MICRO-EPSILON

Schnittstellenanleitung
eddyNCDT 3005

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

Schnittstellenanleitung

MICRO-EPSILON
MESSTECHNIK
GmbH & Co. KG
Königbacher Strasse 15
94496 Ortenburg / Deutschland

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

Inhalt

1. Einleitung

Das Messsystem eddyNCDT 3005 misst mit Hilfe von Wirbelströmen präzise Abstände, Bewegung und Position von metallischen Gegenständen. Die Messwerte können sowohl analog als Spannung, als auch digital über das RS485-Interface ausgelesen werden. Diese Anleitung beschreibt, wie das RS485-Interface an gängigen Schnittstellen (USB, EtherNet, EtherCAT, Ethernet/IP oder PROFINET) angebunden werden kann.

Die PC-Software sensorTOOL erlaubt die Konfiguration des Sensors und die Visualisierung der gemessenen Daten.

Zusätzlich können Sie den IF2035 Schnittstellen-Konverter von Micro-Epsilon nutzen, um die gemessenen Daten über PROFINET, EtherCAT oder EtherNet/IP auszulesen.

Weitere Informationen zum IF2035 Schnittstellenmodul können Sie in den Betriebsanleitungen nachlesen. Diese finden Sie Online unter:

https://www.micro-epsilon.de/industrie-sensoren/schnittstellen-verrechnung/if2035-industrial-ethernet/

Des Weiteren können Sie den IF1032/ETH Konverter von Micro-Epsilon nutzen, um die gemessenen Daten über Ethernet oder EtherCAT auszulesen.

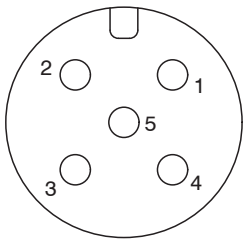
Weitere Informationen zum IF1032/ETH können Sie in der Betriebsanleitung nachlesen. Diese finden Sie Online unter:

https://www.micro-epsilon.de/industrie-sensoren/schnittstellen-verrechnung/if1032-eth/

2. Anschlussbelegung

Die Konverter können in Verbindung mit dem Kabel PCx/5-M12 verwendet werden [1].

Pin	Beschreibung	PC5/5-M12
1	Braun	12 ... 32 VDC
2	Weiß	Abstandssignal
3	Blau	Masse
4	Schwarz	RS485 A / +
5	Grau	RS485 B / -



Ansicht Steckerseite

Abb. 1 5-poliger M12 A-codierter Stecker am Controller

3. Software sensorTOOL

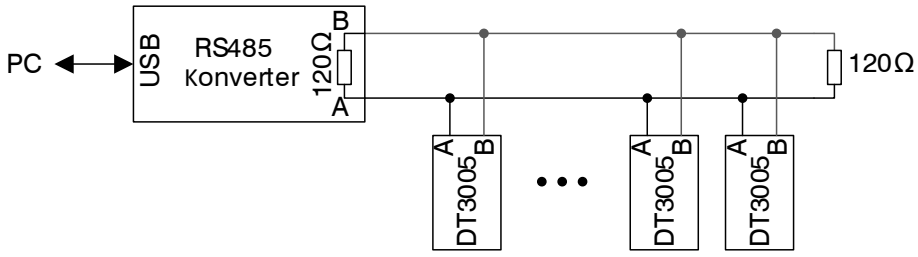


Abb. 2 Anschluss des eddyNCDT 3005 an den PC mittels eines USB/RS485 Konverters

Zwischen der A- und B-Leitung der RS485-Schnittstelle am Anfang und am Ende des RS485-Busses ist ein Abschlusswiderstand von 120 Ω erforderlich. Ein Abschlusswiderstand der RS485-Leitung ist nicht in den DT3005 Sensor integriert. Daher ist der Anschluss verschiedener Sensoren an ein Buskabel möglich.

Mit der PC-Software sensorTOOL steht Ihnen eine dokumentierte Software zur Verfügung, mit der Sie den Sensor einstellen, visualisieren und dokumentieren können.

Dieses Programm finden Sie online unter https://www.micro-epsilon.de/download/software/sensorTOOL.exe.

- Verbinden Sie den DT3005-Controller über einen USB/RS485-Konverter mit einem freien USB-Anschluss Ihres PCs und schließen Sie das Netzteil an den DT3005 an.
- Starten Sie das Programm sensorTOOL.
- Stellen Sie in den Dropdown Menüs die Sensorgruppe eddyNCDT und den Sensortyp eddyNCDT 3005 ein.

1) x = Kabellänge in Meter

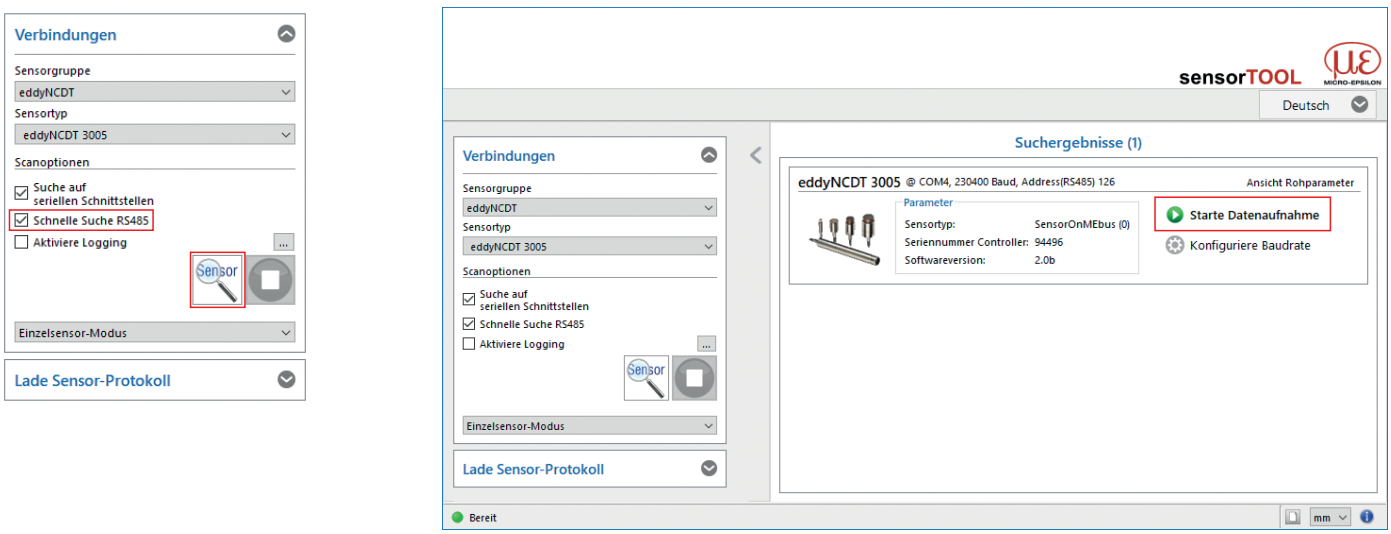


Abb. 3 Erste interaktive Seite nach Aufruf des `sensorTOOL`

- Wählen Sie den angeschlossenen Sensor aus.
- Setzen Sie den Haken bei `Suche auf seriellen Schnittstellen`.
- Falls nur 1 Controller am ME-Bus betrieben wird, setzen Sie den Haken bei `Schnelle Suche RS485`.

In diesem Fall werden alle Adressen gleichzeitig angefragt.

