurement settings into the flash.

The saved parameter will be applied after the next device reset.

This function is only for service purposes. Please don't use.

Input parameters

devNumber Unique ID for the device

Output parameters

There are no return parameters

IfmSetMeasurementRawValue

Syntax

int IfmSetMeasurementRawValue(int devNumber,unsigned int measurementFlags, int outputWordRate)

Description

Alike the function *IfmSetMeasurement* is the function *IfmSetMeasurementRawValue* for the setting of measurement parameters. But in this function all filter settings will be ignored and the firmware works in a transparent mode. It means, the sample frequency and the output word rate are equal.

This function should be called directly **before** the measurement start (see *IfmStart*).

The intended use is for testing purposes only. Please don't use it. For normal reasons *IfmSetMeasurement* with the measurement flag IFM_FILTER_NONE will do the same.

IfmUpdateDevice

Syntax

void IfmUpdateDevice(int devNumber)

Description

The function IfmUpdateDevice causes the update-mode of the card.

Attention!!! *After starting of the update-mode it is not possible without the firmware - upgrade to set back the device to the run-mode. A SIOS-bootloader software should be applied for the upgrading of the firmware.*

This function should only be used by bootloader programs!

Input parameters

devNumber Unique ID for the device

Output parameters

There are no return parameters.

9 Error codes

Symbolic constant	value	Explanation
IFM_ERROR_NONE	0	The function was executed successfully. No error has occurred.
IFM_ERROR_DEVICE_INVALID	-1	The function has tried to access a device which is not valid. The devNumber parameter was wrong (out of range, device was not opened, device was closed...)
IFM_ERROR_BAD_CHANNEL	-2	The channel number was wrong. Up to four channels are supported for each device, so channel numbers can be 0,1, 2 or 3.
IFM_ERROR_BAD_ DEVICETYPE	-3	The DLL supports more device types than the RE-10 for which it was created. But this operation is not possible with the given device because of the type doesn't support it. Some operations can for instance only be made with RE-10 devices.
IFM_ERROR_DATALEN	-4	The requested block of data is too large. It may be device dependent what amount is supported.
IFM_ERROR_UNKNOWN	-6	An error with a non specified case has occurred.
IFM_ERROR_DEVICECOUNT_ OVERFOW	-10	The maximum amount of opened devices has exceeded. It's not likely that this occurs under normal conditions. But there is a delay in closing a device an destroying the internal structures. Be sure not to open/close the same device in a fast loop.

Symbolic constant	value	Explanation
IFM_ERROR_BAD_ REQUESTTYPE	-11	You have requested an information which is not available.
IFM_INVALID_USB_ID	-12	The id to identify the USB device (found with IfmSearchUSBDevices) is not valid.
IFM_ERROR_CREATE_ HANDLE	-13	The device could not be opened. Perhaps the resource (com-number, device file ..) is not present or is already in use.
IFM_ERROR_NOT_ IMPLEMENTED	-14	This function exists as prototype but is not yet implemented. Update to a newer version of the library.
IFM_ERROR_I2C_IN_USE	-15	The I2C subsystem is already in use. Wait some time, before requesting an I2C-operation.
IFM_ERROR_I2C_WRITE	-16	The write operation was not successful. This can have several reasons, permanent (target card not present) or temporal (I2C-bus busy, target card busy). Try it again at least one time.
IFM_ERROR_I2C_TIMEOUT	-17	The I2C operation could not be carried out in the given time.
IFM_ERROR_OWR_TO_HIGH	-18	The requested output word rate (OWR) was too high or the resulting sample rate (with user filter settings) was too high.
IFM_ERROR_INFO_NOT_ AVAILABLE	-19	The requested information is not yet available. Try it again later.
IFM_BAD_SENSOR	-20	The sensorNumber in this function is invalid.

Symbolic constant	value	Explanation
IFM_ERROR_I2C_READ	-21	During a read operation occurs an error. Try again later.
IFM_ERROR_BAD_ PARAMETER	-22	A parameter of the function was out of range or doesn't make sense