MICRO-EPSILON

Interface Instructions

eddyNCDT 3005

RS485 to USB, Ethernet, EtherCAT, EtherNet/IP, PROFINET

Interface Instructions

MICRO-EPSILON
MESSTECHNIK
GmbH & Co. KG
Koenigbacher Strasse 15
94496 Ortenburg / Germany

Tel. +49 (0) 8542 / 168-0
Fax +49 (0) 8542 / 168-90
info@micro-epsilon.com
www.micro-epsilon.com

Contents

1. Introduction

The eddyNCDT 3005 measuring system uses eddy currents to precisely measure the distance, movement and position of metallic objects. The measured values can be read out both analog as voltage and digitally via the RS485 interface. These instructions describe how the RS485 interface can be connected to common interfaces (USB, EtherNet, EtherCAT, Ethernet/IP or PROFINET).

The PC software `sensorTOOL` allows you to configure the sensor and view the measured data.

You can also use the IF2035 converter from Micro-Epsilon to read out the measured data via PROFINET, EtherCAT or EtherNet/IP.

You can find more information about the IF2035 interface module in the operating instructions. They are available online at

https://www.micro-epsilon.com/industry-sensors/interfaces/if2035-for-industrial-ethernet/

You can also use the IF1032/ETH converter from Micro-Epsilon to read out the measured data via Ethernet or EtherCAT.

You can find more information about the IF1032/ETH in the operating instructions. They are available online at:

https://www.micro-epsilon.com/industry-sensors/interfaces/if1032-eth/

2. Pin Assignment

The converters can be used in conjunction with the cable PCx/5-M12 [1].

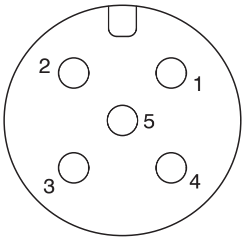
Pin	Description	PC5/5-M12	
1	Brown	12 ... 32 VDC	
2	White	Distance signal	
3	Blue	Ground	
4	Black	RS485 A / +	
5	Gray	RS485 B / -	View on connector side

Fig. 1 5-pole M12 A-coded male connector on controller

3. sensorTOOL Software

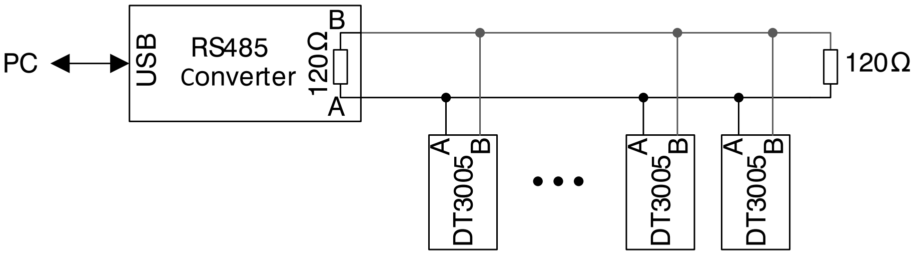


Fig. 2 Connection of the eddyNCDT 3005 to the PC using a USB/RS485 converter

A terminating resistor of 120 Ω is required between the A and B line of the RS485 interface at the start and end of the RS485 bus. A terminating resistor of the RS485 line is not incorporated in the DT3005. It is therefore allowed to connect several sensors to one bus cable.

`sensorTOOL` is a documented PC software package with which you can adjust the sensor as well as visualize and document measurement data.

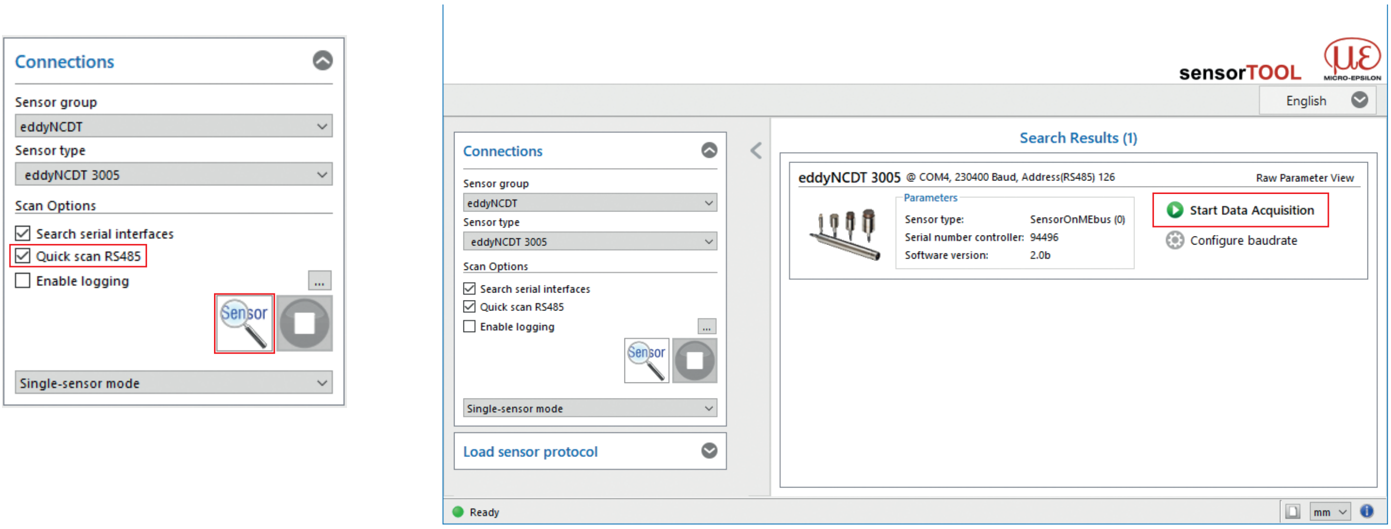
You can find this program online at https://www.micro-epsilon.com/download/software/sensorTOOL.exe.

➡ Connect the DT3005 controller with an USB to RS485 converter to a free USB port of your PC and connect the power supply to the DT3005.

➡ Start the `sensorTOOL` program.

➡ Set the `eddyNCDT` sensor group and the `eddyNCDT 3005 sensor type` in the drop-down menus.

[1] x = cable length in meters



Connections

Sensor group: eddyNCDT

Sensor type: eddyNCDT 3005

Scan Options

☑ Search serial interfaces

☑ Quick scan RS485

☐ Enable logging

Single-sensor mode

Search Results (1)

eddyNCDT 3005 @ COM4, 230400 Baud, Address(RS485) 126

Parameters

Sensor type: SensorOnMEbus (0)

Serial number controller: 94496

Software version: 2.0b

Ready

Fig. 3 First interactive site after calling the `sensorTOOL`

➡ Select the connected sensor.

➡ Check the box `Search serial interfaces.`

➡ If only 1 controller is operated on the ME bus, check the box next to `Quick scan RS485`.

In this case, all addresses are requested at the same time.

If several controllers are operated on the ME bus, the check mark for `Quick scan RS485` must be deactivated. The search then takes longer, as the program scans the entire address range individually, see also chapter `Configuration of Baud Rate`, see 3.2.

➡ Click on the `Sensor` button with the magnifying glass icon in order to start the search.

All available channels will now be displayed in the `Search Results (x)` overview.

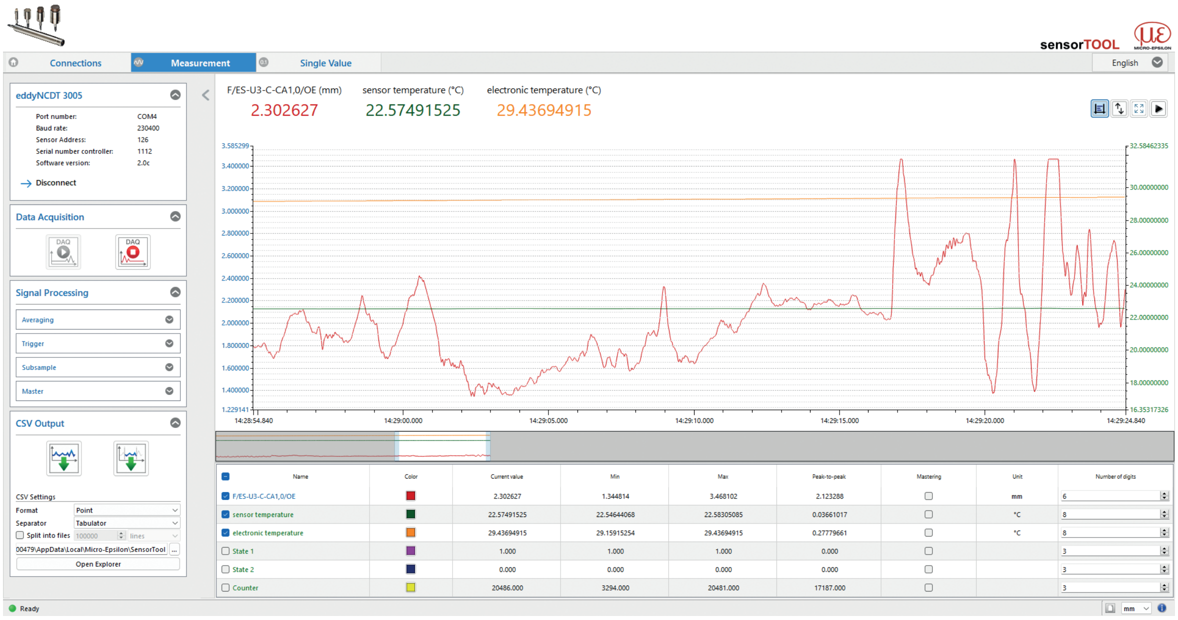
➡ Select a desired sensor.

Further menus can now be called up using the `Start Data Acquisition` and `Configure baudrate` buttons.

3.1 Measurement Menu

- ➡ Click on the `Start Data Acquisition` button or on the controller symbol, see Fig. 3 to make further settings and start data acquisition.