Falls mehrere Controller am ME-Bus betrieben werden, muss der Haken bei `Schnelle Suche RS485` deaktiviert werden. Die Suche dauert dann länger, da das Programm den gesamten Adressbereich einzeln abscannt, siehe auch Kapitel `Konfiguriere Baudrate`, siehe 3.2.

- Klicken Sie auf die Schaltfläche `Sensor` mit dem Lupensymbol, um die Suche zu starten.

In der Übersicht `Suchergebnisse (x)` werden nun alle verfügbaren Kanäle angezeigt.

- Wählen Sie einen gewünschten Sensor aus.

Über die Schaltflächen `Starte Datenaufnahme` und `Konfiguriere Baudrate` können nun weitere Menüs aufgerufen werden.

3.1 Menü Datenaufnahme

➡ Klicken Sie auf die Schaltfläche `Starte Datenaufnahme` oder auf das Controllersymbol, siehe Abb. 3, um weitere Einstellungen vorzunehmen und die Datenaufnahme zu starten.

Zur Überprüfung Ihrer Messungen steht Ihnen eine einfache Datenaufnahme zur Verfügung.

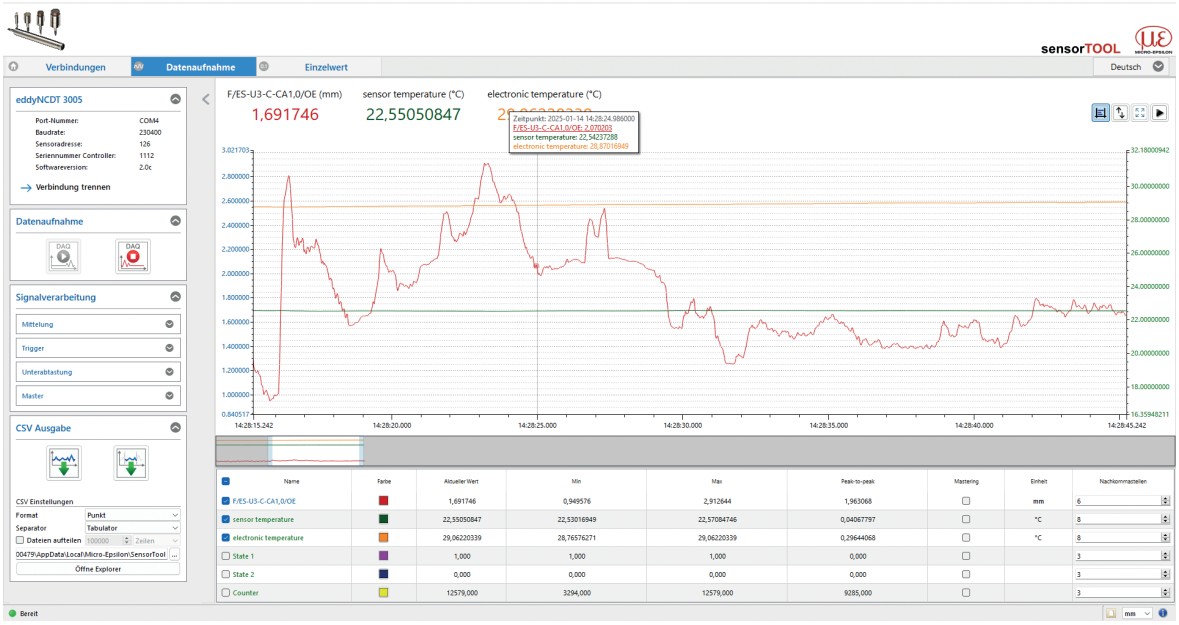
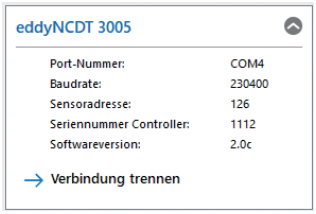


Abb. 4 Ansicht Menü Datenaufnahme



Bei Drücken der Schaltfläche `Verbindung trennen`, springt das Menü zur Controllersuche, siehe Abb. 3, zurück.

Abb. 5 Ansicht Verbindung trennen

	➡ Drücken Sie diese Schaltfläche `Skalierung zurücksetzen`, um die Y-Skala auf die ursprüngliche Einstellung zurückzusetzen (z.B. nach Zoom).
	➡ Drücken Sie diese Schaltfläche `Zum Jetzigen Zeitpunkt springen`, um den aktuellen Signalverlauf anzuzeigen.

3.1.1 Datenaufnahme

➡ Starten Sie die Datenaufnahme, indem Sie auf die Schaltfläche `Start` drücken, siehe Abb. 6. Die Aufnahme wird komplett neu gestartet, und die vorher angehaltene Aufnahme geht verloren.

➡ Stoppen Sie die Datenaufnahme, indem Sie auf die Schaltfläche `Stop` drücken, siehe Abb. 7.

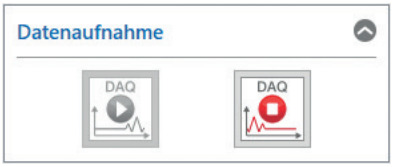


Abb. 6 Start *Abb. 7 Stop*

3.1.2 Signalverarbeitung

i Einstellungen zur Signalverarbeitung im `sensorTOOL` wirken sich nur auf die Daten im `sensorTOOL` und auf die CSV-Ausgabe aus. Die Signalverarbeitung im DT3005 Controller bleibt davon unberührt.

Abb. 8 Ausschnitt Signalverarbeitung

Folgende Auswahlmöglichkeiten bei der Signalverarbeitung stehen zur Verfügung:

Datenaufnahme	Signalverarbeitung	*Unterabtastung*	*Deaktiviert*	*Deaktiviert; Grundeinstellung*
			Messwertbasierend	*Anzahl der Samples ist einstellbar; jede x-te Messung wird erfasst.*
			Zeitbasierend	*Zeitbasiert; Zeit im Millisekundenbereich einstellbar* [1]
		Trigger	*Deaktiviert*	*Deaktiviert; Grundeinstellung*
			Kontinuierlich	*Manueller Trigger*
			Einmalig (messwertbasierend)	*Sample einstellbar; zeichnet Signalverlauf entsprechend den eingestellten Samples auf; je mehr Samples, desto länger der Verlauf*
			Einmalig (zeitbasierend)	*Millisekunden einstellbar; zeichnet Signalverlauf entsprechend der eingestellten Zeit auf.*
		Master	*Jetzt mastern*	*Setzt den Master, siehe Abb. 10.*
			Zurücksetzen	*Setzt den Master wieder zurück.*

1) Zum Beispiel alle 5000 ms: Nach dieser Zeit aktualisiert sich der angezeigt Verlauf.

3.1.3 CSV Ausgabe

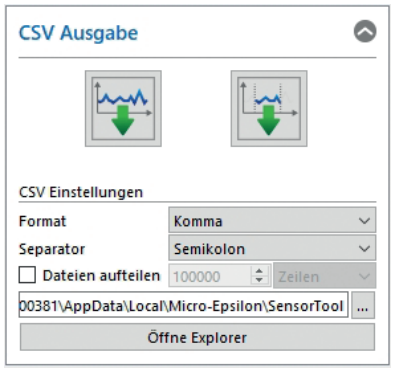


Abb. 9 Ausschnitt CSV Ausgabe

	Drücken Sie diese Schaltfläche, um die Messdatenaufzeichnung zu starten.
	Drücken Sie diese Schaltfläche, um die aktuelle Messwertauswahl zu speichern.

