

# SCC1-RS485/USB Sensor Cable

## Driver DLL Documentation

# Common DLL Commands for SF04-based Liquid Flow Sensors

### Summary

This document describes the communication with Sensirion's SF04 based liquid flow sensors products via the SCC1-RS485 and SCC1-USB Sensor Cables and its dedicated Microsoft Windows driver DLL.

The communication between this DLL and the RS485 sensor hardware is based on the product's Sensirion-HDLC commands. (see separate documentation)

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>4</b>
1.1	Required Files	4
1.2	Applicability of Commands to Different Sensor Types	4
<b>2</b>	<b>SHDLCDRIVER.DLL FUNCTION REFERENCE</b>	<b>5</b>
2.1.1	OpenPort()	5
2.1.2	ClosePort()	5
2.1.3	DeviceReset()	5
2.1.4	GetDeviceAddress ()	6
2.1.5	SetDeviceAddress ()	6
<b>3</b>	<b>SENSORCABLEDRIVER.DLL FUNCTION REFERENCE</b>	<b>7</b>
3.1	Sensor Cable Commands	7
3.1.1	GetTermination ()	7
3.1.2	SetTermination ()	7
3.1.3	GetSensorVoltage ()	8
3.1.4	SetSensorVoltage ()	8
3.1.5	GetSensorType ()	8
3.1.6	SetSensorType ()	9
3.1.7	GetSensorAddress ()	9
3.1.8	SetSensorAddress ()	9
3.2	Measurement Functions	10
3.2.1	GetSensorStatus()	10
3.2.2	StartSingleMeasurement()	11
3.2.3	GetSingleMeasurementSigned/Unsigned()	11
3.2.4	StartContinuousMeasurement()	12
3.2.5	GetContinuousMeasurementStatus()	13
3.2.6	StopContinuousMeasurement()	13
3.2.7	GetLastMeasurementSigned/Unsigned()	14
3.2.8	GetLastMeasurementWithoutClearSigned/Unsigned()	14
3.2.9	GetMeasurementBufferSigned/Unsigned()	15
3.2.10	GetExtendedBufferSigned/Unsigned()	15
3.2.11	GetExtendedBufferSize()	16

3.2.12 ClearBuffer()	16
3.2.13 SetTotalizatorStatus ()	16
3.2.14 ResetTotalizator ()	17
3.2.15 GetTotalizatorValue ()	17
<b>3.3 Sensor settings Functions</b>	<b>18</b>
3.3.1 GetMeasurementType()	18
3.3.2 SetMeasurementType()	18
3.3.3 GetResolution()	18
3.3.4 SetResolution()	19
3.3.5 GetCalibField ()	19
3.3.6 SetCalibField ()	19
<b>3.4 Sensor Infos Functions</b>	<b>20</b>
3.4.1 GetSensorPartName()	20
3.4.2 GetFlowUnit()	20
3.4.3 GetFlowUnitString()	21
3.4.4 GetScaleFactor()	21
3.4.5 GetSensorSerialNumber()	22
3.4.6 GetMeasurementDataType()	22
<b>3.5 Advanced Sensor Functions</b>	<b>22</b>
3.5.1 SensorSoftReset()	22
<b>4 DLL ERROR CODES</b>	<b>23</b>
4.1 Common Device Errors	23
4.2 Device Errors	23
4.3 Common System Errors	23
4.4 System Errors	24
<b>5 DATA TYPES</b>	<b>24</b>
<b>6 SAMPLECODE</b>	<b>25</b>
6.1 C++ Sample Code	25
6.2 C# Sample Code	26
<b>REVISION HISTORY</b>	<b>27</b>

# 1 INTRODUCTION

This document describes the use of the 32Bit and 64Bit C-dlls to communicate with the RS485 Sensor Cable.

## 1.1 REQUIRED FILES

To use the SHDLC Driver with the RS485 Sensor Cable, the following files are required in the same directory as the .exe file:

- ShdlcDriver.dll
- SensorCableDriver.dll

## 1.2 APPLICABILITY OF COMMANDS TO DIFFERENT SENSOR TYPES

The SHDLC command reference (see separate document) lists in a table which commands apply to which sensor type. Supported sensor types include SHTxx (Humidity and Temperature), SF04 (Flow), SF05 (Flow), SF06 (Flow).