To check your measurements, a simple data acquisition is available.



eddyNCDT 3005

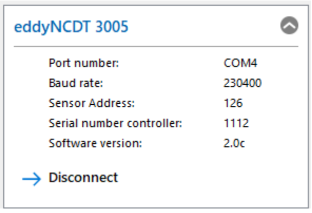
Port number: COM4
Baud rate: 230400
Sensor Address: 126
Serial number controller: 1112
Software version: 2.0c

F/ES-U3-C-CA1,0/OE (mm) 2.302627
sensor temperature (°C) 22.57491525
electronic temperature (°C) 29.43694915

CSV Settings
Format: Point
Separator: Tabulator
□ Split into files 100000 lines
00479\AppData\Local\Micro-Epsilon\SensorTool

Name	Color	Current value	Min	Max	Peak-to-peak	Mastering	Unit	Number of digits
☑ F/ES-U3-C-CA1,0/OE	🟥	2.302627	1.344814	3.468102	2.123288	□	mm	6
☑ sensor temperature	🟩	22.57491525	22.54644068	22.58305085	0.03661017	□	°C	8
☑ electronic temperature	🟧	29.43694915	29.15915254	29.43694915	0.27779661	□	°C	8
□ State 1	🟪	1.000	1.000	1.000	0.000	□		3
□ State 2	🟦	0.000	0.000	0.000	0.000	□		3
□ Counter	🟨	20486.000	3294.000	20481.000	17187.000	□		3

Fig. 4 View Measurement menu



eddyNCDT 3005

Port number: COM4
Baud rate: 230400
Sensor Address: 126
Serial number controller: 1112
Software version: 2.0c

→ Disconnect

Fig. 5 View Disconnect

By clicking the `Disconnect` button you return to the controller search, see Fig. 3.

	➡ Click on the `Reset scale` button to reset the Y-scale to the original setting (e.g. after zoom).
▶	➡ Click on the `Jump to Head` button to display the current signal course.

3.1.1 Data Acquisition

- ➡ Start the data acquisition by clicking the `Start` button, see Fig. 6. The acquisition is completely restarted and the record stopped before will be deleted.
- ➡ Stop the data acquisition by clicking the `Stop` button, see Fig. 7.

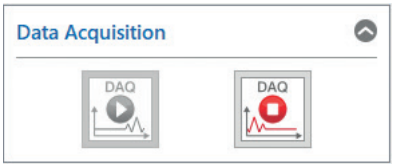
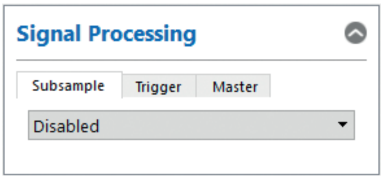


Fig. 6 Start *Fig. 7 Stop*

3.1.2 Signal Processing

i Settings for signal processing in `sensorTOOL` only affect the data in `sensorTOOL` and the CSV output. The signal processing in the DT3005 controller remains unaffected.



Signal Processing

Subsample | Trigger | Master

Disabled

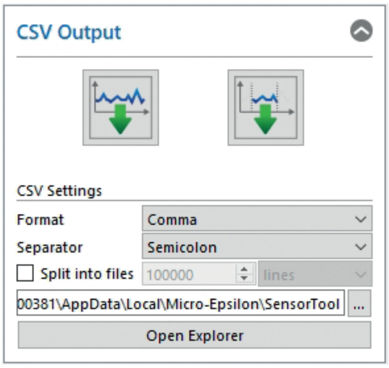
Fig. 8 Signal processing

You can select the following options for signal processing:

Data acquisition	Signal processing	*Subsampling*	*Deactivated*	*Deactivated; basic settings*
			Sample-based	*Number of samples is adjustable; every xth measurement is recorded.*
			Time-based	*Time-based; time can be set in milliseconds*[1]
		Trigger	*Deactivated*	*Deactivated; basic settings*
			Continuous	*Manual trigger*
			One-shot (sample-based)	*Sample can be set; records the signal course according to the set samples; the more samples, the longer the course*
			One-shot (time-based)	*Milliseconds can be set; records the signal course according to the time set.*
		Master	*Master now*	*Sets the master, see Fig. 10.*
			Resetting	*Resets the master.*

[1] For example every 5000 ms: The signal course displayed is updated after this period has elapsed.

3.1.3 CSV Output



CSV Output

CSV Settings

Format: Comma

Separator: Semicolon

□ Split into files 100000 lines

00381\AppData\Local\Micro-Epsilon\SensorTool ...

Open Explorer

Fig. 9 CSV output

	➡ Click this button to start recording measurement data.
	➡ Click this button to save the currently selected measurement value.

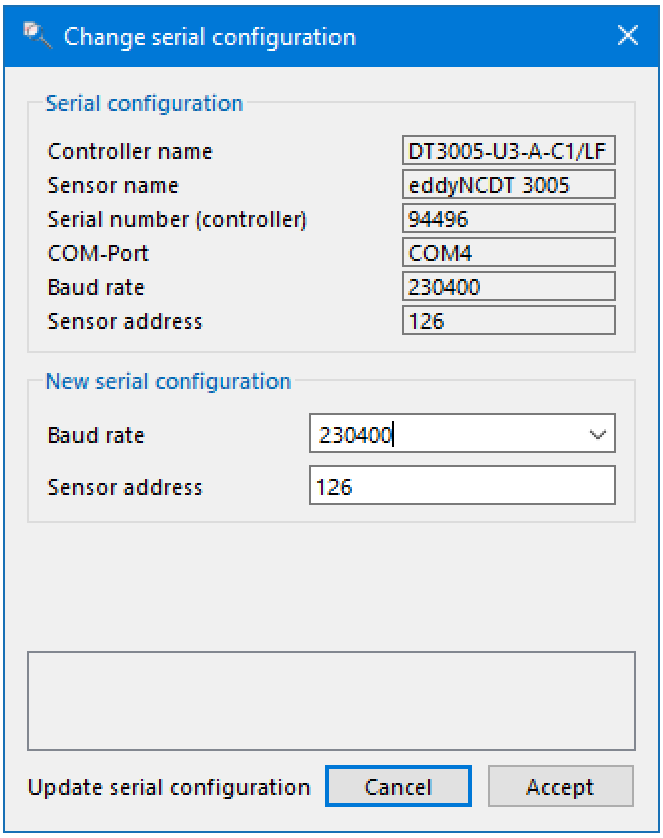
Data acquisition	CSV output	*Format*	*Point / Comma*
		Separator	*Comma / Semicolon / Tabulator*

3.1.4 Data Acquisition Table

Name	Show or hide signal curves of the sensors used.
Color	Here you can change the color settings for the individual curves.
Mastering	By activating the Mastering checkbox you can manually enter the master value. Master now in the menu Measurement > Signal Processing in the Master tab sets the master value, see Fig. 8.
Unit	*Selection of the output to be displayed. The outputs are set before in the Settings menu under Output / Output range and Adjustment.*
Decimal places	0 - 12

Fig. 10 Data Acquisition table

3.2 Configuration of Baud Rate



Change serial configuration

Serial configuration

Controller name	DT3005-U3-A-C1/LF
Sensor name	eddyNCDT 3005
Serial number (controller)	94496
COM-Port	COM4
Baud rate	230400
Sensor address	126

New serial configuration

Baud rate	230400
Sensor address	126

➡ To view the current configuration of the serial interface and change it if necessary, click on the Configure baudrate, see Fig. 3 button.

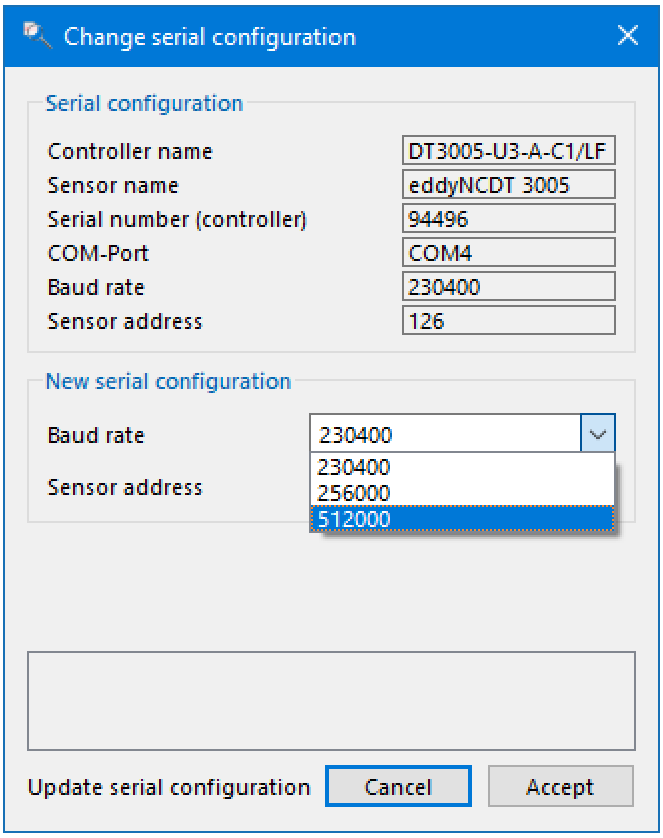
The Change serial configuration window then opens. Here you can change the baud rate, see Fig. 12 and assign a new address for the device, see Fig. 13.

Fig. 11 Window Change serial configuration

3.2.1 Changing the Baud Rate

The baud rate can be selected from a drop-down menu in the `Change serial configuration` window.

i The available baud rates are the baud rates supported by the `sensorTOOL` and not the baud rates supported by the eddyNCDT 3005.



