HYDAC FILTER SYSTEMS

FluidMonitoring Toolkit FluMoT

Version V02.00Rxx

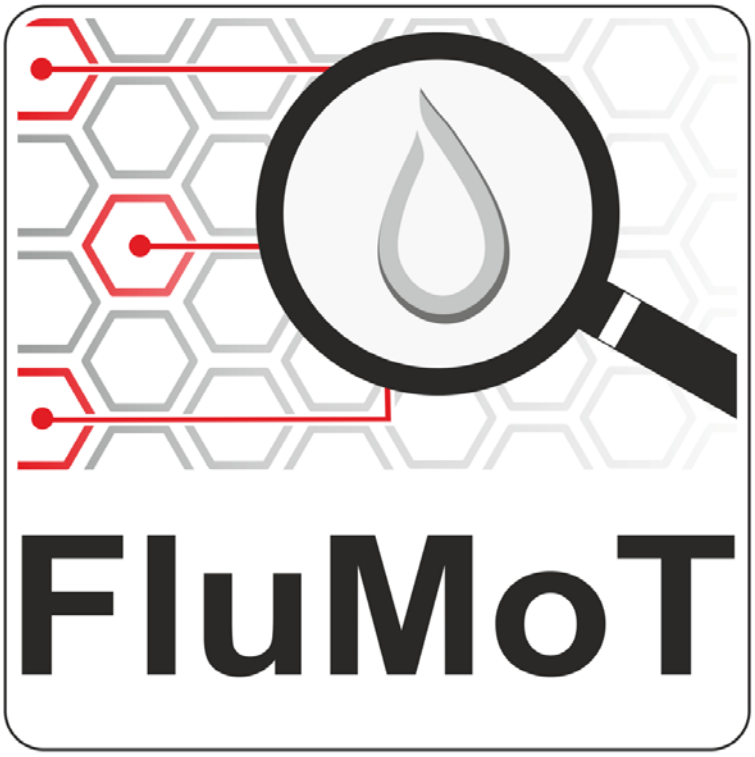
Für:

- CS 1000 / CS 2000 Serie
- AS 1000 Serie
- HLB 1000 Serie
- HMG 3000 Serie
- CMU 1000 Serie
- FCU 1000 / 2000 / 8000 Serie
- CSM 1000 / 2000 Serie
- FMM Serie
- CSI-C-11

Bedienungsanleitung

Deutsch (Originalanleitung)

Dokument-Nr.: 3377564b



Warenzeichen

Die verwendeten Warenzeichen anderer Firmen bezeichnen ausschließlich die Produkte dieser Firmen.

Copyright © 2021 by HYDAC FILTER SYSTEMS GMBH Alle Rechte vorbehalten

Haftungsausschluss

Wir haben unser Möglichstes getan, die Richtigkeit des Inhalts dieses Dokuments zu gewährleisten, dennoch können Fehler nicht ausgeschlossen werden. Deshalb übernehmen wir keine Haftung für Fehler und Mängel in diesem Dokument, auch nicht für Folgeschäden, die daraus entstehen können. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, und notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Anregungen und Verbesserungsvorschläge sind wir dankbar.

Technische Änderungen bleiben vorbehalten.

Inhaltliche Änderungen dieses Handbuchs behalten wir uns ohne Ankündigung vor.

HYDAC FILTER SYSTEMS GMBH
Postfach 12 51
66273 Sulzbach / Saar

Deutschland

Dokumentationsbevollmächtigter

Herr Günter Harge

c/o HYDAC International GmbH, Industriegebiet, 66280 Sulzbach / Saar

Telefon:	+49 (0)6897 509 1511
Telefax:	+49 (0)6897 509 1394
E-Mail:	guenter.harge@hydac.com

Inhalt

FluMoT Registrierkarte ausfüllen

Registrierung FluMoT Version 2.0x

Mit dem Öffnen der Datenträgerverpackung bzw. Installieren der Software haben Sie sich mit den im Software-Überlassungsvertrag aufgeführten Nutzungsbedingungen einverstanden erklärt.

Senden Sie ergänzend diese Registrierkarte ausgefüllt an uns zurück und Sie werden bei uns als Benutzer des Programms registriert.

Nutzen Sie die Vorteile die Ihnen eine Registrierung bietet:

- Kostenloser Support via E-Mail: filtersysteme_support@hydac.com
- News zur Software
- Informationen über Updates der Software

Wir garantieren Ihnen, Ihre Daten nicht an Dritte weiterzugeben.

FluMoT Registrierungs-Schlüssel

Firma

Straße

Postleitzahl, Ort

Name des Benutzers

E-Mail Adresse

Ort, Datum, Unterschrift

Bitte senden Sie die vollständig ausgefüllte Registrierungskarte per Post, Fax oder E-Mail zurück an:

HYDAC FILTER SYSTEMS GMBH
Industriestraße, Werk 6, D-66280 Sulzbach / Saar,
Fax: ++49 (0) 6897 / 509-846, E-Mail: filtersystems_support@hydac.com

HINWEIS

Ohne Registrierung wird ein Software Support durch HYDAC verweigert!

FluMoT Merkmale

Das FluidMonitoring Toolkit **-** FluMoT ist ein Treiberpaket, welches der Einbindung von HYDAC Fluidsensoren oder Sensoren der neuen Generation mit HSI-Schnittstelle in kundeneigene PC-Software dient.

FluMoT löst somit alle bisherigen Treiber (z.B. für Windows, LabVIEW) zur Einbindung von ContaminationSensoren sowie FluidControl Units ab.

Das Treiberpaket FluMoT besteht aus folgenden Bausteinen:

- Dynamische Programmbibliotheken (DLLs) in 32- und 64-bit Format	
→	HSI/HeCom (hecom32.dll, hecom 64.dll)
→	HSITP (hsitp32.dll, hsitp64.dll)
→	DinMessBus (dinmessbus32.dll, dinmessbus64.dll)
- Beispielprogramme in verschiedenen Programmiersprachen für die Einbindung von DLLs:	
→	Delphi 10, 10.4
→	LabView 7, 2017
→	VB/VBA
→	Visual Studio 2017 C#
- OPC Server (DA/UA) zur Einbindung von HYDAC-Sensoren	
- Programmbausteine für SIEMENS Steuerungen	
- Programmbausteine für CODESYS (Beckhoff TWINCAT 3)	

Dieses Treiberpaket stellt einem Programmierer, der Anwendungen entwickelt, einige Standardfunktionen sowie Schnittstellen zur Verfügung.

Es handelt sich dabei um komplett Lösungen als auch um Schnittstellen, die für eine Softwareentwicklung gedacht sind.

Mit **FluMoT** können folgende Geräte abgefragt werden:

- ContaminationSensor CS 1000, CS 2000
- FluidControl Unit FCU1000, FCU2000, FCU8000
- AquaSensor AS 1000 Serie
- ContaminationSensor Module CSM 1000, CSM 2000
- FluidMonitoring Module
- HYDACLab (HLB 1000)
- HMG 3000
- CMU 1000
- CSI-C-11

Wie die Sensoren bzw. Geräte angeschossen werden, entnehmen Sie bitte der jeweiligen Bedienungsanleitung.

Sicherheitshinweise

Wir setzen voraus, dass Sie mit der Bedienung von WINDOWS Betriebssystem, dem Aufbau und der Installation Windows typischer Programme vertraut sind.

Symbol- und Hinweiserklärung

In dieser Bedienungsanleitung werden folgende Benennungen und Zeichen für Hinweise verwendet:

HINWEIS

Der gelbe Streifen gibt wichtige **Hinweise** für den sachgerechten Umgang mit dem Produkt. Das Nichtbeachten dieser Hinweise kann zu Fehlbedienung bzw. Funktionsstörungen führen.

INFO

Unter blauem Streifen werden die wichtigen **Informationen** zusammengefasst.

TIPP

Mit dem grünen Streifen werden die **Anwendungstipps** und besonders nützliche Informationen bezeichnet.

Bei Fragen, Problemen und Anregungen zu **FluMoT** wenden Sie sich bitte an unseren Technischen Vertrieb.

HYDAC FILTER SYSTEMS GMBH
Postfach 12 51
D-66273 Sulzbach / Saar - Deutschland

E-Mail: filtersystems_support@hydac.com
Fax.: +49 (0) 6897 509 9046

Systemvorrausetzungen

Hardware

- Pentium Prozessor 200 MHz oder höher
- 64 MB RAM-Speicher.
- VGA-Grafikkarte (800x600 min.)
- Festplatte mit mindestens 30 MB freiem Speicherplatz
- Eine freie serielle Schnittstelle (RS232 / USB):
  1. Nicht mit einem Stecker belegt ist
  2. Nicht vom Betriebssystem benutzt wird
  3. Von keinem anderen Programm benutzt wird
     (wie z. B. Terminal-, Modem- oder Netzwerksoftware)
- Microsoft Windows kompatible Maus

Software

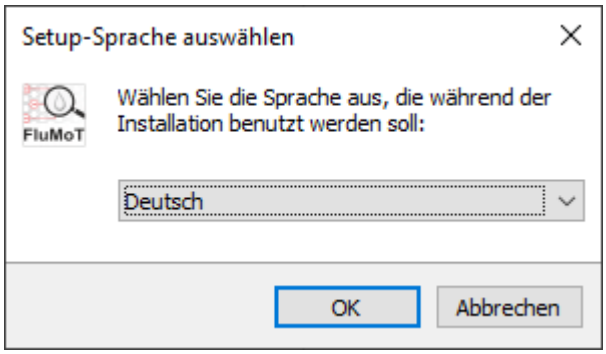
- WINDOWS 98, 2000, ME, XP, Server 2003, Windows Vista (32bit), Windows 7, Windows 8, Windows 10
- Microsoft Internet Explorer 4.0 oder höher
- Administrator Rechte zur Softwareinstallation

FluMoT installieren

Deinstallieren Sie vor der Installation alle älteren Versionen von FluMoT.

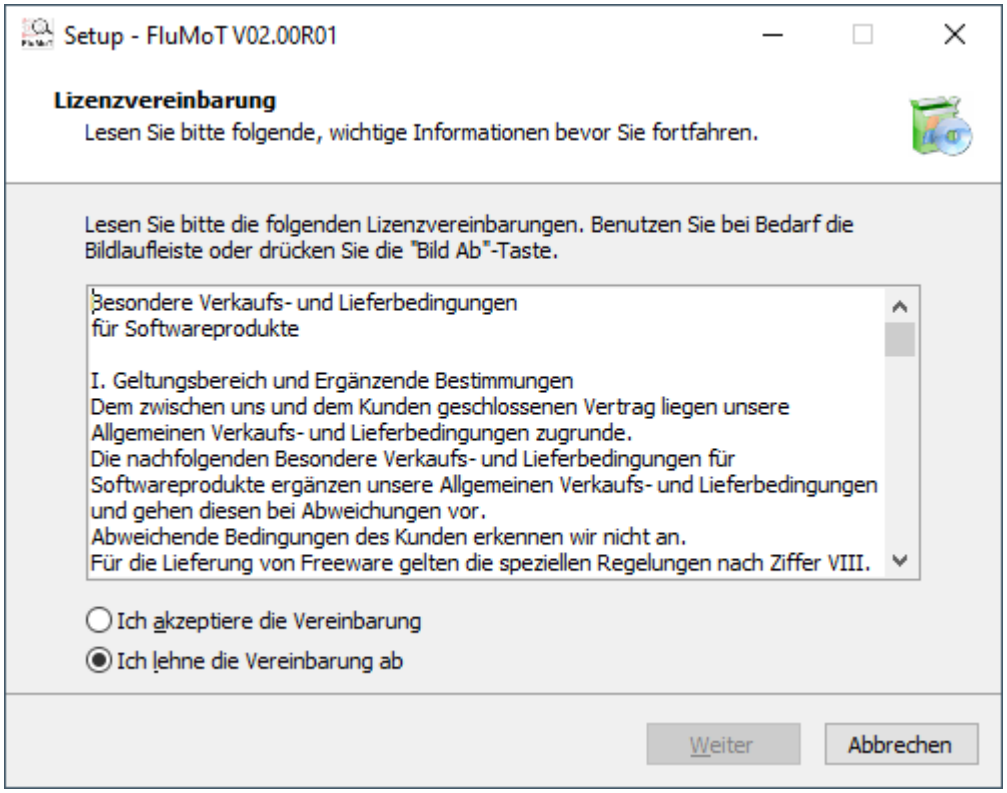
Entnehmen Sie die Details zum Anschluss des Gerätes / Senors der jeweiligen Bedienungsanleitung.

Zur Installation von FluMoT, starten Sie das Programm SETUP_FLUMOT_V02xx.EXE.

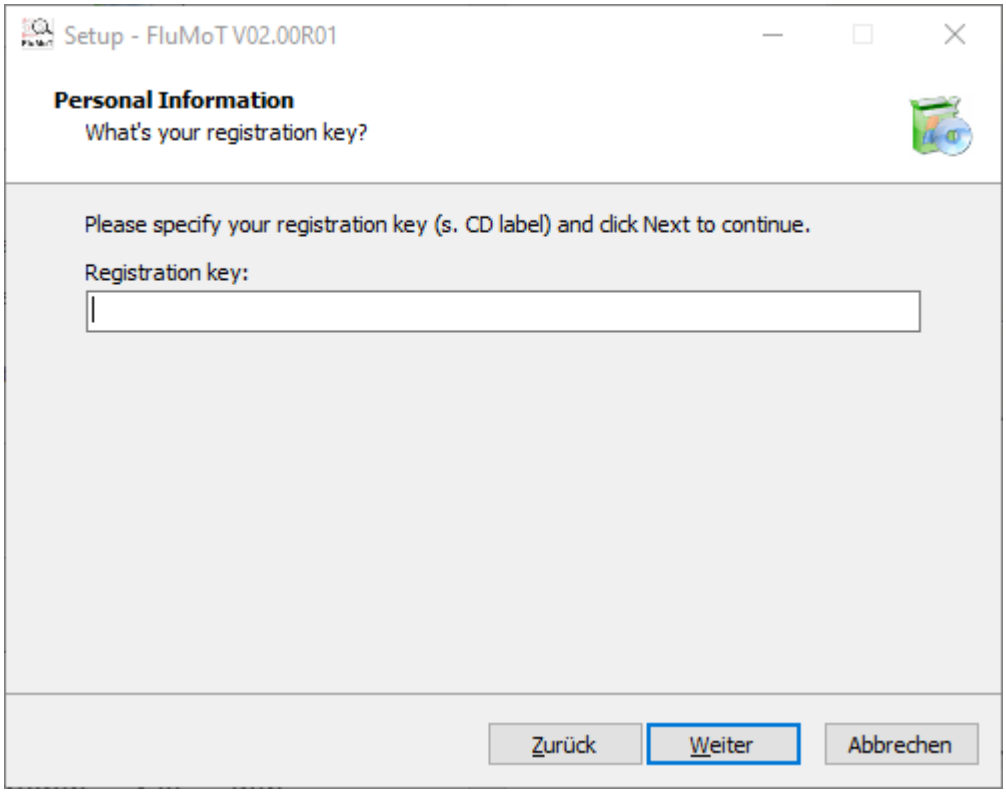


Zuerst wird die Sprache für die Installation ausgewählt, danach führt Sie der Setup-Assistent durch die gesamte Installation. Zum Fortfahren drücken Sie „OK“.

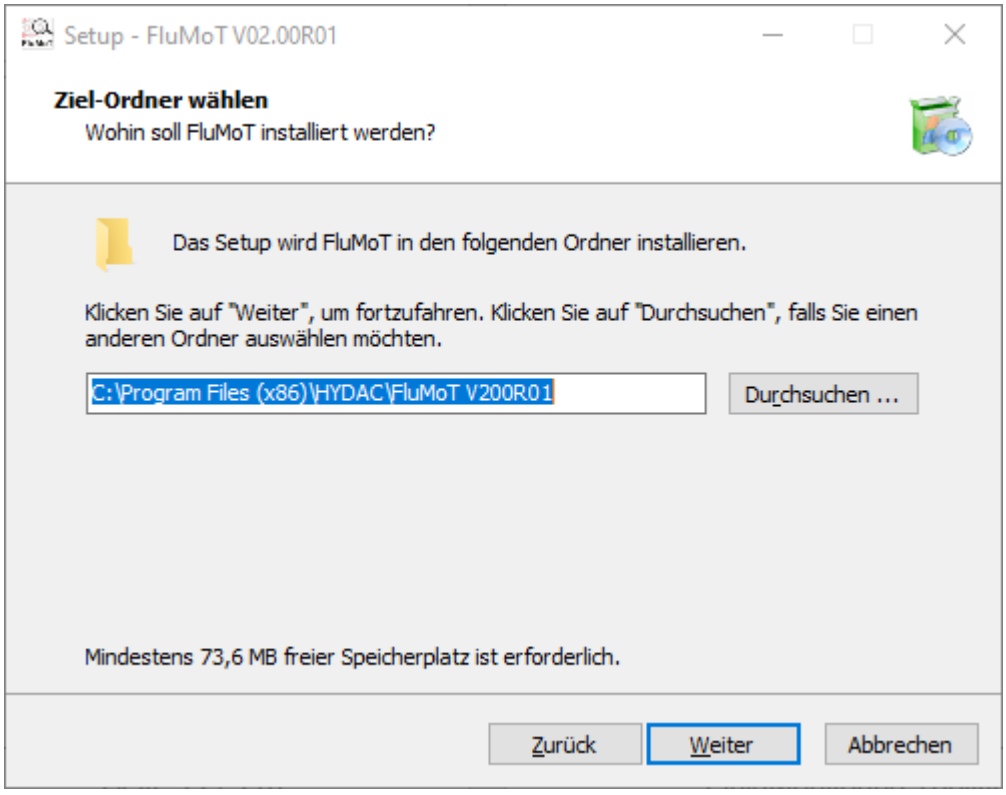
Um die Installation fortzusetzen müssen Sie die Lizenzvereinbarung in dem nachfolgenden Fenster sorgfältig durchlesen und anschließend akzeptieren.



Um Ihre FluMoT Software zu aktivieren, tragen Sie den Registrierungsschlüssel von der FluMoT-CD ein.

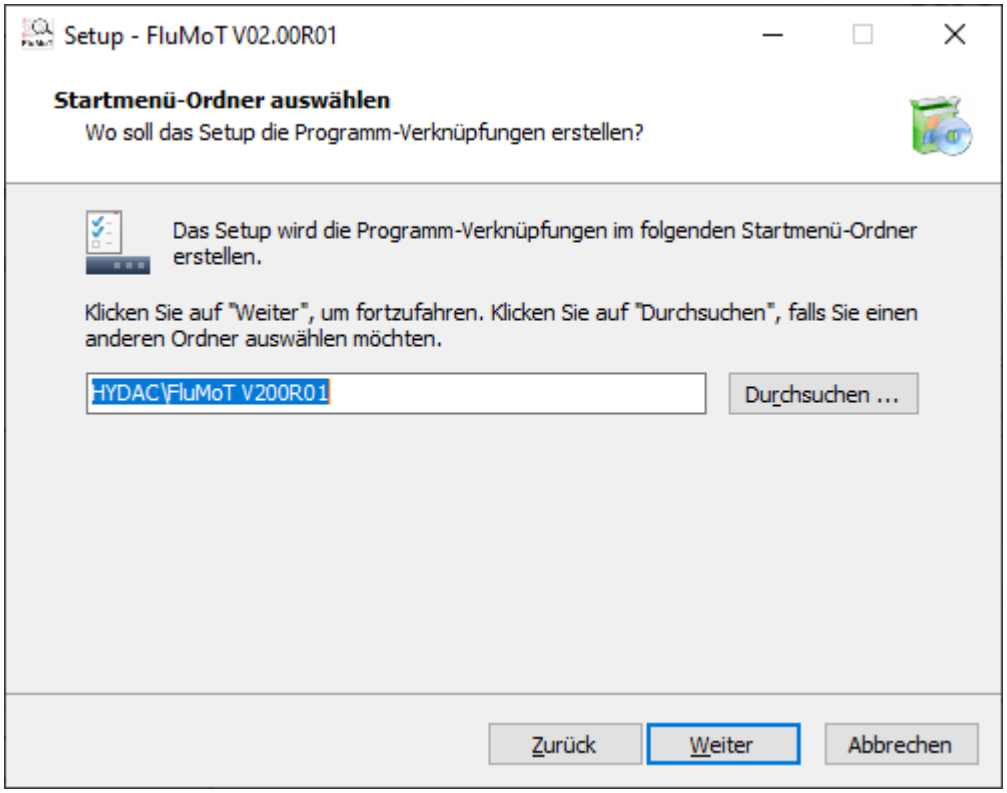


Bei der Installation werden Programmdateien in das Installationsverzeichnis kopiert. Anschließend legen Sie das das Installationsverzeichnis fest.

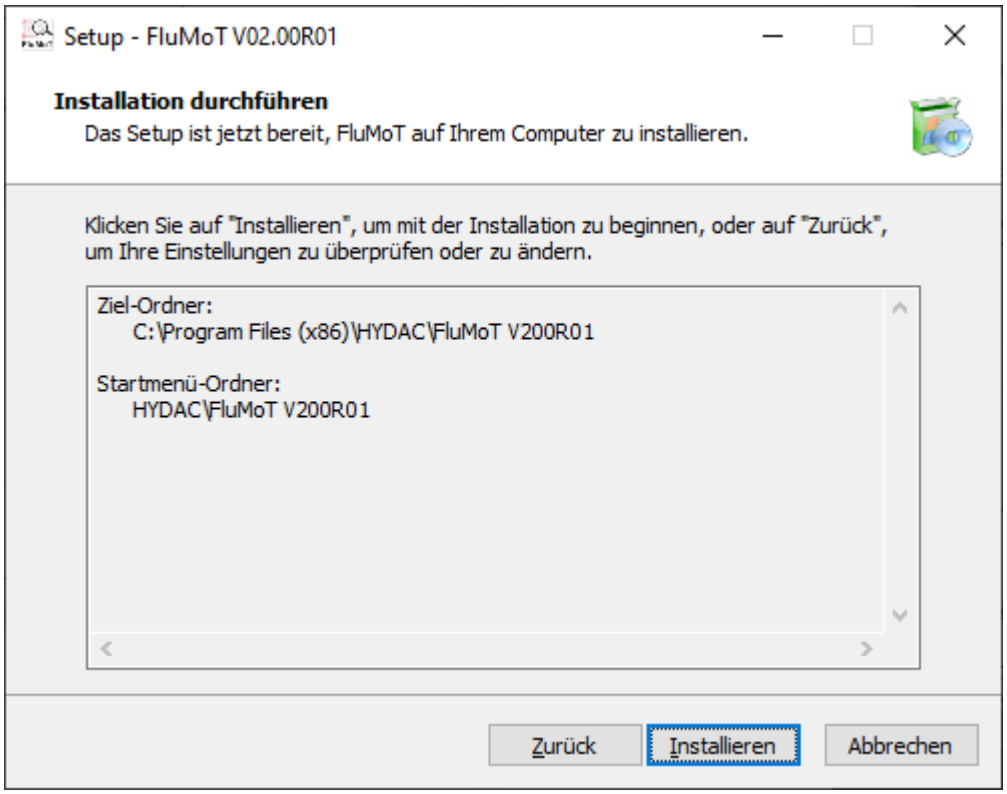


Existiert das Installationsverzeichnis bereits, müssen Sie Entscheiden, ob diese überschrieben werden soll.

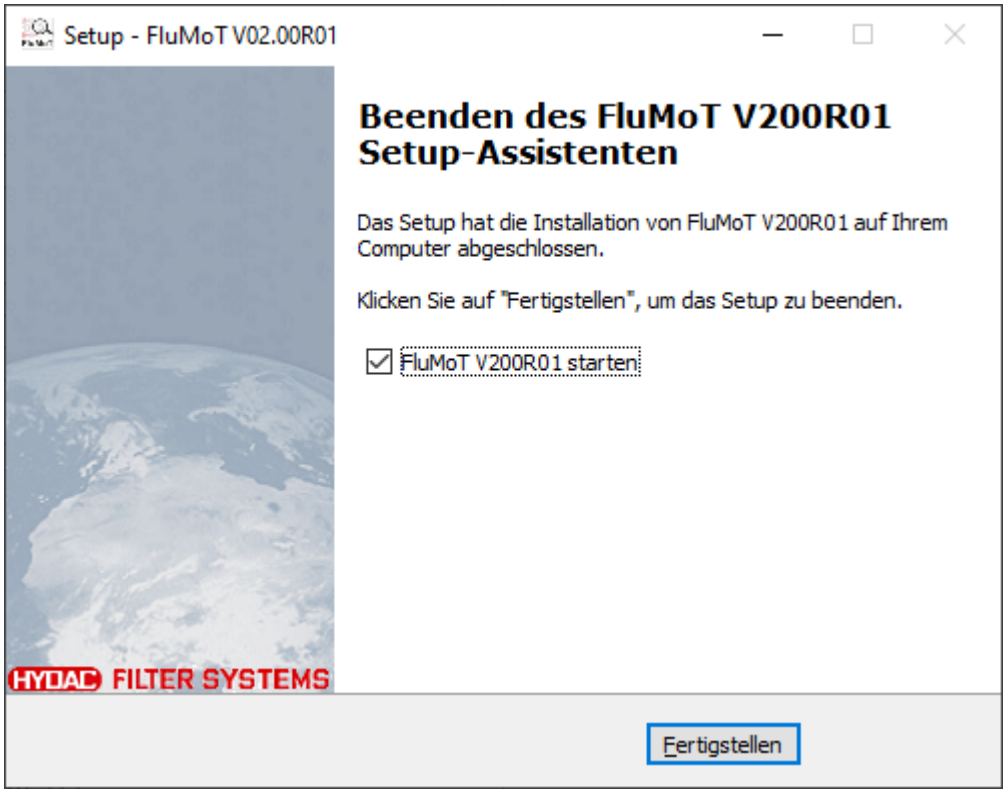
Anschließend wird der Startmenü-Ordner erstellt.



Nach der Bestätigung durch klicken auf Weiter > wird der Installationsprozess gestartet.

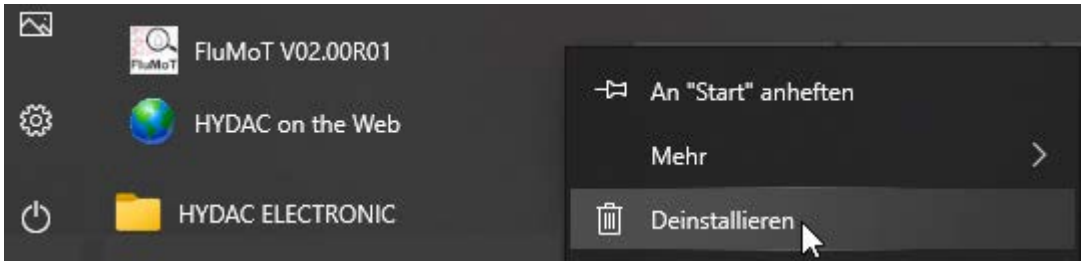


Der Setup – Assistent wird mit dem „Fertigstellen“ Button geschlossen.