This documentation contains the most relevant commands for use with SF04-based liquid flow sensors. For a complete list of the commands available in the DLL, see the complete DLL command documentation available from your Sensirion contact.

## 2 SHDLCDRIVER.DLL FUNCTION REFERENCE

The functions in this chapter are in the file ShdlcDriver.dll.

### 2.1.1 OPENPORT()

#### Description

Opens the desired port and initializes the DLL.

#### Prototype

```
u32t OpenPort(u8t aPortType, char* aPortConfig, u32t* aPortHandle);
```

Parameter	Meaning
aPortType	Defines which kind of port should be opened: - 0: Serial (RS232, RS485,...)
aPortConfig	String which defines the port configuration. The string format depends on the used port type: - 0 (Serial): "<ComPortName>,<Baudrate>,<EchoMode>" example: "COM1, 115200, EchoOn" EchoOn: Data sent by the master is also received by the master EchoOff: Data sent by the master is not received by the master
aPortHandle	Returned port handle

### 2.1.2 CLOSEPORT()

#### Description

Closes a port.

#### Prototype

```
u32t ClosePort(u32t aPortHandle);
```

Parameter	Meaning
aPortHandle	Handle of the port

### 2.1.3 DEVICERESET()

#### Description

Perform a Reset on the Device, i.e. the Sensor Cable. This will also hard-reset any attached sensor.

#### Prototype

```
u32t DeviceReset(u32t aPortHandle, u8t aSlaveAdr);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address

## 2.1.4 GETDEVICEADDRESS ()

### Description

Returns the RS485 device Address.

### Prototype

```
u32t GetDeviceAddress(u32t aPortHandle, u8t aSlaveAdr, u8t* aAddress);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aAddress	Returned slave address

## 2.1.5 SETDEVICEADDRESS ()

### Description

Set new RS485 address of the SHDLC device. If the current address of the device is unknown, a new address can be set by using this command with broadcast mode=1.

IMPORTANT NOTE: Use this command in broadcast mode only with one single device on the bus, otherwise all devices will get the same address.

### Prototype

```
u32t SetDeviceAddress(u32t aPortHandle, u8t aBroadcastMode, u8t aSlaveAdr,
u8t aAddress);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast 0: normal (send command to aSlaveAdr and receive return) 1: Broadcast send (send with broadcast address (aSlaveAdr is ignored , no response from any slave is received. Allow sufficient time for the execution of the command before sending the next command, see the SHDLC command reference for details.) 2: Get last response (Get slave response to the previously sent broadcast command from aSlaveAdr)
aSlaveAdr	Slave address
aAddress	New address to be set

## 3 SENSORCABLEDRIVER.DLL FUNCTION REFERENCE

The functions in this chapter are in the file SensorCable.dll.

### 3.1 SENSOR CABLE COMMANDS

#### 3.1.1 GETTERMINATION ()

##### Description

Get the current setting (enabled/disabled) of the internal termination resistor (120 Ohm).

##### Prototype

```
u32t GetTermination(u32t aPortHandle, u8t aSlaveAdr, u8t* aTermination);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aTermination	Returned Termination setting: 0: Termination resistor disabled 1: Termination resistor enabled

#### 3.1.2 SETTERMINATION ()

##### Description

Enable / disable the internal termination resistor (120 Ohm).

##### Prototype

```
u32t SetTermination(u32t aPortHandle, u8t aSlaveAdr, u8t aTermination);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aTermination	Termination to be set: 0: Termination resistor disabled 1: Termination resistor enabled

### 3.1.3 GETSENSORVOLTAGE ()

#### Description

Get the sensor supply voltage setting

#### Prototype

```
u32t GetSensorVoltage(u32t aPortHandle, u8t aSlaveAdr,
                     u8t* aVoltageSetting);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aVoltageSetting	Returned voltage setting, 0: 3.5V, 1: 5V

### 3.1.4 SETSENSORVOLTAGE ()

#### Description

Set the sensor supply voltage.