Change serial configuration

Serial configuration

Controller name	DT3005-U3-A-C1/LF
Sensor name	eddyNCDT 3005
Serial number (controller)	94496
COM-Port	COM4
Baud rate	230400
Sensor address	126

New serial configuration

Baud rate: 230400

Sensor address

The eddyNCDT 3005 supports the following baud rates:

- 230400 Bit/s
- 256000 Bit/s
- 460800 Bit/s
- 512000 Bit/s

Byte frame: 1 start bit, 8 data bits, 1 parity bit (parity = even), 1 stop bit

Fig. 12 Window Change serial configuration - Baud rate view

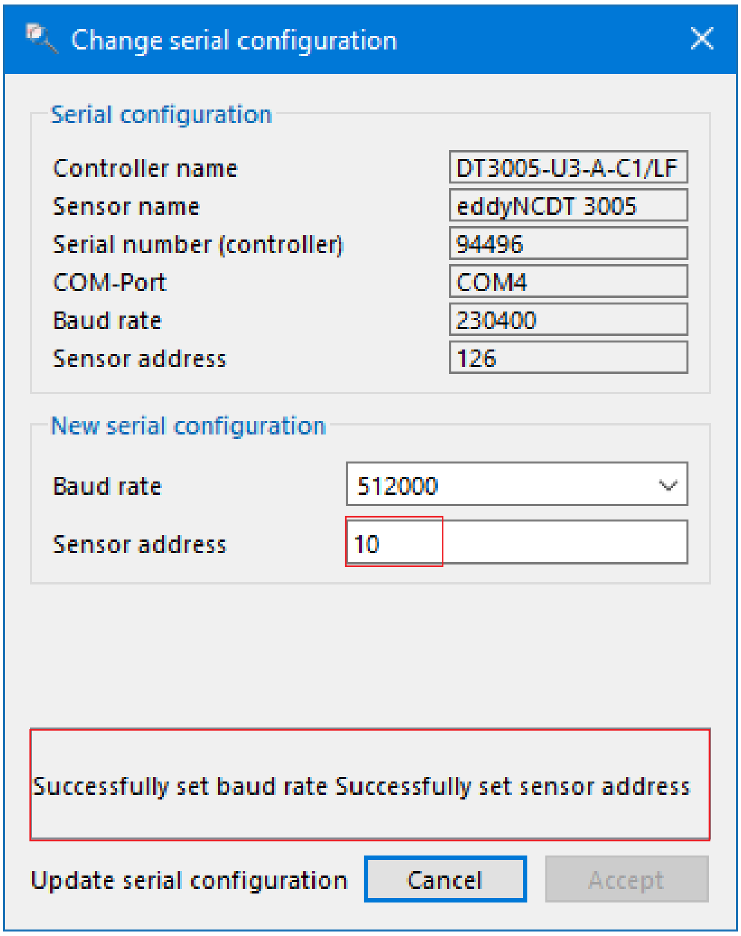
3.2.2 Address Assignment

The eddyNCDT 3005 systems are supplied with the address 126 as standard. The address of a connected device can also be changed in the `Change serial configuration`, see Fig. 13 window.

Valid addresses are addresses 1 to 126 inclusive. To set an address, enter the desired address in the `Sensor address` field and then confirm by clicking on the `Accept` button.

i If there are several devices on the ME bus, it is essential to ensure that each address on the bus is only assigned once.

If the address written is a valid address, the address is accepted by the device and the following message appears:



Change serial configuration

Serial configuration

Controller name: DT3005-U3-A-C1/LF
Sensor name: eddyNCDT 3005
Serial number (controller): 94496
COM-Port: COM4
Baud rate: 230400
Sensor address: 126

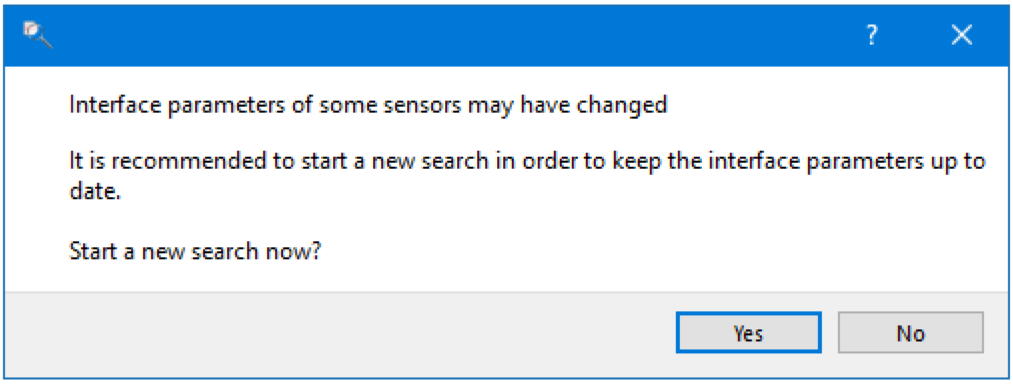
New serial configuration

Baud rate: 512000
Sensor address: 10

Successfully set baud rate Successfully set sensor address

Fig. 13 Message when the sensor address has been successfully changed

If the `Change serial configuration` window is then closed, the following message appears:



Interface parameters of some sensors may have changed

It is recommended to start a new search in order to keep the interface parameters up to date.

Start a new search now?

Fig. 14 Message for new interface parameters

3.3 Multi-Sensor DAQ Mode

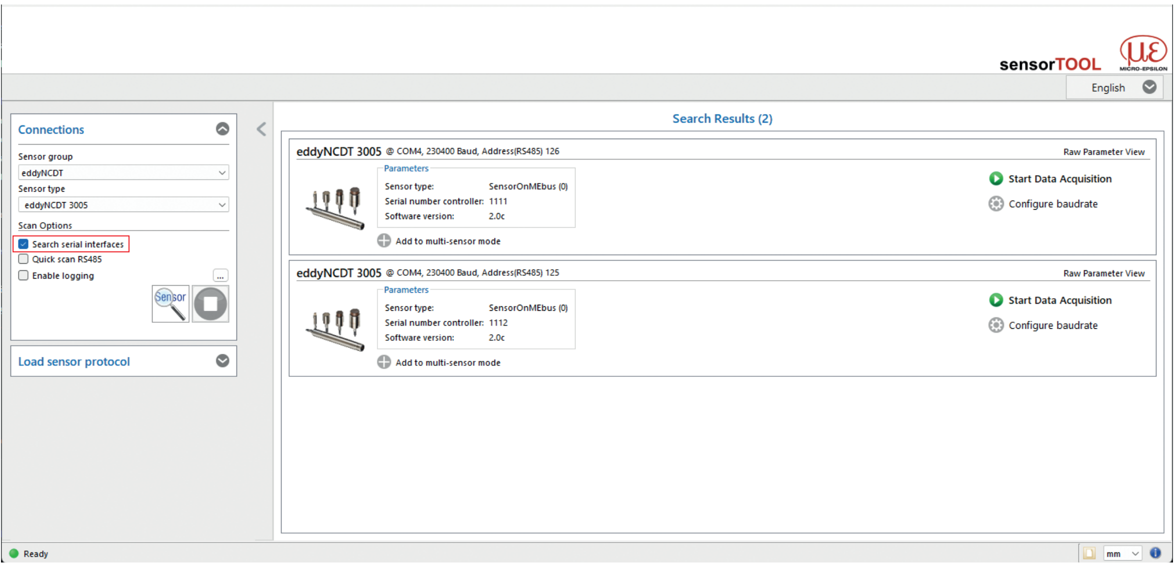
The `sensorTOOL` program also offers the possibility to output the data from several channels of the eddyNCDT 3005 series.

i Please note that the RS485 interface is a serial bus.
Even if the measured values are output simultaneously in `sensorTOOL`, they are recorded with a time delay.

To output the data of several bus participants into one graph, please proceed as follows:

➡ Search for the controller via the `sensorTOOL` program, see Fig. 3.

i Please note that the checkbox `Quick scan RS485` must be deactivated, see Fig. 15 to find multiple channels.



Sensor group: eddyNCDT
Sensor type: eddyNCDT 3005
Scan Options
☑ Search serial interfaces
☐ Quick scan RS485
☐ Enable logging

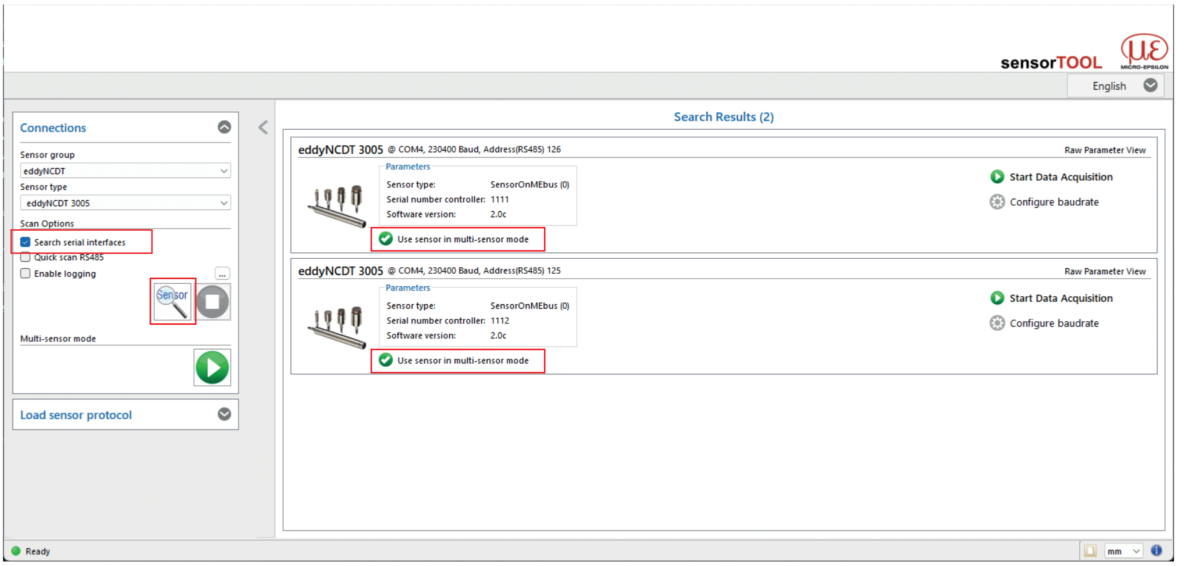
Search Results (2)

eddyNCDT 3005 @ COM4, 230400 Baud, Address(RS485) 126 — Raw Parameter View
Parameters
Sensor type: SensorOnMEbus (0)
Serial number controller: 1111
Software version: 2.0c
Start Data Acquisition
Configure baudrate
Add to multi-sensor mode

eddyNCDT 3005 @ COM4, 230400 Baud, Address(RS485) 125 — Raw Parameter View
Parameters
Sensor type: SensorOnMEbus (0)
Serial number controller: 1112
Software version: 2.0c
Start Data Acquisition
Configure baudrate
Add to multi-sensor mode

Fig. 15 First interactive site after calling the `sensorTOOL`

➡ Then enable the individual checkboxes `Use sensor in multi-sensor mode` of the respective channels.