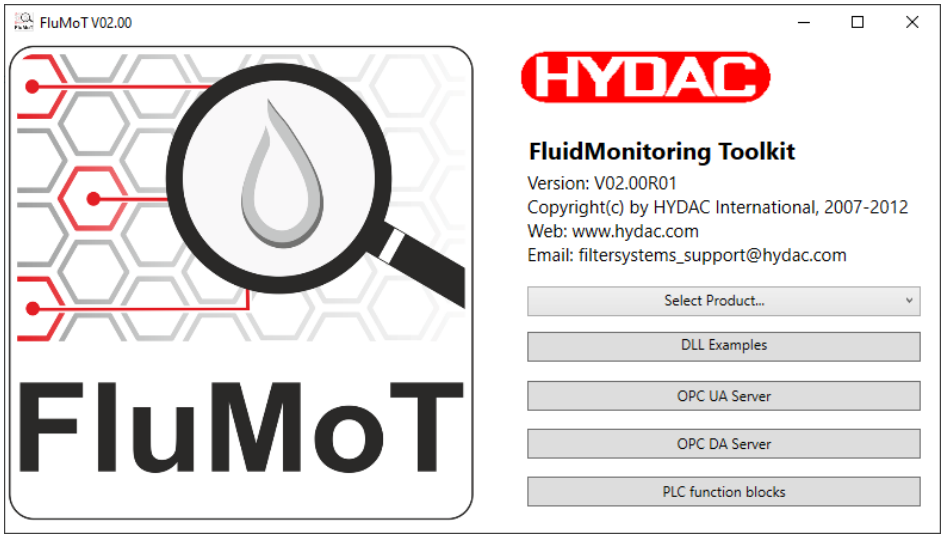
FluMoT deinstallieren

Zur Deinstallation von **FluMoT** führen Sie die im Installationsverzeichnis abgelegte Datei UNINS000.EXE aus oder starten Sie Deinstallation aus dem Startmenü:



Arbeiten mit FluMoT

Nach der Installation von **FluMoT** erscheint das Startfenster vom Toolkit. Hier befindet sich die Auswahl von den Werkzeugen, die im **FluMoT** vorhanden sind. Sie ist hach HYDAC - Gerätefamilien eingeordnet. Dementsprechend werden alle Verknüpfungen im Programm automatisch angepasst. Im Startmenü von Windows erscheint einen Eintrag zum Startfenster.



Das Startfenster beinhaltet die Verknüpfungen zum **FluMoT** OPC DA/UA Server, DLLs, SPS Komponenten und den entsprechenden Verzeichnissen mit Beispielen auf der Festplatte.

Ein Bestandteil von **FluMoT** sind die Programmbibliotheken (DLL).

Diese DLLs stellen die Schnittstellen zur Kommunikation mit unterschiedlichen HYDAC Sensoren bereit. Diese Sensoren sind mit verschiedenen Protokollen ansprechbar.

FluMoT beinhaltet folgende DLLs:

DLL	Beschreibung in Kapitel	Seite
dinmessbus32.dll dinmessbus64.dll	DIN Messbus - DLL	16
hecom32.dll hecom64.dll	HSI– DLL	30
hsitp32.dll hsitp64.dll	HSITP - DLL	45

Um die Hochsprachenprogrammierung zu erleichtern, werden einfache Beispiele als kleine Projekte in Delphi7, Delphi10.4, LabView 2012 und Excel -Macros (VBA 6) im Quellcode mitgeliefert. Diese befinden sich im Installationsverzeichnis von **FluMoT**.

In allen Programmbibliotheken von **FluMoT** werden immer die gleichen Datentypen verwendet. Sie werden in nachfolgender Tabelle aufgelistet.

Typ	Beschreibung	Delphi	C/C++	VB/VBA	Labview
Integer	Bereich: -2147483648 … 2147483647	Integer	Integer	Long	Long

	Format: 32 Bit, mit Vorzeichen				
Double	Bereich: 5.0 x 10^-324 … 1.7 x 10^308 Format: 8 Byte	Double	Float	Double	Double
String	Repräsentiert einen Zeiger auf einen Char - Wert. Das Ende der Zeichenkette wird durch ein Null – Zeichen festgelegt. Format: 1 Byte pro Zeichen.	PChar	Char*	String	String (C String Pointer)

INFO

Mit einer String werden manchmal mehrere Informationen übertragen. In diesem Fall wird ein Steuerzeichen verwendet. Das ist ein Wagenrücklauf (Carriage Return, ASCII Code 13).
Dieses Zeichen wird als <CR> in der Beschreibung gekennzeichnet.

Alle DLL – Funktionen werden in dieser Anleitung mit Hilfe von Pascal – Syntax beschrieben. Die Funktiosaufrufe bei 32- und 64bit-DLLs sind identisch.

Beschreibung DIN Messbus - DLL

Die Dateien Dinmessbus32.dll und Dinmessbus64.dll beinhaltet Schnittstellen, welcher zur Erleichterung der Kommunikation zwischen einem PC und folgenden HYDAC Sensoren dienen:

- FCU2000 Serie ab Index G
- FCU8000 Serie ab Index G
- CS 2000 Serie

Diese Programmbibliothek kapselt in sich die gesamte Protokollstruktur entsprechend den in DIN 66348 Teil 2 festgelegten Kriterien. Sie wandelt die DIN Messbus – Kommunikationsmechanismen in einige vereinfachte Funktionsschnittstellen.

An einem BUS können mehrere FCU / CS angeschlossen werden, wiesen Sie deshalb jedem Sensor eine eigene Adresse zu, unter welcher Sie diesen

ansprechen. Der Adressbereich nach DIN 66348 erstreckt sich von 1 bis 31. Achten Sie darauf, dass jede Adresse nur einmal im Bus vorkommt.

Daraus ergibt sich eine maximale Sensoranzahl je COM Schnittstelle von 31.

API – Funktionen

In folgender Tabelle wird die Übersicht der Funktionen in der *dinmessbus32.dll* dargestellt.

Funktion	Kurzbeschreibung
GetDLLVersion_DMB	DLL – Version als Zahl
GetDLLVersionText_DMB	DLL – Version als Text
SearchBusDevice_DMB	SensorID (auch in einem Bussystem) lesen
GetDeviceSerialNumber_DMB	Seriennummer des Gerätes lesen
GetDeviceSensorNumber_DMB	Sensornummer des Gerätes lesen
GetDeviceCalibrationDate_DMB	Datum der letzten Kalibrierung lesen
GetDeviceChannelCount_DMB	Anzahl der Messkanäle lesen
GetDeviceChannelInfo_DMB	Messkanal – Eigenschaften lesen
SetBusAddress_DMB	Busadresse setzen
SetMeasuringState_DMB	Messung starten/stoppen
GetDeviceState_DMB	Gerätestatus ermitteln
GetDeviceMeasuringValues_DMB	Messwerte lesen
GetDeviceLogListDataBlock_DMB	Dateiverzeichnis vom Gerät lesen
GetDeviceLogDataBlock_DMB	Datei lesen
EraseDeviceLog_DMB	Datei löschen

Fehlerbehandlung / Fehlercode

Viele DLL-Funktionen liefern im Fehlerfall einen Fehlercode. Dieser Fehlercode kann nachfolgende Werte beinhalten.

Allgemein

Statuscode	Statustext	Beschreibung
0	no error	Kein Fehler. Gerät ist betriebsbereit
1	new measuring is done (no error!)	Neue Messwerte sind vorhanden
2	filter contaminated	Filter verschmutzt
3	battery voltage too low	Batteriespannung zu gering
4	EXIN	Fehler am Analogeingang

Statuscode	Statustext	Beschreibung
5	Water warning	LED Strom an Obergrenze, Trübung durch Wasser, Luft, usw. Eventuell ist die LED defekt
6	Memory is full	Speicher voll
7	BSU error	Signal vom Volumenstromsensor liegt vor

Kommunikation mit dem Gerät

Statuscode	Statustext	Beschreibung
10	transmit error	Fehler bei der Übertragung der Daten zum Gerät.
11	receive error	Fehler bei der Datenübertragung vom Gerät.
12	invalid mode	Falsche Moduseinstellung
13	invalid bus address	Falsche Busadresse
14	invalid device model	Unbekannte Geräteserie
15	invalid channel index	Kanalnummer falsch
16	no device found	Kein Gerät gefunden
17	protocol error	DIN Messbus Protokollfehler
18	com port error	COM Port ist gesperrt
19	tx completed (no error!)	Übertragung erfolgreich abgeschlossen
20	invalid fileID	FileID falsch
21	invalid file part	FilePart falsch
22	no channel active	Kein Messkanal aktiv

Fehlermeldungen von FCU / CS

Statuscode	Statustext	Beschreibung
30	calibration values incorrect, fatal	Kalibrierfaktoren falsch

Statuscode	Statustext	Beschreibung
31	constant parameter incorrect: serial no., sensor no., …	Konstante Parameter falsch (z.B. Seriennummer)
32	normal parameter incorrect	Variable Parameter falsch
33	error I²C - bus handling	I²C - Busfehler
34	checksum in EEPROM incorrect	Checksumme in EEPROM falsch
35	error in bus command: syntax	Syntaktischer Fehler im Befehl
36	error in bus command: semantic	Semantischer Fehler im Befehl
37	log memory incorrect	Log beschädigt
38	error in transmission log	Fehler im Übertragungsprotokoll
39	error flow rate	Durchflussfehler
40	error ±VDD	Fehler ±VDD
41	error supply current particle sensor	Fehler LED-Strom Partikelsensor
42	error power supply voltage	Batteriespannung zu gering

Statuswert in der Protokolldatei

Statuscode	Statustext	Beschreibung
50	error flow rate	Durchflussfehler
51	no flow	Kein Durchfluss
52	M3: limit reached	M3: Grenze erreicht
52	M4: limit reached	M4: Grenze erreicht
54	M4: measuring started	M4: Messung gestartet
55	M4: test cycle time started	M4: Messzyklus läuft

Statuskontrolle

GetErrorStateText_DMB()

Mit dieser Funktion kann anhand von Stauscode eine passende Statusmeldung (auf Englisch) ausgegeben werden.

Syntax:	**function** GetErrorStateText_DMB (State: **Integer**): **String**;
Parameter:	*State* – Kommunikationsstatus.
Rückgabewert:	Statusmeldung vom Sensor (Englisch).
Antwort:	16: Gerät nicht gefunden

Versionskontrolle

GetDLLVersion_DMB()

Mit dieser Funktion kann die Bibliothekversion (Double – Wert) ermittelt werden.

Syntax:	**function** GetDLLVersion_DMB(): **double**;
Rückgabewert:	Die Versionsnummer wird als Double – Zahl zurückgeliefert.
Antwort:	1,1: Version V01.10

GetDLLVersionText_DMB()

Mit dieser Funktion kann die Bibliothekversion (String – Wert) ermittelt werden.

Syntax:	function GetDLLVersionText_DMB(): String;
Rückgabewert:	Die Versionsnummer und Ausgabedatum werden als Text zurückgeliefert.
Antwort:	V1.01 09.11.2007

Serielle Schnittstelle

Das DIN-Messbussystem basiert auf der EIA RS 485-Schnittstelle. Bei der RS 485 handelt es sich um eine serielle Schnittstelle. Um eine Übertragung über diese Schnittstelle zu ermöglichen, muss ein sogenannter COM – Port geöffnet werden. Jede nachfolgende Funktion öffnet zuerst einen COM Port, führt seine Routine und schließt den COM - Port. Bei mehreren Anfragen eines Gerätes nimmt dieser Sachverhalt gewisse Zeit in Anspruch. Der Aufwand kann reduziert werden, indem man einen COM Port einmalig (z.B. beim Start des Programms) öffnet, führt alle Anfragen aus und schließt den COM – Port. (z.B. beim Schließen des Programms) Folgende 3 Funktionen dienen zum Arbeiten mit einem/mehreren COM - Porten:

Syntax:	**procedure** OpenPort_DMB(PortNumber: **Integer**; **var** State: **Integer**); **procedure** ClosePort_DMB(PortNumber: **Integer**; **var** State: **Integer**); **procedure** CloseAllPorts_DMB();
Parameter:	*PortNumber* – Die Nummer des COM Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Bemerkung:	Die Baudrate ist fest definiert und beträgt 9600 Baud.

Gerätesuche und Geräteinformation

SearchBusDevice_DMB()

Diese Funktion dient zur Suche eines Gerätes an einem bestimmten COM Port mit einer bestimmten Busadresse.

Syntax:	**function** SearchBusDevice_DMB (PortNumber: **Integer**; Address: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	SensorID im Erfolgsfall.
Antwort:	CS2200 V04.01
Bemerkung:	die Zahl 4.01 die Firmwareversion des Gerätes beschreibt

GetDeviceSerialNumber_DMB()

Diese Funktion liefert die Seriennummer eines Gerätes als String.

Syntax:	function GetDeviceSerialNumber_DMB (PortNumber, Address: Integer; var State: Integer): String;
Parameter:	PortNumber – Die Nummer des COM Portes. Address – Busadresse des Gerätes als Integerzahl von 1 bis 31. State – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Seriennummer des Gerätes als String – Variable.
Antwort:	406C120456

GetDeviceSensorNumber_DMB()

Diese Funktion liefert die Sensornummer eines Gerätes als String.

Syntax:	**function** GetDeviceSensorNumber_DMB(PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Sensornummer des Gerätes als String – Variable.
Antwort:	120456
Bemerkung:	Sensornummer ist auch in Seriennummer enthalten

GetDeviceCalibrationDate_DMB()

Diese Funktion liefert das letzte Kalibrierdatum eines Gerätes.

Syntax:	function GetDeviceCalibrationDate_DMB(PortNumber, Address: Integer; var State: Integer): String;
Parameter:	PortNumber – Die Nummer des COM Portes. Address – Busadresse des Gerätes als Integerzahl von 1 bis 31. State – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Datum der letzten Kalibrierung
Antwort:	03.02.2005

GetDeviceChannelCount_DMB()

Diese Funktion liefert die Anzahl der Messkanäle eines Gerätes.

Syntax:	**function** GetDeviceChannelCount_DMB(PortNumber, Address: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Anzahl der Messkanäle.
Antwort:	5
Bemerkung:	Vom CS 2000: 4 Kanäle mit Partikelzahlen bzw. Verschmutzungsklassen und Durchfluss

GetDeviceChannelInfo_DMB()

Diese Funktion dient zur Ermittlung der Messkanal – Eigenschaften eines Gerät. (z.B. Kanalname, Messeinheit, usw.)

Syntax:	**function** GetDeviceChannelInfo_DMB(PortNumber, Address, ChNumber, Mode: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *ChNumber* – Kanalnummer. (von 0 beginnend) *State* – Referenz auf Kommunikationsstatus - Variable. *Mode* – Messdateneinheiten (wird auch in Funktion „GetDeviceMeasuringValues_DMB“ verwendet) 0 – Partikelzahlen (differenziell) 1 – NAS/SAE - Klassen 2 – ISO – Code 3 – Partikelzahlen (kumulativ)
	HINWEIS

	Die FCU liefern nicht in allen Messkanälen den ISO-Code als Messwerte im Mode 2 (siehe Geräteübersicht in Kapitel *Übersicht Messkanäle*)		
Rückgabewert:	Die Antwort besteht aus 5 Subzeilen, die mit einem Trennzeichen voneinander getrennt sind. Die Struktur einer solchen Antwort wird in der folgenden Tabelle dargestellt:		
Zeilennummer		Parameter	Anmerkung
1		Name	Kanalname
2		Unit	Messbereich, Einheit
3		Decimals	Nachkommastellen Alle Zahlenangaben erfolgen Ganzzahlig. Der Parameter Decimals gibt an, wie viele Stellen der Zahl hinter dem Dezimalpunkt stehen. Z.B. bedeutet die Zahlenangabe: LowerRange = -250, UpperRange = 1000 und Decimals = 1 einen Messbereich von –25,0 bis 100,0.
4		LowerRange	Messbereich, untere Grenze
5		UpperRange	Messbereich, obere Grenze
Antwort:	Temp<CR> °C<CR> 2<CR> -2500<CR> 10000<CR>		

SetBusAddress_DMB()

Mit folgender Funktion wird eine neue Busadresse im Gerät gesetzt. Im Erfolgsfall gibt die Funktion neue Busadresse als ganze Zahl zurück, sonst – die alte Busadresse.

Syntax:	**function** SetBusAddress_DMB (PortNumber, Address, NewAddress: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *NewAddress* – gewünschte neue Busadresse des Gerätes als Integerzahl von 1 bis 31. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Neue Busadresse des Gerätes als Integer – Zahl im Intervall [1…31].
Beispiel:	30 (Neue Busadresse)
	HINWEIS Bei einigen Geräten ist ein Neustart oder Reset (Stromversorgung ein/aus) erforderlich.

Messwerte lesen

SetMeasuringState_DMB()

Dieser Befehl kann eine Messung starten oder stoppen. Zudem können Sie mit diesem Befehl den Fehlerstatus des Gerätes rückzusetzen, sofern kein fataler / schwerer Fehler vorliegt.

Syntax:	**function** SetMeasuringState_DMB(PortNumber, Address, Mode: **Integer**; **var** State**: Integer): Integer**;
Parameter:	PortNumber – Die Nummer des COM Portes. Address – Busadresse des Gerätes als Integerzahl von 1 bis 31. Mode – Modus, bezeichnet folgende Aktionen: 0 – Start Messung 1 – Stop Messung 2 – Reset Errorstatus State – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	*1.1.1.1.1.1.1.1 Betriebszustand* Die erste Stelle zeigt die Nummer des Messmodus: 1x --> M1, 2x --> M2, usw. Die zweite Stelle definiert den genauen Zustand: für M1 (Messen), M2 (Messen u. Schalten), M3 (Filtern bis) gilt: x0 Messung aus x1 Warten auf gültigen Durchfluss x2 Messung läuft für M4 (Filtern von bis) gilt: 40 Messung aus 41 Warten auf gültigen Durchfluss 42 Messung läuft, Test auf untere Grenze 43 Wartezeit läuft 44 Wartezeit abgelaufen, warten auf gültigen Durchfluss 45 Messung läuft, Test auf obere Grenze
Antwort:	20 (Messmodus M2, Messung aus)

GetDeviceState_DMB()

Mit diesem Befehl kann der aktuelle Status des Gerätes ermittelt werden.

Syntax:	**function** GetDeviceState_DMB(PortNumber, Address: **Integer**; **var** State**: Integer): Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Status des Gerätes. Mögliche Werte: 0 – kein Fehler

	1 – neue Messung liegt vor
	2 – Filter verschmutzt
	4 – Batteriespannung zu gering
Antwort:	1 (neue Messwerte sind vorhanden)

GetDeviceMeasuringValues_DMB()

Mit diesem Befehl werden die Messwerte angefordert und übertragen. Die Ansicht der Messwerte muss mit den Namen von Messkanälen übereinstimmen. (siehe Befehl „GetDeviceChannelInfo_DMB) Als Rückgabewert bekommt man eine String mit Messdaten, die anhand von Kanal – Eigenschaften zu interpretieren ist.

Syntax:	**function** GetDeviceMeasuringValues_DMB (PortNumber, Address, Mode: **Integer**; const DeviceID: **String**; **var** State**: Integer): String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als Integerzahl von 1 bis 31. *Mode* – Messdateneinheiten (wird auch in Funktion „GetDeviceChannelInfo_DMB“ verwendet) DeviceID – Ergebnis der Funktion „SerachBusDevice_DMB“ (zum Beispiel: „CS2200 V04.01“ 0 – Partikelzahlen (differenziell) 1 – NAS/SAE - Klassen 2 – ISO-Code 3 – Partikelzahlen (kumulativ) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Messwerte
Bemerkung:	Im Modus 1 bedeutet der Messwert ″-1″ die NAS-Klasse „00“. Die SAE-Klasse „00“ und „000“ ebenfalls mit Hilfe von negativen Zahlen gekennzeichnet.
Antwort:	7680 <CR> 1860 <CR> 5 <CR> 0 <CR> 122 <CR> 0 <CR> 0 <CR>

Dateien auslesen

Zum Auslesen großer Datenmengen aus den Sensorikmemoryspeicher verwenden wir die Block – Befehle. Dabei wird die gesamte zu übertragenen Information auf kleinere Einzelpakete verteilt. Solche Einzelpakete werden vom Sensor übertragen und in einem Puffer gespeichert. Die entsprechenden Funktionen liefern in der Regel nur ein Teil der gesamten Informationen und beinhalten das Wort „Block“ in deren Namen.

HINWEIS

In Beispielprogrammen wird als Puffer eine String – Variable verwendet. Es ist aber zu beachten, dass die zu übertragende Informationsmenge sehr groß sein kann. Deshalb muss das Programm die maximale String – Größe

der jeweiligen Programmiersprache beachten, um einen Überlauf zu verhindern.

Eine Sensordatei besteht immer aus 2 Teilen: Kopfdaten und Messdaten.

Die Kopfdaten beinhalten die Struktur und Größe von Messdaten. Die eigentlichen Messdaten sind in der Regel sehr groß und werden anhand von Kopfdaten ausgewertet. Sie werden immer explizit übertragen.

Die Operationen mit Dateien werden nicht von allen HYDAC Sensoren unterstützt. Nähere Informationen entnehmen Sie den jeweiligen Bedienungsanleitungen der Geräte / Sensoren.

GetDeviceLogDirectory_DMB()

Mit dieser Funktion wird Dateiverzeichnis des Sensors gelesen.

Syntax:	GetDeviceLogDirectory_DMB(PortNumber, Address: Integer; var State: Integer): String;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Zeiger auf eine Zeichenvariable, die Dateiverzeichnis des Sensors enthält.
Beispiel:	Antwort (falls nur ein Protokoll im Speicher vorhanden): „1<CR>HYDAC FCU 8110<CR>5<CR>14.12.2006 10:01<CR>“ Die Bedeutung der einzelnen Einträgen wird in der nachfolgenden Tabelle näher erklärt.

Zeilennummer	Parameter	Anmerkung
1	FileID	Identifiziert eindeutig eine Datei. Diese ID wird bei den späteren Dateioperationen benutzt
2	Measuring point	Messstelle
3	RecordCount	Anzahl der Datensätze in der Datei
4	FileDate	Erstelldatum der Datei

GetDeviceLogHeader_DMB()

Mit dieser Funktion werden Informationen zur Aufnahme (Dateikopf) gelesen. Dabei handelt es sich um die Struktur der Daten im Sensor. Diese Information gilt als Voraussetzung für die richtige Auswertung von Messdaten.

Syntax:	**GetDeviceLogHeader_DMB** (PortNumber, Address, FileID, Mode: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes im ASCII Code. *FileID* – Datei – Identifikator (siehe GetDeviceLogDirectory_DMB) *Mode* – Messdateneinheiten (wird auch in Funktion „GetDeviceChannelInfo_DMB“ verwendet) 0 – Partikelzahlen (differenziell) 1 – NAS/SAE - Klassen 2 – ISO – Code 3 – Partikelzahlen (kumulativ) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Zeiger auf eine Zeichenvariable mit Dateikopf - Information.
Beispiel:	Rückgabewert: „7<CR><CR>1<CR>3<CR>14.12.2006 14:36:00<CR>0<CR> SAE A<CR><CR>2 <CR>-200<CR>1500<CR> SAE B<CR><CR>2 <CR>-200<CR>1500<CR> SAE C<CR><CR>2 <CR>-200<CR>1500<CR> SAE D<CR><CR>2 <CR>-200<CR>1500<CR> SAE E<CR><CR>2 <CR>-200<CR>1500<CR> SAE F<CR><CR>2 <CR>-200<CR>1500<CR> Flow<CR>ml/min<CR>1<CR>0<CR>8000<CR>“ Die Bedeutung der einzelnen Einträgen wird in der nachfolgenden Tabelle näher erklärt.

Zeilennummer	Parameter	Anmerkung
1	ChannelCount	Anzahl der Messkanäle
2	HasTimeStamps	Zeitbezug der Messwerte (0/1, Minutenwechsel bei 1)
3	RecordCount	Anzahl der Datensätze
4	StartDate	Zeitstempel: Start Messung oder 0
5	StopDate	Zeitstempel: Stop Messung oder 0
(für Messkanal 1)		
6	Name	Kanalname
7	Unit	Messeinheit
8	Decimals	Anzahl der Nachkommastellen

9	LowerRange	untere Messgrenze
10	UpperRange	obere Messgrenze
(eventuell für Messkanal 2)		
17	Name	
18	Unit	
…	…	
…	…	
24	…	

GetDeviceLogDataBlock_DMB()

Mit dieser Funktion werden die Messdaten gelesen. Um die Struktur dieser Daten zu wissen muss immer zuerst der Dateikopf gelesen werden.

Syntax: **GetDeviceLogDataBlock_DMB** (PortNumber, Address, FileID, Mode: **Integer**; **var** Offset, State: **Integer**): **String**;

Parameter:
PortNumber – Die Nummer des COM Portes.
Address – Busadresse des Gerätes als ASCII - Code des Zeichens.
FileID – Datei – Identifikator (siehe GetDeviceLogDirectory_DMB)
Mode – Messdateneinheiten (wird auch in Funktion „GetDeviceChannelInfo_DMB“ verwendet)
0 – Partikelzahlen (differenziell)
1 – NAS/SAE - Klassen
2 – ISO – Code
3 – Partikelzahlen (kumulativ)
Offset – Referenz auf eine Variable, die aktuelle Position im Sensor – Speicher enthält. Inhalt dieser Variable beim Start gleich 0 sein und wird intern im DLL verwendet.
State – Referenz auf Kommunikationsstatus - Variable.

Rückgabewert: Zeiger auf eine Zeichenvariable. Ein Teil der Messdaten aus einer Log-Datei. Innerhalb eines Datensatzes gilt immer folgende Anordnung:

		Kanal 1	Kanal 2	…
[Status]	[Zeitstempel]	Messwert 1	Messwert 2	…

Der Zeitstempel ist optional. Dieser Wert ist immer gleich 0 oder 1. Eins bedeutet in diesem Fall Minutenwechsel. Alle Statuscodes finden Sie auf Seite 17.

Beispiel: Antwort (evtl. nach mehreren Durchläufen):

„0 <CR>0 <CR>1623 <CR>1418 <CR>1033 <CR>848 <CR>572 <CR>388 <CR>730 <CR>“

Die Bedeutung der einzelnen Einträgen wird in der nachfolgenden Tabelle näher erklärt. Diese String wird anhand von Dateikopf – Informationen ausgewertet.

Beispiele für die Auswertung der Log – Daten:.

Log - Datensatz:

0<CR>0<CR>1623<CR>1418<CR>1033<CR>848<CR>572<CR>388<CR>730<CR>

Aus Dateikopf sind z.B. folgende Informationen zu entnehmen:

Parameter	Wert
ChannelCount	7
HasTimeStamps	1
RecordCount	1
Decimals Channel 1	2
Decimals Channel 2	2
Decimals Channel 3	2
Decimals Channel 4	2
Decimals Channel 5	2
Decimals Channel 6	2
Decimals Channel 7	1

Deshalb sind die Werte folgendermaßen zu interpretieren:

	Wert
Status	0
Zeitstempel	0
Channel 1	16.23
Channel 2	14.18
Channel 3	10.33
Channel 4	8.48
Channel 5	5.72
Channel 6	3.88
Channel 7	73.0

Dateien löschen

EraseDeviceLog_DMB ()

Diese Funktion entfernt eine Log – Datei aus dem Gerätspeicher.

Syntax:	**function** EraseDeviceLog_DMB (PortNumber, Address, FileID: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *FileID* – Datei – Identifikator (siehe GetDeviceLogDirectory_DMB)

	State – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	1 falls Datei erfolgreich gelöscht oder 0 im Fehlerfall
Antwort:	0 (Fehler: siehe Inhalt der Variable „State“)

Beschreibung HSI - DLL

Die Kommunikation zwischen digitalen Sensoren / Geräten und den Auswertegeräten von HYDAC wird via **H**ydac **S**ensor **I**nterface (HSI) realisiert.

Dabei handelt es sich um folgende Sensoren / Auswertegeräte:

Sensor		Auswertegerät
AquaSensor AS 1000 Serie		
HYDACLab HLB 1000 Serie		HMG 3000 Serie
ContaminationSensor CS 1000 Serie		CMU 1000 Serie
FluidControl Unit FCU 1000 Serie	HSI	CSI-C-11

Das HSI ist eine digitale 1-Draht Schnittstelle, welche es ermöglicht, Sensoren und PCs miteinander zu verbinden. Ein Sensor / Gerät sendet Messwerte über diese Schnittstelle an eine angeschlossene Auswerteeinheit (zum Beispiel: PC). Die Art und Weise, wie die Daten dabei verpackt werden, wird als HECOM – Protokoll bezeichnet.