#### Prototype

```
u32t SetSensorVoltage(u32t aPortHandle, u8t aSlaveAdr,
                     u8t aVoltageSetting);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aVoltageSetting	Voltage setting, 0: 3.5V, 1: 5V

### 3.1.5 GETSENSORTYPE ()

#### Description

Returns the Sensor Type selected on the device.

#### Prototype

```
u32t GetSensorType(u32t aPortHandle, u8t aSlaveAdr, u8t* aSensorType);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aSensorType	Returned Sensortype, 0: Flow Sensor (SF04 based products) 1: Humidity Sensor (SHTxx products) 2: Flow Sensor (SF05A based products) 3: Flow Sensor (SF06 based products)

### 3.1.6 SETSENSORATYPE ()

#### Description

Select new Sensortype.

#### Prototype

```
u32t SetSensorType(u32t aPortHandle, u8t aSlaveAdr, u8t aSensorType);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aSensorType	New Sensortype to be set 0: Flow Sensor (SF04 based products) 1: Humidity Sensor (SHTxx products) 2: Flow Sensor (SF05A based products) 3: Flow Sensor (SF06 based products)

### 3.1.7 GETSENSORADDRESS ()

#### Description

Get the I2C Sensor Address on the cable for communication between Sensor Cable and Sensor.

#### Prototype

```
u32t GetSensorAddress(u32t aPortHandle, u8t aSlaveAdr,  
u8t* aSensorAddress);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aSensorAddress	Returned I2C sensor address [0...127]

### 3.1.8 SETSENSORADDRESS ()

#### Description

Set the I2C Sensor Address on the cable for communication between Sensor Cable and Sensor.

#### Prototype

```
u32t SetSensorAddress(u32t aPortHandle, u8t aSlaveAdr, u8t aSensorAddress);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aSensorAddress	New I2C sensor address [0...127]

## 3.2 MEASUREMENT FUNCTIONS

### 3.2.1 GETSENSORSTATUS()

#### Description

Get the status of the sensor and continuous measurement. See the separate application note for a detailed description of the Auto-Detection Mode. Applies to sensor types 0, 1, 2, 3.

#### Prototype

```
u32t GetSensorStatus(u32t aPortHandle, u8t aSlaveAdr, u8t* aSensorStatus);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aSensorStatus	Returned status of sensor or device: Bit 0:     0: Sensor idle 1: Sensor Busy Bit 1:     0: Continuous Measurement disabled, Sensor is idle or in Detect Mode 1: Continuous Measurement enabled, Sensor is in Measurement Mode Bits 2...4: For advanced settings only.

### 3.2.2 STARTSINGLEMEASUREMENT()

#### Description

Start a single measurement. The result must be read out with command “GetSingleMeasurement” (3.2.3) for flow sensors or “GetSingleTempAndHumi” (**Error! Reference source not found.**) for SHTxx Sensor. Applies to sensor types 0, 1, 2.

#### Prototype

```
u32t StartSingleMeasurement(u32t aPortHandle, u8t aBroadcastMode,
                           u8t aSlaveAdr);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see also 2.1.5) 0: normal (send command to aSlaveAdr and receive return) 1: Broadcast send (send with broadcast address (aSlaveAdr is ignored , no response from any slave is received. Allow sufficient time for the execution of the command before sending the next command.) 2: Get last response (Get slave response to the previously sent broadcast command from aSlaveAdr)
aSlaveAdr	Slave address

### 3.2.3 GETSINGLEMEASUREMENTSIGNED/UNSIGNED()

#### Description

Read out single measurement with signed or unsigned data type. If measurement is not yet finished, error 1376 is returned. A single measurement must be started before, the completion of the measurement can be polled with this command. Applies to sensor types 0, 2.

#### Prototype signed

```
u32t GetSingleMeasurementSigned(u32t aPortHandle, u8t aSlaveAdr,
                                i16t* aMeasureResult);
```

#### Prototype unsigned

```
u32t GetSingleMeasurementUnsigned(u32t aPortHandle, u8t aSlaveAdr,
                                   u16t* aMeasureResult);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aMeasureResult	Returned measurement result as signed or unsigned integer.