Sensor group: eddyNCDT
Sensor type: eddyNCDT 3005
Scan Options
☑ Search serial interfaces
☐ Quick scan RS485
☐ Enable logging

Search Results (2)

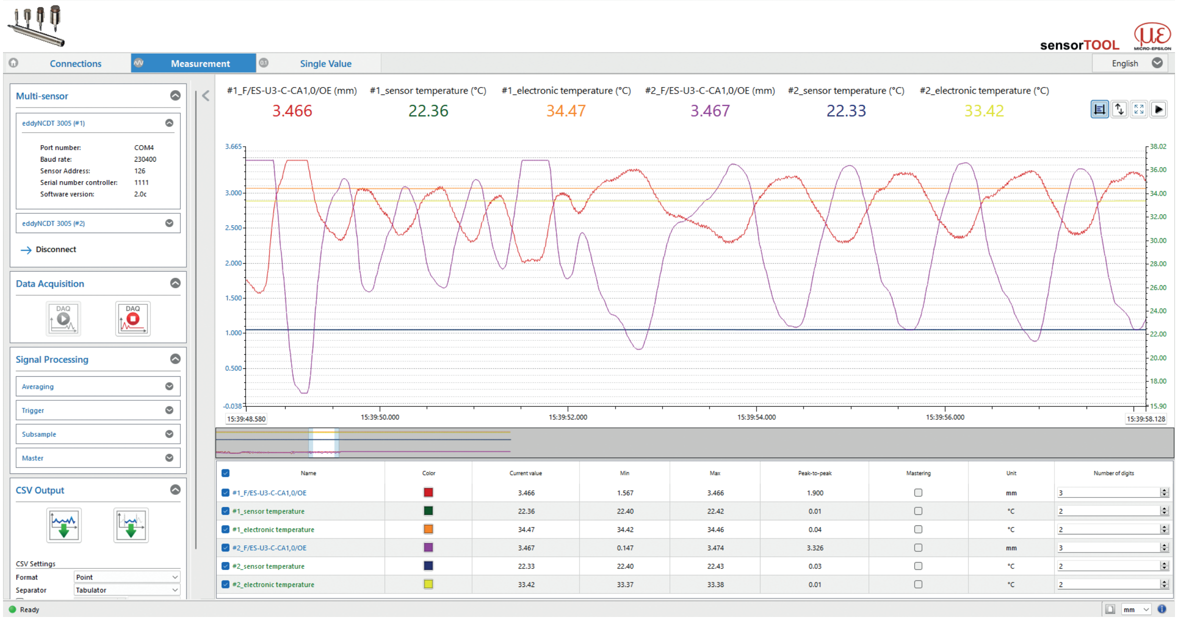
eddyNCDT 3005 @ COM4, 230400 Baud, Address(RS485) 126 — Raw Parameter View
Parameters
Sensor type: SensorOnMEbus (0)
Serial number controller: 1111
Software version: 2.0c
Start Data Acquisition
Configure baudrate
✓ Use sensor in multi-sensor mode

eddyNCDT 3005 @ COM4, 230400 Baud, Address(RS485) 125 — Raw Parameter View
Parameters
Sensor type: SensorOnMEbus (0)
Serial number controller: 1112
Software version: 2.0c
Start Data Acquisition
Configure baudrate
✓ Use sensor in multi-sensor mode

Fig. 16 First interactive site after calling the `sensorTOOL` *for the Multi-Sensor DAQ Mode*

➡ Now press the ▶ button.

In the `Measurement`, see Fig. 17 menu, the data output of the selected channels is displayed.



Port number:	COM4
Baud rate:	230400
Sensor Address:	126
Serial number controller:	1111
Software version:	2.0c

CSV Settings

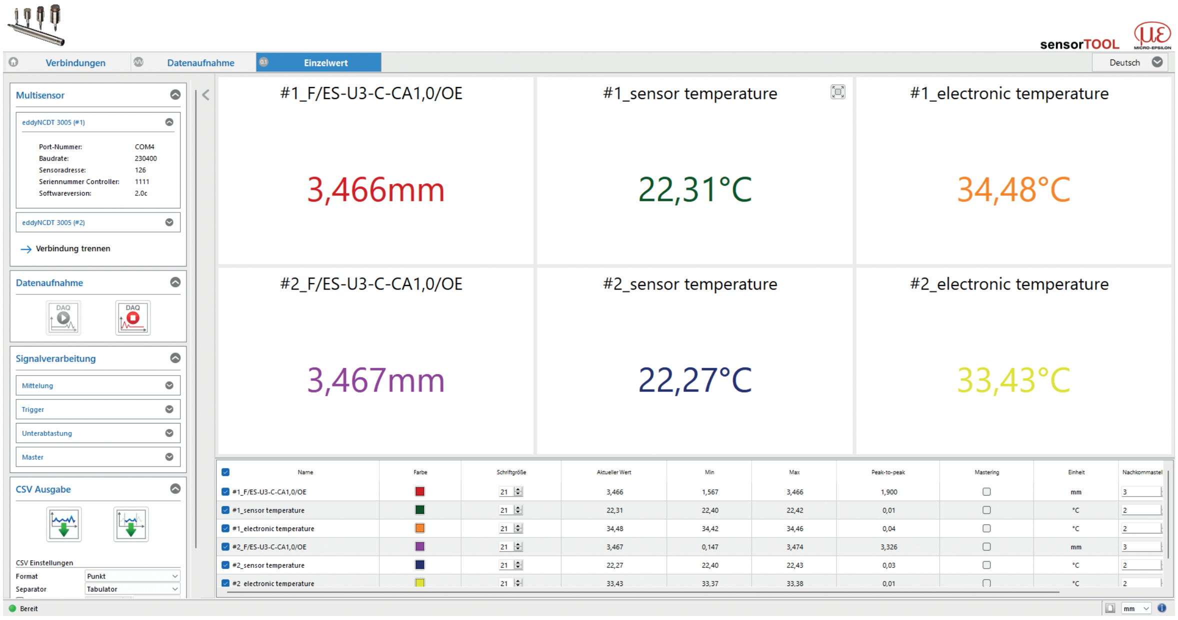
Format	Point
Separator	Tabulator

#1_F/ES-U3-C-CA1,0/OE (mm)	#1_sensor temperature (°C)	#1_electronic temperature (°C)	#2_F/ES-U3-C-CA1,0/OE (mm)	#2_sensor temperature (°C)	#2_electronic temperature (°C)
3.466	22.36	34.47	3.467	22.33	33.42

☑ Name	Color	Current value	Min	Max	Peak-to-peak	Mastering	Unit	Number of digits
☑ #1_F/ES-U3-C-CA1,0/OE	🟥	3.466	1.567	3.466	1.900	☐	mm	3
☑ #1_sensor temperature	🟩	22.36	22.40	22.42	0.01	☐	°C	2
☑ #1_electronic temperature	🟧	34.47	34.42	34.46	0.04	☐	°C	2
☑ #2_F/ES-U3-C-CA1,0/OE	🟪	3.467	0.147	3.474	3.326	☐	mm	3
☑ #2_sensor temperature	🟦	22.33	22.40	22.43	0.03	☐	°C	2
☑ #2_electronic temperature	🟨	33.42	33.37	33.38	0.01	☐	°C	2

Fig. 17 `Measurement` *menu, Multi-sensor DAQ mode*

In addition, the `Single Value` menu displays the data as numerical value.



Port-Nummer:	COM4
Baudrate:	230400
Sensoradresse:	126
Seriennummer Controller:	1111
Softwareversion:	2.0c

CSV Einstellungen

Format	Punkt
Separator	Tabulator

#1_F/ES-U3-C-CA1,0/OE	#1_sensor temperature	#1_electronic temperature
3,466mm	22,31°C	34,48°C
#2_F/ES-U3-C-CA1,0/OE	**#2_sensor temperature**	**#2_electronic temperature**
3,467mm	22,27°C	33,43°C

☑ Name	Farbe	Schriftgröße	Aktueller Wert	Min	Max	Peak-to-peak	Mastering	Einheit	Nachkommastel
☑ #1_F/ES-U3-C-CA1,0/OE	🟥	21	3,466	1,567	3,466	1,900	☐	mm	3
☑ #1_sensor temperature	🟩	21	22,31	22,40	22,42	0,01	☐	°C	2
☑ #1_electronic temperature	🟧	21	34,48	34,42	34,46	0,04	☐	°C	2
☑ #2_F/ES-U3-C-CA1,0/OE	🟪	21	3,467	0,147	3,474	3,326	☐	mm	3
☑ #2_sensor temperature	🟦	21	22,27	22,40	22,43	0,03	☐	°C	2
☑ #2_electronic temperature	🟨	21	33,43	33,37	33,38	0,01	☐	°C	2

Fig. 18 `Single value` *menu, Multi-sensor DAQ mode*

4. Interfaces

4.1 IF1032/ETH vs. IF2035

RS485 interface to DT3005	
IF1032/ETH	Supports only one participant on the RS485 bus.
IF2035	Supports up to 32 participants on the RS485 bus.
Interface to the customer	
IF1032/ETH	Can be configured by the customer (switch on circuit board and via software) between Ethernet and EtherCAT. It is factory set to Industrial Ethernet, see 4.3.
IF2035	Available in 3 variants: EtherCAT, PROFINET, EtherNet/IP, see 4.2

4.2 IF2035

4.2.1 Connection Diagram

The supply voltage is daisy-chained from the supply port (terminal 1) to the sensor port (terminal 2). Positive voltage must be between 12 V and 32 V.

Pin assignment of the eddyNCDT 3005 connector, see Fig. 1.