Datenaufnahme	CSV Ausgabe	*Format*	*Punkt / Komma*
		Separator	*Komma / Semikolon / Tabulator*

3.1.4 Tabelle Datenaufnahme

Name	Hier können Signalverläufe der eingesetzten Sensoren ein- und ausgeblendet werden.
Farbe	Hier können Farbeinstellungen der einzelnen Verläufe geändert werden.
Mastering	Durch Aktivieren der `Mastering` Checkbox kann der Masterwert manuell eingetragen werden. Die Masterwerte werden durch `Jetzt mastern` im Menü `Datenaufnahme` > `Signalverarbeitung` im Reiter `Master` gesetzt, siehe Abb. 8.
Einheit	*Auswahl des Ausgangs, der angezeigt werden soll. Die Ausgänge werden im Menü `Einstellungen` unter `Ausgabe` / `Ausgabebereich` und `Justierung` vorher eingestellt.*
Nachkommastellen	0 - 12

Abb. 10 Tabelle Datenaufnahme

3.2 Konfiguriere Baudrate

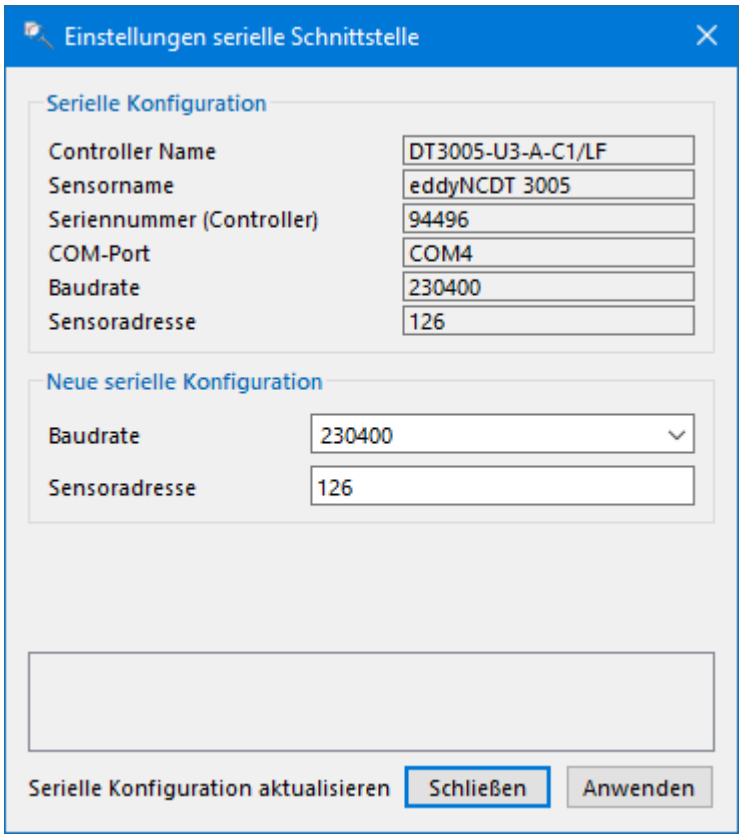


Abb. 11 Fenster Einstellungen serielle Schnittstelle

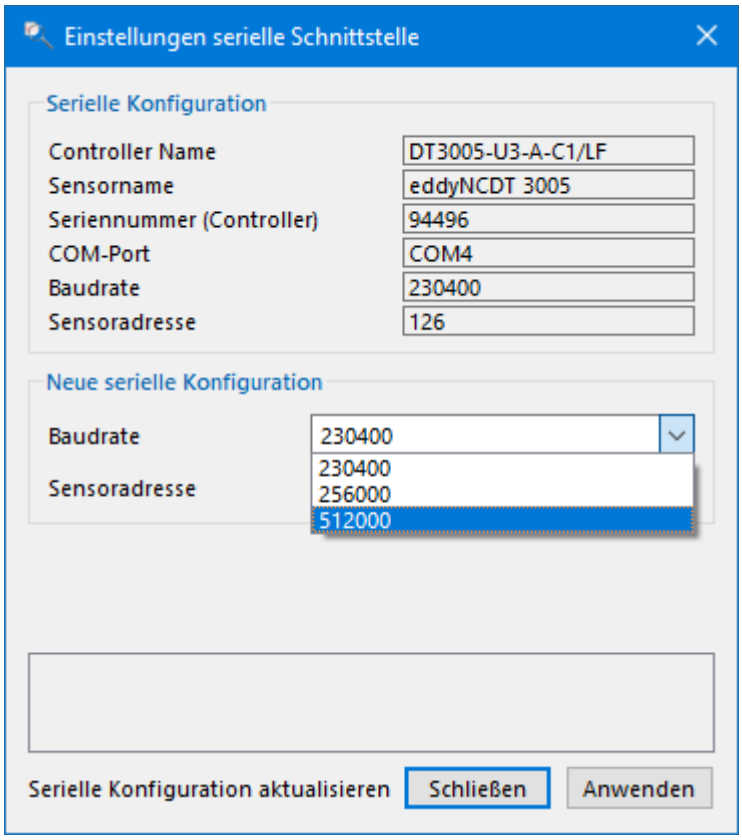
Um die aktuelle Konfiguration der seriellen Schnittstelle einzusehen und diese gegebenenfalls zu ändern, klicken Sie auf die Schaltfläche `Konfiguriere Baudrate`, siehe Abb. 3.

Daraufhin öffnet sich das Fenster `Einstellungen serielle Schnittstelle`. Hier kann die Baudrate geändert, siehe Abb. 12, und eine neue Adresse für das Gerät vergeben werden, siehe Abb. 13.

3.2.1 Baudrate ändern

Im Fenster `Einstellungen serielle Schnittstelle` kann die Baudrate aus einem Dropdown-Menü ausgewählt werden.

i Bei den zur Verfügung stehenden Baudraten handelt es sich um diejenigen Baudraten, die vom `sensorTOOL` unterstützt werden und nicht um die Baudraten, die vom eddyNCDT 3005 unterstützt werden.



Vom eddyNCDT 3005 werden folgende Baudraten unterstützt:

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

Byte Rahmen: 1 Start Bit, 8 Daten Bits, 1 Parity Bit (parity = even), 1 Stop Bit

Abb. 12 Fenster Einstelllungen serielle Schnittstelle - Ansicht Baudrate

3.2.2 Adressvergabe

Die eddyNCDT 3005-Systeme werden standardmäßig mit der Adresse 126 ausgeliefert. Die Adresse eines angeschlossenen Geräts kann ebenso im Fenster `Einstellungen serielle Schnittstelle`, siehe Abb. 13, geändert werden.

Gültige Adressen sind die Adressen 1 bis einschließlich 126. Um eine Adresse einzustellen, wird die gewünschte Adresse in das Feld `Sensoradresse` geschrieben und anschließend mit Klick auf die Schaltfläche `Anwenden` bestätigt.

i Bei mehreren Geräten am ME-Bus ist unbedingt darauf zu achten, dass jede Adresse am Bus nur einmal vergeben wird.