Der Adressbereich gemäß HECOM erstreckt sich von 97 bis 122 (ASCII - Code der Zeichen: ′a′ … ′z′). So ergibt sich eine maximale Geräteanzahl von 26 je COM - Schnittstelle. Jeder Sensor muss eine eindeutige Adresse (′a′ … ′z′) zugewiesen werden. Dies gewährleistet, dass der Sensor im BUS angesprochen werden kann.

Sollte nur ein Sensor angeschlossen sein, kann auch das Zeichen ’+’ (ASCII Code 43) benutzt werden.

API - Funktionen

Folgende Funktionen stehen in den Dateien *hecom32.dll* und *hecom64.dll* zur Verfügung:

Funktion	**Kurzbeschreibung**
GetDLLVersion_HSI	DLL – Version als Zahl
GetDLLVersionText_HSI	DLL – Version als Text
SearchOneDevice_HSI	SensorID ermitteln, falls kein Bussystem vorhanden
SearchBusDevice_HSI	SensorID ermitteln in einem Bussystem
GetDeviceChannelCount_HSI	Anzahl der Messkanäle ermitteln
GetDeviceSerialNumber_HSI	Seriennummer ermitteln

GetDeviceChannelInfo_HSI	Messkanal – Eigenschaften ermitteln
GetBusAddress_HSI	Busadresse ermitteln
SetBusAddress_HSI	Busadresse setzen
GetDeviceChannelsMask_HSI	Struktur der Messwerte auslesen
GetDeviceMeasuringValues_HSI	Messwerte auslesen
GetDeviceState_HSI	Sensorstatus ermitteln
GetDeviceLogDirectoryBlock_HSI	Log - Verzeichnis lesen
GetDeviceLogHeaderBlock_HSI	Log – Kopfinformation lesen
GetDeviceLogDataBlock_HSI	Log – Inhalt lesen (Messwerte)
EraseDeviceLog_HSI	Log aus Sensorspeicher entfernen

Fehlerbehandlung

Fast alle Funktionen liefern im Fehlerfall einen Fehlercode. Dieser Fehlercode kann folgende Werte beinhalten:

Statuscode	Statustext	Beschreibung
0	no error	Kein Fehler. Gerät ist betriebsbereit
1	transmit error	Fehler bei der Übertragung von Daten zum Gerät
2	receive error	Fehler bei der Datenübertragung vom Gerät
3	too much devices	Zu viele Geräte gefunden
4	search error	Fehler in SensorID des Gerätes
5	no channels	Kein Messkanal aktiv
6	invalid channel index	Falsche Kanalnummer
7	invalid checksum	Checksumme falsch
8	com port blocked	COM Port ist gesperrt
9	invalid channels mask	„Gerätemaske“ falsch
10	no device found	Kein Gerät gefunden
11	protocol error	HSI – Protokollfehler
12	invalid device	Falsches gerät
13	multipacket tx not supported	Multipaket-Übertragung wird vom Gerät nicht unterstützt
14	no logs supported	Dateien werden nicht unterstützt
15	tx completed	Übertragung erfolgreich beendet
16	no logs found	Keine Datei gefunden

17	invalid FileID	FileID falsch
18	invalid FilePart (0 -> fileheader, 1 -> measurement data)	FilePart falsch
19	no smart sensor	Kein Smart - Sensor
20	invalid log mask	Dateimaske falsch

Statuskontrolle

GetErrorStateText_HSI()

Mit dieser Funktion kann anhand von Stauscode eine passende Statusmeldung (auf Englisch) ausgegeben werden.

Syntax:	**function** GetErrorStateText_HSI(State: **Integer**): **String**;
Parameter:	*State* – Kommunikationsstatus.
Rückgabewert:	Statusmeldung vom Sensor (Englisch).
Beispiel:	10: no device

Versionskontrolle - DLL

GetDLLVersion_HSI()

Mit dieser Funktion kann die Bibliothekversion ermittelt werden.

Syntax:	**function** GetDLLVersion_HSI(): **double**;
Rückgabewert:	Die Versionsnummer wird als Double – Zahl zurückgeliefert.
Beispiel:	1,03

GetDLLVersionText_HSI()

Mit dieser Funktion kann die Bibliothekversion ermittelt werden.

Syntax:	**function** GetDLLVersionText_HSI(): **String**;
Rückgabewert:	Die Versionsnummer und Ausgabedatum werden als Text zurückgeliefert.
Beispiel:	v.1,03 03.07.2007

Serielle Schnittstelle

Wenn HSI über RS 485-Schnittstelle gefahren wird, muss immer zuerst ein sogenannter COM – Port geöffnet werden. Jede nachfolgende Funktion öffnet zuerst einen COM Port, führt seine Routine aus und schließt den COM

- Port. Bei mehreren Anfragen eines Gerätes nimmt dieser Sachverhalt gewisse Zeit in Anspruch. Der Aufwand kann reduziert werden, indem man einen COM Port einmalig (z.B. beim Start des Programms) öffnet, führt alle seinen Anfragen und schließt den COM – Port z.B. beim Schließen des Programms. Folgende 4 Funktionen dienen zum Arbeiten mit einem/mehreren COM - Port:

Syntax:	**procedure** OpenPort_HSI(PortNumber: **Integer**; **var** State: **Integer**);
	procedure OpenPortExt_HSI(PortNumber, Baudrate: **Integer**; **var** State: **Integer**);
	procedure ClosePort_HSI(PortNumber: **Integer**; **var** State: **Integer**);
	procedure CloseAllPorts_HSI();
Parameter:	*PortNumber* – Die Nummer des COM Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Bemerkung:	Standardmäßig wird die Baudrate 9600 Baud verwendet.

Gerätesuche und Geräteinformation

SearchOneDevice_HSI()

Diese Funktion dient zur Suche eines Gerätes an einem bestimmten COM – Port ohne BUS. Es muss sichergestellt werden, dass an COM – Port **genau ein** Gerät angeschlossen ist.

Syntax:	**function** SearchOneDevice_HSI(PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	SensorID im Erfolgsfall.
Beispiel:	Eine Antwort kann z.B. folgendermaßen aussehen: „CS1320 V02.21“, wobei die Zahl 2.21 die Firmwareversion des Sensors beschreibt.

SearchBusDevice_HSI()

Diese Funktion dient zur Suche eines Gerätes an einem bestimmten COM Port mit einer bestimmten Busadresse.

Syntax:	**function** SearchBusDevice_HSI (PortNumber: **Integer**; Address: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	SensorID im Erfolgsfall.

Beispiel:	Eine Antwort kann z.B. folgendermaßen aussehen: „AS1000 V02.00“, wobei die Zahl 2.00 die Firmwareversion des Gerätes bezeichnet.

GetDeviceChannelCount_HSI()

Diese Funktion liefert die Anzahl der Kanäle in einem Gerät.

Syntax:	**function** GetDeviceChannelCount_HSI (PortNumber, Address: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Anzahl der Messkanäle.
Antwort:	10 (vom CS 1000)

GetDeviceSerialNumber_HSI()

Diese Funktion liefert die Seriennummer eines Gerätes.

Syntax:	**function** GetDeviceSerialNumber_HSI(PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Seriennummer des Gerätes als String – Variable.
Beispiel:	4711

GetDeviceChannelInfo_HSI()

Diese Funktion dient zur Ermittlung der Kanal – Eigenschaften in einem Gerät. (z.B. Kanalname, Messeinheit usw.)

Syntax:	**function** GetDeviceChannelInfo_HSI(PortNumber, Address, ChNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. ChNumber – Kanalnummer. (von 0 beginnend) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Die Antwort besteht aus 5 Subzeilen, die mit einem Trennzeichen voneinander getrennt sind. Die Struktur einer solchen Antwort wird in der folgenden Tabelle dargestellt:

Zeilennummer	Parameter	Anmerkung
1	Name	Kanalname
2	Unit	Messbereich, Einheit
3	Decimals	Nachkommastellen Alle Zahlenangaben erfolgen ganzzahlig. Der Parameter Decimals gibt an, wie viele Stellen der Zahl hinter dem Dezimalpunkt stehen. Z.B. bedeutet die Zahlenangabe: LowerRange = -250, UpperRange = 1000 und Decimals = 1 einen Messbereich von -25,0 bis 100,0
4	LowerRange	Messbereich, untere Grenze
5	UpperRange	Messbereich, obere Grenze

Antwort: Temp<CR>°C<CR>2<CR>-2500<CR>10000<CR>

Busadressen verwalten

HINWEIS

Nicht alle HSI – Sensoren unterstützen die nachfolgenden zwei Befehle.

GetBusAddress_HSI()

Diese Funktion gibt die Busadresse eines Gerätes zurück.

HINWEIS

Es darf nur ein einziges Gerät an dem COM Port angeschlossen sein.

Syntax:	**function** GetBusAddress_HSI(PortNumber: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Busadresse des Gerätes als ASCII - Code des Zeichens.
Beispiel:	97 (Zeichen ‘a’)

SetBusAddress_HSI()

Setzt eine neue Busadresse im Sensor.

Syntax:	**function** SetBusAddress_HSI (PortNumber, Address, NewAddress: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *NewAddress* – gewünschte neue Busadresse des Gerätes als ASCII - Code des Zeichens. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Neue Busadresse des Gerätes als ASCII - Code des Zeichens. Im Erfolgsfall wird neue Adresse zurückgegeben, sonst alte.
Beispiel:	98 (Neue Busdresse ist ‘b’)

Messwerte auslesen

GetDeviceChannelsMask_HSI()

Mit diesem Befehl ermittelt Sie, wie sich die Messwerte zusammensetzen, z.B. ob es sich um 8-bit oder 16-bit Zahlen handelt. Führen Sie diesen Befehl nur einmal aus, damit die „Gerätemaske“ weiter verwenden können.

Syntax:	**function** GetDeviceChannelsMask_HSI(PortNumber, Address: **Integer**; **var** State**: Integer): String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Zeiger auf eine Zeichenvariable, die sogenannte „Gerätemaske“.
Antwort:	3<CR>7<CR>0<CR>0<CR>2<CR>4<CR>2<CR>

Die Struktur einer solchen „Gerätemaske“ wird in der folgenden Tabelle dargestellt:

Zeilennummer	Parameter	Anmerkung
1	ChannelCount	Anzahl der Messkanäle
2	ActivityMask	Jedem Kanal ist ein Bit zugeordnet, das anzeigt ob der Kanal aktiv oder nicht aktiv ist
3	MinMask	Jedem Kanal ist ein Bit zugeordnet, das anzeigt ob der Kanal Min – Werte besitzt oder nicht
4	MaxMask	Jedem Kanal ist ein Bit zugeordnet, das anzeigt ob der Kanal Max – Werte besitzt oder nicht

5	DataSize, Channel 1	Datengröße im 1. Messbereich
6	DataSize, Channel 2	Datengröße im 2. Messbereich
7	…	… (für jeden Kanal)

Der Parameter „DataSize“ kann nur die Werte 1, 2 oder 4 besitzen. Diese Werte entsprechen 8-, 16- oder 32-bit Werten.

Die „ActivityMask“ gibt an, welche Kanäle tatsächlich aktiv sind. Bei der Messwertübertragung werden inaktive Kanäle nicht übertragen. Bit 0 der Maske zeigt an ob Kanal 0 aktiv ist, Bit 1 Kanal 1 und so weiter.

Mit der Min- und Maxmask wird festgelegt, ob es zu dem jeweiligen Messwert auch noch einen minimal Wert und/oder einen maximal Wert gibt. Bit 0 gehört auch hier zu Kanal 0.
Ist ein Kanal nicht aktiv, so sind auch die minimal oder maximal Werte grundsätzlich nicht aktiv. Das bedeutet, ein minimal oder maximal Wert darf ohne Messwert nicht vorkommen.

GetDeviceMeasuringValues_HSI()

Mit diesem Befehl werden die Messwerte angefordert und übertragen. Stellen Sie die zusammensetzung der Messwerte vorher mit dem Befehl „GetDeviceChannelsMask_HSI“ fest. Dabei bekommen Sie die Gerätemaske als Antwort. Diese müssen Sie jedes Mal mit dem Befehl „GetDeviceMeasuringValues_HSI“ bestätigen. Der Grund dafür ist eine Möglichkeit, die Struktur der Messwerte im Betrieb dynamisch anzupassen.

Die Struktur der „Gerätemaske“ wurde bereits für die Funktion „GetDeviceChannelsMask_HSI“ beschrieben.

Syntax:	**function** GetDeviceMeasuringValues_HSI(PortNumber, Address: **Integer**; **const** DeviceChannelsMask: **String**; **var** State**: Integer): String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *DeviceChannelsMask* – „Gerätemaske“. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Messwerte
Antwort:	127<CR>104<CR>80<CR>20<CR>21<CR>22<CR>19<CR>246<CR>100<CR>0<CR>

Messwerte interpretieren (Beispiele)

1) Messwerte vom Gerät: 135<CR>47<CR>7<CR>

Gerätemaske: 3<CR>7<CR>0<CR>0<CR>2<CR>4<CR>2<CR>, also:

Parameter	Wert

ChannelCount	3		
	Bit0 Kanal0	Bit1 Kanal1	Bit2 Kanal2
ActivityMask	1	1	1
MinMask	0	0	0
MaxMask	0	0	0
DataSize, Channel 1	2		
DataSize, Channel 2		4	
DataSize, Channel 3			2

Die Werte sind wie folgt auszuwerten:

	Wert
Channel 1	135
Channel 2	47
Channel 3	7

2) Messwerte vom Gerät: 135<CR>47<CR>7<CR>

Gerätemaske: 3<CR>6<CR>2<CR>0<CR>2<CR>4<CR>2<CR>, also:

Parameter	Wert		
ChannelCount	3		
	Bit0 Kanal0	Bit1 Kanal1	Bit2 Kanal2
ActivityMask	1	1	0
MinMask	0	1	0
MaxMask	0	0	0
DataSize, Channel 1	2		
DataSize, Channel 2		4	
DataSize, Channel 3			2

Die Werte sind wie folgt auszuwerten:

	Wert
Channel 1	135
Channel 2	47
Channel 2, Minimum	7

GetDeviceState_HSI()

Der Sensorstatus dient dazu festzustellen, ob das angeschlossene Gerät betriebsbereit ist, oder in einen Fehlerzustand eingetreten ist. Der Sensorstatus hat folgenden Aufbau:

- 8-bit Statusbyte,
- 16-bit Statuscode bzw. Fehlercode (mit Vorzeichen)
- Optionaler Statustext

Das *Statusbyte* gibt den aktuellen Zustand des Gerätes an. Die einzelnen Zustände können über den folgenden Statuscode näher spezifiziert werden.

Folgende Werte für das Statusbyte sind definiert:

0: Betriebsbereit	Kein aktiver Fehler vorhanden, Gerät ist betriebsbereit.
1: Stand-by	Kein aktiver Fehler vorhanden, Gerät ist aber zur Zeit nicht betriebsbereit, eventuell sind einzelne Gerätefunktionen abgeschaltet, oder Gerät ist in einer Anlaufphase, etc.
2: Leichter Fehler	Es ist ein leichter Fehler vorhanden, der quittiert werden kann.
3: Mittlerer Fehler	Es ist ein mittelschwerer Fehler vorhanden, der durch Ein/Ausschalten eventuell behebbar ist.
4: Schwerer Fehler	Es ist ein schwerer Fehler vorhanden, das Gerät muss zum Hersteller zurück.

Der *Statuscode* spezifiziert den aktuellen Zustand näher. Es ist ein 16-bit Wert. Die genaue Bedeutung ist von Gerät zu Gerät unterschiedlich. Der Anwender kann dann dem Handbuch nähere Infos zu dem Statuscode entnehmen.

Der *Statustext* ist optional und maximal 32 Zeichen lang. Er dient dazu, dass ein Bediengerät den Status eines Sensors im Klartext anzeigen kann.

Syntax:	**function** GetDeviceState_HSI(PortNumber, Address: **Integer**; **var** StateByte, StateCode, State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *StateByte* – StatusByte. *StateCode* – Statuscode. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Statustext.
Antwort:	ASIC-CRC-Error, *StateByte* = 3, *StateCode* =17

Dateien exportieren

Aufgrund großen Datenmengen werden zum Auslesen des Sensorikmemoryspeichers die sogenannten Block – Befehle verwendet. Dabei wird die gesamte zu übertragenen Information auf kleinere Einzelpakete verteilt. Solche Einzelpakete werden vom Sensor übertragen und in einem Puffer gespeichert. Die entsprechenden Funktionen liefern in der Regel nur ein Teil der gesamten Informationen und beinhalten das Wort „Block“ in deren Namen.

HINWEIS

Die Operationen mit Dateien werden nicht von allen HYDAC Sensoren unterstützt. Nähere Informationen entnehmen Sie den jeweiligen Bedienungsanleitungen.

HINWEIS

In Beispielprogrammen wird als Puffer eine String – Variable verwendet. Es ist aber zu beachten, dass die zu übertragende Informationsmenge sehr groß sein kann. Deshalb muss das Programm die maximale String – Größe der jeweiligen Programmiersprache beachten, um einen Überlauf verhindern.

Eine Sensordatei besteht immer aus 2 Teilen: Kopfdaten und Messdaten. Die Kopfdaten beinhalten die Struktur und Größe von Messdaten. Die eigentlichen Messdaten sind in der Regel sehr groß und werden anhand von Kopfdaten ausgewertet. Sie werden immer explizit übertragen.

GetDeviceLogDirectoryBlock_HSI()

Mit dieser Funktion wird Dateiverzeichnis des Sensors gelesen.

Syntax:	**GetDeviceLogDirectoryBlock_HSI**(PortNumber, Address: Integer; var Offset, State: Integer): String;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *Offset* – Referenz auf eine Variable, die aktuelle Position im Sensor – Speicher enthält. Inhalt dieser Variable beim Start gleich 0 sein und wird intern im DLL verwendet. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Zeiger auf eine Zeichenvariable. Ein Teil der Verzeichnisliste im Sensor.
Beispiel:	Antwort (evtl. nach mehreren Durchläufen): 1<CR>0<CR>HLB1000 - LOGDAT<CR>1<CR>0<CR> Die Bedeutung der einzelnen Einträgen wird in der nachfolgenden Tabelle näher erklärt.

Zeilennummer	Parameter	Anmerkung
1	FileID	Identifiziert eindeutig eine Datei. Diese ID wird bei den späteren Dateioperationen benutzt
2	FileType	Dateityp: 0 = Log – Datei 1 = Konfigurationsdatei
3	FileName	Dateiname (max. 32 Zeichen)
4	FileNumber	Nummer der Messung im Sensor
5	FileDate	Erstelldatum der Datei

GetDeviceLogHeaderBlock_HSI()

Mit dieser Funktion werden Informationen zur Aufnahme (Dateikopf) gelesen. Dabei handelt es sich um die Struktur der Daten im Sensor. Diese Information gilt als Voraussetzung für die richtige Auswertung von Messdaten.

Syntax:	**GetDeviceLogHeaderBlock_HSI** (PortNumber, Address, FileID: **Integer**; **var** Offset, State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *FileID* – Datei – Identifikator (s. GetDeviceLogDirectoryBlock_HSI) *Offset* – Referenz auf eine Variable, die aktuelle Position im Sensor – Speicher enthält. Inhalt dieser Variable beim Start gleich 0 sein und wird intern im DLL verwendet. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Zeiger auf eine Zeichenvariable. Ein Teil des Dateikopfes im Sensor.
Beispiel:	Antwort (evtl. nach mehreren Durchläufen): 4<CR>0<CR>0<CR>0<CR>5075<CR>0<CR>0<CR>0<CR>-2500<CR>10000<CR>0<CR>0<CR>Temp<CR>°C<CR>2<CR>2<CR>12400<CR>18250<CR>0<CR>0<CR>FreqVal<CR><CR>0<CR>2<CR>0<CR>1000<CR>0<CR>0<CR>DkVal<CR><CR><CR>2<CR>2<CR>0<CR>10000<CR>0<CR>0<CR>RelHum <CR>%<CR>2<CR>2<CR> Die Bedeutung der einzelnen Einträgen wird in der nachfolgenden Tabelle näher erklärt.

Zeilennummer	Parameter	Anmerkung
1	ChannelCount	Anzahl der Messkanäle
2	HasTimeStamps	Zeitbezug der Messwerte (0/1)
3	HasStates	Status zum Messwert (0/1)
4	HasMinMax	Min/Max – Werte (0/1)

5	RecordCount	Anzahl der Datensätze
6	Samplerate	Abtastrate (als Vielfaches von 100µs) oder 0
7	StatusCodeId	Kodierung der Status – Angabe
8	PreTriggerCount	Wie viele Datensätze liegen vor einem Triggerereignis. Das Triggerereignis wird auf der Zeitachse immer mit 0 dargestellt.
(für Messkanal 1)		
9	LowerRange	untere Messgrenze
10	UpperRange	obere Messgrenze
11	LowerRawRange	Evtl. Umrechnung
12	UpperRawRange	evtl. Umrechnung
13	Name	Kanalname
14	Unit	Messeinheit
15	Decimals	Anzahl der Nachkommastellen
16	DataSize	Datengröße
(eventuell für Messkanal 2)		
17	LowerRange	untere Messgrenze
18	UpperRange	obere Messgrenze
19-24	…	
(usw für alle Messkanäle)		
8 + 8 * ChannelCount + 1	InfoText	Kommentar

GetDeviceLogDataBlock_HSI()

Mit dieser Funktion werden die Messdaten gelesen. Um die Struktur dieser Daten zu wissen muss immer zuerst der Dateikopf gelesen werden.

Syntax:	**GetDeviceLogDataBlock_HSI** ()PortNumber, Address, FileID: **Integer**; **const** LogChannelsMask: **String**;**var** Offset, State: **Integer**): **String**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *FileID* – Datei – Identifikator (siehe GetDeviceLogDirectoryBlock_HSI) *LogChannelsMask – eine Zeichenkette, die interne Datenstruktur beschreibt. Diese Maske kann entweder aus Dateikopf – Informationen zusammengestellt werden, oder auch leer sein. (Dann wird die Ausführung des Befehls jeweils länger dauern) Die*

Struktur einer solchen Maske ist der nachfolgenden Tabelle zu entnehmen.

Offset – Referenz auf eine Variable, die aktuelle Position im Sensor – Speicher enthält. Inhalt dieser Variable beim Start gleich 0 sein und wird intern im DLL verwendet.

State – Referenz auf Kommunikationsstatus - Variable.

Rückgabewert: Zeiger auf eine Zeichenvariable. Ein Teil der Messdaten aus einer Log-Datei. Innerhalb eines Datensatzes gilt immer folgende Anordnung:

		Kanal 1		Kanal 2		…	
[Zeitstempel]	[Status]	Messwert 1	[Minwert 1]	[Maxwert 1]	Messwert 2	[Minwert 2]	[Maxwert 2]

Die ausgeklammerten Einträge in dieser Tabelle sind optional (siehe LogChannelsMask).

Beispiel: Antwort (evtl. nach mehreren Durchläufen):

„0<CR>-31730<CR>250<CR>240<CR>230<CR>220<CR>29<CR>-1<CR>0<CR>10<CR>1038<CR>250<CR>240<CR>230<CR>220<CR>29<CR>-1<CR>0<CR>“

Die Bedeutung der einzelnen Einträgen wird in der nachfolgenden Tabelle näher erklärt. Diese String wird anhand von der Maske ausgewertet.

Zeilennummer	Parameter	Anmerkung
1	ChannelCount	Anzahl der Messkanäle
2	HasTimeStamps	Zeitbezug der Messwerte (0/1)
3	HasStates	Dieser Flag zeigt, ob die aktuelle Aufzeichnung zu jedem Datensatz noch ein Status – Zahl enthält (0/1)
4	HasMinMax	Dieser Flag zeigt, ob die Minimale UND Maximale (immer zusammen) Messwerte separat zu jedem Kanal mitgespeichert wurden (0/1)
5	DataSize1	Datengröße Messkanal 1
6	DataSize2	Datengröße Messkanal 2
7	…	(für jeden Messkanal)

Der Parameter „DataSize“ kann nur die Werte 1,2 oder 4 besitzen. Diese Werte entsprechen 8-, 16- oder 32-bit Werten. Diese Werte sind nur für den internen Gerbrach im DLL gedacht.

Beispiele für die Auswertung der Log – Daten:

1) Log - Datensatz: 0<CR>47<CR>7<CR>

Gerätemaske: 2<CR>0<CR>1<CR>0<CR>2<CR>2<CR> (deshalb ein Datensatz = 2 Zahlen), also

Parameter	Wert
ChannelCount	2
HasTimeStamps	0
HasStates	1
HasMinMax	0
DataSize, Channel 1	2
DataSize, Channel 2	2

Werten Sie die Antworten wie folgt aus:

	Wert
Status	0
Channel 1	47
Channel 2	7

2) Log - Datensatz: 13556<CR>47<CR>7<CR>56<CR>6<CR>1<CR>100<CR>

Gerätemaske: 2<CR>6<CR>2<CR>0<CR>2<CR>4<CR>2<CR> (deshalb ein Datensatz = 7 Zahlen), also

Parameter	Wert
ChannelCount	2
HasTimeStamps	1
HasStates	0
HasMinMax	1
DataSize, Channel 1	2
DataSize, Channel 2	2

Werten Sie die Antworten wie folgt aus:

	Wert
Zeitstempel	13556
Channel 1	47
Minimum Channel 1	7
Maximum Channel 1	56
Channel 2	6
Minimum Channel 2	1
Maximum Channel 2	100

Dateien löschen

EraseDeviceLog_HSI ()

Diese Funktion entfernt eine Log – Datei aus dem Gerätspeicher.

Syntax:	**EraseDeviceLog_HSI** (PortNumber, Address, FileID: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*PortNumber* – Die Nummer des COM Portes. *Address* – Busadresse des Gerätes als ASCII - Code des Zeichens. *FileID* – Datei – Identifikator (siehe GetDeviceLogDirectoryBlock_HSI) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	FileID der gelöschten Datei oder -1 im Fehlerfall
Beispiel:	Antwort: 10 (Datei 10 wurde erfolgreich gelöscht)

Beschreibung HSITP - DLL

Die Kommunikation zwischen digitalen Sensoren / Geräten von HYDAC und entsprechenden Auswertegeräten über Modem- und TCP-IP- Verbindung wird mit Hilfe von HSI Text Protokoll (HSI–TP) realisiert. Das Protokoll benutzt reines peer-to-peer Verfahren (es gibt keine Busadressen). Es ist ein Master – Slave Protokoll, das bedeutet der Master sendet ein Paket und der Slave antwortet mit einem Paket.