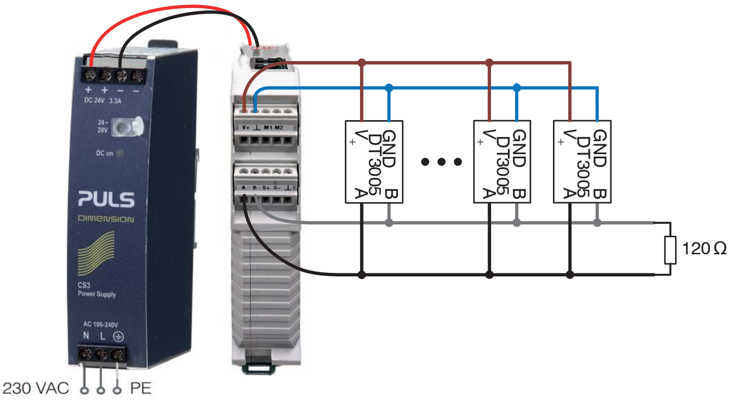


Fig. 19 Connection of the eddyNCDT 3005 controller to the IF2035 interface module with optional PS2020 power supply unit

Micro-Epsilon recommends a 120 Ω terminating resistor between the signal lines at both the bus start and end. In the IF2035, a 120 Ω terminating resistor is already permanently incorporated.

4.2.2 Hardware Interface

Physical interface:	RS485 half-duplex
Baud rate:	230400 Bit/s (default); additionally supported are 256000, 460800 and 512000 Bit/s
Byte frame:	1 Start Bit, 8 Data Bits, 1 Parity Bit (parity = even), 1 Stop Bit
ME bus address:	126 (default)

Internal acquisition rate of the eddyNCDT 3005 is 75 kSPS. RS485 is a bus interface. Up to 32, DT3005 can be connected to the same IF2035. However, the ME-Bus address has be unique on the bus, otherwise the sent data cannot be interpreted. To change the ME bus address of a DT3005, see Fig. 13.

4.2.3 Data Format

The structure data is 6 bytes long and contains the measurement values. It is structured as follows:

Data type	Name	Description
Uint16	distance	Target distance
Uint16	temperature_sensor	Temperature of the sensor
Uint16	temperature_electronic	Temperature of the controller

4.2.3.1 Calculation of Distance Values

Calculating of digital distance values for a U3 sensor with a measuring range of 3 mm.
(SMR = 0.3 mm, EMR = 3.3 mm)

	Distance in Digits	Distance in mm
SMR	3000	0.3
EMR	62000	3.3

Conversion of digital values

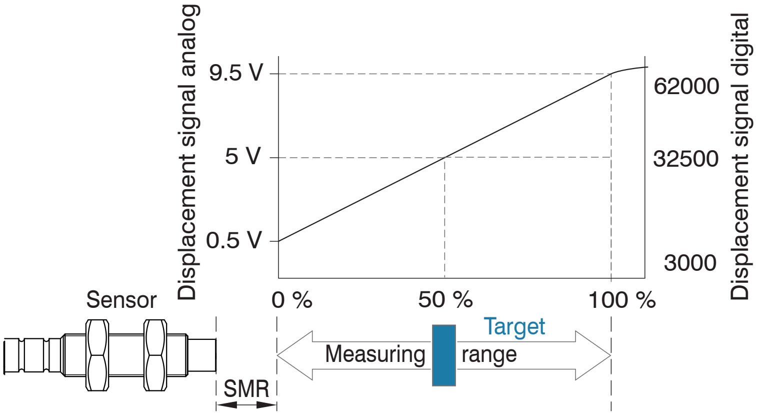


Fig. 20 Start of measuring range (SMR), the shortest distance between the front surface of the sensor and the target.

Distance with SMR (start of measuring range)

$$d = \frac{(x - 3000) \cdot \mathrm{MR}}{59000} \, \mathrm{SMR} \qquad x = [3000 \ldots 62000]$$

i The formula including the start of the measuring range is displayed in the `sensorTOOL`.

Distance without SMR (start of measuring range)

$$d = \frac{(x - 3000) \cdot \mathrm{MR}}{59000} \qquad x = [3000 \ldots 62000]$$

The start of measuring range SMR is 10 % FSO as standard. This sensor-specific value can be found in the respective eddyNCDT 3005 setup guide.

4.2.3.2 Calculating the Sensor Temperature

	Temperature in digits	Temperature in °C
SMR	3000	-40
EMR	62000	200

Conversion of digital values

$$\vartheta_s = \frac{(x_s - 3000) \cdot 240\,^{\circ}\mathrm{C}}{59000} \, -40\,^{\circ}\mathrm{C} \qquad \begin{array}{l} x_s = [3000 \ldots 62000] \\ \vartheta_s = [-40\,^{\circ}\mathrm{C} \ldots +200\,^{\circ}\mathrm{C}] \end{array}$$

4.2.3.3 Calculation of the Electronics (Controller) Temperature

	Temperature in digits	Temperature in °C
SMR	3000	-25
EMR	62000	85

Conversion of digital values

$$\vartheta_c = \frac{(x_c - 3000) \cdot 110\text{ °C}}{59000} - 25\text{ °C}$$

x_c = [3000 ... 62000]

ϑ_c = [-25 °C ... +85 °C]

4.2.3.4 Example of the Transmission of a Measurement Value

Byte no.	Byte	Description	Meaning / Value
0 ... 1	0xC0 0x8B	distance	0x8BC0 = 35776 = 1.967 mm
2 ... 3	0x49 0x7A	temperature_sensor	0x749 = 31305 = 75.14 °C
4 ... 5	0x04 0x78	temperature_electronic	0x7804 = 30724 = 26.69 °C

Distance to target is 3 mm/59000 x (35776 - 3000) + 0.3 mm = **1.967 mm**.

Sensor temperature is 4.0678e-3 °C x 31305 - 52.20 °C = **75.14 °C**.

Electronic temperature is 1.86441e-3 °C x 30724 - 30.59 °C = **26.69 °C**

4.3 IF1032/ETH

4.3.1 Connection Diagram

The supply voltage is daisy-chained from the supply port of the IF1032/ETH to the sensor port terminal. Positive voltage must be between 12 V and 32 V.

Pin assignment of the DT3005 connector, see Fig. 2.

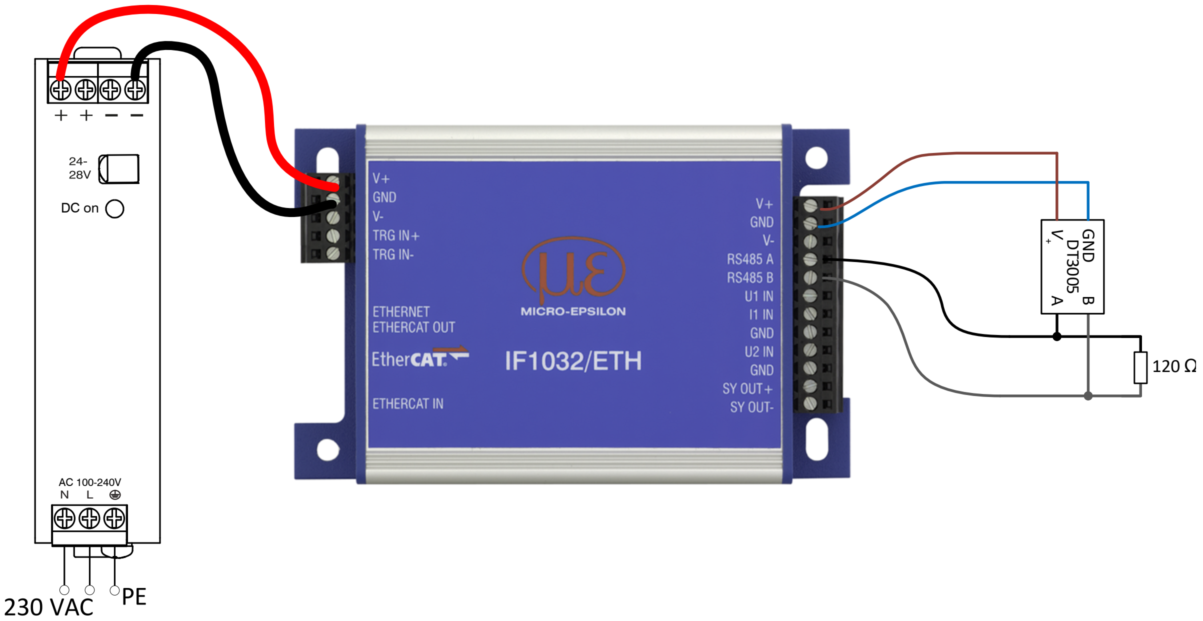


Fig. 21 Connection of the eddyNCDT 3005 controller to the IF1032/ETH interface module with optional PS2020 power supply unit

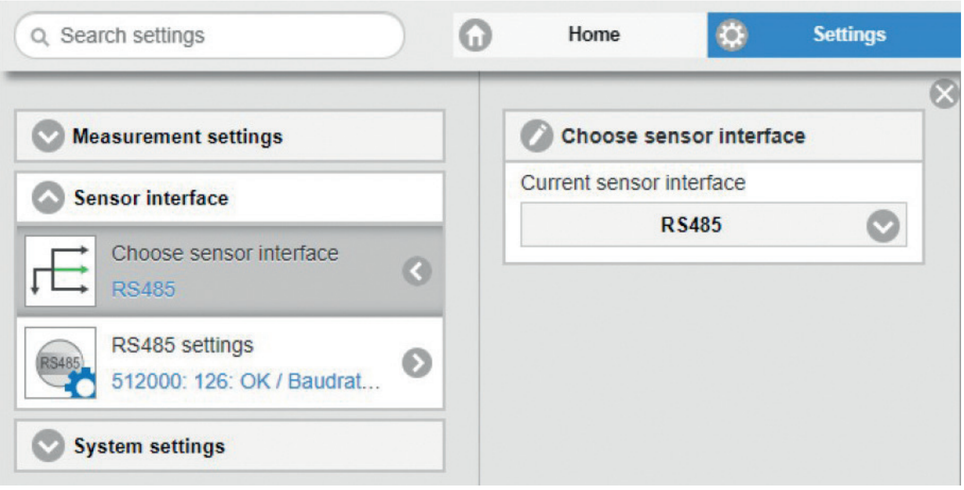
Only one DT3005 can be connected to the RS485 Bus of the IF1032/ETH.