Handelt es sich bei der geschriebenen Adresse um eine gültige Adresse, so wird die Adresse von Gerät übernommen und es erscheint folgende Meldung:

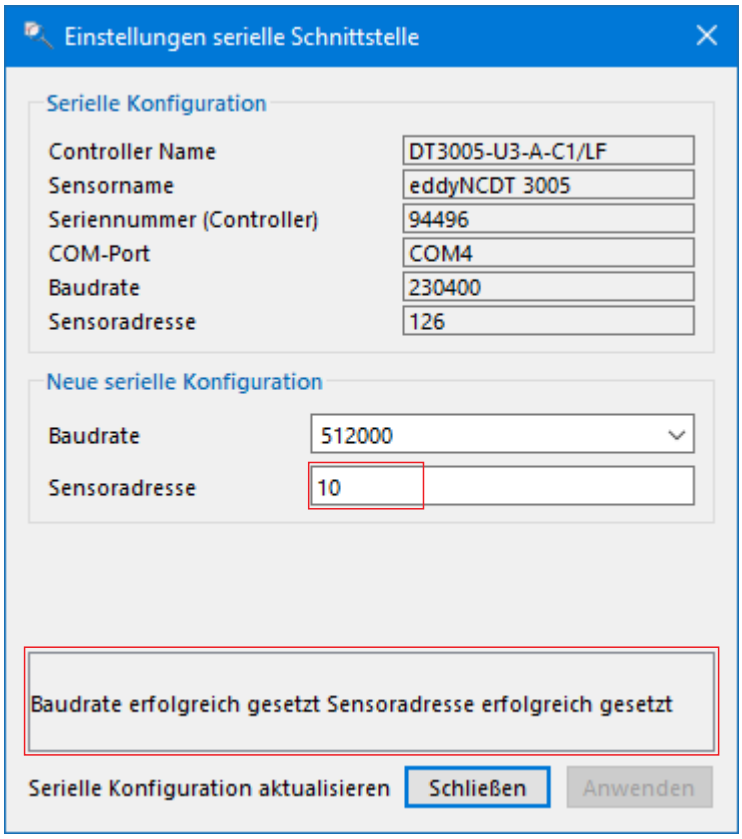


Abb. 13 Meldung bei erfolgreicher Änderung der Sensoradresse

Wird das Fenster `Einstellungen serielle Schnittstelle` anschließend geschlossen, so erscheint folgende Meldung:

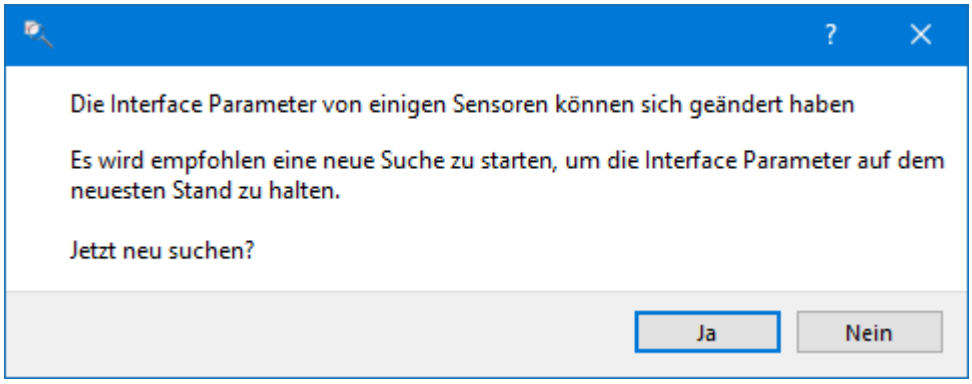


Abb. 14 Meldung bei neuen Interface-Parametern

3.3 Multi-Sensor DAQ Modus

Das Programm sensorTOOL bietet auch die Möglichkeit, die Daten von mehreren Kanälen der Serie eddyNCDT 3005 auszugeben.

i Bitte beachten Sie, dass es sich bei der RS485-Schnittstelle um einen seriellen Bus handelt.
Auch, wenn die Messwerte im sensorTOOL gleichzeitig ausgegeben werden, werden sie jedoch zeitversetzt aufgenommen.

Um die Daten von mehreren Busteilnehmern in einen Graphen auszugeben, gehen Sie bitte wie folgt vor:

➡ Suchen Sie den Controller über das Programm sensorTOOL, siehe Abb. 3.

i Beachten Sie hierbei, dass die Checkbox Schnelle Suche RS485 deaktiviert sein muss, siehe Abb. 15, um mehrere Kanäle zu finden.

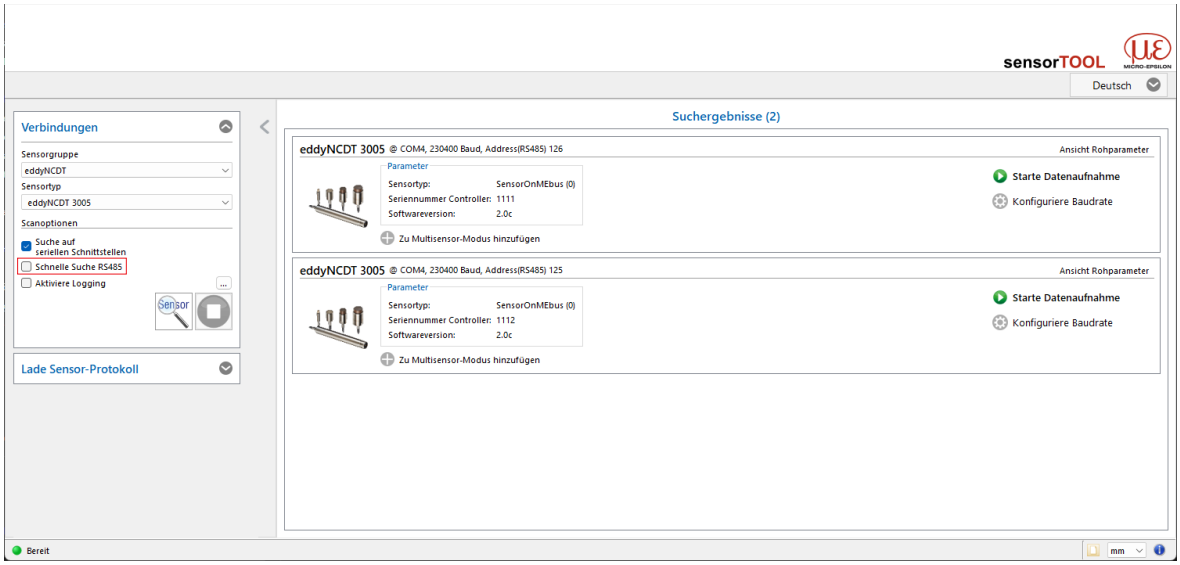


Abb. 15 Erste interaktive Seite nach Aufruf des sensorTOOL

➡ Aktivieren Sie nun die einzelnen Checkboxen In Multisensor-Modus verwenden der jeweiligen Kanäle.