API - Funktionen

Alle DLL – Funktionen werden in dieser Anleitung mit Hilfe von Pascal – Syntax beschrieben. Folgende Funktionen stehen in der Datei *hsitp32.dll* zur Verfügung:

Funktion	Kurzbeschreibung
GetDLLVersion_HTP	DLL – Version als Zahl
GetDLLVersionText_HTP	DLL – Version als Text
OpenConnection_HTP	Verbindung mit Gerät aufbauen
CloseConnection_HTP	Verbindung schließen
CloseAllConnections_HTP	Alle Verbindungen schließen
SearchOneDevice_HTP	Gerätebezeichnung ermitteln
GetDeviceChannelCount_HTP	Anzahl der Messkanäle ermitteln
GetDeviceSerialNumber_HTP	Seriennummer ermitteln
GetDeviceChannelInfo_HTP	Messkanal – Eigenschaften ermitteln
GetDeviceChannelsMask_HTP	Struktur der Messwerte lesen
GetDeviceMeasuringValues_HTP	Messwerte lesen
GetDeviceState_HTP	Sensorstatus ermitteln

Fehlerbehandlung

Fast alle Funktionen liefern im Fehlerfall einen Fehlercode. Dieser Fehlercode kann folgende Werte beinhalten:

Statuscode	Statustext	Beschreibung
0	no error	Gerät ist betriebsbereit
1	invalid IP address	IP – Adresse falsch
2	invalid port number	Portnummer falsch
3	no connection	Es besteht keine Verbindung
4	invalid checksum	Checksumme falsch
5	no device found	Kein Gerät gefunden
6	protocol error	HSI – TP Protokollfehler
7	invalid channel mask	“Gerätemaske” falsch
8	invalid sensor info	Sensorinformation inkonsistent

Statuskontrolle

GetErrorStateText_HTP()

Mit dieser Funktion kann anhand von Stauscode eine passende Statusmeldung (auf Englisch) ausgegeben werden.

Syntax:	**function** GetErrorStateText_HTP(State: Integer): PChar;
Parameter:	*State* – Kommunikationsstatus.
Rückgabewert:	Statusmeldung vom Sensor (Englisch).
Antwort:	10: no device

DLL – Versionskontrolle

GetDLLVersion_HTP()

Mit dieser Funktion kann die Bibliothekversion ermittelt werden.

Syntax:	**function** GetDLLVersion_HTP(): **double**;
Rückgabewert:	Die Versionsnummer wird als Double – Zahl zurückgeliefert.
Antwort:	1,03

GetDLLVersionText_HSI()

Mit dieser Funktion kann die Bibliothekversion ermittelt werden.

Syntax:	**function** GetDLLVersionText_HTP(): **String**;
Rückgabewert:	Die Versionsnummer und Ausgabedatum werden als Text zurückgeliefert.
Antwort:	v.1,03 03.07.2007

Ethernet Schnittstelle verbinden

Bevor Sie eine Übertragung starten, bauen Sie zuerst eine Verbindung zu dem Anschluss auf. Jede nachfolgende Funktion öffnet zuerst die Verbindung mit Host, führt seine Routine und schließt die Verbindung. Bei mehreren Anfragen eines Gerätes nimmt dieses eine gewisse Zeit in Anspruch.

Sie können den Aufwand reduziert, indem man die Verbindung zum Gerät einmalig (z.B. beim Start des Programms) öffnet und alle Anfragen ausführt. Anschließend wird die Verbindung z.B. beim Schließen von FluMoT geschlossen.

Folgende 3 Funktionen dienen zum Aufbauen/Schließen einer Verbindung im Ethernet:

Syntax:	**procedure** OpenConnection_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**); **procedure** CloseConnection_HTP(**const** IpAddress: **String**; **var** State: **Integer**); **procedure** CloseAllConnections_HTP();
Parameter:	IpAddress – IP-Adresse des Gerätes *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Bemerkung:	Für die Kommunikation mit der CMU 1000 ist die Portnummer 5000 erforderlich.

Gerätesuche und Geräteinformation

SearchOneDevice_HTP()

Diese Funktion dient zur Suche eines Gerätes mit einer bestimmten IP – Adresse.

Syntax:	**function** SearchOneDevice_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	SensorID im Erfolgsfall.
Beispiel:	Eine Antwort kann z.B. folgendermaßen aussehen: „CMU1000 V00.20“, wobei die Zahl 0.20 die Firmwareversion des Sensors bezeichnet.

GetDeviceChannelCount_HTP()

Diese Funktion liefert die Anzahl der Kanäle in einem Gerät.

Syntax:	**function** GetDeviceChannelCount_HTP (**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*IpAddress* – IP Addresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Anzahl der Messkanäle.
Antwort:	10 (vom CS 1000)

GetDeviceSerialNumber_HTP()

Diese Funktion liefert die Seriennummer eines Gerätes.

Syntax:	**function** GetDeviceSerialNumber_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Seriennummer des Gerätes als String – Variable.
Antwort:	4711

GetDeviceChannelInfo_HTP()

Diese Funktion dient zur Ermittlung der Kanal – Eigenschaften in einem Gerät. (z.B. Kanalname, Messeinheit usw.)

Syntax:	**function** GetDeviceChannelInfo_HTP(**const** IpAddress: **String**; PortNumber, ChNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. ChNumber – Kanalnummer. (von 0 beginnend) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Die Antwort besteht aus 5 Subzeilen, die mit einem Trennzeichen voneinander getrennt sind. Die Struktur einer solchen Antwort wird in der folgenden Tabelle dargestellt:

Zeilennummer	Parameter	Anmerkung
1	Name	Kanalname
2	Unit	Messbereich, Einheit
3	Decimals	Nachkommastellen

		Alle Zahlenangaben erfolgen ganzzahlig. Der Parameter Decimals gibt an, wie viele Stellen der Zahl hinter dem Dezimalpunkt stehen. Z.B. bedeutet die Zahlenangabe: LowerRange = -250, UpperRange = 1000 und Decimals = 1 einen Messbereich von –25,0 bis 100,0.
4	LowerRange	Messbereich, untere Grenze
5	UpperRange	Messbereich, obere Grenze
Beispiel:	Temp<CR>°C<CR>2<CR>-2500<CR>10000<CR>	

Messwerte auslesen

GetDeviceChannelsMask_HTP()

Mit diesem Befehl kann ermittelt werden, wie sich die Messwerte zusammensetzen, z.B. ob es sich um 8-bit oder 16-bit Zahlen handelt. Dieser Befehl soll nur einmal ausgeführt werden, damit die sog. „Gerätemaske“ im Weiteren verwendet werden kann.

Syntax:	**function** GetDeviceChannelsMask_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State**: Integer): String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Zeiger auf eine Zeichenvariable, die sog. „Gerätemaske“.
Antwort:	3<CR>7<CR>0<CR>0<CR>2<CR>4<CR>2<CR>

Die Struktur einer solchen „Gerätemaske“ wird in der folgenden Tabelle dargestellt:

Zeilennummer	Parameter	Anmerkung
1	ChannelCount	Anzahl der Messkanäle
2	ActivityMask	Jedem Kanal ist ein Bit zugeordnet, das anzeigt ob der Kanal aktiv oder nicht aktiv ist
3	MinMask	Jedem Kanal ist ein Bit zugeordnet, das anzeigt ob der Kanal Min – Werte besitzt oder nicht
4	MaxMask	Jedem Kanal ist ein Bit zugeordnet, das anzeigt ob der Kanal Max – Werte besitzt oder nicht

5	DataSize, Channel 1	Datengröße im 1.Messbereich
6	DataSize, Channel 2	Datengröße im 2.Messbereich
7	…	für jeden Kanal

Der Parameter „DataSize“ kann nur die Werte 1, 2 oder 4 besitzen. Diese Werte entsprechen 8-, 16- oder 32-bit Werten. Diese Werte sind nur für den internen Gebrauch im DLL nötig.

Die „ActivityMask“ gibt an, welche Kanäle tatsächlich aktiv sind. Bei der Messwerte- Übertragung werden inaktive Kanäle nicht mit übertragen. Bit0 der Maske zeigt an ob Kanal 0 aktiv ist, Bit1 Kanal 1 und so weiter.

Mit der Min- und Maxmask wird festgelegt, ob es zu dem jeweiligen Messwert auch noch einen Minwert und/oder einen Maxwert gibt. Bit0 gehört auch hier zu Kanal 0. Ist ein Kanal nicht aktiv, so sind auch die Min- oder Maxwerte grundsätzlich nicht aktiv. Das bedeutet, ein Min- oder Maxwert darf ohne aktuellen Messwert nicht vorkommen.

GetDeviceMeasuringValues_HTP()

Mit diesem Befehl werden die Messwerte angefordert und übertragen.

Die Zusammensetzung der Messwerte muss vorher mit dem Befehl „GetDeviceChannelsMask_HTP“ festgestellt werden. Dabei bekommt man als Antwort sie sog. „Gerätemaske“, die jedes Mal mit dem Befehl „GetDeviceMeasuringValues_HTP“ bestätigt werden muss. Der Grund dafür ist eine Möglichkeit, die Struktur der Messwerte dynamisch anzupassen. Die Struktur der „Gerätemaske“ wurde bereits für die Funktion „GetDeviceChannelsMask_HTP“ beschrieben.

Syntax:	**function** GetDeviceMeasuringValues_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **const** DeviceChannelsMask: **String**; **var** State**: Integer): String**;
Parameter:	*IpAddress* – IP Addresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *DeviceChannelsMask* – „Gerätemaske“. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Messwerte
Antwort:	127<CR>104<CR>80<CR>20<CR>21<CR>22<CR>19<CR>246<CR>100<CR>0<CR>

GetDeviceState_HTP()

Die Abfrage des Sensorstatus hilft Ihnen dabei festzustellen, ob das angeschlossene Gerät betriebsbereit ist, oder in einen Fehlerzustand ist.

Der Sensorstatus hat folgenden Aufbau:

- 8-bit Statusbyte
- 16-bit Statuscode bzw. Fehlercode (mit Vorzeichen)
- Optionaler Statustext

Das *Statusbyte* gibt den aktuellen Zustand des Gerätes an. Die einzelnen Zustände können Sie über den folgenden Statuscode näher spezifizieren.

Folgende Werte für das Statusbyte sind definiert:

0:	Betriebsbereit	Kein aktiver Fehler vorhanden. Das Gerät / der Sensor ist betriebsbereit.
1:	Stand-by	Kein aktiver Fehler vorhanden. Das Gerät / der Sensor ist aber zur Zeit nicht betriebsbereit. Eventuell sind einzelne Gerätefunktionen abgeschaltet oder das Gerät befindet sich in der Anlaufphase, etc..
2:	Leichter Fehler	Ein leichter Fehler liegt vor. Quittiert Sie den Fehler.
3:	Mittlerer Fehler	Ein mittelschwerer Fehler liegt vor. Versuchen Sie den Fehler durch Ein- / Ausschalten des Gerätes / Sensors zu beheben.
4:	Schwerer Fehler	Ein schwerer Fehler liegt vor. Senden Sie das Gerät / den Sensor an HYDAC.

Der *Statuscode* spezifiziert den aktuellen Zustand näher. Es handelt sich um einen 16-bit Wert. Die genaue Bedeutung ist abhängig vom angeschlossenen Gerät / Sensor. Entnehmen Sie dazu weiter Informationen in der Anleitung zum Gerät / Sensor.

Der *Statustext* ist optional und maximal 32 Zeichen lang. Er dient dazu, dass ein Bediengerät den Status eines Sensors im Klartext anzeigen kann.

Syntax:	**function** GetDeviceState_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** StateByte, StateCode, State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes.*StateByte* – StatusByte. *StateCode* – Statuscode. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Statustext
Antwort:	ASIC-CRC-Error“, *StateByte* = 3, *StateCode* =17

Sensoren der CS 2000 Serie mit Ethernet-Schnittstelle

Die Geräte der CS 2000 Serie stellen einen Sonderfall in der Kommunikation dar. Diese Geräte können optional ein Ethernet-Modul haben. In diesem Fall

können Sie die Datenübertragung via dem TCP/IP – Kommunikationsprotokoll führen. Um den CS 2000 mit Ethernetschnittstelle anzusprechen, benötigen Sie einige Sonderbefehle.

Für die DLL – Versionskontrolle können die Befehle GetDLLVersion_HTP und GetDLLVersionText_HTP weiterbenutzt werden.

Die Verbindung mit Gerät wird mit der oben beschriebenen Funktionen OpenConnection_HTP, CloseConnection_HTP und CloseAllConnections_HTP gesteuert. Verwenden Sie für die Kommunikation mit dem CS 2000 die Portnummer 49322.

Folgende Funktionen zur Kommunikation mit einem CS 2000 stehen in der Datei *hsitp32.dll* zur Verfügung:

Funktion	Kurzbeschreibung
SearchOneDevice_CSTCP	Gerätebezeichnung ermitteln
GetDeviceChannelCount_CSTCP	Anzahl der Messkanäle ermitteln
GetDeviceSerialNumber_CSTCP	Seriennummer ermitteln
GetDeviceChannelInfo_CSTCP	Messkanal – Eigenschaften ermitteln
SetMeasuringState_CSTCP	Messung starten / stoppen
GetDeviceMeasuringValues_CSTCP	Messwerte lesen
GetDeviceState_CSTCP	Sensorstatus ermitteln

Das Ethernet-Modul ist für die standardisierte Verkabelungsvariante „10Base-T“ ausgelegt und entspricht den in IEEE 802.3 festgelegten Kriterien (Bitrate 10Mbps, Übertragungsmedium: Twisted Pair (UTP/STP)- Kabel).

Geräte suchen und Geräteinformation auslesen

SearchOneDevice_CSTCP()

Diese Funktion dient zur Suche eines Gerätes mit einer bestimmten IP – Adresse.

Syntax:	**function** SearchOneDevice_CSTCP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist)
	PortNumber – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	SensorID im Erfolgsfall.
Beispiel:	Eine Antwort kann z.B. folgendermaßen aussehen: „CS2210 V03.25“, wobei die Zahl 03.25 die Firmwareversion des Sensors bezeichnet.

GetDeviceChannelCount_CSTCP()

Diese Funktion liefert die Anzahl der Kanäle in einem Gerät.

Syntax:	**function** GetDeviceChannelCount_CSTCP(**const** IpAddress: **String**; PortNumber, Mode: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*IpAddress* – IP Addresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *Mode* – Messdateneinheiten (wird auch in Funktion „GetDeviceMeasuringValues_CSTCP“ verwendet) 0 – Partikelzahlen (differenziell) 1 – NAS/SAE - Klassen 2 – ISO – Code 3 – Partikelzahlen (kumulativ) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Anzahl der Messkanäle
Antwort:	5

GetDeviceSerialNumber_CSTCP()

Diese Funktion liefert die Seriennummer eines Gerätes.

Syntax:	**function** GetDeviceSerialNumber_CSTCP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Seriennummer des Gerätes als String – Variable.
Antwort:	406C120456

GetDeviceChannelInfo_CSTCP()

Diese Funktion dient zur Ermittlung der Kanal – Eigenschaften in einem Gerät. (z.B. Kanalname, Messeinheit usw.)

Syntax:	**function** GetDeviceChannelInfo_CSTCP(**const** IpAddress: **String**; PortNumber, ChNumber, Mode: **Integer**; **var** State: **Integer**): **String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *ChNumber* – Kanalnummer. (von 0 beginnend)

	Mode – Messdateneinheiten (wird auch in Funktion „GetDeviceMeasuringValues_CSTCP“ verwendet) 0 – Partikelzahlen (differenziell) 1 – NAS/SAE - Klassen 2 – ISO – Code 3 – Partikelzahlen (kumulativ) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Die Antwort besteht aus 5 Subzeilen, die mit einem Trennzeichen voneinander getrennt sind. Die Struktur einer solchen Antwort wird in der folgenden Tabelle dargestellt:

Zeilennummer	Parameter	Anmerkung
1	Name	Kanalname
2	Unit	Messbereich, Einheit
3	Decimals	Nachkommastellen Alle Zahlenangaben erfolgen ganzzahlig. Der Parameter Decimals gibt an, wie viele Stellen der Zahl hinter dem Dezimalpunkt stehen. Z.B. bedeutet die Zahlenangabe: LowerRange = -250, UpperRange = 1000 und Decimals = 1 einen Messbereich von -25,0 … 100,0.
4	LowerRange	Messbereich, untere Grenze
5	UpperRange	Messbereich, obere Grenze

Beispiel: Temp<CR>°C<CR>2<CR>-2500<CR>10000<CR>

Messwerte lesen

SetMeasuringState_CSTCP()

Mit diesem Befehl wird eine Messung gestartet oder gestoppt. Zudem könne Sie mit diesem Befehl den Fehlerstatus des Gerätes rückzusetzen, sofern kein schwerer / fataler Fehler vorliegt.

Syntax:	**function** SetMeasuringState_CSTCP(**const** IpAddress: **String**; PortNumber, Mode: **Integer**; **var** State**: Integer): String**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *Mode* – Modus, bezeichnet folgende Aktionen: 0 – Start Messung 1 – Stop Messung

	2 –Errorstatus rücksetzen *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Betriebszustand Die erste Stelle zeigt die Nummer des Messmodus: 1x → M1, 2x → M2, usw. Die zweite Stelle definiert den genauen Zustand: für M1 (Messen), M2 (Messen u. Schalten), M3 (Filtern bis) gilt: x0 Messung aus x1 Warten auf gültigen Durchfluss x2 Messung läuft für M4 (Filtern von bis) gilt: 40 Messung aus 41 Warten auf gültigen Durchfluss 42 Messung läuft, Test auf untere Grenze 43 Wartezeit läuft 44 Wartezeit abgelaufen, warten auf gültigen Durchfluss 45 Messung läuft, Test auf obere Grenze
Antwort:	20 (Messmodus M2, Messung aus)

GetDeviceMeasuringValues_CSTCP()

Mit diesem Befehl werden die Messwerte angefordert und übertragen.

Syntax:	**function** GetDeviceMeasuringValues_CSTCP(**const** IpAddress: **String**; PortNumber, Mode: **Integer**; **var** State**: Integer): String**;
Parameter:	*IpAddress* – IP Addresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) PortNumber – Die Nummer des Portes. Mode – Messdateneinheiten 0 – Partikelzahlen (differenziell) 1 – NAS/SAE - Klassen 2 – ISO-Code 3 – Partikelzahlen (kumulativ) *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Messwerte
Antwort:	127<CR>104<CR>80<CR>20<CR>100<CR>

GetDeviceState_CSTCP()

Mit diesem Befehl kann der aktuelle Status des Gerätes ermittelt werden.

Syntax:	**function** GetDeviceState_CSTCP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **Integer**;
Parameter:	*IpAddress* – IP Adresse des Gerätes („XXX.XXX.XXX.XXX“, wobei XXX – eine Zahl zwischen 0 und 255 ist) *PortNumber* – Die Nummer des Portes. *State* – Referenz auf Kommunikationsstatus - Variable.
Rückgabewert:	Die möglichen Fehlerwerte sind dem Kapitel „DIN Messbus DLL: Fehlerbehandlung“ zu entnehmen
Beispiel:	Antwort: 1 (neue Messwerte sind vorhanden)

Beispiele

Um die Hochsprachenprogrammierung mit FluMoT DLLs zu erleichtern, werden einfache Beispiele als kleine Projekte in Delphi 7/10.4, LabView 7/2012 und Excel -Makros (VBA 6) im Quellcode mitgeliefert.

Diese Beispiele befinden sich nach der Installation im Verzeichnis:

[LW]:\…\FluMoT\Dlls\Examples

Dabei handelt es sich nicht um die fertigen Softwareprodukten, sondern um die kleinen Demo – Programmen.

INFO

LabView – Beispiel (EXE - Datei) ist nur ausführbar, wenn LabView Runtime Engine installiert ist. Ist keine Runtime Engine installiert sein, finden Sie im Verzeichnis „Examples“ eine vollständige Installation.

OPC – Server Schnittstelle

OPC (Openness, Productivity, Collaboration, früher OLE for Process Control) ist eine standardisierte Schnittstelle zum Zugriff auf Prozessdaten. Sie basiert auf dem Microsoft Standard DCOM und wurde für die Bedürfnisse des Datenzugriffs in der Automatisierung erweitert. Sie wird vorwiegend zum Lesen von Messwerten aus der Steuerung verwendet. Clients sind in der Regel Visualisierungen, Programme zur Betriebsdatenerfassung usw. OPC Server werden typischerweise als Treiber mit unterschiedlichen Feldgeräten oder Sensoren zur Verfügung gestellt.

Der OPC Server ist ein ausführbares Programm, das bei einem Verbindungsaufbau zwischen Client und Sensor gestartet wird. Der Server sammelt alle verfügbare Messdaten von Sensoren und stellt diese als Variablen dem Client zur Verfügung. Auch mehrere Clients können gleichzeitig auf diese Daten zugreifen.

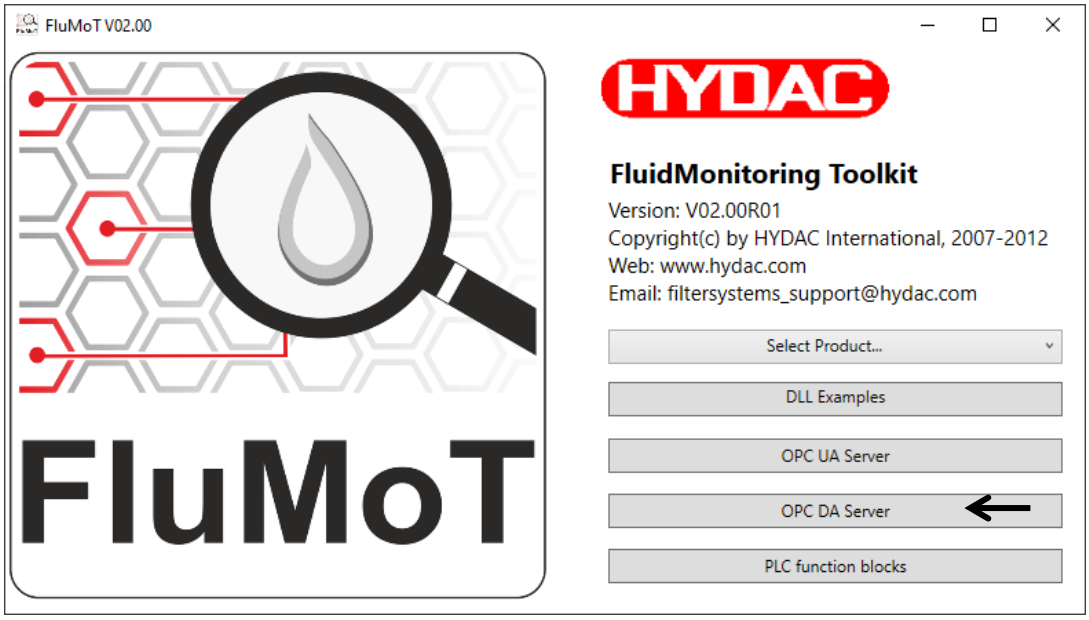
FluMoT kommuniziert mit folgenden Geräte- / Sensortypen:

- **HSI Devices** (Serielle Kommunikation mittels HSI – Protokoll, unterschiedliche Einstellungen für Baudrate möglich)
- **DIN Messbus Devices** (Serielle Kommunikation mittels DIN Messbus)
- **HSITP Devices** (TCP/IP Kommunikation mittels HSI-TP Protokoll)
- CS 2000 mit einer Ethernet-Schnittstelle

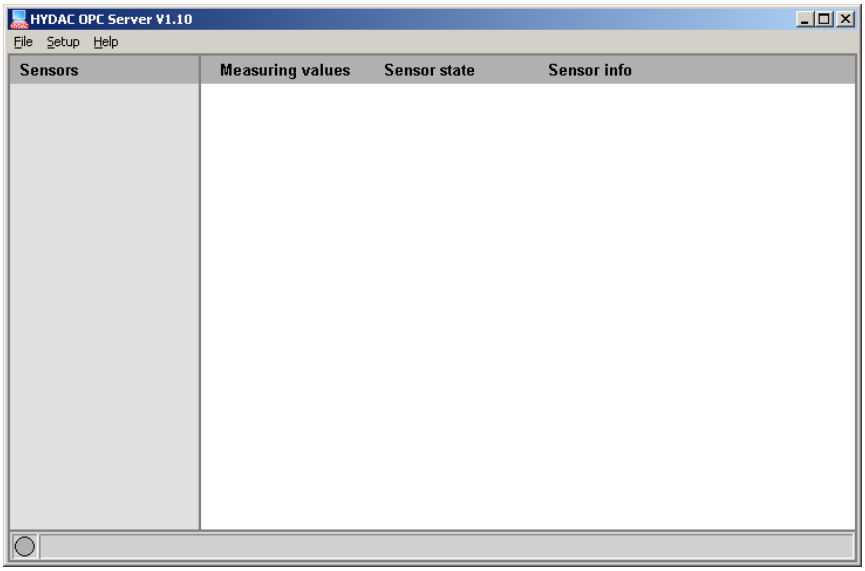
DCOM - Technologie ermöglicht auch einen Zugriff auf Server, der auf einem anderen PC läuft. Dazu sind allerdings die gewissen DCOM – Einstellungen nötig. Die Konfiguration der DCOM Sicherheitseinstellungen wird mit dem Dienstprogramm "DCOMCNFG.EXE" durchgeführt. Nähere Informationen sind der entsprechenden WINDOWS Dokumentation zu entnehmen.

OPC-DA

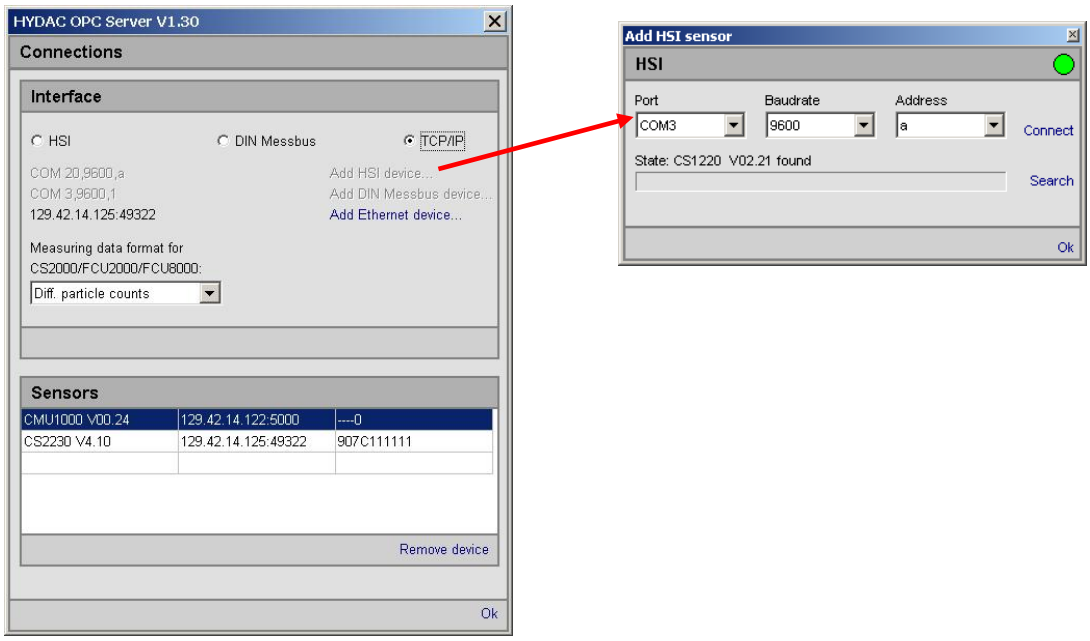
HYDAC OPC Server V1.30 basiert auf OPC Data Access 2.0 (Spezifikation zur Übertragung von Echtzeitwerten über Microsoft Standard DCOM). Der Server wird während FluMoT Installation auf den PC kopiert und im System registriert. Bei der Deinstallation von FluMoT wird auch der HYDAC OPC Server entfernt. Sie können die Registrierung / Deregistrierung mit Hilfe der beiliegenden Dateien *reg_opc.bat* und *unreg_opc.bat* auch manuell durchführen.



Konfigurieren Sie den HYDAC OPC Server nach der Installation. Definieren Sie dazu die Schnittstellen aller abzufragenden Sensoren in FluMoT.