Micro-Epsilon recommends a 120 Ω terminating resistor between the signal lines at both the bus start and end. In the IF1032/ETH a 120 Ω terminating resistor has already been permanently incorporated.

4.3.2 Sensor Interface

The IF1032/ETH only supports one eddyNCDT 3005 on the RS485 interface.

On the Webinterface of the IF1032/ETH the sensor interface has to be changed to RS485.



Choose sensor interface
RS485

RS485 settings
512000: 126: OK / Baudrat...

Choose sensor interface

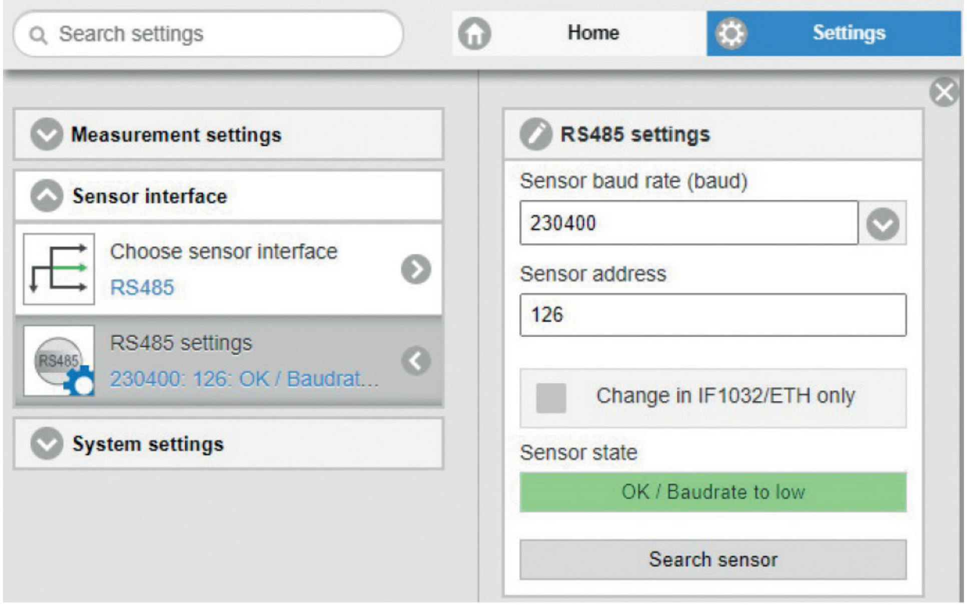
Current sensor interface: RS485

Fig. 22 View switch to RS485

The standard settings of the DT3005 on the RS485 interface are sensor baud rate 230400 baud and sensor address 126.

The baud rate of the sensor can be changed via the drop-down menu or by editing the field `Sensor baud rate (baud)` to one of the possible baud rates 230400, 256000, 460800 or 512000.

The sensor address is also changed when the field sensor address is edited.



Choose sensor interface
RS485

RS485 settings
230400: 126: OK / Baudrat...

RS485 settings

Sensor baud rate (baud): 230400

Sensor address: 126

□ Change in IF1032/ETH only

Sensor state: OK / Baudrate to low

Fig. 23 View changing the baud rate

Internal sampling rate of the eddyNCDT 3005 is 75 kSa/s. It is not possible to transfer every sample via the RS485 interface. The current data rate is shown in `Measurement settings` > `Measurement Mode`, see Fig. 24.

The possible transmission rate depends on the baud rate set for the RS485 interface. To achieve the highest transmission rate, the highest baud rate must also be set.

Baud rate	Maximum transmission rate
230400	831.6 Sa/s
256000	907.0 Sa/s
460800	1425.5 Sa/s
512000	1534.9 Sa/s

4.3.3 Measurement Settings - Measurement Mode

In the DT3005, an arithmetic mean value can be calculated over 2 to 65535 values. This reduces the rate at which the DT3005 outputs distance values. When averaging over 1000 values, the measuring rate is still 75 Sa/s, for example.

The arithmetic mean value M is calculated and output using the selectable number N of consecutive measured values.

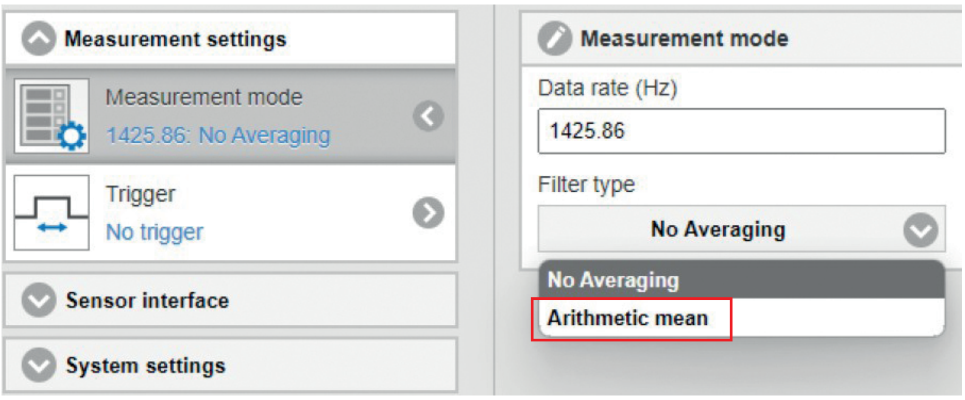
Method

Measured values are collected based on which the average is calculated. This method leads to a reduced amount of data because an average value is only output after every Nth measured value.

Example with N = 3:

.... 0 1 $\boxed{2\ 3\ 4}$ ... becomes $\frac{2+3+4}{3}$ average n

.... 3 4 $\boxed{5\ 6\ 7}$ ... becomes $\frac{5+6+7}{3}$ average n + 1



Measurement settings

Measurement mode
1425.86: No Averaging

Trigger
No trigger

Sensor interface

System settings

Measurement mode

Data rate (Hz)
1425.86

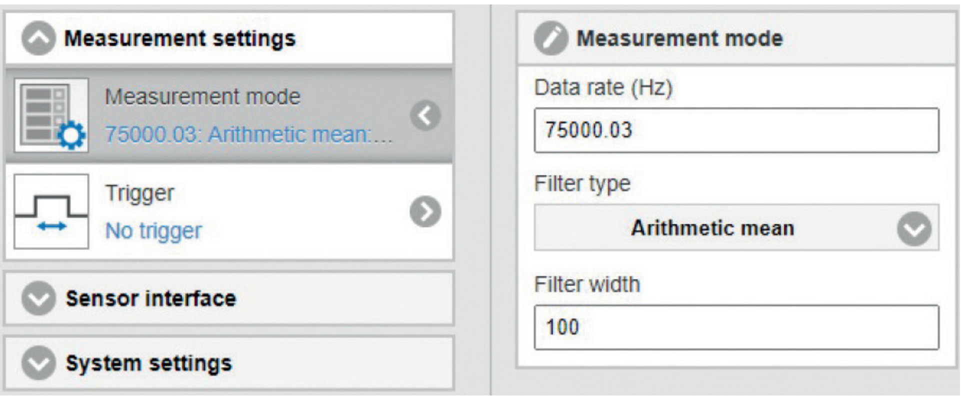
Filter type
No Averaging

No Averaging
Arithmetic mean

Fig. 24 View Measurement mode - Arithmetic mean

The value under `Data rate (Hz)` corresponds to the data rate before averaging.

After setting the `Arithmetic mean`, the data rate is 75000 Hz/100 = 750 Hz.



Measurement settings

Measurement mode
75000.03: Arithmetic mean:...

Trigger
No trigger

Sensor interface

System settings

Measurement mode

Data rate (Hz)
75000.03

Filter type
Arithmetic mean

Filter width
100

Fig. 25 View Measurement mode - Arithmetic mean

5. MEDAQLib

MEDAQLib is a documented driver DLL. This allows you to integrate sensors from Micro-Epsilon in conjunction with a converter or interface module into existing or customer-specific PC software.

MEDAQLib

- contains a DLL that can be imported into C, C++, VB, Delphi and many other programs,
- takes care of data conversion for you,
- works regardless of the type of interface used,
- uses the same functions for communication (commands),
- provides a uniform transmission format for all Micro-Epsilon sensors.

For C/C++ programmers, an additional header file and a library file are integrated into MEDAQLib.

- You can download the MEDAQLib installation files to your computer via the link https://www.micro-epsilon.com/link/software/medaqlib.
- For further information on MEDAQLib, please use the page https://www.micro-epsilon.com/service/software-sensorintegration/medaqlib.

5.1 Supported ME-Bus Sensor Commands

Command	**Supported**
Logout	no
Login	no
Get_UserLevel	no
Set_Password	no
Set_Samplerate	no
Get_Samplerate	yes
Set_Trigger	no
Get_Trigger	no
Set_Averaging	yes
Get_Averaging [1]	yes
Get_Measure	yes
Get_AlternateMeasure	no
Set_ContinuousMode	no
Get_ContinuousMode	yes
Set_Range	no
Test_Baudrate	yes
Set_Baudrate	yes
Get_Baudrate	yes
Set_SensorAddress	yes
Get_SensorInfo	yes
Get_Channelinfo	yes
Get_Channelinfos	yes
Get_ControllerInfo	yes
Get_DiagnosticInfo	no
Get_DiagnosticInfo	no
Get_ConfigDescription	no
Set_ConfigParameter	no
Get_ConfigParameter	no
Read_AllBlocks	yes

1) AveragingType = {0; 2}, AveragingValue = [0; 65535]

5.2 Examples

The following examples read the name, serial number of the DT3005 and the description of the measurement values. Then some measurement values are read from the DT3005.