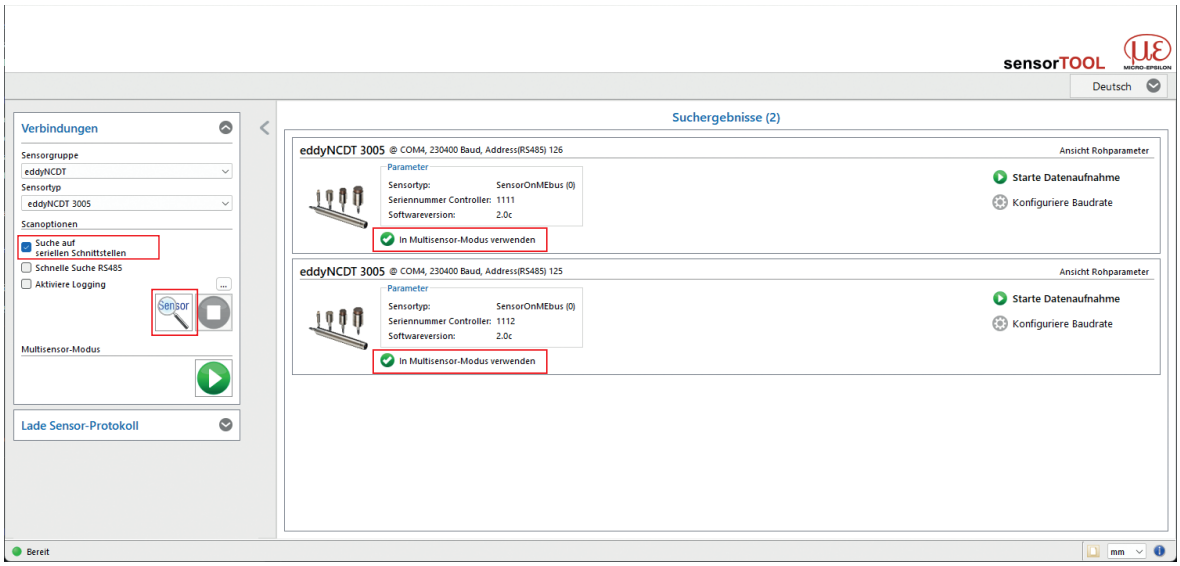


Abb. 16 Erste interaktive Seite nach Aufruf des sensorTOOL für den Multi-Sensor DAQ Modus

Drücken Sie nun die Schaltfläche .

Im Menü `Datenaufnahme`, siehe Abb. 17, erscheint nun die Datenausgabe mit den Daten der ausgewählten Kanäle.

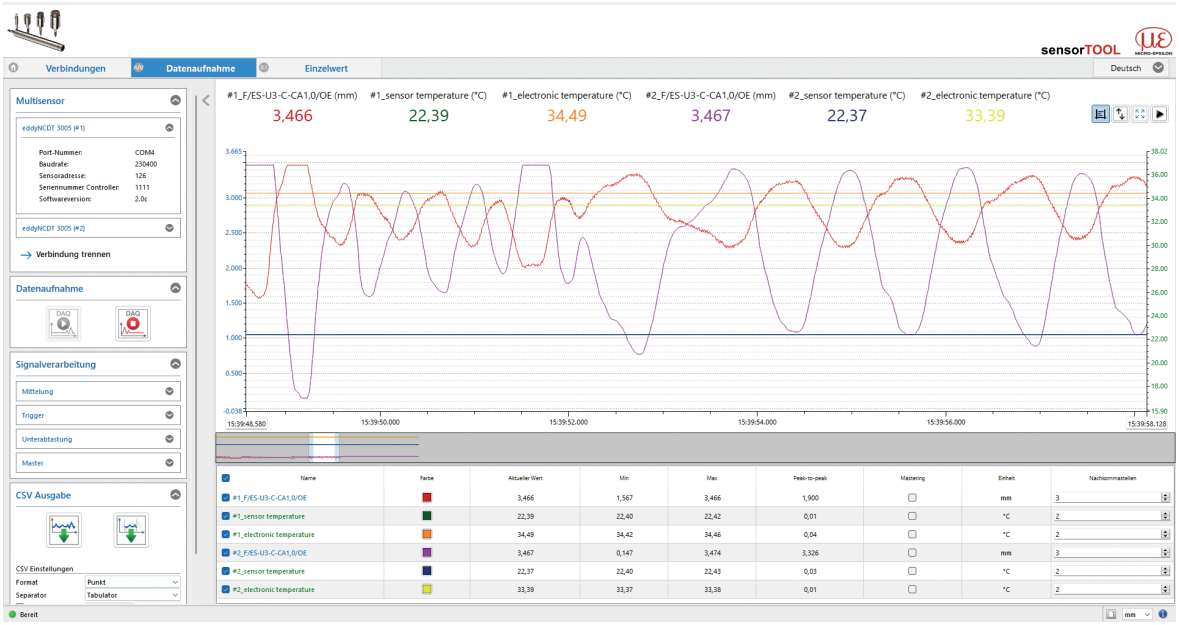


Abb. 17 Menü `Datenaufnahme`, Multi-Sensor DAQ Modus

Im Menü `Einzelwert` können die Daten auch als Zahlenwert dargestellt werden.

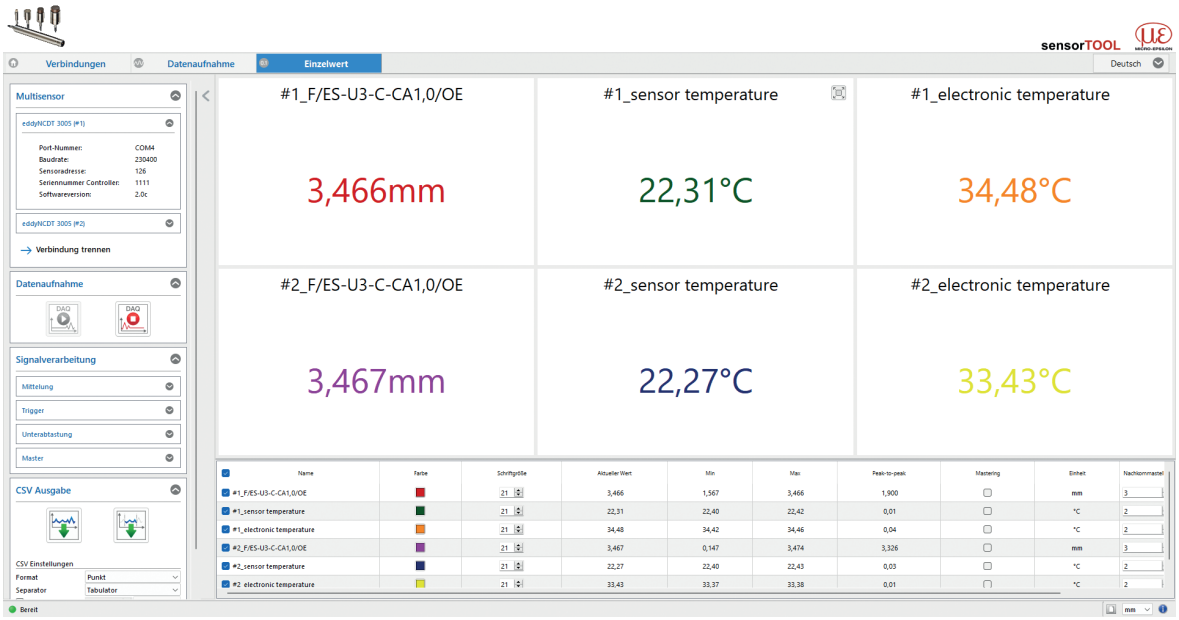


Abb. 18 Ansicht Menü `Einzelwert`, Multi-Sensor DAQ Modus

4. Schnittstellen

4.1 Gegenüberstellung IF1032/ETH und IF2035

RS485 Schnittstelle zum DT3005	
IF1032/ETH	Unterstützt nur einen Teilnehmer am RS485 Bus.
IF2035	Unterstützt bis zu 32 Teilnehmer am RS485 Bus.
Schnittstelle zum Kunden	
IF1032/ETH	Ist durch den Kunden konfigurierbar (Schalter auf Platine und per Software) zwischen Ethernet und EtherCAT. Werkseinstellung ist Ethernet, siehe 4.3.
IF2035	In 3 Varianten verfügbar: EtherCAT, PROFINET, EtherNet/IP, siehe 4.2

4.2 IF2035

4.2.1 Anschlussdiagramm

Die Versorgungsspannung wird von der Versorgungsbuchse (Klemme 1) zur Sensorbuchse (Klemme 2) durchgeschleift. Die positive Spannung muss zwischen 12 V und 32 V liegen.