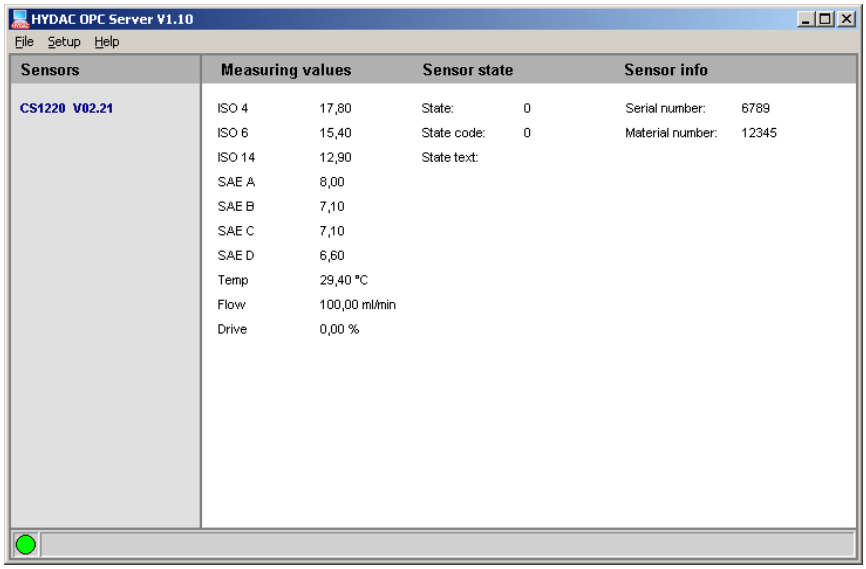


Über den Menüpunkt SETUP -> CONNECTIONS im Hauptmenü des Programms stellen Sie die Verbindung zu den einzelnen Sensoren her. Die Geräte / Sensoren werden einer Liste hinzugefügt. Diese Liste wird bei der Bestätigung durch die Ok Taste gespeichert. Anschließend wird die Online Messung gestartet.



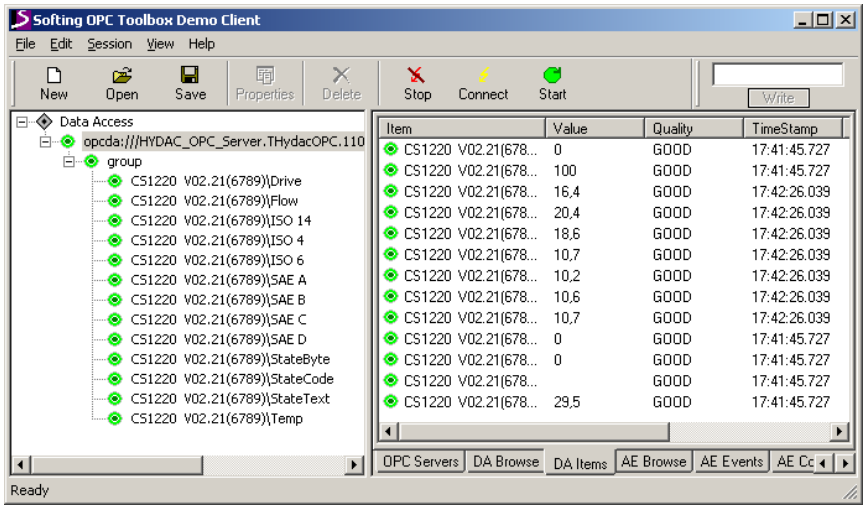
Spricht ein OPC – Client den Server an, wir diese Verbindung gespeichert und immer wiederhergestellt.

Die Anzahl von Sensoren, die auf diese Weise angesprochen werden, ist unbegrenzt. Beachten Sie, dass durch die Kommunikation mit vielen Geräten / Sensoren die Übertragung deutlich langsamer wird.



Nun kann HYDAC – OPC Server mit einem OPC - Client kommunizieren. Starten Sie einen OPC Client. (z.B. Softing OPC Demo Client) Unter dem Auswahlpunkt „*OPC Servers*“ selektieren Sie unter LOCAL → DATA ACCESS V2 und erzeugen Sie eine Verbindung zum “HYDAC OPC – Server V1.30”. Der OPC – Server wird automatisch gestartet und beginnt direkt, alle Geräte aus seiner Liste abzufragen. So stehen die Messwerte möglichst schnell zur Verfügung. Sobald die Messung gestartet wurde, geht das Programm ins Tray und läuft im Hintergrund als ein Dienst.

Jetzt kann die Client – Anwendung die einzelnen Kanäle vom Sensor einbinden (siehe Registerkarte DA Browse im Softing OPC Demo Client) und die entsprechenden Messwerte anzeigen / bearbeiten. (Registerkarte DA Items)

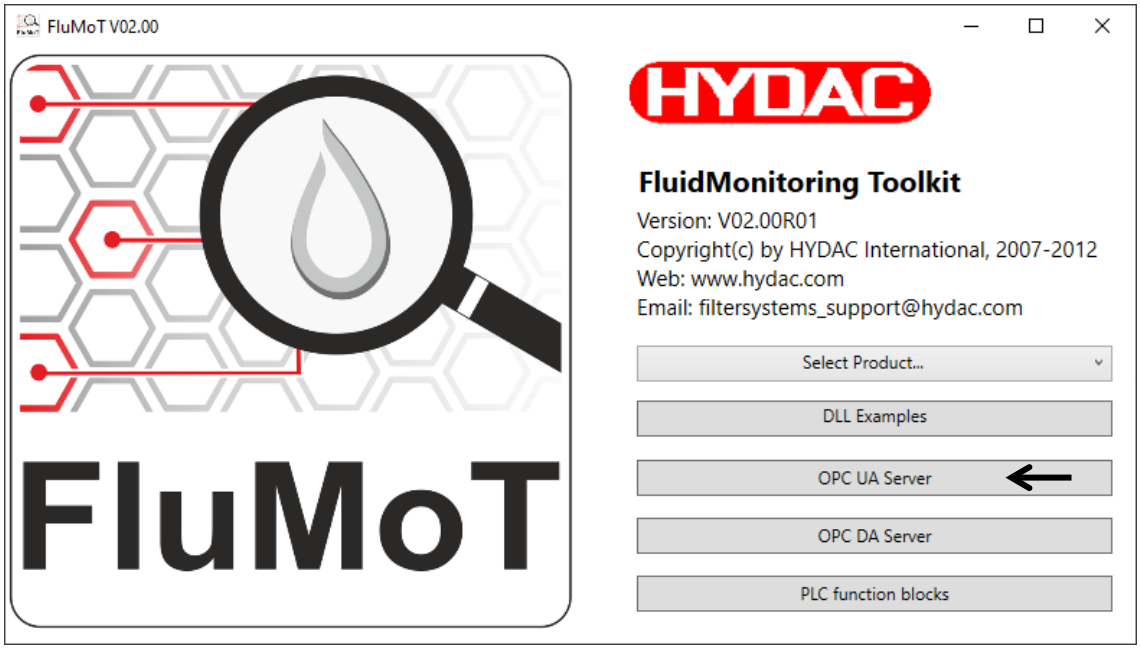


OPC-UA Server

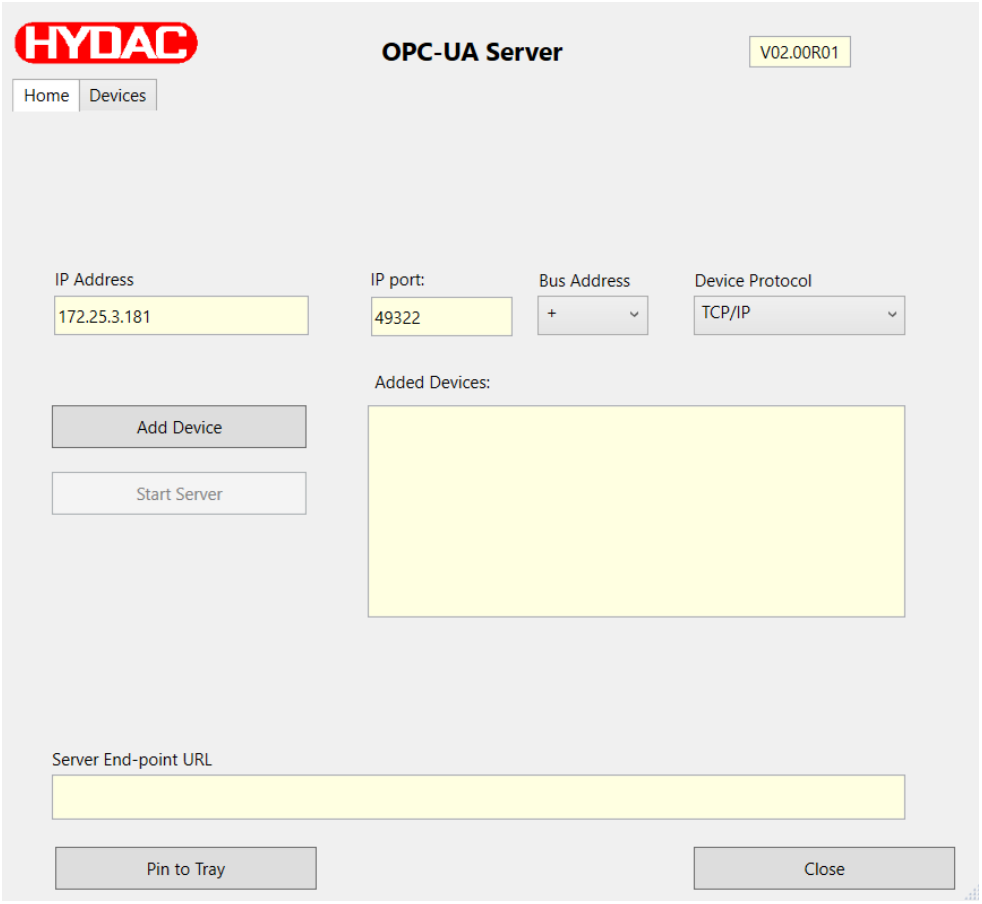
Klassische OPC-Server legen ihre internen Strukturen als hierarchische Bäume offen. OPC DA-Server stellen eine Hierarchie von Verzweigungen und Elementen bereit, wobei Verzweigungen verwendet werden, um andere Verzweigungen und Elemente zu organisieren, und nur Elemente können einen Wert haben. Dies gilt nur für den Einsatz von OPC AE/DA Server.

Das OPC UA verwendet aber breiteren und allgemeineren Adressraum als klassisches OPC. OPC UA betrachtet Metadaten als Teil des Adressraums und ist nicht auf hierarchische Bäume beschränkt. Es bietet eine allgemeine Möglichkeit, jede Art von Beziehung aufzudecken. Das HYDAC OPC UA Server V2.00 basiert, wie beim OPC-DA, auf OPC Data Access (Spezifikation zur Übertragung von Echtzeitwerten).

Um den OPC-UA Server zu starten, klicken Sie auf den Knopf „OPC UA Server“

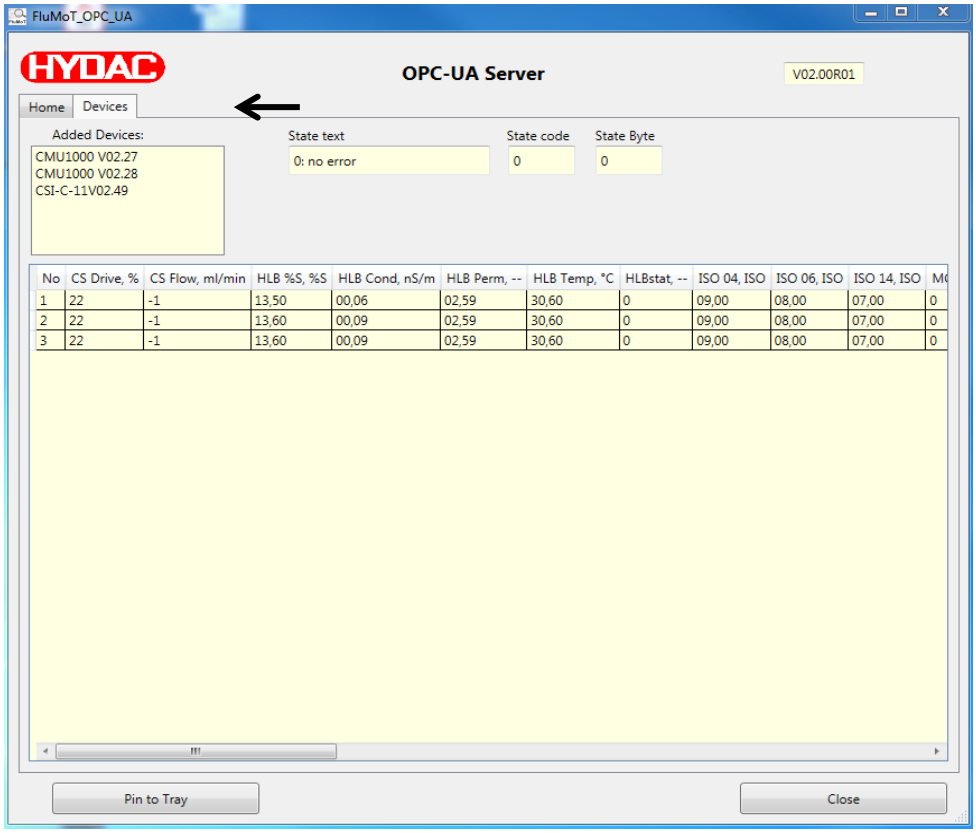


Das Konfigurationsfenster wird nun angezeigt(sehe folgendes Bild). Hier können Sie die Geräte durch die Eingabe der richtigen Parameter und dann das klicken auf dem Knopf „Add Device“ definieren. Die hinzugefügten Geräte werden auf dem Fenster „Devices“ angezeigt. Es ist zu merken, dass nur fünf Geräte hinzugefügt werden dürfen.



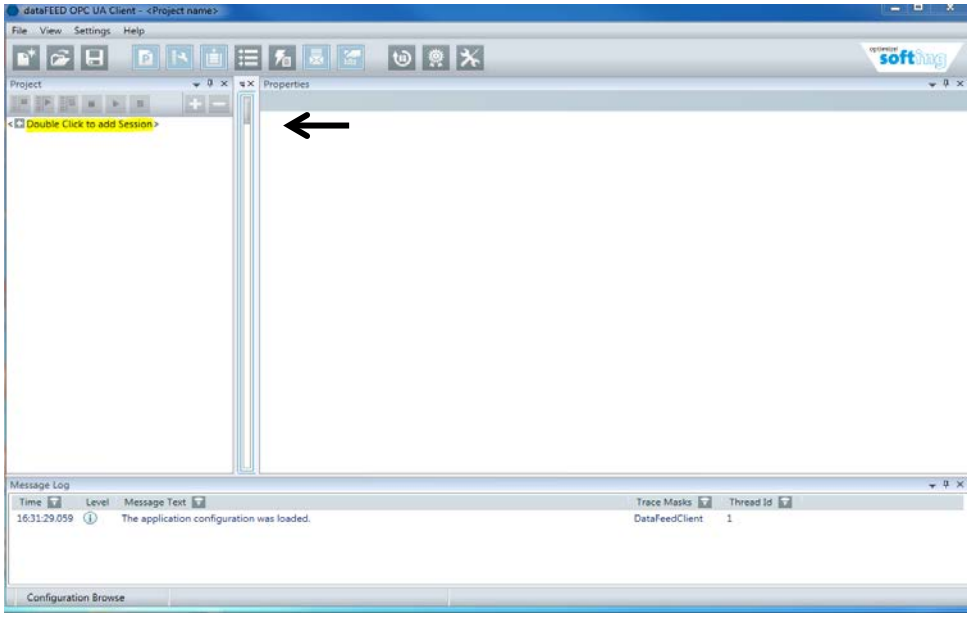
Der „Start Server“ Knopf wird nur aktivierte, wenn mindestens ein Gerät definiert ist. Beim Klicken auf diesen Knopf starten Sie das OPC UA Server. Nach Starten des Servers können keine weiteren Geräte hinzugefügt werden. Ein Server End-point URL wird nun angegeben.

Unter dem Teilfenster „Devices“ kann der Benutzer Gerätekanäle und ihre Messwerte lesen sowie das Gerätzustand durch das Gerät-Statustext Statusbyte überprüfen.

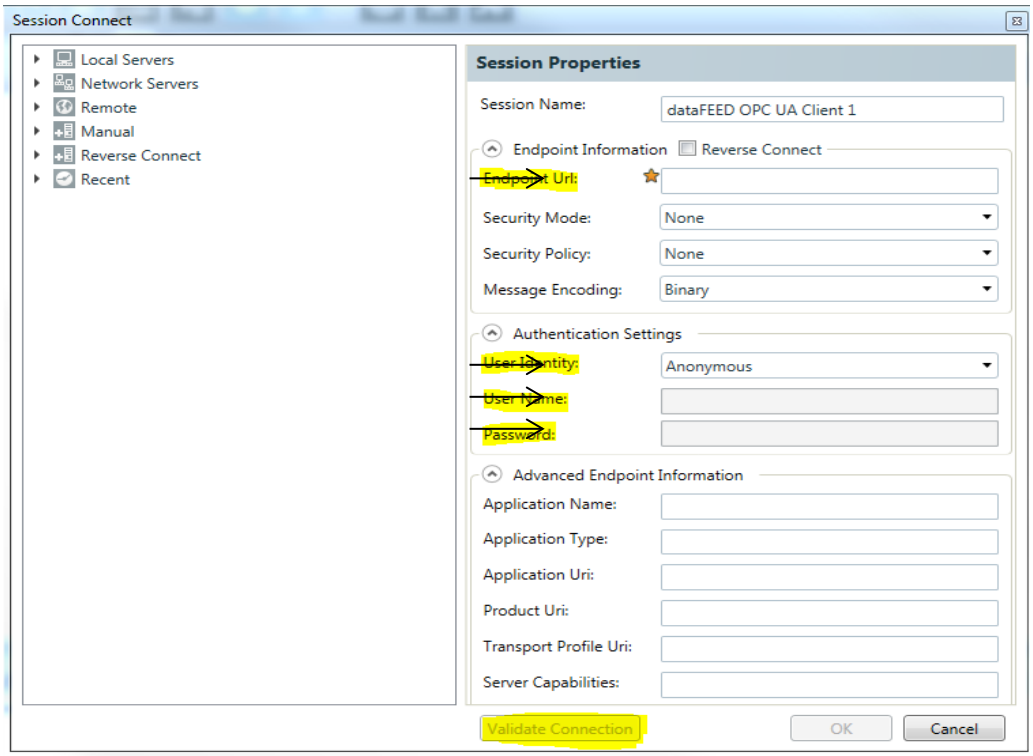


Merke: durch den Knopf „Pin to Tray“ können Sie das Programm laufen lassen, aber im Treiber-Tray versteckt halten. Das schließen vom Programm beendet auch den OPC-UA Server.

Nach dem Starten des Servers können Sie die Daten von den hinzugefügten Geräten durch einen OPC Client (z.B. Softing OPC Demo Client) gelesen werden. Dafür starten Sie das OPC Client und addieren Sie ein „Session“.



Ein Konfigurationsfenster vom Client wird wie im folgenden Bild angegeben.



Um das Client zu konfigurieren sind die markierten Felder zu bedienen. Diese sind:

a) Fügen Sie das Endpunkt-URL hinzu.

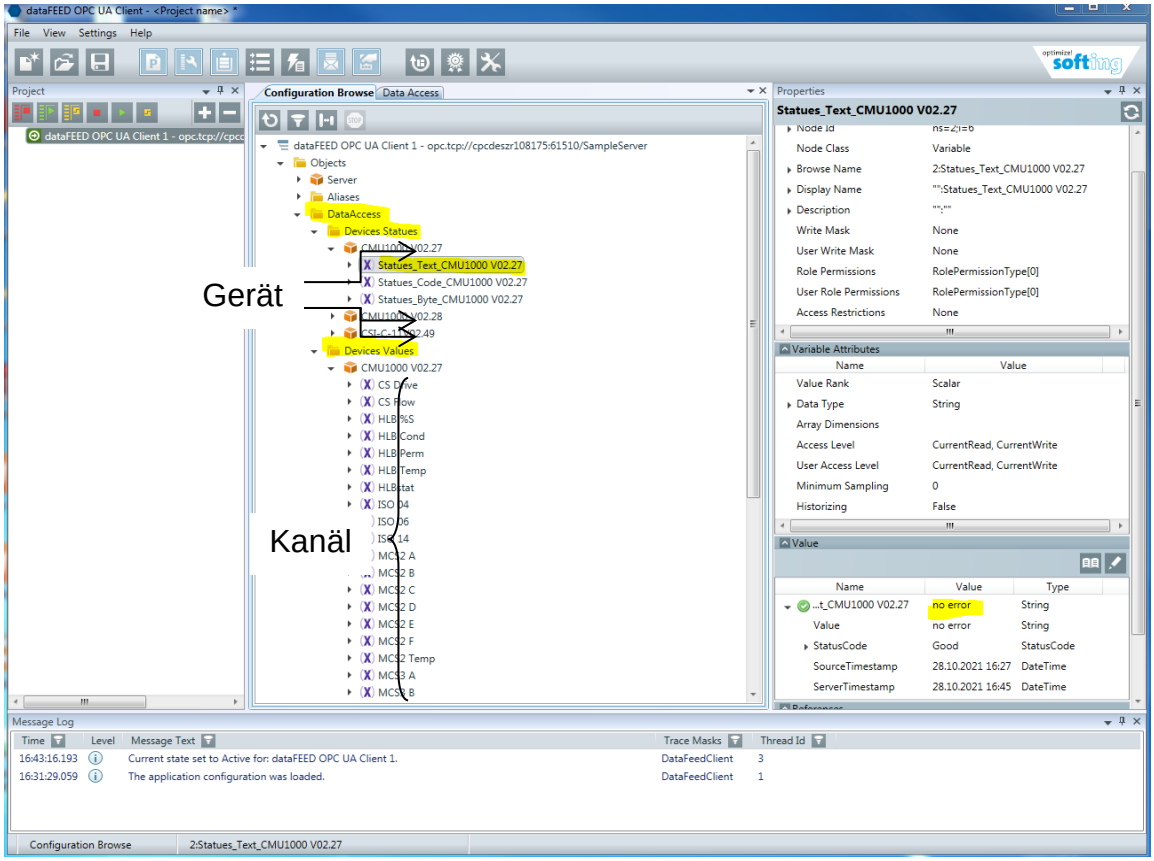
b) Ändern Sie die Benutzeridentität „User Identity“ in Benutzername.

c) Geben Sie Ihren Benutzernamen und Ihr Passwort ein.

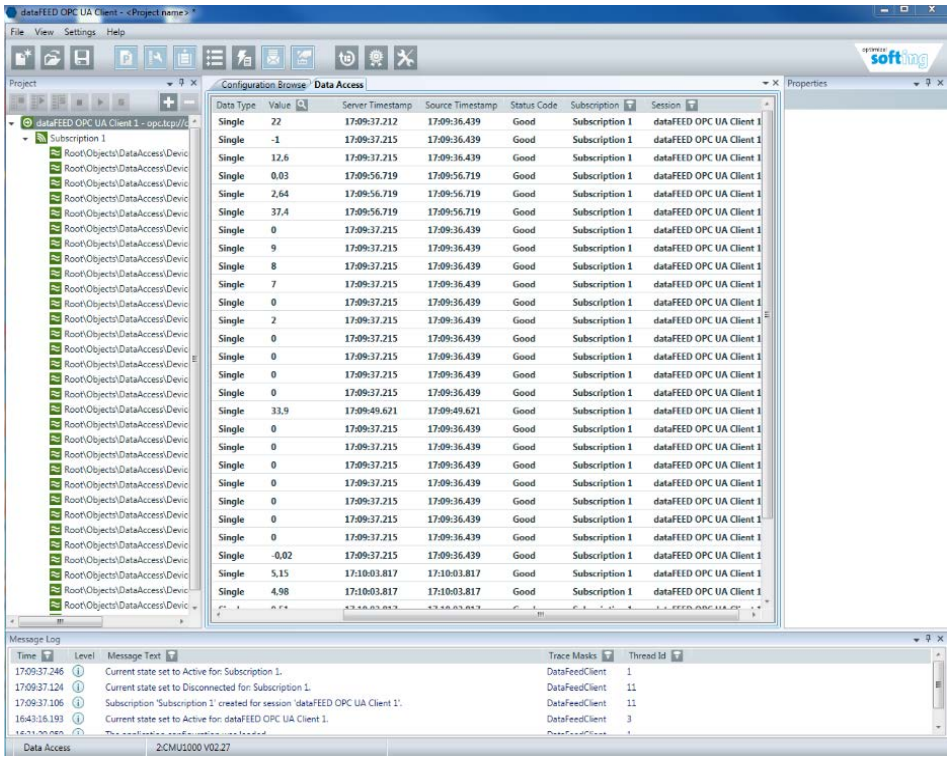
d) Sie können Ihre Daten sowie die Verbindung überprüfen, indem Sie auf „Verbindung validieren“ klicken.

Aus Sicherheitsgründen können nur registrierte Benutzer auf die Daten der Geräte im Client zugreifen.

Auf dem Fenster „Configuration Browse“ können Sie alle Geräte, Ereignisse, Kanalwerte, …usw. überprüfen. Unter dem Ordner „DataAccess“ sind zwei Unterordner zu finden. Im ersten Unterordner „Devices Statues“ können Sie den Gerätestatus (Statustext, Statuscode und Statusbyte) überprüfen. Unter dem zweiten Unterordner „Devices Values“ können Sie auf die Geräte, Gerätkanäle und die Messwerte sowie die dazu nötige Informationen (z.B. Einheit, Messbereiche... usw.) zugreifen.



Sie können bestimmte Kanäle wählen und die auf dem Fenster „Data Access“ einfacher überprüfen. Um alle Kanäle eins Geräts zu überwachen:
a) Klicken Sie mit der rechten Maustaste auf das Gerät.
b) Wählen Sie „Alle Kinder überwachen“.



Bausteine für CoDesys V3 / Beckhoff TWINCAT3

Folgende Bausteine ermöglichen das Einlesen der Sensoren mit HSI Schnittstellen in die Steuerung. Zur Verwendung der Bausteinen benötigt man seriellen Schnittstellenklemmen EL600x, EL602x mit der Bibliothek Tc2_SerialCom von Beckhoff.Die Schnittstellen liegen in der Verantwortung der Haupt-Anwendung (Kundenseite) und der Baustein ist durch die Byte-Arrays TX/RX zunächst unabhängig von der verwendeten Schnittstelle.

FUNCTION_BLOCK Hsi_Dev_GetId

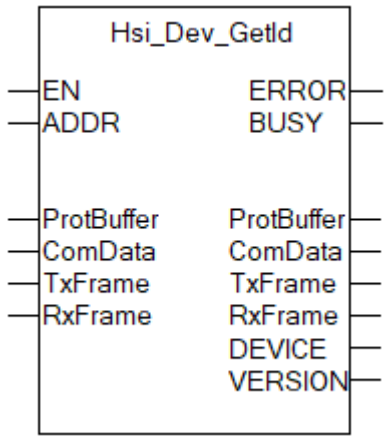
Der Baustein Hsi_Dev_GetId liest die Sensor-Id von einem Hydac-Gerät aus. Die Sensor-Id dient dazu festzustellen, welches Gerät angeschlossen ist.

Die Sensor-Id besteht aus 14 Zeichen mit folgendem Aufbau:

- 8 Zeichen Gerätebezeichnung,

- 6 Zeichen Versionsbezeichnung.