5.2.1 Python

➡ Copy the two files `MEDAQLib.dll` and `MEDAQLib.py` from the `Snippets/Python` subdirectory in the MEDAQLib installation directory to the same directory as the Python source code.

```
#
# This is a very simple sample following MEDAQLib.pdf section 4 Using MEDAQLib
#
# Please adjust to your setup (interface card and sensor used)
#

from MEDAQLib import MEDAQLib, ME_SENSOR, ERR_CODE
import time

number_of_reads = 10;

# Tell MEDAQLib about sensor type to be used
MEDAQLib_object = MEDAQLib.CreateSensorInstByName ("MEBus")



# Tell MEDAQLib about interface to be used
MEDAQLib_object.SetParameterString("IP_Interface", "RS232")
MEDAQLib_object.SetParameterString("IP_Port", "COM4")
MEDAQLib_object.SetParameterInt("IP_SensorAddress", 126)
MEDAQLib_object.SetParameterInt("IP_Baudrate", 230400)

# Enable Logfile writing
# MEDAQLib_object.SetParameterInt("IP_EnableLogging", 1)

# Try to open communication to sensor via interface specified
MEDAQLib_object.OpenSensor()
if MEDAQLib_object.GetLastError() != ERR_CODE.ERR_NOERROR:
    raise RuntimeError("OpenSensor: " + MEDAQLib_object.GetError())

MEDAQLib_object.ExecSCmd("Get_ControllerInfo")
if MEDAQLib_object.GetLastError() != ERR_CODE.ERR_NOERROR:
    raise RuntimeError("Get_ControllerInfo: " + MEDAQLib_object.GetError())

controller_name = MEDAQLib_object.GetParameterString("SA_ControllerName")
serial_number = MEDAQLib_object.GetParameterString("SA_SerialNumber")
print(f"Controller Name: {controller_name}")
print(f"Controller Serial Number: {serial_number}")

MEDAQLib_object.ExecSCmd("Get_TransmittedDataInfo")
if MEDAQLib_object.GetLastError() != ERR_CODE.ERR_NOERROR:
    raise RuntimeError("Get_TransmittedDataInfo: " + MEDAQLib_object.GetError())
```

```
number_of_channels = MEDAQLib_object.GetParameterInt("IA_ValuesPerFrame")

if number_of_channels == 0:
    raise RuntimeError("No data channels available")

for i in range(1, number_of_channels+1):
    index = MEDAQLib_object.GetParameterInt("IA_Index"+str(i))
    raw_name = MEDAQLib_object.GetParameterString("IA_Raw_Name"+str(i))
    scaled_name = MEDAQLib_object.GetParameterString("IA_Scaled_Name"+str(i))
    raw_unit = MEDAQLib_object.GetParameterString("IA_Raw_Unit"+str(i))
    scaled_unit = MEDAQLib_object.GetParameterString("IA_Scaled_Unit"+str(i))
    raw_range_min = MEDAQLib_object.GetParameterDouble("IA_Raw_RangeMin"+str(i))
    scaled_range_min = MEDAQLib_object.GetParameterDouble("IA_Scaled_RangeMin"+str(i))
    raw_range_max = MEDAQLib_object.GetParameterDouble("IA_Raw_RangeMax"+str(i))
    scaled_range_max = MEDAQLib_object.GetParameterDouble("IA_Scaled_RangeMax"+str(i))
    print(f"{index}: {raw_name} [{raw_range_min} .. {raw_range_max} {raw_unit}], " \
        f"{scaled_name} in {scaled_unit} [{scaled_range_min} .. {scaled_range_max}]")


print(f"Read {number_of_reads} measurements from {number_of_channels} channels ...")
# If no error then try to acquire data
if MEDAQLib_object.GetLastError() == ERR_CODE.ERR_NOERROR:
    for num_read in range(number_of_reads):
        # Sleep for 10 ms
        time.sleep(0.01)
        # Ask sensor for new data
        MEDAQLib_object.ExecSCmd("Get_Measure")
        # Check whether there is enough data to read in
        currently_available = MEDAQLib_object.DataAvail()
        # Check if DataAvail causes an Error
        if (MEDAQLib_object.GetLastError() != ERR_CODE.ERR_NOERROR):
            print(MEDAQLib_object.GetError())
        # If data is available?
        if currently_available >= number_of_channels:
            # Transfer/Move data from MEDAQLib internal buffer to own buffer
            transfered_data = MEDAQLib_object.TransferData(currently_available)
            # Check if TransferData causes an error
            if MEDAQLib_object.GetLastError() == ERR_CODE.ERR_NOERROR:
                # contains original values form sensor
                raw_data = transfered_data[0]
                # contains scaled data values
                scaled_data = transfered_data[1]
                # get number of data values received,
                # should be equal to currently_available
                nr_values_transfered = transfered_data[2]
                # output raw and scaled value of very first measurement
                for j in range(0,nr_values_transfered,number_of_channels):
                    print(scaled_data[j:j+number_of_channels], sep=', ')


                # do your computation on data ....
            else:
                # Print TransferData error
                print(MEDAQLib_object.GetError())
else:
    # Print OpenSensor Error
    print(MEDAQLib_object.GetError())

# Closing down by closing interface and releasing sensor instance
MEDAQLib_object.CloseSensor()
MEDAQLib_object.ReleaseSensorInstance()
```

5.2.2 C#

➡ Copy the two files `MEDAQLib.dll` and `MEDAQLib.Net.dll` from the subdirectory `Release` in the MEDAQLib installation directory into the same directory as the C# code.

```
using System;
using System.Diagnostics;
s
using MicroEpsilon; // MEDAQLib

namespace C_Sharp_Example
{
    class Program
    {
        static ERR_CODE Error(string location, ref MEDAQLib sensor)
        {
            string errText = "";
            ERR_CODE err = sensor.GetError(ref errText);
            Console.WriteLine(location + " returned error: " + errText);
            Console.WriteLine("Demo failed, press any key ...)");
            Console.ReadKey(true);
            return err;
        }

        static int sValsPerFrame = 0;

        static string StrWithIndex(string name, int index)
        {
            return name + index.ToString();
        }

        static ERR_CODE GetControllerInfo(ref MEDAQLib sensor)
        {
            string controllerName = "", controllerSerialNumber = "";

            if (sensor.ExecSCmd("Get_ControllerInfo") != ERR_CODE.ERR_NOERROR)
```

```
                return Error("Get_ControllerInfo", ref sensor);

            sensor.GetParameterString("SA_ControllerName", ref controllerName);
            sensor.GetParameterString("SA_SerialNumber", ref controllerSerialNumber);

            Console.WriteLine("Controller Name: {0}", controllerName);
            Console.WriteLine("Controller Serial Number: {0}", controllerSerialNumber);

            return ERR_CODE.ERR_NOERROR;
        }

        static ERR_CODE GetTransmittedDataInfo(ref MEDAQLib sensor)
        {
            int maxValsPerFrame = 0, maxOutputIndex = 0;

            if (sensor.ExecSCmdGetInt("Get_TransmittedDataInfo", "IA_ValuesPerFrame",
                    ref sValsPerFrame) != ERR_CODE.ERR_NOERROR)
                return Error("Get_TransmittedDataInfo", ref sensor);

            sensor.GetParameterInt("IA_MaxValuesPerFrame", ref maxValsPerFrame);
            sensor.GetParameterInt("IA_MaxOutputIndex", ref maxOutputIndex);
            Console.WriteLine("Sensor transmits {0} of {1} possible values," +
                "maximum output index is {2}",
                sValsPerFrame, maxValsPerFrame, maxOutputIndex);

            for (int i = 0; i < sValsPerFrame; i++)
            {
                int index = 0;
                double rawRangeMin = 0.0, rawRangeMax = 0.0;
                double scaledRangeMin = 0.0, scaledRangeMax = 0.0;
                string rawName = "", scaledName = "", rawUnit = "", scaledUnit = "";
                sensor.GetParameterString(
                  StrWithIndex("IA_Raw_Name", i + 1), ref rawName);
                sensor.GetParameterString(
                  StrWithIndex("IA_Scaled_Name", i + 1), ref scaledName);
                sensor.GetParameterString(
                  StrWithIndex("IA_Raw_Unit", i + 1), ref rawUnit);
                sensor.GetParameterString(
                  StrWithIndex("IA_Scaled_Unit", i + 1), ref scaledUnit);
                sensor.GetParameterInt(
                  StrWithIndex("IA_Index", i + 1), ref index);
                sensor.GetParameterDouble(
                  StrWithIndex("IA_Raw_RangeMin", i + 1), ref rawRangeMin);
                sensor.GetParameterDouble(
                  StrWithIndex("IA_Scaled_RangeMin", i + 1), ref scaledRangeMin);
                sensor.GetParameterDouble(
                  StrWithIndex("IA_Raw_RangeMax", i + 1), ref rawRangeMax);
                sensor.GetParameterDouble(
                  StrWithIndex("IA_Scaled_RangeMax", i + 1), ref scaledRangeMax);
                Console.WriteLine(
                    " {0,2}: {1} [{2} .. {3} {4}], {5} in {8} [" +
                    "{6} .. {7}" +
                    "]",
                    index, rawName, rawRangeMin, rawRangeMax, rawUnit, scaledName,
                    scaledRangeMin, scaledRangeMax, scaledUnit
                );
                Console.WriteLine(" {0,2}: {1} [{2} .. {3} {4}], {5} in {8} " +
                    "[{6} .. {7}]", index, rawName, rawRangeMin, rawRangeMax, rawUnit,
                    scaledName, scaledRangeMin, scaledRangeMax, scaledUnit);
            }

            return ERR_CODE.ERR_NOERROR;
        }

        static ERR_CODE TransferData(ref MEDAQLib sensor)
        {
            Console.WriteLine("Transfer data ...");

            while (!Console.KeyAvailable)
            {
                System.Threading.Thread.Sleep(10);
                sensor.ExecSCmd("Get_Measure");

                int avail = 0;
                if (sensor.DataAvail(ref avail) != ERR_CODE.ERR_NOERROR)
                    return Error("DataAvail", ref sensor);

                int[] rawData = new int[avail];
                double[] scaledData = new double[avail];
                int read = 0;
                if (sensor.TransferData(rawData, scaledData, avail, ref read)
                    != ERR_CODE.ERR_NOERROR)
                        return Error("TransferData", ref sensor);

                int num_values = read/sValsPerFrame;
                for (int i = 0; i < num_values; i++)
                {
                    Console.Write("{0:F3}", scaledData[i*sValsPerFrame]);
                    for (int j = 1; j < sValsPerFrame; j++)
                    {
                      Console.Write(", {0:F3}", scaledData[i*sValsPerFrame+j]);
                    }
                    Console.WriteLine("");
                }
            }
            Console.ReadKey(true);
            Console.WriteLine("");

            return ERR_CODE.ERR_NOERROR;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Start Demo...");

            MEDAQLib sensor = new MEDAQLib("ME-Bus");
            sensor.SetParameterString("IP_Interface", "RS232");
            sensor.SetParameterString("IP_Port", "COM4");
            sensor.SetParameterInt("IP_Baudrate", 230400);
            sensor.SetParameterInt("IP_SensorAddress", 126);
            // Enables logging of additional debugging information to TXT file
            //sensor.SetParameterInt("IP_EnableLogging", 1);

            if (sensor.OpenSensor() != ERR_CODE.ERR_NOERROR)
            {
                Error("OpenSensor", ref sensor);
                return;
            }

            if (GetControllerInfo(ref sensor) != ERR_CODE.ERR_NOERROR)
                return;

            if (GetTransmittedDataInfo(ref sensor) != ERR_CODE.ERR_NOERROR)
                return;

            if (sValsPerFrame == 0)
            {
                Console.WriteLine("No data channels available");
                Console.WriteLine("Demo failed, press any key ...)");
                Console.ReadKey(true);
                return;
            }

            if (TransferData(ref sensor) != ERR_CODE.ERR_NOERROR)
                return;

            Console.WriteLine("Demo successfully finished, press any key ...");
            Console.ReadKey(true);
        }
    }
}
```