Die Anschlussbelegung des eddyNCDT 3005-Steckers, siehe Abb. 1.

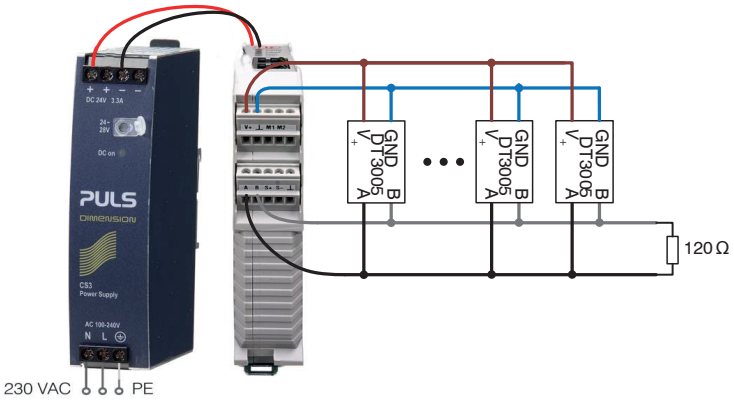


Abb. 19 Anschluss der eddyNCDT 3005 Controller an das Schnittstellenmodul IF2035 mit optionalem Netzteil PS2020

Micro-Epsilon empfiehlt einen Abschlusswiderstand von 120 Ω zwischen den Signalleitungen sowohl am Busanfang und -ende. Im IF2035 ist bereits ein 120 Ω Abschlusswiderstand fest integriert.

4.2.2 Hardware-Schnittstelle

Physikalische Schnittstelle:	RS485 halbduplex
Baudrate:	230400 Bit/s (Standard); zusätzlich unterstützt werden 256000, 460800 und 512000 Bit/s
Byte-Rahmen:	1 Startbit, 8 Datenbits, 1 Paritätsbit (Parität = gerade), 1 Stoppbit
ME-Busadresse:	126 (Standard)

Die interne Erfassungsrate des eddyNCDT 3005 beträgt 75 kSPS. RS485 ist eine Busschnittstelle. Bis zu 32 DT3005 können an denselben IF2035 angeschlossen werden. Die ME-Bus-Adresse darf nur einmal auf dem Bus vorkommen, da die gesendeten Daten sonst nicht interpretiert werden können. Um die ME-Bus-Adresse eines DT3005 zu ändern, siehe Abb. 13.

4.2.3 Datenformat

Die Strukturdaten sind 6 Bytes lang und enthalten die Messwerte. Aufbau wie folgt:

Datentyp	Bezeichnung	Beschreibung
Uint16	distance	Abstand Messobjekt
Uint16	temperature_sensor	Sensortemperatur
Uint16	temperature_electronic	Elektroniktemperatur

4.2.3.1 Berechnung von Abstandswerten

Berechnung digitaler Abstandswerte für einen U3-Sensor mit einem Messbereich von 3 mm.
(MBA = 0,3 mm, MBE = 3,3 mm)

	Abstand in Ziffern	Abstand in mm
MBA	3000	0,3
MBE	62000	3,3

Umrechnung Digitalwerte

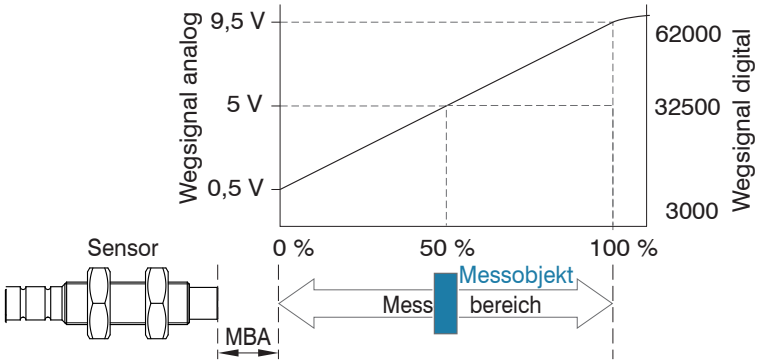


Abb. 20 Messbereichsanfang (MBA), der kleinste Abstand zwischen Sensorstirnfläche und Messobjekt

Abstand mit MBA (Messbereichsanfang):

$$d = \frac{(x - 3000) \cdot MB}{59000} + MBA \qquad x = [3000 \ldots 62000]$$

i Im `sensorTOOL` wird die Formel inklusive dem Messbereichsanfang angezeigt.

Abstand ohne MBA (Messbereichsanfang):

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

Der Messbereichsanfang MBA beträgt standardmäßig 10 % des Messbereiches. Diesen sensorspezifischen Wert finden Sie in der jeweiligen Montageanleitung eddyNCDT 3005.

4.2.3.2 Berechnen der Sensortemperatur

	Temperatur in Ziffern	Temperatur in °C
MBA	3000	-40
MBE	62000	200

Umrechnung Digitalwerte

$$\vartheta_s = \frac{(x_s - 3000) \cdot 240\ °C}{59000} - 40\ °C \qquad x_s = [3000 \ldots 62000]$$
$$\vartheta_s = [-40\ °C \ldots +200\ °C]$$

4.2.3.4 Berechnen der Elektronik- (Controller-) Temperatur

	Temperatur in Ziffern	Temperatur in °C
MBA	3000	-25
MBE	62000	85

Umrechnung Digitalwerte

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

x_c = [3000 ... 62000]
ϑ_c = [-25 °C ... +85 °C]

4.2.3.5 Beispiel für die Übertragung eines Messwertes

Anzahl Bytes	Byte	Beschreibung	Bedeutung / Wert
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

Der Abstand zum Messobjekt beträgt 3 mm/59000 x (35776 - 3000) + 0,3 mm = **1,967 mm**.

Die Sensortemperatur beträgt 4,0678e-3 °C x 31305 - 52,20 °C = **75,14 °C**.

Die Elektroniktemperatur beträgt 1,86441e-3 °C x 30724 - 30,59 °C = **26,69 °C**.

4.3 IF1032/ETH

4.3.1 Anschluss-Schaltbild

Die Versorgungsspannung wird vom Versorgungsanschluss des IF1032/ETH zum Anschluss des Sensoranschlusses durchgeschleift. Die positive Spannung muss zwischen 12 V und 32 V liegen.

Anschlussbelegung des DT3005 Steckers, siehe Abb. 2.