Die Versionsbezeichnung hat folgenden Aufbau: Vxx.yy, wobei xx für die Versionsnummer und yy für die Releasenummer steht. Beide Nummern sind 2-stellig, d.h. bei Nummern <10 wird eine führende Null vorangestellt.




```
VAR_INPUT
    EN    : Bool;          // Über eine positive Flanke an diesem Eingang
                           wird der Baustein aktiviert.
    ADDR  : STRING(1);     // Busadresse des HSI-Sensors.
END_VAR
```

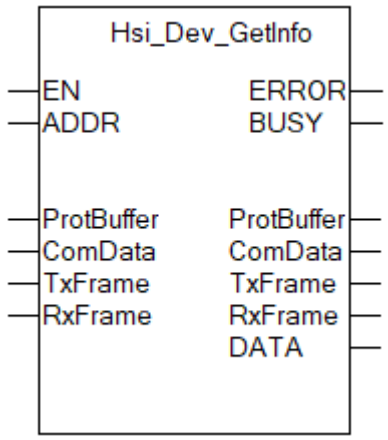
Am Ausgang sind interne Datenstrukturen zusammengefasst, die für eine Reibungslose Kommunikation zuständig sind. Diese muss man nur einmal beim Start instanziieren.

```
VAR_IN_OUT
    ProtBuffer:HSI_DATA_PROT;    // Interne Daten für HSI Kommunkation.
    ComData:   HSI_DATA_COM;     // Enthält 2 Datenpuffer für die
                                 serielle Kommunikation.
    TxFrame:   HSI_DATA_FRAME;   // Enthält einen Datenpaket zum Senden.
    RxFrame:   HSI_DATA_FRAME;   // Enthält einen Datenpaket zum
                                 Empfangen.
END_VAR
```

```
VAR_OUTPUT
    BUSY: Bool;                  // Dieser Ausgang bleibt solange auf TRUE,
                                 bis der Baustein seine Ausführung beendet
                                 hat.
    ERROR : Bool;                // Dieser Ausgang wird auf TRUE geschaltet,
                                 wenn ein Fehler während der
                                 Befehlsübertragung auftritt.
    DEVICE: STRING(8);           // Gerätebezeichnung (max. 8 Zeichen).
    VERSION     : STRING(6);     // Versionsbezeichnung (max. 6 Zeichen).
END_VAR
```

FUNCTION_BLOCK Hsi_Dev_GetInfo

Die Sensorinfos beschreiben detailliert einen Sensor und seine Funktionen. Sie dienen einem Bediengerät dazu, herauszufinden welche Funktionen ein Sensor bietet, wie seine Daten strukturiert sind und noch einiges mehr. Während die bereits beschriebene Sensor-Id mehr oder minder nur einen Namen darstellt, sind die Sensorinfos wesentlich detaillierter und strukturierter.



```
VAR_INPUT
    EN    : Bool;          // Über eine positive Flanke an diesem Eingang
                           wird der Baustein aktiviert.
    ADDR  : STRING(1);     // Busadresse des HSI-Sensors.
END_VAR
```

Am Ausgang sind interne Datenstrukturen zusammengefasst, die für eine Reibungslose Kommunikation zuständig sind. Diese muss man nur einmal beim Start instanziieren.

```
VAR_IN_OUT
    ProtBuffer:HSI_DATA_PROT;    // Interne Daten für HSI Kommunkation.
    ComData:   HSI_DATA_COM;     // Enthält 2 Datenpuffer für die
                                 serielle Kommunikation.
    TxFrame:   HSI_DATA_FRAME;   // Enthält einen Datenpaket zum Senden.
    RxFrame:   HSI_DATA_FRAME;   // Enthält einen Datenpaket zum
                                 Empfangen.
END_VAR
```

```
VAR_OUTPUT
    BUSY:   Bool;             // Dieser Ausgang bleibt solange auf TRUE,
                              bis der Baustein seine Ausführung beendet
                              hat.
    ERROR:  Bool;             // Dieser Ausgang wird auf TRUE geschaltet,
                              wenn ein Fehler während der
                              Befehlsübertragung auftritt.
    DATA:   DEVICE_HSI_INFO;  // Geräteinformationen in Struct.
END_VAR
```

Die Rückgabe enthält die Sensorinformationen in folgenden Datenstrukturen:

```
TYPE DEVICE_HSI_INFO:
STRUCT
    SERIALNO: UDINT;                // Seriennummer des Gerätes.
    CHCOUNT:  BYTE;                 // Anzahl Messkanäle.
    CHINFO:   ARRAY [0..31] OF      // Geräteinformationen in Struct.
    DEVICE_HSI_CHANNEL_INFO;
END_VAR
```

```
TYPE DEVICE_HSI_CHANNEL_INFO:
STRUCT
```

```
    Name        : STRING(10);    // Name des Messkanales.
    Unit        : STRING(8);     // Einheit des Messkanales.
    Decimals    : BYTE;          // Anzahl Dezimalstellen.
    LowerRange  : DINT;          // Unterer Messbereich.
    UpperRange  : DINT;          // Oberer Messbereich.
END_VAR
```

FUNCTION_BLOCK Hsi_Dev_GetState

Der Sensorstatus dient dazu festzustellen, ob das angeschlossene Gerät betriebsbereit ist, oder in einen Fehlerzustand eingetreten ist. Der Sensorstatus hat folgenden Aufbau:

- 8-bit Statusbyte,
- 16-bit Statuscode bzw. Fehlercode (mit Vorzeichen)
- Optionaler Statustext

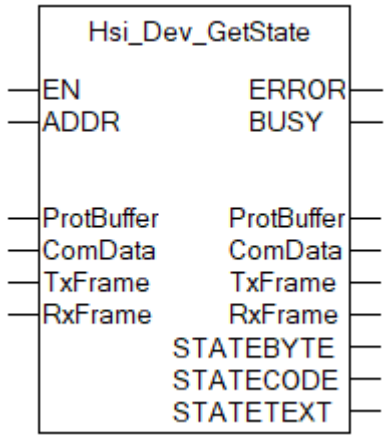
Das *Statusbyte* gibt den aktuellen Zustand des Gerätes an. Die einzelnen Zustände können über den folgenden Statuscode näher spezifiziert werden.

Folgende Werte für das Statusbyte sind definiert:

0:	Betriebsbereit	Kein aktiver Fehler vorhanden, Gerät ist betriebsbereit.
1:	Stand-by	Kein aktiver Fehler vorhanden, Gerät ist aber zur Zeit nicht betriebsbereit, eventuell sind einzelne Gerätefunktionen abgeschaltet, oder Gerät ist in einer Anlaufphase, etc.
2:	Leichter Fehler	Es ist ein leichter Fehler vorhanden, der quittiert werden kann.
3:	Mittlerer Fehler	Es ist ein mittelschwerer Fehler vorhanden, der durch Ein/Ausschalten eventuell behebbar ist.
4:	Schwerer Fehler	Es ist ein schwerer Fehler vorhanden, das Gerät muss zum Hersteller zurück.

Der *Statuscode* spezifiziert den aktuellen Zustand näher. Es ist ein 16-bit Wert. Die genaue Bedeutung ist von Gerät zu Gerät unterschiedlich. Der Anwender kann dann dem Handbuch nähere Infos zu dem Statuscode entnehmen.

Der *Statustext* ist optional und maximal 32 Zeichen lang. Er dient dazu, dass ein Bediengerät den Status eines Sensors im Klartext anzeigen kann.




```
VAR_INPUT
    EN:   BOOL;       // Über eine positive Flanke an diesem Eingang wird
                      der Baustein aktiviert.
    ADDR:             // Busadresse des HSI-Sensors.
    STRING(1);
END_VAR
```

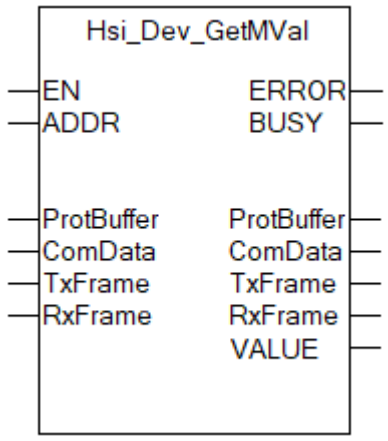
Am Ausgang sind interne Datenstrukturen zusammengefasst, die für eine Reibungslose Kommunikation zuständig sind. Diese muss man nur einmal beim Start instanziieren.

```
VAR_IN_OUT
    ProtBuffer: HSI_DATA_PROT;    // Interne Daten für HSI Kommunkation.
    ComData:    HSI_DATA_COM;     // Enthält 2 Datenpuffer für die
                                  serielle Kommunikation.
    TxFrame:    HSI_DATA_FRAME;   // Enthält einen Datenpaket zum Senden.
    RxFrame:    HSI_DATA_FRAME;   // Enthält einen Datenpaket zum
                                  Empfangen.
END_VAR
```

```
VAR_OUTPUT
    BUSY:       BOOL;         // Dieser Ausgang bleibt solange auf TRUE,
                              bis der Baustein seine Ausführung beendet
                              hat.
    ERROR:      BOOL;         // Dieser Ausgang wird auf TRUE geschaltet,
                              wenn ein Fehler während der
                              Befehlsübertragung auftritt.
    STATEBYTE:  BYTE;         // 8-bit Statusbyte.
    STATECODE:  INT;          // 16-bit Statuscode.
    STATETEXT:  STRING(16);   // Optionaler Statustext.
END_VAR
```

FUNCTION_BLOCK Hsi_Dev_GetMVal

Messwerte können von digitalen HSI-Sensoren, sogenannten Smart-Sensoren zu einem Bediengerät übertragen werden. Analoge HSI-Sensoren sind dazu nicht in der Lage.




```
VAR_INPUT
    EN:   BOOL;       // Über eine positive Flanke an diesem Eingang wird
                      der Baustein aktiviert.
    ADDR:             // Busadresse des HSI-Sensors.
    STRING(1);
END_VAR
```

Hier sind interne Datenstrukturen zusammengefasst, die für eine Reibungslose Kommunikation zuständig sind. Diese muss man nur einmal beim Start instanziieren.

```
VAR_IN_OUT
    ProtBuffer: HSI_DATA_PROT;    // Interne Daten für HSI Kommunkation.
    ComData:    HSI_DATA_COM;     // Enthält 2 Datenpuffer für die
                                  serielle Kommunikation.
    TxFrame:    HSI_DATA_FRAME;   // Enthält einen Datenpaket zum Senden.
    RxFrame:    HSI_DATA_FRAME;   // Enthält einen Datenpaket zum
                                  Empfangen.
END_VAR
```

```
VAR_OUTPUT
    BUSY:   BOOL;           // Dieser Ausgang bleibt solange auf TRUE,
                            bis der Baustein seine Ausführung beendet
                            hat.
```

```
    ERROR:  BOOL;           // Dieser Ausgang wird auf TRUE geschaltet,
                            wenn ein Fehler während der
                            Befehlsübertragung auftritt.
    VALUE:                  // Die Struktur mit einem Messwert und Anzahl
    DEVICE_HSI_MEASVALUES;  Dezimalstellen beim Messkanal.
END_VAR
```

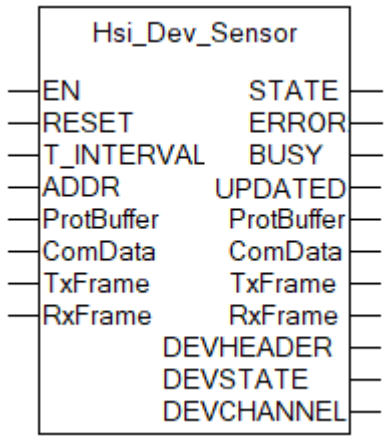
Die Rückgabe enthält die aktuellen Messwerten in folgenden Datenstrukturen:

```
TYPE DEVICE_HSI_MEASVALUES :
STRUCT
      CHANNEL     : ARRAY [0..31] OF DEVICE_HSI_CHANNEL_MEAS;
END_STRUCT
END_TYPE
```

```
TYPE DEVICE_HSI_CHANNEL_MEAS :
STRUCT
      Value       : DINT; // Messwert als ganze Zahl.
      Decimals    : BYTE; // Anzahl Dezimalstellen.
END_STRUCT
END_TYPE
```

FUNCTION_BLOCK HSI_DEV_SENSOR

Der Baustein **Hsi_Dev_Sensor** übernimmt nach der Freigabe die zyklische Abfrage des HSI Sensors (in der richtigen Abfragereihenfolge der HSI Befehle, d.h. Id und Sensorinfos nur einmalig bei Start, danach Status und Messwerte zyklisch) und bereitet die erhaltenen Daten für die Hauptanwendung sensorspezifisch.



```
VAR_INPUT
   EN:           BOOL;       Über eine positive Flanke an diesem Eingang
                             wird der Baustein aktiviert.
   RESET:        BOOL;       Über eine positive Flanke an diesem Eingang
                             wird der ERROR Ausgang zurückgesetzt.
   T_INTERVAL:   DWORD;      Abtastrate für Aktualisierung der
                             Sensordaten (ms).
   ADDR:         STRING(1);  Busadresse des HSI-Sensors.
END_VAR
```

Am Ausgang sind interne Datenstrukturen zusammengefasst, die für eine Reibungslose Kommunikation zuständig sind. Diese muss man nur einmal beim Start instanziieren.

```
VAR_IN_OUT
    ProtBuffer: HSI_DATA_PROT;    // Interne Daten für HSI Kommunkation.
    ComData:    HSI_DATA_COM;     // Enthält 2 Datenpuffer für die
                                  serielle Kommunikation.
    TxFrame:    HSI_DATA_FRAME;   // Enthält einen Datenpaket zum Senden.
    RxFrame:    HSI_DATA_FRAME;   // Enthält einen Datenpaket zum
                                  Empfangen.
END_VAR
```

```
VAR_OUTPUT
     STATE:     INT;                Status Kommunikation.
     ERROR:     BOOL;               Dieser Ausgang wird auf TRUE
                                    geschaltet, wenn ein Fehler während
                                    der Befehlsübertragung auftritt.
     BUSY:      BOOL;               Dieser Ausgang bleibt solange auf
                                    TRUE, bis der Baustein seine
                                    Ausführung beendet hat.
     UPDATED:   BOOL;               Dieser Ausgang wird auf TRUE
                                    geschaltet, wenn die Sensordaten
                                    inkl.
                                    Messwerten aktualisiert sind.

     DEVHEADER: DEVICE_SNS_HEADER;
     DEVSTATE:  DEVICE_SNS_STATE;
     DEVCHANNEL: ARRAY [0..31] OF
     DEVICE_SNS_CHANNEL;
END_VAR
```

Die Rückgabe enthält die aktuellen Messwerten in folgenden Datenstrukturen:

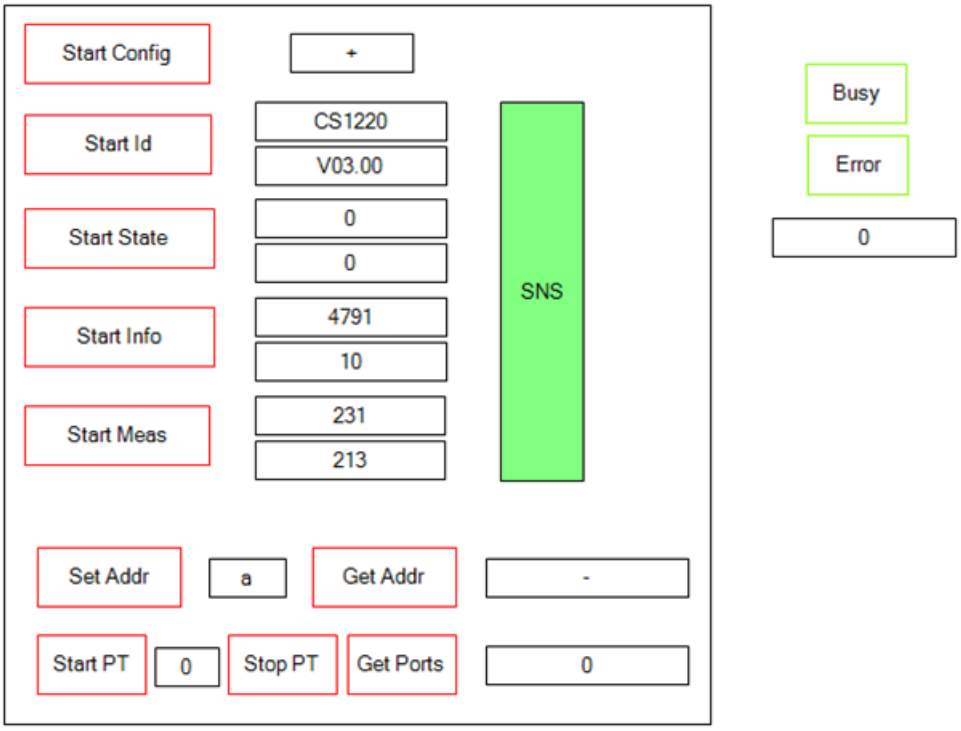
```
TYPE DEVICE_SNS_HEADER:
STRUCT
     Name:     STRING(8);   // SensorID des Gerätes.
     Firmware: STRING(6);   // Firmware-Version.
     SerNo:    UDInt;       // Seriennummer.
     ChCount:  Byte;        // Anzahl Messkanäle.
END_STRUCT
END_TYPE
```

```
TYPE DEVICE_SNS_STATE:
STRUCT
     StateByte:  Byte;      // Sensor StatusByte.
     StateCode:  Int;       // Sensor StatusCode.
END_STRUCT
END_TYPE
```

```
TYPE DEVICE_SNS_CHANNEL:
STRUCT
     Name:                  // Name des Messkanals.
     STRING(10);
     Unit:     STRING(8);   // Messeinheit.
     Decimals: Byte;        // Anzahl Dezimalstellen.
     Value:    DINT;        // Aktueller messwert.
END_STRUCT
END_TYPE
```

Eine Einfache Visualisierung

Das TWINCAT 3 Projekt enthält auch eine einfache Visualisierung, um die einzelnen Fuktionen zu testen. Damit kann man entweder einzelnen Funktionen als auch den Baustein **Hsi_Dev_Sensor** ausführen. Wenn der Buttun SNS grün ist, dann ist der Baustein aktiv, sonst kann man die Funktionen einzeln testen.


Start Config
+
Start Id
CS1220
V03.00
Start State
0
0
Start Info
4791
10
Start Meas
231
213
SNS
Busy
Error
0
Set Addr
a
Get Addr
-
Start PT
0
Stop PT
Get Ports
0

Siemens Bausteine

FluMoT bietet folgende Projekte für die SIEMENS Steuerungen an:

Projektname	Entw.-Umgebung	SPS Familie	Funktion
Flumot_S71x00	TIA Portal V13SP1	S7 1200/1500	1
HLB_Analog HDA.ISO	TIA Portal V13SP1	S7 1200	2
TIA_Hydac_Sensoren	TIA Portal V15SP1	S7 1200	3
HLB 400_A_1	SIMATIC MANAGER STEP 7	S7 300/400	4
S7312_RS	SIMATIC MANAGER STEP 7	S7 300/1200V2	5

Funktionen:

1 Digitale Kommunikation mit Hydac Smartsensoren.

2 Analoge Kommunikation mit HLB1400 (Signalkodierung s. Datenblatt von HLB1400).

3 ModbusTCP Kommunikation mit CSI-C-11, analoge Kommunikation CS1000.

4 Analoge Kommunikation mit HLB1400(Signalkodierung s. Datenblatt von HLB1400).

5 Digitale Kommunikation mit Hydac Smartsensoren.

Projekt Flumot_S71x00 (Zielsystem S7 1200/1500)

Das Projekt Flumot_S71x00 enthält den Programmbaustein HSI_FB. Hier wurden die einzelnen Funktionsblöcken zur digitalen Kommunikation mit den Hydac-Smartsensoren umgesetzt. Dieser Programmbaustein wurde in der TIA Portal V13 entwickelt. Als Zielsystem sind S7 1200 und S71500 definiert mit Version ≥4.0. Die Schnittstellen liegen in der Verantwortung der Haupt-Anwendung (Kundenseite).

FUNCTION_BLOCK HSI_ReadSensorId

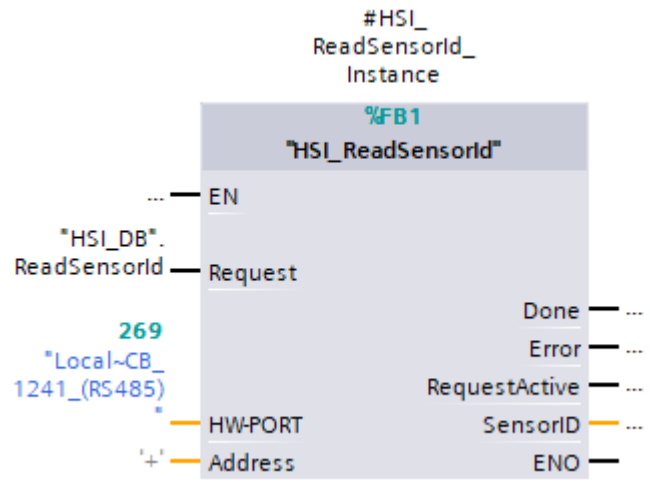
Der Baustein Hsi_ReadSensorId liest die Sensor-Id von einem Hydac-Gerät aus. Die Sensor-Id dient dazu festzustellen, welches Gerät angeschlossen ist.

Die Sensor-Id besteht aus 14 Zeichen mit folgendem Aufbau:

- 8 Zeichen Gerätebezeichnung,

- 6 Zeichen Versionsbezeichnung.

Die Versionsbezeichnung hat folgenden Aufbau: Vxx.yy, wobei xx für die Versionsnummer und yy für die Releasenummer steht. Beide Nummern sind 2-stellig, d.h. bei Nummern <10 wird eine führende Null vorangestellt.




```
VAR_INPUT
      EN:      Bool;       // Wenn EN=TRUE ist, dann wird der Baustein
                           erst bearbeitet.
      Request: Bool;       // Anfrage starten. Über eine positive Flanke
                           an diesem Eingang werden Sensoren gesucht.
      HW-PORT: HW_ANY;     // Hardware-Kennung der lokalen
                           Schnittstelle.
      Address: Char;       // Busadresse des HSI-Sensors.
END_VAR
```

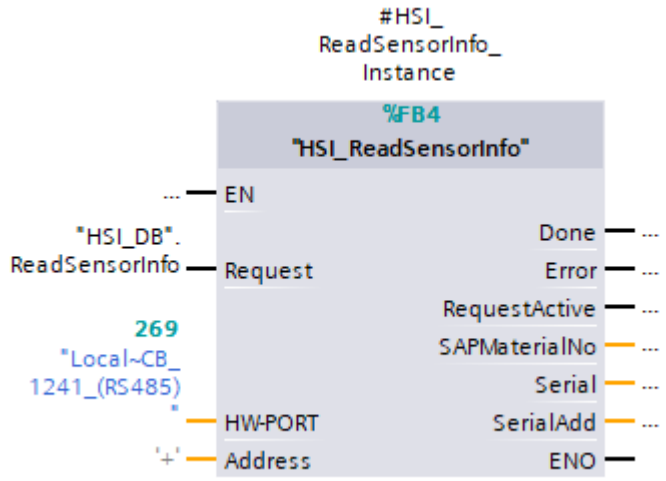
```
VAR_OUTPUT
      Done:          Bool;         // Dieser Ausgang bleibt solange auf
                                   FALSE, bis der Baustein seine
                                   Ausführung beendet hat.
      Error:         Bool;         // Dieser Ausgang wird auf TRUE
                                   geschaltet, wenn ein Fehler während
                                   der Befehlsübertragung auftritt.
      RequestActvie: Bool;         // Dieser Ausgang ist TRUE, wenn eine
                                   Anfrage läuft.

      SensorId:      STRING(14);   // Die Sensor-Id.
      RequestActvie: Bool;         // Freigabeausgang.


END_VAR
```

FUNCTION_BLOCK HSI_ReadSensorInfo

Die Sensorinfos beschreiben detailliert einen Sensor und seine Funktionen. Sie dienen einem Bediengerät dazu, herauszufinden welche Funktionen ein Sensor bietet, wie seine Daten strukturiert sind und noch einiges mehr. Während die bereits beschriebene SensorId mehr oder minder nur einen Namen darstellt, sind die Sensorinfos wesentlich detaillierter und strukturierter.




```
VAR_INPUT
      EN:       Bool;       // Wenn EN=TRUE ist, dann wird der Baustein
                            erst bearbeitet.
      Request:  Bool;       // Anfrage starten. Über eine positive Flanke
                            an diesem Eingang werden Sensoren gesucht.
      HW-PORT:  HW_ANY;     // Hardware-Kennung der lokalen
                            Schnittstelle.
      Address:  Char;       // Busadresse des HSI-Sensors.
END_VAR
```

```
VAR_OUTPUT
      Done:           Bool;         // Dieser Ausgang bleibt solange auf
                                    FALSE, bis der Baustein seine
                                    Ausführung beendet hat.
      Error:          Bool;         // Dieser Ausgang wird auf TRUE
                                    geschaltet, wenn ein Fehler während
                                    der Befehlsübertragung auftritt.
      RequestActvie:  Bool;         // Dieser Ausgang ist TRUE, wenn eine
                                    Anfrage läuft.
      SAPMaterialNo:  UDint;        // SAP Material Nummer.
      Serial:         UDInt;        // Die Serialnummer des Sensors.
      SerialAdd:      String(4);    // Der Serialnummer-Zusatz.
      ENO:            Bool;         // Freigabeausgang.
```

```
END_VAR
```

FUNCTION_BLOCK HSI_ReadSensorStatus

Der Sensorstatus dient dazu festzustellen, ob das angeschlossene Gerät betriebsbereit ist, oder in einen Fehlerzustand eingetreten ist. Der Sensorstatus hat folgenden Aufbau:

- 8-bit Statusbyte,
- 16-bit Statuscode bzw. Fehlercode (mit Vorzeichen)
- Optionaler Statustext

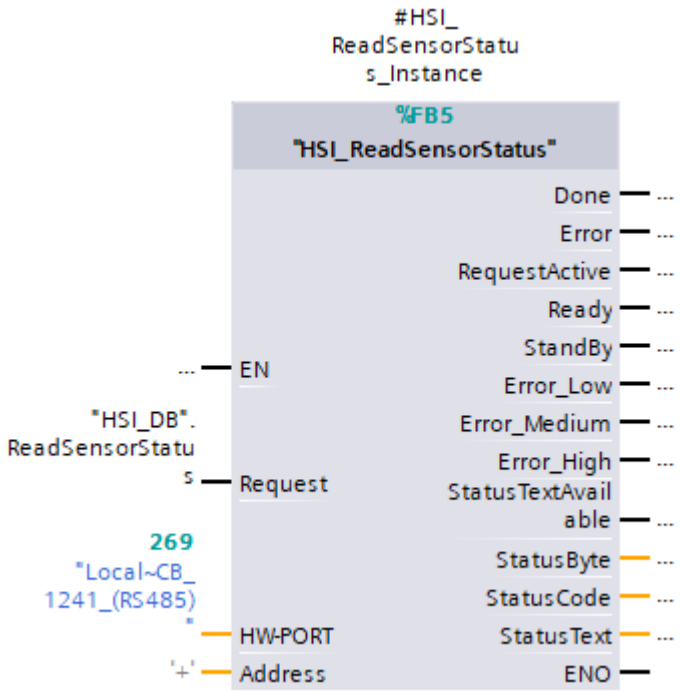
Das *Statusbyte* gibt den aktuellen Zustand des Gerätes an. Die einzelnen Zustände können über den folgenden Statuscode näher spezifiziert werden.

Folgende Werte für das Statusbyte sind definiert:

0:	Betriebsbereit	Kein aktiver Fehler vorhanden, Gerät ist betriebsbereit.
1:	Stand-by	Kein aktiver Fehler vorhanden, Gerät ist aber zur Zeit nicht betriebsbereit, eventuell sind einzelne Gerätefunktionen abgeschaltet, oder Gerät ist in einer Anlaufphase, etc.
2:	Leichter Fehler	Es ist ein leichter Fehler vorhanden, der quittiert werden kann.
3:	Mittlerer Fehler	Es ist ein mittelschwerer Fehler vorhanden, der durch Ein/Ausschalten eventuell behebbar ist.
4:	Schwerer Fehler	Es ist ein schwerer Fehler vorhanden, das Gerät muss zum Hersteller zurück.