### 3.2.4 STARTCONTINUOUSMEASUREMENT()

#### Description

Start continuous measurement with given interval. The measurements will be saved in a buffer. The newest 127 measurements can be read out with “Get Measurement Buffer” (3.2.9). Single measurements while continuous measurement can be read out with command “Get Last Measurement” (3.2.7). If buffering of more than 127 measurements is required, the GetExtendedBuffer (3.2.10) command has to be used. Any old measurements in buffer are cleared by the StartContinuosMeasurement command.

If the interval is set to zero, the measurement is as fast as possible. Otherwise the available minimum Interval depends on the actually set resolution in the sensor. Applies to sensor types 0, 2.

#### Prototype

```
u32t StartContinuousMeasurement(u32t aPortHandle, u8t aBroadcastMode,
                                u8t aSlaveAdr, u16t aInterval);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aInterval	Interval between measurements in ms 0: as fast as possible >0: Interval in ms, available minimum interval depends on the resolution. Sensor type 0: 9 Bit : min. 1ms 10 Bit : min. 2ms 11 Bit : min. 3ms 12 Bit : min. 6ms 13 Bit : min. 10ms 14 Bit : min. 20ms 15 Bit : min. 40ms 16 Bit : min. 80ms Sensor type 2: min. 1 ms for all settings.

### 3.2.5 GETCONTINUOUSMEASUREMENTSTATUS()

#### Description

Get the interval or/and status of the continuous measurement. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t GetContinuousMeasurementStatus(u32t aPortHandle, u8t aSlaveAdr,
                                     u8t* aStatus, u16t* aInterval);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aStatus	Returned status 0: continuous measurement disabled (aInterval not available) >1: continuous measurement enabled with aInterval
aInterval	Returned interval of continuous measurement, if started 0: as fast as possible >0: Interval in ms

### 3.2.6 STOPCONTINUOUSMEASUREMENT()

#### Description

Stop the continuous measurement after the current measurement is finished. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t StopContinuousMeasurement(u32t aPortHandle, u8t aBroadcastMode,
                                u8t aSlaveAdr);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address

### 3.2.7 GETLASTMEASUREMENTSIGNED/UNSIGNED()

#### Description

Read out last measurement while in continuous measurement. Use start Continuous Measurement (3.2.4) before using this command. If measurement is not started, not yet finished or no new measurement is available, error 1376 is returned. Applies to sensor types 0, 2, 3.

#### Prototype signed

```
u32t GetLastMeasurementSigned(u32t aPortHandle, u8t aBroadcastMode,
                             u8t aSlaveAdr, i16t* aMeasureResult);
```

#### Prototype unsigned

```
u32t GetLastMeasurementUnsigned(u32t aPortHandle, u8t aBroadcastMode,
                                u8t aSlaveAdr, u16t* aMeasureResult);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aMeasureResult	measurement result signed/unsigned. Note: for sensor type 3, only the first result in the sensor output (typically the flow measurement) is returned.

### 3.2.8 GETLASTMEASUREMENTWITHOUTCLEARSigned/UNSIGNED()

#### Description

(for Firmware  $\geq 1.4$ ) Read out last measurement during continuous measurement. Use start Continuous Measurement (3.2.4) before using this command. If continuous measurement is not started an error is returned. This command returns always the newest measurement, if no new value is available since the last call to this function, the same measurement is returned more than once. Before the first measurement is available, the function returns zero. Applies to sensor types 0, 2, 3.