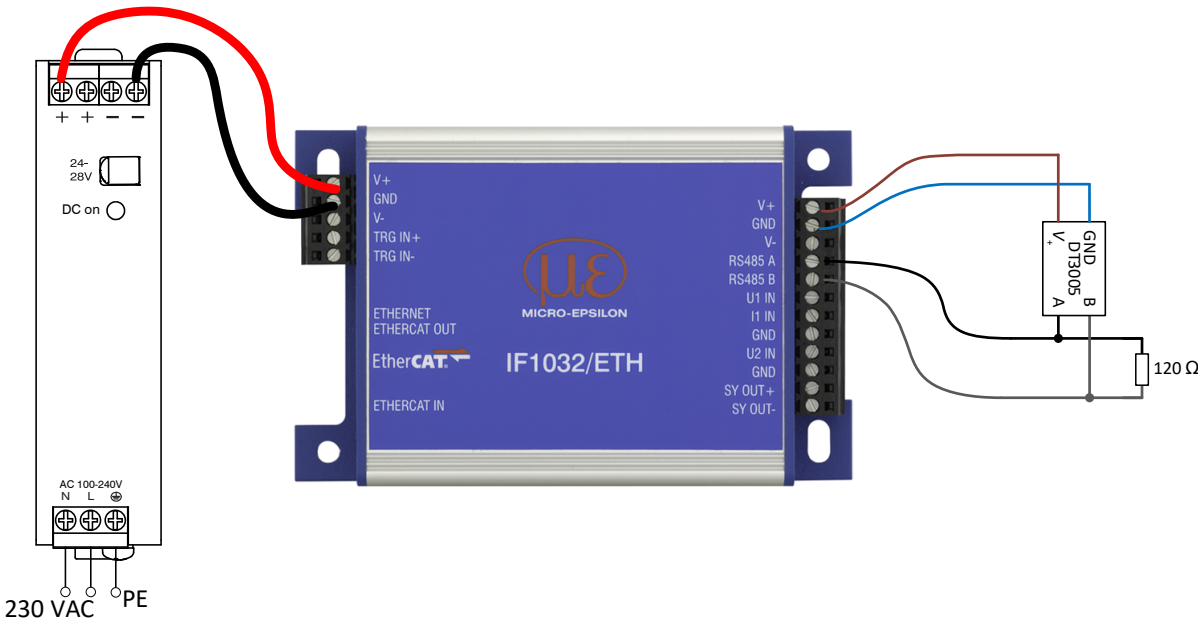


Abb. 21 Anschluss der eddyNCDT 3005 Controller an das Schnittstellenmodul IF1032/ETH mit optionalem Netzteil PS2020

Es kann nur ein DT3005 an den RS485-Bus des IF1032/ETH angeschlossen werden.

Micro-Epsilon empfiehlt einen Abschlusswiderstand von 120 Ω zwischen den Signalleitungen sowohl am Busanfang und -ende. In der IF1032/ETH ist ein 120 Ω Abschlusswiderstand bereits fest eingebaut.

4.3.2 Sensorschnittstelle

Das IF1032/ETH unterstützt nur einen eddyNCDT 3005 an der RS485-Schnittstelle.

Im Webinterface des IF1032/ETH muss die Sensorschnittstelle auf RS485 umgestellt werden.

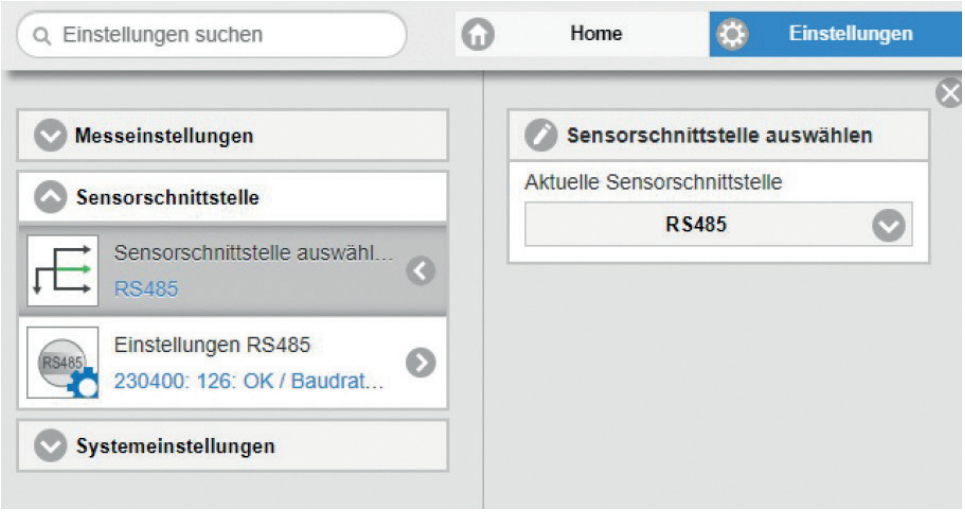


Abb. 22 Ansicht Wechsel zu RS485

Die Standardeinstellungen des DT3005 an der RS485-Schnittstelle sind Sensor-Baudrate 230400 Baud und Sensoradresse 126.

Die Baudrate des Sensors kann über das Dropdown-Menü oder durch Bearbeiten des Feldes `Sensor Baudrate (baud)` auf eine der möglichen Baudraten 230400, 256000, 460800 oder 512000 geändert werden.

Die Sensoradresse wird auch geändert, wenn die Feldsensoradresse bearbeitet wird.

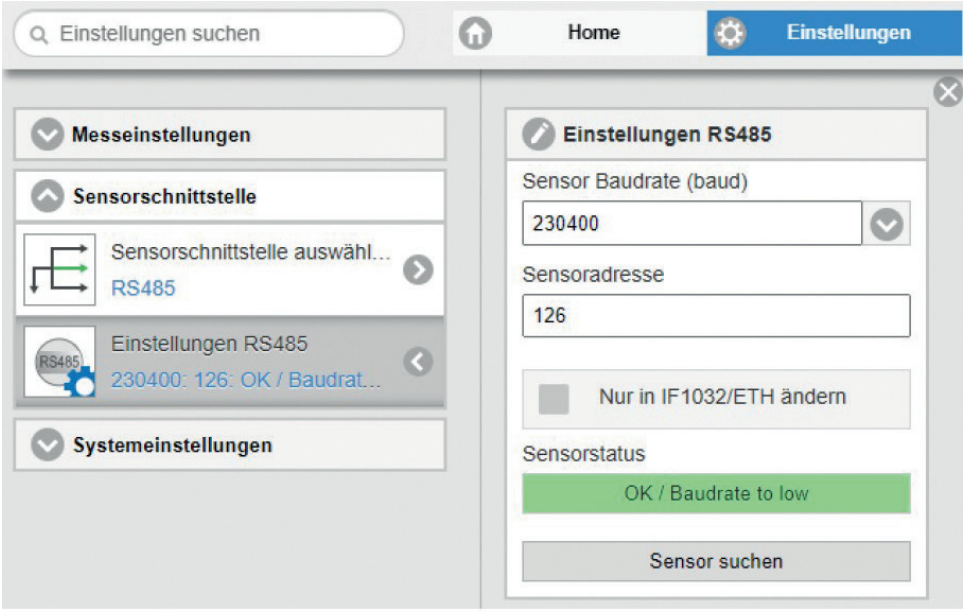


Abb. 23 Ansicht Wechsel der Baudrate

Die interne Erfassungsrate des eddyNCDT 3005 beträgt 75 kSa/s. Es ist nicht möglich, jede Probe über die RS485-Schnittstelle zu übertragen. Die aktuelle Datenrate wird unter `Messeinstellungen > Messmodus` angezeigt, siehe Abb. 24.

Die mögliche Übertragungsrate hängt von der eingestellten Baudrate der RS485 Schnittstelle ab. Um die größte Übertragungsrate zu erreichen, muss auch die höchste Baudrate eingestellt werden.

Baudrate	Maximale Übertragungsrate
230400	831.6 Sa/s
256000	907.0 Sa/s
460800	1425.5 Sa/s
512000	1534.9 Sa/s

4.3.3 Messeinstellungen - Messmodus

Im DT3005 kann ein arithmetischer Mittelwert über 2 bis 65535 Werte berechnet werden. Dadurch reduziert sich die Rate, mit welcher das DT3005 Abstandswerte ausgibt. Bei einer Mittelung über 1000 Werte beträgt die Messrate z. B. noch 75 Sa/s.

Über die wählbare Anzahl N aufeinanderfolgender Messwerte wird der arithmetische Mittelwert M gebildet und ausgegeben.