Der *Statuscode* spezifiziert den aktuellen Zustand näher. Es ist ein 16-bit Wert. Die genaue Bedeutung ist von Gerät zu Gerät unterschiedlich. Der Anwender kann dann dem Handbuch nähere Infos zu dem Statuscode entnehmen.

Der *Statustext* ist optional und maximal 32 Zeichen lang. Er dient dazu, dass ein Bediengerät den Status eines Sensors im Klartext anzeigen kann.



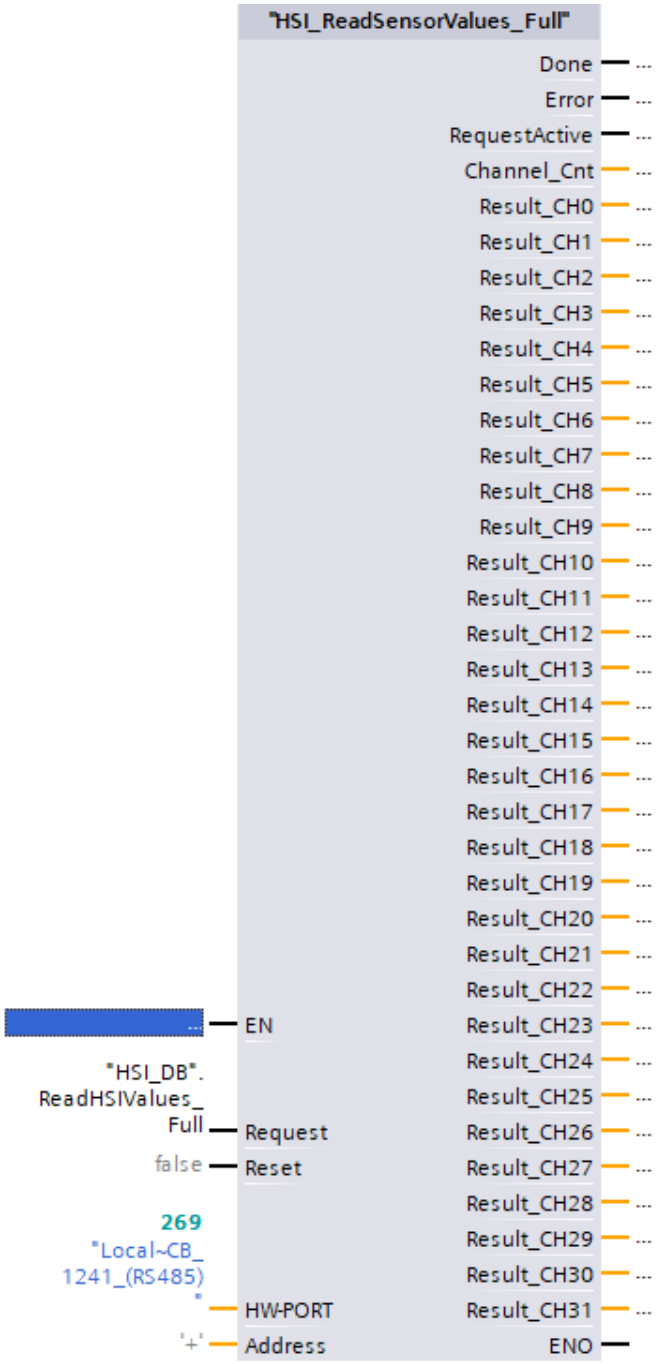

```
VAR_INPUT
      EN:       Bool;       // Wenn EN=TRUE ist, dann wird der Baustein
                            erst bearbeitet.
      Request:  Bool;       // Anfrage starten. Über eine positive Flanke
                            an diesem Eingang werden Sensoren gesucht.
      HW-PORT:  HW_ANY;     // Hardware-Kennung der lokalen
                            Schnittstelle.
      Address:  Char;       // Busadresse des HSI-Sensors.
END_VAR
```

```
VAR_OUTPUT
      Done:                 Bool;       // Dieser Ausgang bleibt solange
                                        auf FALSE, bis der Baustein seine
                                        Ausführung beendet hat.
      Error:                Bool;       // Dieser Ausgang wird auf TRUE
                                        geschaltet, wenn ein Fehler
                                        während der Befehlsübertragung
                                        auftritt.
      RequestActvie:        Bool;       // Dieser Ausgang ist TRUE, wenn
                                        eine Anfrage läuft.
      Ready:                Bool;       // Kein aktiver Fehler vorhanden,
                                        Gerät ist betriebsbereit.
      StandBy:              Bool;       // Gerät ist in einer Anlaufphase.
      Error_Low:            Bool;       // Es ist ein leichter Fehler
                                        vorhanden, der quittiert werden
                                        kann.
      Error_Medium:         Bool;       // Es ist ein mittelschwerer
                                        Fehler vorhanden, der durch
```

```
                                    Ein/Ausschalten eventuell behebbar
                                    ist.
      Error_High:          Bool;    // Es ist ein schwerer Fehler
                                    vorhanden, das Gerät muss zum
                                    Hersteller zurück.
      StatusTextAvailable: Bool;    // Dieser Ausgang wird auf TRUE
                                    gesetzt, wenn ein Status-Text
                                    vorhanden ist.
      StatusByte:        Byte;      // 8-bit StatusByte des Gerätes.
      StatusCode:        Int;       // 16-bit StatusCode des Gerätes.
      StatusText:        String(32); // Opt. Statustext des Gerätes.
      ENO:               Bool;      // Freigabeausgang.
END_VAR
```

FUNCTION_BLOCK HSI_ReadSenosorValues

Messwerte können von digitalen HSI-Sensoren, sogenannten Smart-Sensoren zu einem Bediengerät übertragen werden. Analoge HSI-Sensoren sind dazu nicht in der Lage.




```
VAR_INPUT
      EN:      Bool;       // Wenn EN=TRUE ist, dann wird der Baustein
                           erst bearbeitet.
      Request: Bool;       // Anfrage starten. Über eine positive Flanke
                           an diesem Eingang werden Sensoren gesucht.
      Reset:     Bool;     // Baustein im Fehlerfall zurücksetzen.
      HW_PORT: HW_ANY;     // Hardware-Kennung der lokalen
                           Schnittstelle.
      Address: Char;       // Busadresse des HSI-Sensors.
END_VAR
```

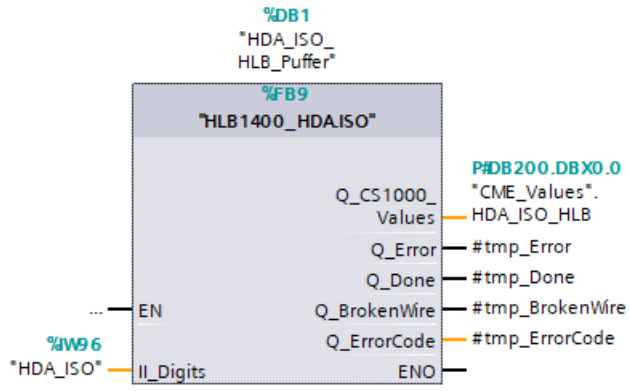
```
VAR_OUTPUT
      Done:           Bool;        // Dieser Ausgang bleibt solange auf
                                   FALSE, bis der Baustein seine
                                   Ausführung beendet hat.
      Error:          Bool;        // Dieser Ausgang wird auf TRUE
                                   geschaltet, wenn ein Fehler während
                                   der Befehlsübertragung auftritt.
      RequestActvie: Bool;         // Dieser Ausgang ist TRUE, wenn eine
                                   Anfrage läuft.
      Channel_Cnt:    USInt;       // Anzahl Messkanäle im Sensor.
      Result_CH0…Result_CH31:      // Aktuelle Messwerte oder „N/A“.
                      Real;
      ENO:            Bool;        // Freigabeausgang.
END_VAR
```

Projekt HLB_Analog_HDA_ISO_V13SP1 (Zielsystem S7 1200)

Das Projekt HLB_Analog_HDA_ISO_V13SP1 enthält den Programmbaustein MAIN. Mit diesem Baustein lässt sich die Kommunikation mit einem HLB14000 hestellen. Das Basissignal (HDA.ISO) wird ausgewertet. Dieser Programmbaustein wurde in der TIA Portal V13 entwickelt. Als Zielsystem ist S7 1200 definiert. Die Einrichtung der Schnittstellen liegen in der Verantwortung der Haupt-Anwendung (Kundenseite).

FUNCTION_BLOCK HLB1400_HDA.ISO

Der Baustein HLB1400_HDA.ISO liest die Messwerte von einem HLB1400 aus.




```
VAR_INPUT
      EN:         Bool;      // Wenn EN=TRUE ist, dann wird der Baustein
                             erst bearbeitet.
      II_Digits: Int;        // Eingangs in Digits (Basissignal)
```

```
END_VAR
```

```
VAR_OUTPUT
      CME_Values:           // Zuletzt ausgewertete Messwerte.
      HDA.ISOAuswertHLB;
      Q_Error:       Bool;  // Fehler während letzter Auswertung.
      Q_Done:        Bool;  // Ablauf einer Messerfassung abgeschlossen.
      Q_BrokenWire:  Bool;  // Leitung gebrochen.
      Q_ErrorCode:   Int;   // Fehlercode während letzter Auswertung.
      ENO:           Bool;  // Freigabeausgang.
END_VAR
```

Die Rückgabe enthält die Messwerte in folgender Datenstruktur:

```
TYPE HDA.ISOAuswertHLB:
STRUCT
     Leitfaehigkeit:               Real;   // Messwerte eines HLB1400
                                           folgen…
     Rel.Leitfaehigkeit:           Real;
     Dielektrizitätskonstante:     Real;
     Rel.Dielektrizitätskonstante: Real;
     Saettigung:                   Real;
     Temperatur:                   Real;
     STATE:                        Real;
END_STRUCT
END_TYPE
```

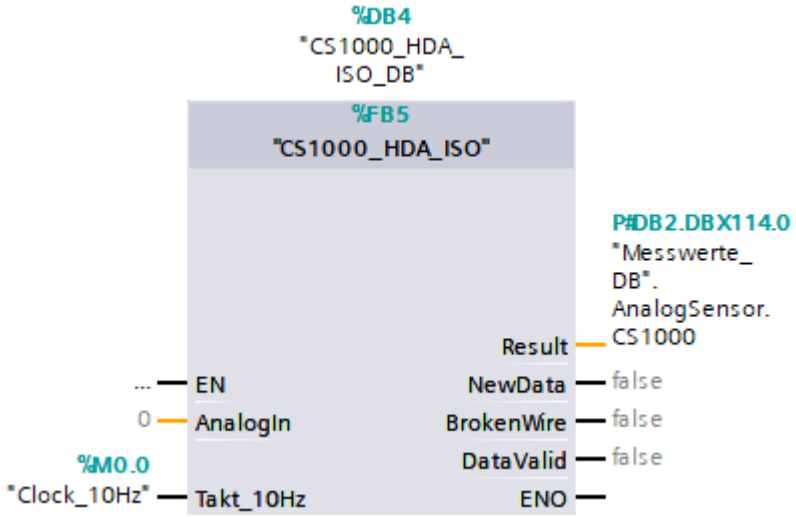
Projekt TIA_HYDAC_Sensoren (Zielsystem S7 1200)

Das Projekt TIA_HYDAC_Sensoren enthält verschiedene Programmbausteine, die zur Kommunikation mit bestimmten Geräte ermöglicht. Das Baustein „CS1000_HDA_ISO_DB“ lässt sich die Kommunikation mit einem CS1000 herstellen. Das Basissignal wird ausgewertet. Die Ethernet Kommunikation mit CSI-C-11 ist mithilfe von dem Baustein “CSI_C11_Main_DB“ realiesiert. Dabei werden die Input Register vom CSI-C-11 per ModbusTCP ausgelesen.

Dieser Programmbaustein wurde in der TIA Portal V15SP1 entwickelt. Als Zielsystem ist S7 1200 definiert. Die Einrichtung der Schnittstellen liegen in der Verantwortung der Haupt-Anwendung (Kundenseite).

FUNCTION_BLOCK CS1000_HDA.ISO

Der Baustein CS1000_HDA.ISO liest die Messwerte von einem CS1000 aus.




```
VAR_INPUT
      EN:         Bool;  // Wenn EN=TRUE ist, dann wird der Baustein
                         erst bearbeitet.
      AnalogIn:   INT;   // Analogeingang (Rohwert bzw. Basissignal)
      Takt_10Hz:  Bool;  // Tankmerker 10Hz.
END_VAR
```

```
VAR_OUTPUT
      Result        : STRUCT;      // Zuletzt ausgewertete Messwerte.
      NewData       : BOOL;        // Neue Daten stehen bereit.
      BrokenWire    : BOOL;        // Drahtbruch erkannt.
      DataValid     : BOOL;        // Daten sind gültig.
      ENO:           Bool;         // Freigabeausgang.
END_VAR
```

Die Rückgabe enthält die Messwerte in folgender Datenstruktur:

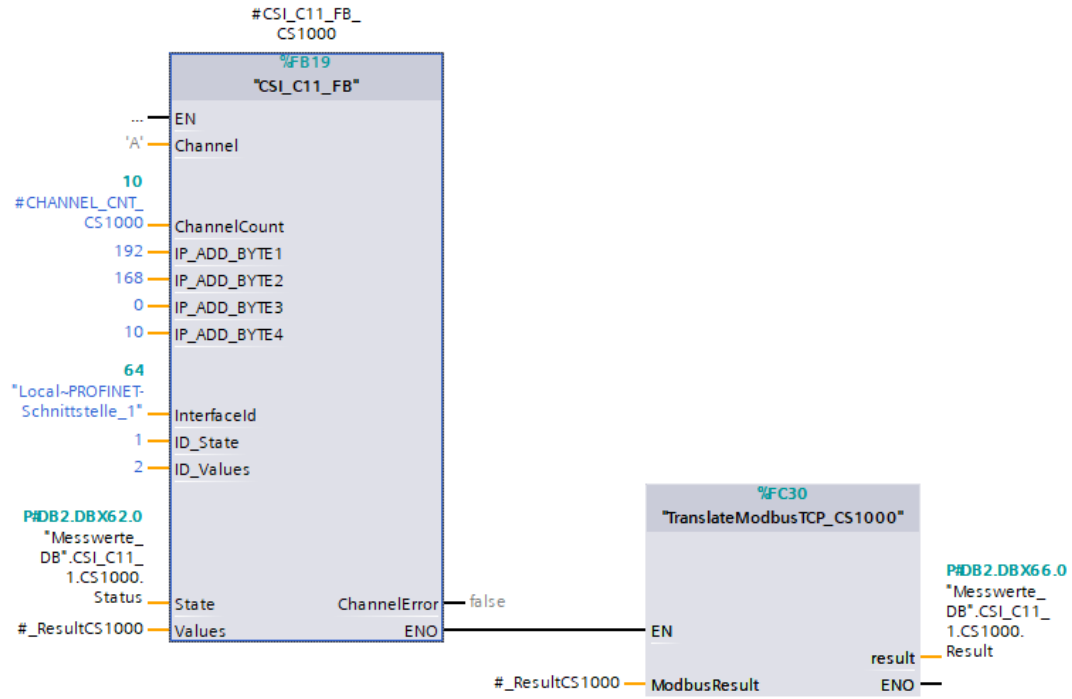
```
TYPE Result:
STRUCT
      ISO_4:      Real;     // Messwerte von CS1000 (siehe Dokumentation)
      ISO_6:      Real;
      ISO_14:     Real;
      SAE_A:      Real;
      SAE_B:      Real;
      SAE_C:      Real;
      SAE_D:      Real;
      Temp:       Real;
      FlowState:  Int;
      State:      Int;
      Drive:      Int;
      StatusWord: Struct;   // Beschreibung StatusWord folgt…
END_STRUCT
END_TYPE
```

Die Datenstruktur „StatusWord“:

```
TYPE StatusWord:
STRUCT
      BrokenWire:          Bool;     // Drahtbruch erkannt.
      OK:                  Bool;     // Auswertung OK.
      Fault:               Bool;     // Auswertung fehlgeschlagen.
      Flow2LOW:            Bool;     // Durchfluss zu niedrig.
      ISO<9<8<7:           Bool;     // Wert Ausserhalb Messbereich.
      No_MeasurementValue: Bool;     // Nummer des Messwertes
      Reserve6:            Bool;
      Reserve7:            Bool;
      Reserve8:            Bool;
      Reserve10:           Bool;
      Reserve11:           Bool;
      Reserve12:           Bool;
      Reserve13:           Bool;
      Reserve14:           Bool;
      Reserve15:           Bool;
END_STRUCT
END_TYPE
```

FUNCTION_BLOCK CSI_C11_FB

Der Baustein CSI_C11_FB liest die Messwerte von einem CS1000 und AS1000 über die ModbusTCP-Schnittstelle eines CSI-C-11. Es werden alle Input Register ausgewertet.




```
VAR_INPUT
      EN:           Bool;       // Wenn EN=TRUE ist, dann wird der
                                Baustein erst bearbeitet.
      Channel:      Char;       // ‘A‘ bedeutet Kanal A, ‘B‘ – Kanal B.
      ChannelCount: INT;        // Anzahl auszulesender Kanälen. (10 bei
                                CS1000, 2 bei AS1000, 7 bei MCS1000…)
      IP_ADD_BYTE1: Byte;       // Erstes Byte IP Adresse
      IP_ADD_BYTE2: Byte;       // Zweites Byte IP Adresse
      IP_ADD_BYTE3: Byte;       // Drittes Byte IP Adresse
      IP_ADD_BYTE4: Byte;       // Viertes Byte IP Adresse
      InterfaceID: HW_ANY;      // Hardware-Kennung der lokalen
                                Schnittstelle.
      ID_State:    CONN_OUC;    // Verbindungsstatus.
      ID_Values:   CONN_OUC;    // Verbindungsinformation.
END_VAR
```

```
VAR_OUTPUT
```

```
      State: T_ModbusTCP_Status;     // Struktur mit Statusvariablen.
      Values: Array[0..31] of        // Word-Array mit aktuellen
      Word;                          Messwerten.
      ChannelError: Bool;            // Falscher Messkanal ausgewählt.
      ENO:           Bool;           // Freigabeausgang.
END_VAR
```

Die Rückgabe enthält die Messwerte in folgender Datenstruktur:

```
TYPE T_ModbusTCP_Status:
STRUCT
      StatusCode: Word;     // Allgemwiner Zustand des Sensors
      ErrorCode:  Word;     // Gerätespezifischer Statuscode
END_STRUCT
END_TYPE
```

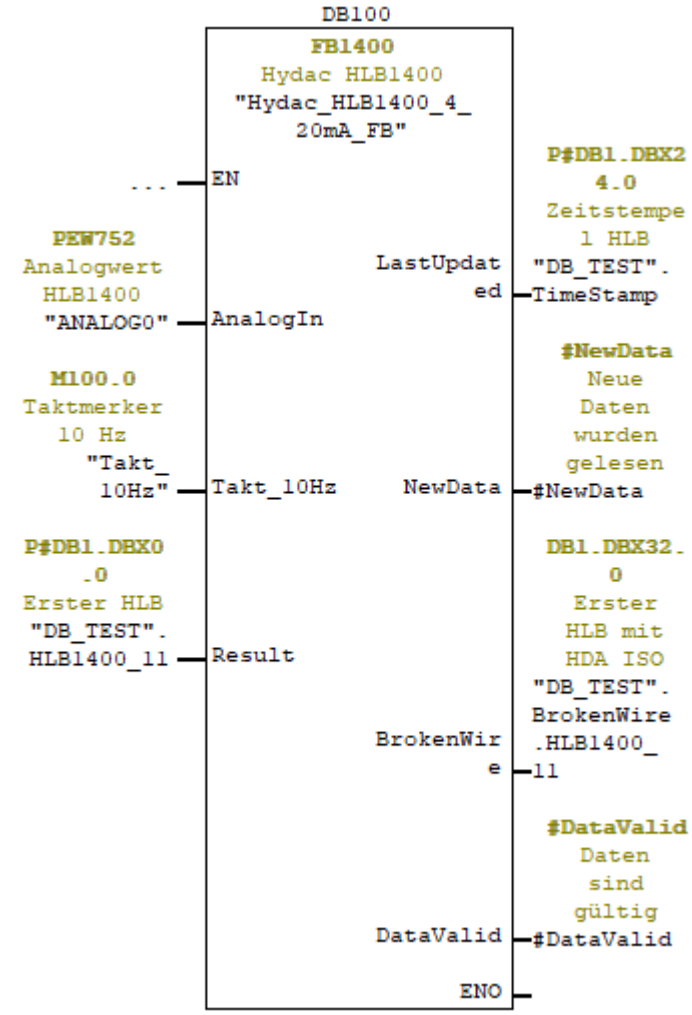
Projekt HLB400_Analog (Zielsystem S7 300)

Das Projekt HLB400_Analog enthält das Programmbaustein FB1400, das die analoge Kommunikation mit einem HLB1400 ermöglicht. Die Signalbeschreibung bzw. Kodierung finden Sie in der Dokumentation zum HLB1400.

Dieser Programmbaustein wurde in der SIMATIC Manager© Step 7 Version 5.6.2.0 entwickelt. Als Zielsystem ist S7 300 definiert. Die Einrichtung der Schnittstellen liegen in der Verantwortung der Haupt-Anwendung (Kundenseite).

FUNCTION_BLOCK FB1400 Hydac_HLB1400_4_20mA_FB

Der Baustein FB1400 (oder seine ungeschützte Version FB1401) liest die Messwerte von einem HLB1400 über analoge Schnittstelle 4..20mA und wertet die aktuellen Messwerte aus.




```
VAR_INPUT
   EN:        Bool;          // Wenn EN=TRUE ist, dann wird der
                             Baustein erst bearbeitet.
   AnalogIn:  Int;           // Analogwert HLB1400.
   Takt_10Hz: Bool;          // Tankmerker 10Hz.
```

```
    Result:    T_HydacHLB1400;   // Die Struktur für zuletzt gelesenen
                                 Daten.
END_VAR
```

```
VAR_OUTPUT
      LastUpdated: DATE_AND_TIME;    // Zeitstempel der zuletzt gelesenen
                                     Daten.
      NewData:     Bool;             // Neue Daten stehen bereit.
      BrokenWire:  Bool;             // Drahtbruch erkannt.
      DataValid:   Bool;             // Daten sind gültig.
      Result:      T_HydacHLB1400;   // Zuletzt gelesene Daten.
      ENO:         Bool;             // Freigabeausgang.
END_VAR
```

Die Rückgabe enthält die Messwerte in folgenden Datenstrukturen:

```
TYPE T_HydacHLB1400:
STRUCT
    Condition:  Real;         // Absolutwert der el. Leitfähigkeit    0 ..
                              100 [nS/m] (1 Nachkommastelle)
    Change_Condition: Int;    // relative Änderung der el. Leitfähigkeit
                              -100 .. 200 [%]
    DC: Real;                 // Absolutwert der DK    1 .. 10 (2
                              Nachkommastellen)
    Change_DC: Int;           // rel. Änderung der DK    -30% ... +30 [%]
    Saturation: Int;          // Sättigungsgrad    0% ... +100 [%]
    Temperatur: Real;         // Temperatur    -25°C... +100 [°C] (1
                              Nachkommastelle)
    State: Real;              // Statussignal mA
    StatusWord: Struct;       // Statusword
END_STRUCT
END_TYPE
```

Die Statusinformationen werder in der Struktur StatusWord (16 boolearische Werte) zur Verfügung gestellt:

```
TYPE StatusWord:
STRUCT
    Work_Phase: Bool;               // Betriebsphase aktiv
    Homogenization_Phase: Bool;     // Homogenisierungsphase aktiv
    Orientation_Phase: Bool;        // Orientierungsphase aktiv
```

```
    Observation_Phase: Bool;        // Wartephase aktiv
    State_Error: Bool;              // Statusfehler
    RESERVE05: Bool;                // Reserve
    RESERVE06: Bool;                // Reserve
    RESERVE07: Bool;                // Reserve
    SP1: Bool;                      // Zustand Schaltpunkt 1
    SP2: Bool;                      // Zustand Schaltpunkt 2
    RESERVE12: Bool;                // Reserve
    RESERVE13: Bool;                // Reserve
    OutOfRange: Bool;               // Betriebsbereich verlassen
    RESERVE15: Bool;                // Reserve
    RESERVE16: Bool;                // Reserve
    RESERVE17: Bool;                // Reserve
END_STRUCT
END_TYPE
```

Projekt S7312_RS (Zielsystem S7 300/1200V2)

Das Projekt S7312_RS enthält die Programmbausteine mit einzelnen Funktionsblöcken zur digitalen Kommunikation mit den Hydac-Smartsensoren. Dieser Programmbaustein wurde in der SIMATIC Manager© Step 7 Version 5.6.2.0 entwickelt. Als Zielsystem sind S7 300/1200V2 definiert. Die Schnittstellen liegen in der Verantwortung der Haupt-Anwendung (Kundenseite).

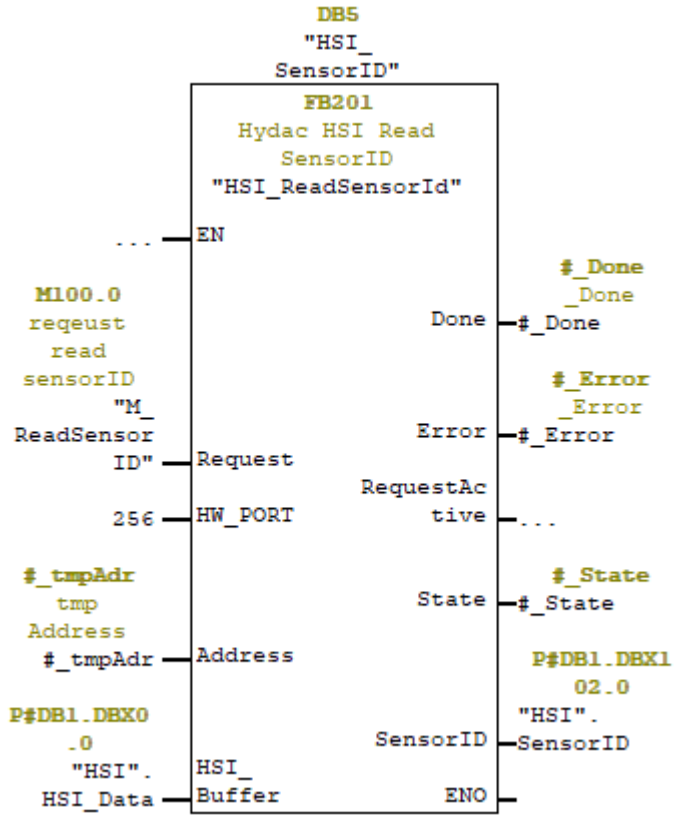
FUNCTION_BLOCK FB201 HSI_ReadSensorID

Der Baustein Hsi_ReadSensorId liest die SensorId von einem Hydac-Gerät aus. Die SensorId dient dazu festzustellen, welches Gerät angeschlossen ist. Die SensorId besteht aus 14 Zeichen mit folgendem Aufbau:

- 8 Zeichen Gerätebezeichnung,

- 6 Zeichen Versionsbezeichnung.