#### Prototype signed

```
u32t GetLastMeasurementWithoutClearSigned(u32t aPortHandle,
                                           u8t aBroadcastMode, u8t aSlaveAdr, i16t* aMeasureResult);
```

#### Prototype unsigned

```
u32t GetLastMeasurementWithoutClearUnsigned(u32t aPortHandle,
                                             u8t aBroadcastMode, u8t aSlaveAdr, u16t* aMeasureResult);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aMeasureResult	Returned measurement result as signed or unsigned integer. Note: for sensor type 3, only the first result in the sensor output (typically the flow measurement) is returned.

### 3.2.9 GETMEASUREMENTBUFFERSigned/UNSIGNED()

#### Description

Read out the newest 127 measurements and clear all measurements from the buffer. Use the "Extended Buffer commands" to work with more than 127 buffered measurements. If the returned length is 0, no new measurements are available. Applies to sensor types 0, 2.

#### Prototype signed

```
u32t GetMeasurementBufferSigned(u32t aPortHandle, u8t aBroadcastMode,
                                u8t aSlaveAdr, i16t aMeasureResult[127],
                                u8t* aLength);
```

#### Prototype unsigned

```
u32t GetMeasurementBufferUnsigned(u32t aPortHandle, u8t aBroadcastMode,
                                   u8t aSlaveAdr, u16t aMeasureResult[127],
                                   u8t* aLength);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aMeasureResult	Pointer to array to write signed or unsigned measurements
aLength	Returned number of measurements

### 3.2.10 GETEXTENDEDBUFFERSigned/UNSIGNED()

#### Description

Read out measurements from the extended Buffer, if more than 127 measurements are available, the oldest 127 measurement will be read out and removed from the buffer. The other measurements can be read out with the next commands. Use this command several times to read out the complete extended buffer (maximum 1000 entries). Applies to sensor types 0, 2.

#### Prototype signed

```
u32t GetExtendedBufferSigned(u32t aPortHandle, u8t aBroadcastMode,
                              u8t aSlaveAdr, i16t aMeasureResult[127],
                              u8t* aLength);
```

#### Prototype unsigned

```
u32t GetExtendedBufferUnsigned(u32t aPortHandle, u8t aBroadcastMode,
                                u8t aSlaveAdr, u16t aMeasureResult[127],
                                u8t* aLength);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aMeasureResult	Pointer to array to write signed or unsigned measurements
aLength	Returned number of measurements

### 3.2.11 GETEXTENDEDBUFFERSIZE()

#### Description

Return the actual number of measurements in the extended buffer. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t GetExtendedBufferSize(u32t aPortHandle, u8t aBroadcastMode, u8t aSlaveAdr, u32t* aBufferSize);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aBufferSize	Returned number of measurements in extended Buffer

### 3.2.12 CLEARBUFFER()

#### Description

Clear all measurements from the buffer. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t ClearBuffer(u32t aPortHandle, u8t aBroadcastMode, u8t aSlaveAdr);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address

### 3.2.13 SETTOTALIZATORSTATUS ()

#### Description

Enable or disable the Totalizator. The value of the Totalizator is not changed with this command. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t SetTotalizatorStatus(u32t aPortHandle, u8t aBroadcastMode, u8t aSlaveAdr, u8t aStatus);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aStatus	Status of the Totalizator, 0: disabled, 1: enabled

### 3.2.14 RESETTOTALIZATOR ()

#### Description

Set the Totalizator value to zero, the Totalizator status (enabled/disabled) is not changed. The Totalizator can be reset anytime. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t ResetTotalizator(u32t aPortHandle, u8t aBroadcastMode,
                     u8t aSlaveAdr);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address

### 3.2.15 GETTOTALIZATORVALUE ()

#### Description

Get the value of the Totalizator. This value is the sum of all unscaled measurements while in continuous measurement mode. See the separate application note for details. Applies to sensor types 0, 2, 3.

Note that for sensor type 3 only the first measurement result (typically the flow measurement) is summed up in the totalizer.

#### Prototype

```
u32t GetTotalizatorValue(u32t aPortHandle, u8t aBroadcastMode,
                        u8t aSlaveAdr, i64t* aTotalizatorValue);
```

Parameter	Meaning
aPortHandle	Handle of the port
aBroadcastMode	Define mode for broadcast (see 3.2.2)
aSlaveAdr	Slave address
aTotalizatorValue	Value of Totalizer

## 3.3 SENSOR SETTINGS FUNCTIONS

### 3.3.1 GETMEASUREMENTTYPE()

#### Description

Get the type of measurement. Applies to sensor types 0, 2.

#### Prototype

```
u32t GetMeasurementType(u32t aPortHandle, u8t aSlaveAdr,
                        u8t* aMeasureType);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aMeasureType	Returned Measurement Type (0: Flow, 1: Temperature, 2: VDD)

### 3.3.2 SETMEASUREMENTTYPE()

#### Description

Set the type of measurement. Applies to sensor types 0, 2.