Verfahren

Es werden Messwerte gesammelt und daraus der Mittelwert berechnet. Diese Methode führt zu einer Reduzierung der anfallenden Datenmenge, weil nur nach jedem N-ten Messwert ein Mittelwert ausgegeben wird.

Beispiel mit N = 3:

.... 0 1 2 3 4 ... wird zu $\frac{2+3+4}{3}$ Mittelwert n

.... 3 4 5 6 7 ... wird zu $\frac{5+6+7}{3}$ Mittelwert n + 1

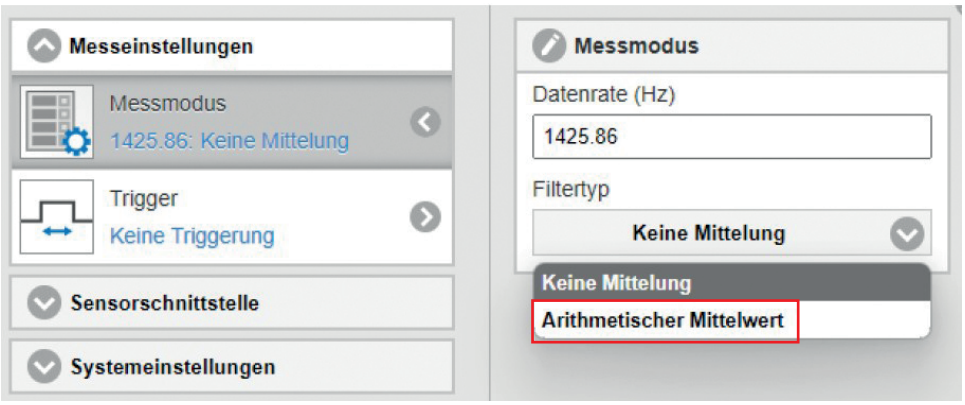


Abb. 24 Ansicht Einstellung Messmodus - Arithmetischer Mittelwert

Der Wert unter `Datenrate (Hz)` entspricht der Datenrate vor der Mittelung.

Nach der Einstellung von `Arithmetischer Mittelwert` beträgt die Datenrate 75000 Hz/100 = 750 Hz.

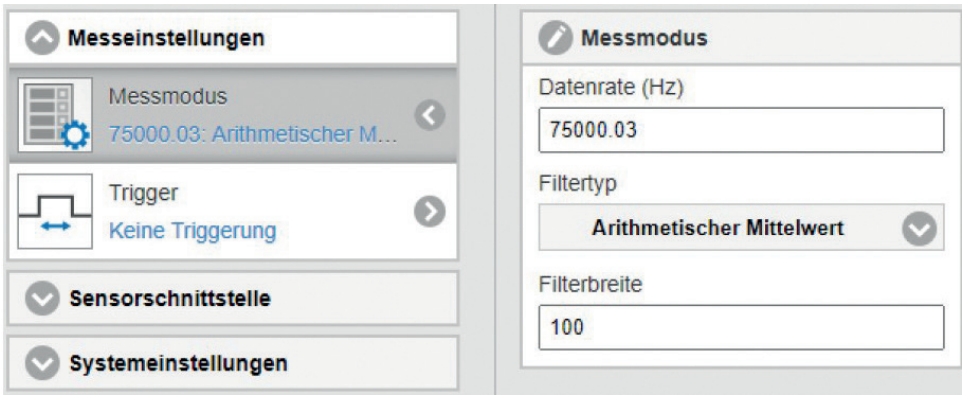


Abb. 25 Ansicht Einstellung Messmodus - Arithmetischer Mittelwert

5. MEDAQLib

Mit MEDAQLib steht Ihnen eine dokumentierte Treiber-DLL zur Verfügung. Damit binden Sie Sensoren von Micro-Epsilon in Verbindung mit einem Konverter oder Schnittstellenmodul in eine bestehende oder kundeneigene PC-Software ein.

MEDAQLib

- enthält eine DLL, die in C, C++, VB, Delphi und viele weitere Programme importiert werden kann,
- nimmt Ihnen die Datenkonvertierung ab,
- funktioniert unabhängig vom verwendeten Schnittstellentyp,
- zeichnet sich durch gleiche Funktionen für die Kommunikation (Befehle) aus,
- bietet ein einheitliches Übertragungsformat für alle Sensoren von Micro-Epsilon.

Für C/C++-Programmierer ist in MEDAQLib eine zusätzliche Header-Datei und eine Library-Datei integriert.

- Die MEDAQLib Installationsdateien können Sie über den Link https://www.micro-epsilon.de/link/software/medaqlib auf Ihren Rechner laden.
- Für weitere Informationen zur MEDAQLib verwenden Sie bitte die Seite https://www.micro-epsilon.de/service/software-sensorintegration/medaqlib.

5.1 Unterstützte ME-Bus-Sensor-Befehle

Befehl	Unterstützt
Logout	nein
Login	nein
Get_UserLevel	nein
Set_Password	nein
Set_Samplerate	nein
Get_Samplerate	ja
Set_Trigger	nein
Get_Trigger	nein
Set_Averaging	ja
Get_Averaging [1]	ja
Get_Measure	ja
Get_AlternateMeasure	nein
Set_ContinuousMode	nein
Get_ContinuousMode	ja
Set_Range	nein
Test_Baudrate	ja
Set_Baudrate	ja
Get_Baudrate	ja
Set_SensorAddress	ja
Get_SensorInfo	ja
Get_Channelinfo	ja
Get_Channelinfos	ja
Get_ControllerInfo	ja
Get_DiagnosticInfo	nein
Get_DiagnosticInfo	nein
Get_ConfigDescription	nein
Set_ConfigParameter	nein
Get_ConfigParameter	nein
Read_AllBlocks	ja

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

5.2 Beispiele

Die folgenden Beispiele lesen den Namen, die Seriennummer des DT3005 und die Beschreibung der Messwerte aus. Dann werden einige Messwerte aus dem DT3005 ausgelesen.

5.2.1 Python

➪ Kopieren Sie die beiden Dateien `MEDAQLib.dll` und `MEDAQLib.py` aus dem Unterverzeichnis `Snippets/Python` im MEDAQLib-Installationsverzeichnis in das gleiche Verzeichnis wie den Python-Quellcode.

```
#
# 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#

➡ Kopieren Sie die beiden Dateien `MEDAQLib.dll` und `MEDAQLib.Net.dll` aus dem Unterverzeichnis `Release` im MEDAQLib-Installationsverzeichnis in das gleiche Verzeichnis wie den 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

Kopieren Sie die beiden Dateien `MEDAQLib.h` und `Release-x64\MEDAQLib.dll` aus dem MEDAQLib-Installationsverzeichnis in das gleiche Verzeichnis wie das MATLAB-Skript.

```
%%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
```

Anhang

Optionales Zubehör

PC5/5-M12		Versorgungs- und Ausgangskabel, 5 m lang
PC10/5-M12		Versorgungs- und Ausgangskabel, 10 m lang
PC20/5-M12		Versorgungs- und Ausgangskabel, 20 m lang
PS2020		Eingang 100...240 VAC Ausgang 24 VDC / 2.5 A, für Schnappmontage auf DIN 50022-Schiene
IF7001		IF7001 Einkanal USB/RS485 Konverter
IF1032/ETH		Mehrkanaliger Ethernet- und EtherCAT-Konverter - drei analoge Eingänge - eine RS485 (einkanalig)
IF2035-EtherCAT		Schnittstellenmodul für EtherCAT
IF2035-PROFINET		Schnittstellenmodul für PROFINET
IF2035-EIP		Schnittstellenmodul für Ethernet/IP

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

X9750337.01-A012025HDR