5.2.3 MATLAB

➡ Copy the two files `MEDAQLib.h` and `Release-x64\MEDAQLib.dll` from the MEDAQLib installation directory to the same directory as the MATLAB script.

```
%%DT3005_READ Example for reading one measurement value from DT3005
clear;
medaqlib_install_dir = 'C:\Program Files (x86)\MEDAQLib';
max_str_length = 32;
max_err_length = uint32(1024); % Reserved maximum length for error messages

number_of_reads = 10; % Number read requests to the sensor

%% Load MEDAQLib
if ~isfile('MEDAQLib.dll')
    copyfile(fullfile(medaqlib_install_dir, 'Release-x64', 'MEDAQLib.dll'), '.');
end
if ~isfile('MEDAQLib.h')
    copyfile(fullfile(medaqlib_install_dir, 'MEDAQLib.h'), '.');
end
if ~libisloaded('medaqlib')
    [notfound, warnings] = loadlibrary('MEDAQLib', 'MEDAQLib.h', 'alias', 'medaqlib');
end

try

    %% Tell MEDAQLib about sensor type to be used.
    h_sensor = uint32(calllib('medaqlib', 'CreateSensorInstByName', 'MEbus'));

    %% Tell MEDAQLib about interface to be used
    calllib('medaqlib', 'SetParameterString', h_sensor, 'IP_Interface', 'RS232');
    calllib('medaqlib', 'SetParameterString', h_sensor, 'IP_Port', 'COM4');
    calllib('medaqlib', 'SetParameterInt', h_sensor, 'IP_SensorAddress', 126);
    calllib('medaqlib', 'SetParameterInt', h_sensor, 'IP_Baudrate', 230400);

    %% Enable Logfile writing
    calllib('medaqlib', 'SetParameterInt', h_sensor, 'IP_EnableLogging', 1);

    %% Try to open communication to sensor via interface specified
    err = calllib('medaqlib', 'OpenSensor', h_sensor);
    if ~strcmp(err, 'ERR_NOERROR')
        error('Unable to open Sensor %s', err);
    end
catch ME
    unloadlibrary('medaqlib');
    rethrow(ME);
end

try

    %% Read information about connected controller
```

```
err = calllib('medaqlib', 'ExecSCmd', h_sensor, 'Get_ControllerInfo');
assert(strcmp(err, 'ERR_NOERROR'), 'medaqlib:ExecSCmd', 'Get_ControllerInfo');

% Display controller name
[~, param_name, controller_name, ~] = calllib('medaqlib', ...
    'GetParameterString', h_sensor, 'SA_ControllerName', ...
    blanks(max_str_length), libpointer('uint32Ptr', max_str_length));
fprintf('%s = %s\n', param_name, controller_name);

% Display controller serial number
[~, param_name, controller_serial_number, ~] = calllib('medaqlib', ...
    'GetParameterString', h_sensor, 'SA_SerialNumber', ...
    blanks(max_str_length), libpointer('uint32Ptr', max_str_length));
fprintf('%s = %s\n', param_name, controller_serial_number);

%% Read information about the transmitted data
err = calllib('medaqlib', 'ExecSCmd', h_sensor, 'Get_TransmittedDataInfo');
assert(strcmp(err, 'ERR_NOERROR'), 'medaqlib:ExecSCmd', 'Get_TransmittedDataInfo');
[~, ~, channel_count] = calllib('medaqlib', 'GetParameterInt', h_sensor, ...
    'IA_ValuesPerFrame', libpointer('int32Ptr', 0));

disp("Read "+string(number_of_reads)+" measurements from "+ ...
    channel_count+" channels ...");
if channel_count == 0
    error('No data channels available');
end

channel_names = cell(1, channel_count);
for k = 1:channel_count
    [~, ~, index] = calllib('medaqlib', 'GetParameterInt', h_sensor, ...
        sprintf('IA_Index%d', k), libpointer('int32Ptr', 0));
    [~, ~, scaled_name, ~] = calllib('medaqlib', 'GetParameterString', ...
        h_sensor, sprintf('IA_Scaled_Name%d', k), blanks(max_str_length), ...
        libpointer('uint32Ptr', max_str_length));
    [~, ~, scaled_unit, len] = calllib('medaqlib', 'GetParameterBinary', ...
        h_sensor, sprintf('IA_Scaled_Unit%d', k), ...
        zeros(1, max_str_length, 'uint8'), ...
        libpointer('uint32Ptr', max_str_length));
    if len > 0
        scaled_unit = char(scaled_unit(1:len));
    else
        scaled_unit = '';
    end
    [~, ~, scaled_range_min] = calllib('medaqlib', 'GetParameterDouble', ...
        h_sensor, sprintf('IA_Scaled_RangeMin%d', k), libpointer('doublePtr', 0));
    [~, ~, scaled_range_max] = calllib('medaqlib', 'GetParameterDouble', ...
        h_sensor, sprintf('IA_Scaled_RangeMax%d', k), libpointer('doublePtr', 0));
    disp(string(index)+": "+scaled_name+" ["+string(scaled_range_min) ...
            +" .. "+string(scaled_range_max)+"]");
        channel_names{k} = strip(strrep(scaled_name, '(scaled)', ''), 'both');
    end
    clear str;
    disp(strjoin(channel_names, ', '));

    %% Read measurement value from sensor
    for i = 1:number_of_reads
        % Ask sensor for new data
        err = calllib('medaqlib', 'ExecSCmd', h_sensor, 'Get_Measure');
        assert(strcmp(err, 'ERR_NOERROR'), 'medaqlib:ExecSCmd', 'Get_Measure');

        % Check whether there is enough data to read in
        [~, currently_available] = calllib('medaqlib', 'DataAvail', h_sensor, ...
            libpointer('int32Ptr', 1));

        % If data is available?
        if currently_available >= channel_count
            % Transfer/Move data from MEDAQLib‘s internal buffer to own buffer
            raw_data = libpointer('int32Ptr', zeros(1, currently_available));
            scaled_data = libpointer('doublePtr', zeros(1, currently_available));

            [err, raw_data, scaled_data, num_read] = calllib('medaqlib', ...
                'TransferData', h_sensor, raw_data, scaled_data, ...
                currently_available, libpointer('int32Ptr', 1));
            assert(strcmp(err, 'ERR_NOERROR'), 'medaqlib:TransferData', '');

            data_read = reshape(scaled_data, channel_count, []).';
            for k = 1:size(data_read, 1)
                disp(strjoin(string(data_read(k, :)), ', '));
            end
        else
            error('No data available');
        end
    end
    calllib('medaqlib', 'CloseSensor', h_sensor);
    calllib('medaqlib', 'ReleaseSensorInstance', h_sensor);
    unloadlibrary('medaqlib');
catch ME
    s = split(ME.identifier, ':');
    if strcmp(s{1}, 'medaqlib')
        [~, error_txt] = calllib('medaqlib', 'GetError', h_sensor, ...
            blanks(max_err_length), max_err_length);
        if isempty(error_txt)
            error('%s: %s', ME.identifier, ME.message);
        else
            error('%s: %s MEDAQLib "%s"', ME.identifier, ME.message, error_txt);
        end
    end
    calllib('medaqlib', 'CloseSensor', h_sensor);
    calllib('medaqlib', 'ReleaseSensorInstance', h_sensor);
    unloadlibrary('medaqlib');
    rethrow(ME);
end
```

Appendix

Optional Accessories

PC5/5-M12		Power supply and output cable, 5 m long
PC10/5-M12		Power supply and output cable, 10 m long
PC20/5-M12		Power supply and output cable, 20 m long
PS2020		Input 100...240 VAC Output 24 VDC / 2.5 A, for snap in mounting on DIN 50022 rail
IF7001		IF7001 Single-channel USB/RS485 converter
IF1032/ETH		Multi-channel Ethernet and EtherCAT converter - three analog inputs - one RS485 (single channel)
IF2035-EtherCAT IF2035-PROFINET IF2035-EIP		Interface module for EtherCAT Interface module for PROFINET Interface module for Ethernet/IP

MICRO-EPSILON

MICRO-EPSILON MESSTECHNIK GmbH & Co. KG
Koenigbacher Strasse 15 · 94496 Ortenburg / Germany
Tel. +49 (0) 8542 / 168-0 · Fax +49 (0) 8542 / 168-90
info@micro-epsilon.com · www.micro-epsilon.com
Your local contact: www.micro-epsilon.com/contact/worldwide/