#### Prototype

```
u32t SetMeasurementType(u32t aPortHandle, u8t aSlaveAdr, u8t aMeasureType);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aMeasureType	Measurement Type (0: Flow, 1: Temperature, 2: VDD)

### 3.3.3 GETRESOLUTION()

#### Description

Get the resolution of the measurement. Applies to sensor types 0, 1, 2.

#### Prototype

```
u32t GetResolution(u32t aPortHandle, u8t aSlaveAdr, u8t* aResolution);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aResolution	Returned Resolution[9...16]

### 3.3.4 SETRESOLUTION()

#### Description

Set the resolution of the measurement. Applies to sensor types 0, 1, 2.

#### Prototype

```
u32t SetResolution(u32t aPortHandle, u8t aSlaveAdr, u8t aResolution);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aResolution	New Resolution to set [9...16]

### 3.3.5 GETCALIBFIELD ()

#### Description

Get the Calibration Field of the flow sensor. Applies to sensor type 0 only.

#### Prototype

```
u32t GetCalibField(u32t aPortHandle, u8t aSlaveAdr, u8t* aCalibField);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aCalibField	Returned Calibration Field setting [0...4]

### 3.3.6 SETCALIBFIELD ()

#### Description

Set the active Calibration Field of the flow sensor. Applies to sensor type 0 only.

#### Prototype

```
u32t SetCalibField(u32t aPortHandle, u8t aSlaveAdr, u8t aCalibField);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aCalibField	Calibration Field to set in sensor [0...4]

## 3.4 SENSOR INFOS FUNCTIONS

### 3.4.1 GETSENSORPARTNAME()

#### Description

Get the part name of the flow sensor. Applies to sensor types 0, 3.

#### Prototype

```
u32t GetSensorPartName(u32t aPortHandle, u8t aSlaveAdr,
                      char* aPartNameString, u32t aStringMaxSize);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aPartNameString	Location where to write the information string (min. length 21)
aStringMaxSize	Maximum number of characters allowed to write to the aPartNameString location (including null-character)

### 3.4.2 GETFLOWUNIT()

#### Description

Get the bit-encoded flow unit of the active calibration field of the flow sensor.  
Applies to sensor types 0, 2.

#### Prototype

```
u32t GetFlowUnit(u32t aPortHandle, u8t aSlaveAdr, u16t* aFlowUnit);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aFlowUnit	Returned encoded flow unit

### 3.4.3 GETFLOWUNITSTRING()

#### Description

Get the flow unit of the active calibration field of the flow sensor as a string. Applies to sensor types 0, 2.

#### Prototype

```
u32t GetFlowUnitString(u32t aPortHandle, u8t aSlaveAdr,
                      char* aFlowUnitString, u32t aStringMaxSize);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aFlowUnitString	Location where to write the string (min. length 10)
aStringMaxSize	Maximum number of characters allowed to write to the aFlowUnitString location (including null-character)

### 3.4.4 GETSCALEFACTOR()

#### Description

Get the scale factor of the sensor for the active measurement type and calibration field. Applies to sensor types 0, 2.

#### Prototype

```
u32t GetScaleFactor(u32t aPortHandle, u8t aSlaveAdr, u16t* aScaleFactor);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aScaleFactor	Returned scale factor.

### 3.4.5 GETSENSORSERIALNUMBER()

#### Description

Get the serial number of the sensor. Applies to sensor types 0, 2, 3.

#### Prototype

```
u32t GetSensorSerialNumber(u32t aPortHandle, u8t aSlaveAdr,
                          u32t* aSensorSerialNumber);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aSensorSerialNumber	Returned serial number

### 3.4.6 GETMEASUREMENTDATATYPE()

#### Description

Get the datatype of the flow measurements (signed or unsigned). Applies to sensor types 0, 2.

#### Prototype

```
u32t GetMeasurementDataType(u32t aPortHandle, u8t aSlaveAdr,
                           u8t* aDataType);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address
aDataType	Returned data type of measurements 0: signed (i16t) use ...Signed() Functions to read measurements 1: unsigned (u16t) use ...Unsigned() Functions to read measurements

## 3.5 ADVANCED SENSOR FUNCTIONS

### 3.5.1 SENSORSOFTRESET()

#### Description

Execute a hard reset with the sensor and check for correct response.

#### Prototype

```
u32t SensorSoftReset(u32t aPortHandle, u8t aSlaveAdr);
```

Parameter	Meaning
aPortHandle	Handle of the port
aSlaveAdr	Slave address

## 4 DLL ERROR CODES

### 4.1 COMMON DEVICE ERRORS

Error code	Error description
1	Device reported an illegal data size
2	Command not accepted from device
3	No access right on device for this command
4	Parameter out of range (report from device)

### 4.2 DEVICE ERRORS

Error code	Error description
32	command could not be executed because sensor is busy
33	Sensor gives no I2C acknowledge
34	CRC error while communication with sensor
35	Timeout of sensor while measurement
36	No measure is started

### 4.3 COMMON SYSTEM ERRORS

Error code	Error description
128	Fatal system error
129	In the Rx data stream, the start or stop byte (0x7E) is missing.
130	Too few bytes in Rx frame (frame content + checksum $\geq$ 5 bytes).
131	The transmitted data length information in the Rx frame does not match with the number of bytes received.
132	The port configuration string has an illegal format.
133	Could not open the COM port.
134	Could not close COM port.
135	Unknown communication type of communication port.
136	Incoming checksum was wrong.
137	The device command in the received frame is not the same as sent.
138	The returned number of Data is wrong for this command
139	Illegal broadcast mode
140	One of the given arguments has an illegal size.
141	The SerialPortOverlapped class reported an error.
142	Do not use the broadcast address when calling the transceive function.
143	The maximum number of open ports which could be handled by the DLL is reached.
144	The given port handle is not valid.

145	The requested functionality is not implemented yet.
146	An error occurred while calling a windows API function.
147	A timeout occurred while waiting for the RX data.
148	The function SerialPortOverlapped.WriteData() could not write all data.
149	The COM port is not open when trying to work with it (in SerialPortOverlapped).

## 4.4 SYSTEM ERRORS

Error code	Error description
1024	There is no connection to ShdlcDriver.dll (library or one of its functions could not be loaded).
1025	The returned number of Data is wrong for this command
1026	Illegal broadcast mode
1027	Wrong device command in response frame
1376	The current measure is not yet finished for read out

## 5 DATA TYPES

In the Documentation, an own notation for the different data types is used. Note that the DLL work with the little endian data format.

notation	C++ type	range
u8t	unsigned char	0 ... 255
i8t	signed char	-128 ... 127
u16t	unsigned int	0 ... 65535
i16t	signed int	-32768 ... 32767
u32t	unsigned long int	0 ... 4'294'967'295
i32t	signed long int	-2'147'483'648 ... 2'147'483'647
u64t	unsigned long long int	0 ... $2^{64}-1$
i64t	signed long long int	$-2^{63}$ ... $2^{63}-1$
ft	float	6 decimals
dt	double	10 decimals
bt	bool	1/0; true/false

## 6 SAMPLECODE

### 6.1 C++ SAMPLE CODE

```
#include <windows.h>
#include <stdio.h>

/*****
 * basic types: making the size of types clear
 *****/
typedef unsigned char    u8t;    ///< range: 0 .. 255
typedef signed char      i8t;    ///< range: -128 .. +127

typedef unsigned short   u16t;   ///< range: 0 .. 65535
typedef signed short     i16t;   ///< range: -32768 .. +32767

typedef unsigned long    u32t;   ///< range: 0 .. 4'294'967'295
typedef signed long      i32t;   ///< range: -2'147'483'648 .. +2'147'483'647

typedef unsigned __int64 u64t;   ///< range: 0 .. 2^64 - 1
typedef __int64          i64t;   ///< range: -2^63 .. 2^63 - 1

typedef float            ft;     ///< range: +-1.18E-38 .. +-3.39E+38
typedef double           dt;     ///< range: .. +-1.79E+308

typedef bool             bt;     ///< values: 0, 1 (real bool used)

// Definition of commands in common Dll
typedef u32t (__cdecl *FctOpenPort) (u8t aPortType, char* aPortConfig, u32t* aPortHandle);
typedef u32t (__cdecl *FctClosePort) (u32t aPortHandle);

// Definition of commands in Sensor Cable Dll
typedef u32t (__cdecl *FctGetSensorPartName) (u32t aPortHandle, u8t aSlaveAdr, char* aPartNameString, u32t
aStringMaxSize);

int _tmain(int argc, _TCHAR* argv[])
{
    FctOpenPort OpenPort;
    FctClosePort ClosePort;
    FctGetSensorPartName GetSensorPartName;

    // Get a handle to the ShdlcDriver DLL module.
    HINSTANCE CommonLib = LoadLibrary(TEXT("ShdlcDriver.dll"));

    // If the handle is valid, try to get the function address.
    if (CommonLib != NULL)
    {
        OpenPort = (FctOpenPort)GetProcAddress(CommonLib, "OpenPort");
        ClosePort = (FctClosePort)GetProcAddress(CommonLib, "ClosePort");
    }
    else
    {
        printf("ShdlcDriver.dll not found");
        getchar();
    }

    // Get a handle to the SensorCableDriver DLL module.
    HINSTANCE SensorCableLib = LoadLibrary(TEXT("SensorCableDriver.dll"));

    // If the handle is valid, try to get the function address.
    if (SensorCableLib != NULL)
    {
        GetSensorPartName = (FctGetSensorPartName)GetProcAddress(SensorCableLib, "GetSensorPartName");
    }
    else
    {
        printf("SensorCableDriver.dll not found");
        getchar();
    }

    u32t xError;
    u32t Connection;

    // open port
    xError = OpenPort(0, "COM11, 115200, ECHOOFF", &Connection);

    // Read SensorPartName from device at port Connection and Address 0
    char Partname[256];
    xError = GetSensorPartName(Connection, 0, Partname, 256);

    printf(Partname);
}
```

```

    getch();

    // close Port
    xError = ClosePort(Connection);

    return 0;
}

```

## 6.2 C# SAMPLE CODE

```

using System;
using System.Text;
using System.Runtime.InteropServices;

namespace CsSampleCode
{
    class Program
    {
        // Import of commands in common Dll
        [DllImport("ShdlcDriver.dll", EntryPoint = "OpenPort", CharSet = CharSet.Ansi,
            CallingConvention = CallingConvention.Cdecl)]
        public static extern UInt32 OpenPort(byte aPortType, string aPortConfig, out UInt32 aPortHandle);

        [DllImport("ShdlcDriver.dll", EntryPoint = "ClosePort", CharSet = CharSet.Ansi,
            CallingConvention = CallingConvention.Cdecl)]
        public static extern UInt32 ClosePort(UInt32 aPortHandle);

        // Import of commands in Sensor Cable Dll
        [DllImport("SensorCableDriver.dll", EntryPoint = "GetSensorPartName", CharSet = CharSet.Ansi,
            CallingConvention = CallingConvention.Cdecl)]
        public static extern UInt32 GetSensorPartName(UInt32 aPortHandle, byte aSlaveAdr,
            StringBuilder aSensorPartName, UInt32 aStringMaxSize);

        static void Main(string[] args)
        {
            UInt32 xPortHandle;
            UInt32 xError;

            // open Port
            xError = OpenPort(0, "COM1, 115200, ECHOOFF", out xPortHandle);

            // Read SensorPartName from device at port xPortHandle and Address 0
            StringBuilder xSensorPartName = new StringBuilder(256);
            xError = GetSensorPartName(xPortHandle, 0, xSensorPartName, 256);

            Console.WriteLine(xSensorPartName);
            Console.ReadLine();

            // close Port
            xError = ClosePort(xPortHandle);
        }
    }
}

```

# REVISION HISTORY

Date	Version	Change
Feb. 20	1	Initial version