Die Versionsbezeichnung hat folgenden Aufbau: Vxx.yy, wobei xx für die Versionsnummer und yy für die Releasenummer steht. Beide Nummern sind 2-stellig, d.h. bei Nummern <10 wird eine führende Null vorangestellt.




```
VAR_INPUT
   EN:          Bool;     // Wenn EN=TRUE ist, dann wird der Baustein
                          erst bearbeitet.
```

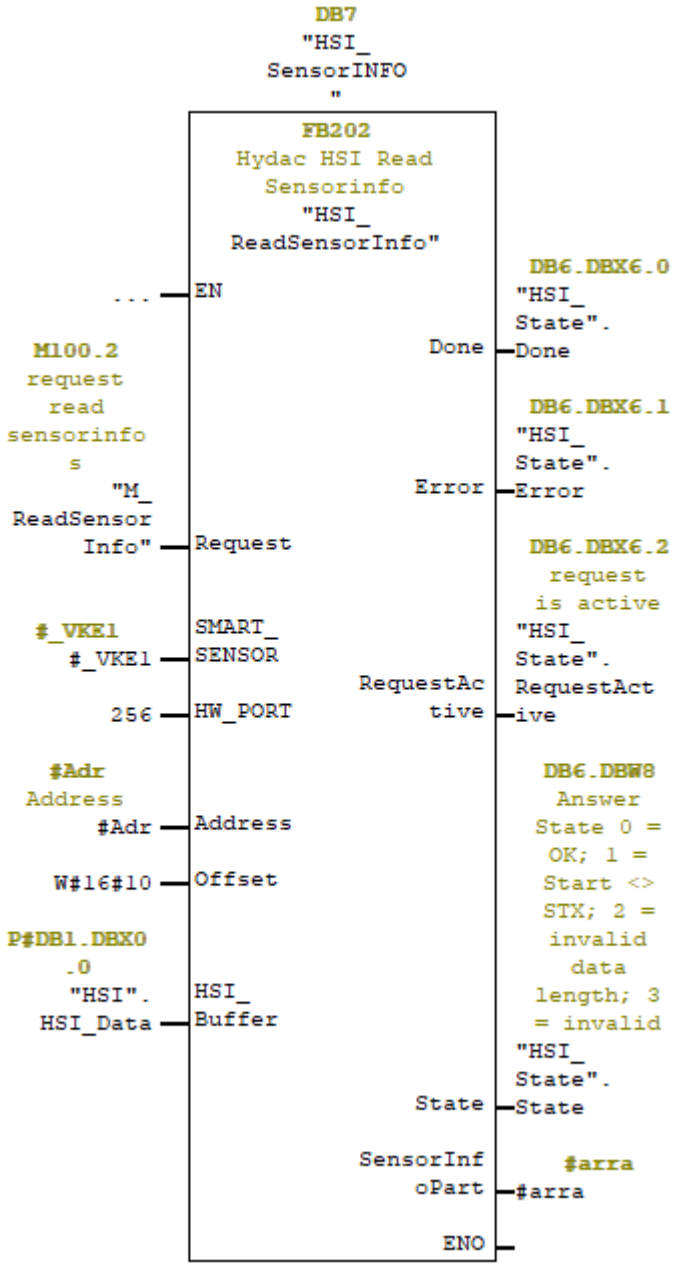
```
    Request:    Bool;    // Anfrage starten. Über eine positive Flanke
                         an diesem Eingang werden Sensoren gesucht.
    HW-PORT:    Int;     // Hardware-Kennung der lokalen Schnittstelle.
    Address:    Char;    // Busadresse des HSI-Sensors.
END_VAR
```

```
VAR_OUTPUT
    Done:          Bool;     // Dieser Ausgang bleibt solange auf FALSE,
                             bis der Baustein seine Ausführung beendet
                             hat.
    Error:         Bool;     // Dieser Ausgang wird auf TRUE geschaltet,
                             wenn ein Fehler während der
                             Befehlsübertragung auftritt.
    RequestActvie: Bool;     // Dieser Ausgang ist TRUE, wenn eine
                             Anfrage läuft.
    State:         Int;      // Antwortstatus 0 = OK, sonst
                             Protokollfehler.
    SensorID: Array          // Rückgabe SensorID.
    [0..15] Of Char
    ENO:                     // Freigabeausgang.
END_VAR
```

```
VAR_IN_OUT
    HSI_Buffer: Array[0..10] of Byte;     // Datenpuffer für HSI-
                                          Kommunikation
END_VAR
```

FUNCTION_BLOCK FB202 HSI_ReadSensorInfo

Die Sensorinfos beschreiben detailliert einen Sensor und seine Funktionen. Sie dienen einem Bediengerät dazu, herauszufinden welche Funktionen ein Sensor bietet, wie seine Daten strukturiert sind und noch einiges mehr. Während die bereits beschriebene SensorId mehr oder minder nur einen Namen darstellt, sind die Sensorinfos wesentlich detaillierter und strukturierter.




```
VAR_INPUT
    EN:             Bool;    // Wenn EN=TRUE ist, dann wird der Baustein
                             erst bearbeitet.
    Request:        Bool;    // Anfrage starten. Über eine positive Flanke
                             an diesem Eingang werden Sensoren gesucht.
    SMART_SENSOR:   Bool;    // TRUE: HSI SmartSensor, sonst HSI Analog
                             Sensor
    HW-PORT:        Int;     // Hardware-Kennung der lokalen Schnittstelle.
    Address:        Char;    // Busadresse des HSI-Sensors.
    Offset:         Word;    // HSI Offset Adresse
END_VAR
```

```
VAR_OUTPUT
    Done:           Bool;     // Dieser Ausgang bleibt solange auf FALSE,
                              bis der Baustein seine Ausführung beendet
                              hat.
    Error:          Bool;     // Dieser Ausgang wird auf TRUE geschaltet,
                              wenn ein Fehler während der
                              Befehlsübertragung auftritt.
    RequestActvie: Bool;      // Dieser Ausgang ist TRUE, wenn eine
                              Anfrage läuft.
    State:          Int;      // Antwortstatus 0 = OK, sonst
                              Protokollfehler.
    SensorInfoPart:           // (Nächstes) Teil der Sensorinfo
    Array[0..15] of Byte;
    ENO:                      // Freigabeausgang.
END_VAR
```

```
VAR_IN_OUT
    HSI_Buffer: Array[0..10] of       // Datenpuffer für HSI-
    Byte;                             Kommunikation
END_VAR
```

FUNCTION_BLOCK FB204 HSI_ReadSensorStatus

Der Sensorstatus dient dazu festzustellen, ob das angeschlossene Gerät betriebsbereit ist, oder in einen Fehlerzustand eingetreten ist. Der Sensorstatus hat folgenden Aufbau:

- 8-bit Statusbyte,
- 16-bit Statuscode bzw. Fehlercode (mit Vorzeichen)
- Optionaler Statustext

Das *Statusbyte* gibt den aktuellen Zustand des Gerätes an. Die einzelnen Zustände können über den folgenden Statuscode näher spezifiziert werden.

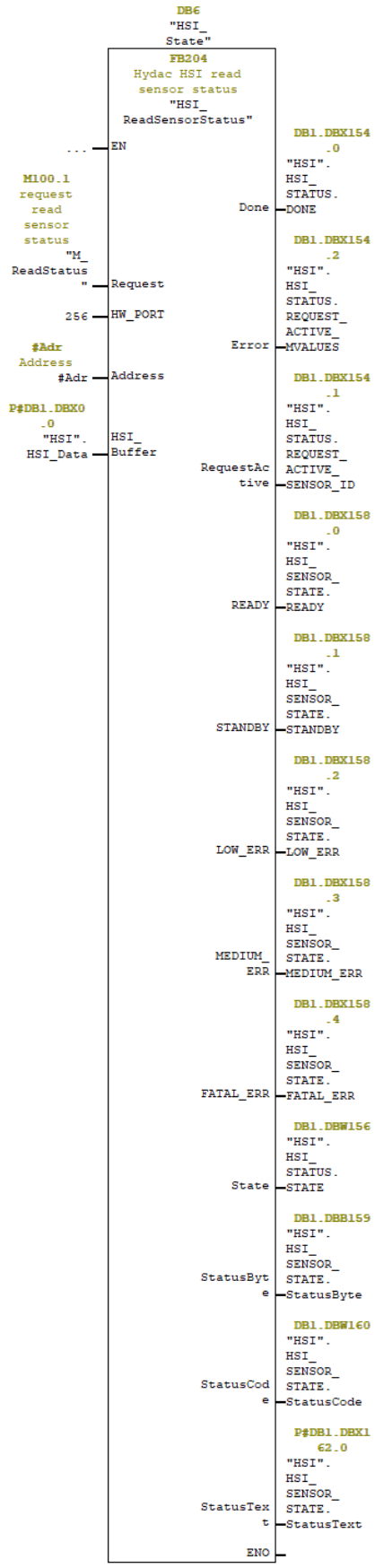
Folgende Werte für das Statusbyte sind definiert:

0:	Betriebsbereit	Kein aktiver Fehler vorhanden, Gerät ist betriebsbereit.
1:	Stand-by	Kein aktiver Fehler vorhanden, Gerät ist aber zur Zeit nicht betriebsbereit, eventuell sind einzelne Gerätefunktionen abgeschaltet, oder Gerät ist in einer Anlaufphase, etc.
2:	Leichter Fehler	Es ist ein leichter Fehler vorhanden, der quittiert werden kann.

3:	Mittlerer Fehler	Es ist ein mittelschwerer Fehler vorhanden, der durch Ein/Ausschalten eventuell behebbar ist.
4:	Schwerer Fehler	Es ist ein schwerer Fehler vorhanden, das Gerät muss zum Hersteller zurück.

Der *Statuscode* spezifiziert den aktuellen Zustand näher. Es ist ein 16-bit Wert. Die genaue Bedeutung ist von Gerät zu Gerät unterschiedlich. Der Anwender kann dann dem Handbuch nähere Infos zu dem Statuscode entnehmen.

Der *Statustext* ist optional und maximal 32 Zeichen lang. Er dient dazu, dass ein Bediengerät den Status eines Sensors im Klartext anzeigen kann.


DB6
"HSI_
State"
FB204
Hydac HSI read
sensor status
"HSI_
ReadSensorStatus"
... EN
M100.1
request
read
sensor
status
"M_
ReadStatus
" Request
256 HW_PORT
#Adr
Address
#Adr Address
P#DB1.DBX0
.0
"HSI".
HSI_Data HSI_
Buffer
Done DB1.DBX154
.0
"HSI".
HSI_
STATUS.
DONE
DB1.DBX154
.2
"HSI".
HSI_
STATUS.
REQUEST_
ACTIVE_
Error MVALUES
DB1.DBX154
.1
"HSI".
HSI_
STATUS.
REQUEST_
ACTIVE_
RequestAc
tive SENSOR_ID
DB1.DBX158
.0
"HSI".
HSI_
SENSOR_
STATE.
READY READY
DB1.DBX158
.1
"HSI".
HSI_
SENSOR_
STATE.
STANDBY STANDBY
DB1.DBX158
.2
"HSI".
HSI_
SENSOR_
STATE.
LOW_ERR LOW_ERR
DB1.DBX158
.3
"HSI".
HSI_
SENSOR_
STATE.
MEDIUM_
ERR MEDIUM_ERR
DB1.DBX158
.4
"HSI".
HSI_
SENSOR_
STATE.
FATAL_ERR FATAL_ERR
DB1.DBW156
"HSI".
HSI_
STATUS.
State STATE
DB1.DBB159
"HSI".
HSI_
SENSOR_
STATE.
StatusByt
e StatusByte
DB1.DBW160
"HSI".
HSI_
SENSOR_
STATE.
StatusCod
e StatusCode
P#DB1.DBX1
62.0
"HSI".
HSI_
SENSOR_
STATE.
StatusTex
t StatusText
ENO

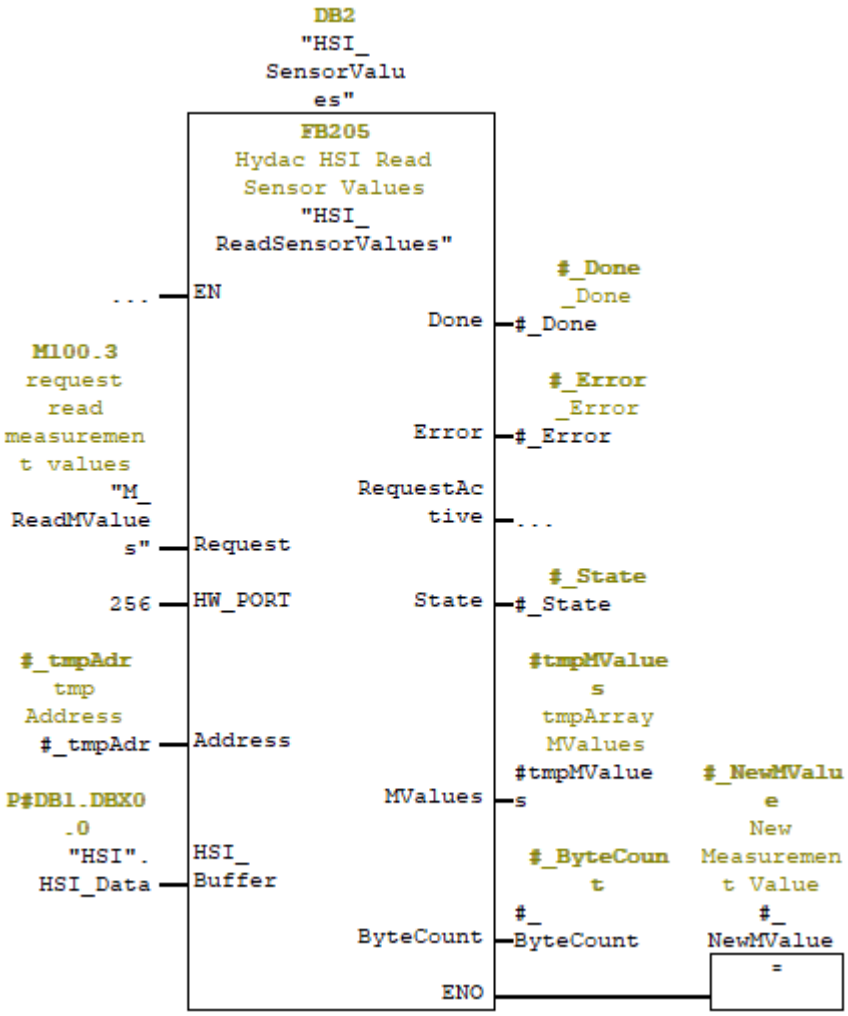
```
VAR_INPUT
    EN:           Bool;    // Wenn EN=TRUE ist, dann wird der Baustein
                           erst bearbeitet.
    Request:      Bool;    // Anfrage starten. Über eine positive Flanke
                           an diesem Eingang werden Sensoren gesucht.
    HW-PORT:      Int;     // Hardware-Kennung der lokalen Schnittstelle.
    Address:      Char;    // Busadresse des HSI-Sensors.
END_VAR
```

```
VAR_OUTPUT
    Done:          Bool;     // Dieser Ausgang bleibt solange auf FALSE,
                             bis der Baustein seine Ausführung beendet
                             hat.
    Error:         Bool;     // Dieser Ausgang wird auf TRUE geschaltet,
                             wenn ein Fehler während der
                             Befehlsübertragung auftritt.
    RequestActvie: Bool;     // Dieser Ausgang ist TRUE, wenn eine
                             Anfrage läuft.
    READY:         Bool;     // Status READY.
    STANDBY:       Bool;     // Status Stand-By.
    LOW_ERR:       Bool;     // Leichter Fehler.
    MEDIUM_ERR:    Bool;     // Mittlerer Fehler.
    FATAL_ERR:     Bool;     // Schwerer Fehler.
    State:         Int;      // Antwortstatus 0 = OK, sonst
                             Protokollfehler.
    StatusByte:    Byte;     // HSI-StatusByte.
    StatusCode:    Word;     // HSI-StatusCode.
    StatusText: Array        // HSI-Statustext.
    [0..31] Of Char;
    ENO:                     // Freigabeausgang.
END_VAR
```

```
VAR_IN_OUT
    HSI_Buffer: Array[0..10] of Byte;    // Datenpuffer für HSI-
                                         Kommunikation
END_VAR
```

FUNCTION_BLOCK FB205 HSI_ReadSensorValues

Messwerte können von digitalen HSI-Sensoren, sogenannten Smart-Sensoren zu einem Bediengerät übertragen werden. Analoge HSI-Sensoren sind dazu nicht in der Lage.




```
VAR_INPUT
    EN:             Bool;    // Wenn EN=TRUE ist, dann wird der Baustein
                             erst bearbeitet.
    Request:        Bool;    // Anfrage starten. Über eine positive Flanke
                             an diesem Eingang werden Sensoren gesucht.
    HW-PORT:        Int;     // Hardware-Kennung der lokalen Schnittstelle.
    Address:        Char;    // Busadresse des HSI-Sensors.
END_VAR
```

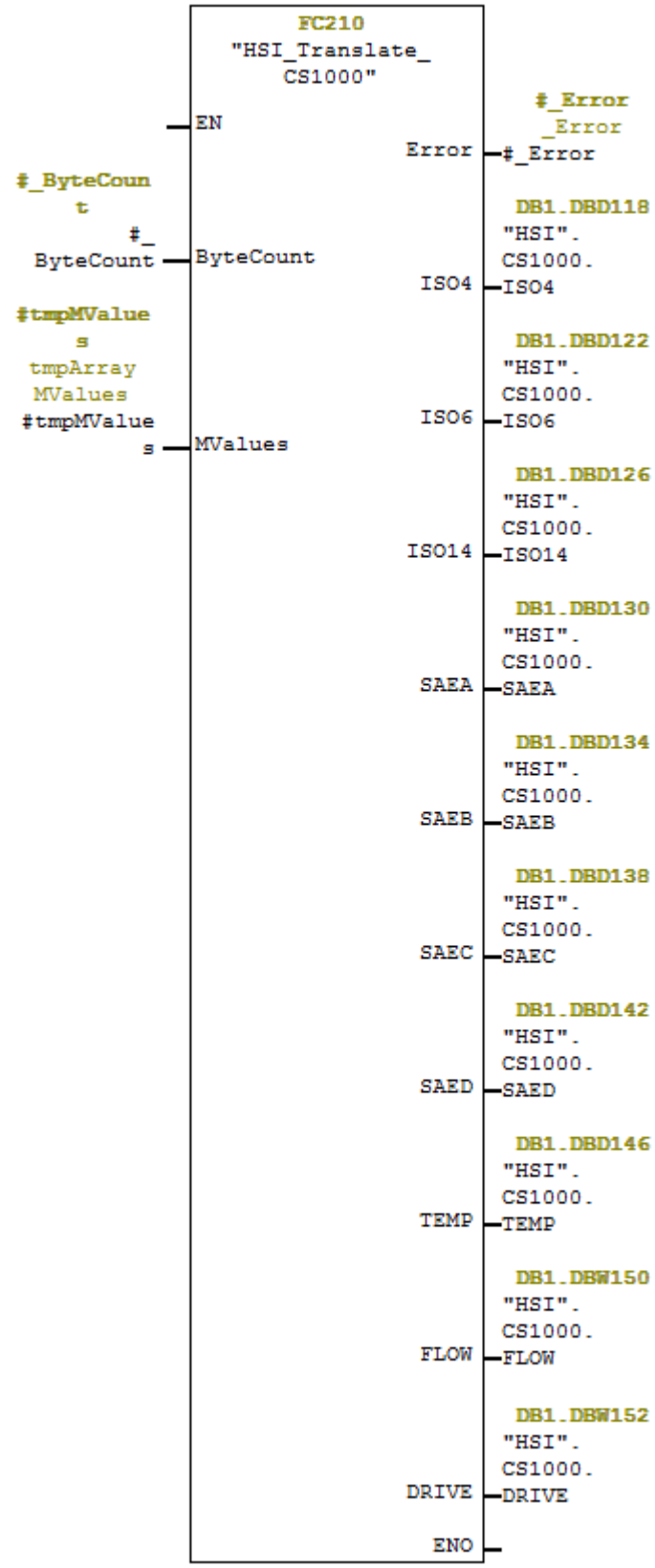
```
VAR_OUTPUT
    Done:           Bool;    // Dieser Ausgang bleibt solange auf FALSE,
                             bis der Baustein seine Ausführung beendet
                             hat.
```

```
    Error:          Bool;     // Dieser Ausgang wird auf TRUE geschaltet,
                              wenn ein Fehler während der
                              Befehlsübertragung auftritt.
    RequestActvie: Bool;      // Dieser Ausgang ist TRUE, wenn eine
                              Anfrage läuft.
    State:          Int;      // Antwortstatus 0 = OK, sonst
                              Protokollfehler.
    MValues:        Array     // Array mit Messwerten.
     [0..100] Of Byte;
    ByteCount:      Int;      // Anzahl belegter Bytes im Array mit
                              Messwerten
    ENO:                      // Freigabeausgang.
END_VAR
```

```
VAR_IN_OUT
    HSI_Buffer: Array[0..10] of Byte;    // Datenpuffer für HSI-
                                         Kommunikation
END_VAR
```

FUNCTION_BLOCK FB210 HSI_Translate_CS1000

Der Array mit Messwerten am Ausgang des Funktionsblockes " FB205 HSI_ReadSensorValues" werden mit Hilfe vom Baustein "HSI_Translate_CS1000" ausgewertet.




```
VAR_INPUT
   EN:          Bool;              // Wenn EN=TRUE ist, dann wird der
                                   Baustein erst bearbeitet.
```

```
    ByteCount:    Int;                // Anzahl Bytes im Array.
    MValues: Array [0..100] Of Byte;  // Array mit Messwerten.
END_VAR
```

```
VAR_OUTPUT
    Error:         Bool;     // Dieser Ausgang wird auf TRUE geschaltet,
                             wenn ein Fehler während der
                             Befehlsübertragung auftritt.
    ISO4:  Real;             // Aktueller Messwert von CS1000: ISO 4
    ISO6:  Real;             // Aktueller Messwert von CS1000: ISO 6
    ISO14: Real;             // Aktueller Messwert von CS1000: ISO 14
    SAEA:  Real;             // Aktueller Messwert von CS1000: SAE A
    SAEB:  Real;             // Aktueller Messwert von CS1000: SAE B
    SAEC:  Real;             // Aktueller Messwert von CS1000: SAE C
    SAED:  Real;             // Aktueller Messwert von CS1000: SAE D
    TEMP:  Real;             // Aktueller Messwert von CS1000: Temperatur
    FLOW:  Int;              // Aktueller Messwert von CS1000: Durchfluss
    DRIVE: Int;              // Aktueller Messwert von CS1000: Drive
    ENO:                     // Freigabeausgang.
END_VAR
```

Übersicht Messkanäle

In der nachfolgenden Tabelle wird ein Übersicht der Messkanäle von unterschiedlichen HYDAC – Sensoren dargestellt.

Für die Geräte / Sensoren: HLB1000, CMU1000, HMG3000 entnehmen Sie die Messkanal Übersicht aus der jeweiligen Bedienungsanleitung.

Messkanäle FCU 2000 Serie

FCU 20xx						
	Partikelzahlen		NAS/SAE Klasse		ISO Code	
Kanal 1	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO >5 µm	[0…25]
Kanal 2	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO >15 µm	[0…25]
Kanal 3	25-50 µm	[0…129600]	NAS 25-50 µm	[-1…15]	ISO >25 µm	[0…25]
Kanal 4	>50 µm	[0…23040]	NAS >50 µm	[-1…15]	ISO >50 µm	[0…25]
Kanal 5	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
-	-	-	-	-	-	-

FCU 21xx						
	Partikelzahlen		NAS/SAE Klasse		ISO Code	
Kanal 1	2-5 µm	[0…20484000]	NAS 2-5 µm	[-1…15]	ISO >2 µm	[0…25]
Kanal 2	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO >5 µm	[0…25]
Kanal 3	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO >15 µm	[0…25]
Kanal 4	>25 µm	[0…129600]	NAS >25 µm	[-1…15]	ISO >25 µm	[0…25]
Kanal 5	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
-	-	-	-	-	-	-

FCU 22xx						
	Partikelzahlen		NAS/SAE Klasse		ISO Code	
Kanal 1	> 4 µm	[0…3200000]	SAE A	[-2…15]	ISO >4 µm	[0…25]
Kanal 2	> 6 µm	[0…1250000]	SAE B	[-2…15]	ISO >6 µm	[0…25]
Kanal 3	> 14 µm	[0…222000]	SAE C	[-2…15]	ISO >14 µm	[0…25]
Kanal 4	> 21 µm	[0…39200]	SAE D	[-2…15]	ISO >21 µm	[0…25]

Kanal 7	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
-	-	-	-	-	-	-

Messkanäle FCU 8000 Serie

FCU 81xx						
	Partikelzahlen		NAS/SAE Klassen		ISO Code	
Kanal 1	2-5 µm	[0…20484000]	NAS 2-5 µm	[-1…15]	ISO > 2 µm	[0…25]
Kanal 2	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO > 5 µm	[0…25]
Kanal 3	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO > 15 µm	[0…25]
Kanal 4	25-50 µm	[0…129600]	NAS 25-50 µm	[-1…15]	ISO > 25 µm	[0…25]
Kanal 5	50-100 µm	[0…23040]	NAS 50-100 µm	[-1…15]	ISO > 50 µm	[0…25]
Kanal 6	>100 µm	[0…4096]	NAS > 100 µm	[-1…15]	ISO > 100 µm	[0…25]
Kanal 7	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]

FCU 82xx						
	Partikelzahlen		NAS/SAE Klassen		ISO Code	
Kanal 1	> 4 µm	[0…3200000]	SAE A	[-2…15]	ISO > 4 µm	[0…25]
Kanal 2	> 6 µm	[0…1250000]	SAE B	[-2…15]	ISO > 6 µm	[0…25]
Kanal 3	>14 µm	[0…222000]	SAE C	[-2…15]	ISO > 14 µm	[0…25]
Kanal 4	> 21 µm	[0…39200]	SAE D	[-2…15]	ISO > 21 µm	[0…25]
Kanal 5	> 38 µm	[0…6780]	SAE E	[-2…15]	ISO > 38 µm	[0…25]
Kanal 6	> 70 µm	[0…1020]	SAE F	[-2…15]	ISO > 70 µm	[0…25]
Kanal 7	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]

Messkanäle CS 1000 Serie

CS 12xx			
Kanal 1	ISO >4	[9…25]	
Kanal 2	ISO >6	[8…24]	
Kanal 3	ISO >14	[7…23]	
Kanal 4	SAE A	[0…14]	
Kanal 5	SAE B	[0…14]	
Kanal 6	SAE C	[0…14]	
Kanal 7	SAE D	[0…14]	
Kanal 8	Temp	[-60…150]	°C
Kanal 9	Flow	[30…300]	ml/min
Kanal 10	Drive	[0…100]	%

CS 13xx			
Kanal 1	ISO >2	[9…25]	
Kanal 2	ISO >5	[8…24]	
Kanal 3	ISO >15	[7…23]	
Kanal 4	NAS 2-5	[0…14]	
Kanal 5	NAS 5-15	[0…14]	
Kanal 6	NAS 15-25	[0…14]	
Kanal 7	NAS >25	[0…14]	
Kanal 8	Temp	[-60…150]	°C
Kanal 9	Flow	[30…300]	ml/min
Kanal 10	Drive	[0…100]	%

Messkanäle AS 1000 Serie

AS1008-C			
Kanal 1	Sat. (RelHum)	[0…100]	%
Kanal 2	Temp	[-25…100]	°C

Messkanäle CS 2000 Serie

CS 20xx						
	Partikelzahlen		NAS/SAE Klasse		ISO Code	
Kanal 1	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO > 5 µm	[0…25]
Kanal 2	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO > 15 µm	[0…25]
Kanal 3	25-50 µm	[0…129600]	NAS 25-50 µm	[-1…15]	ISO > 25 µm	[0…25]

Kanal 4	>50 µm	[0…23040]	NAS >50 µm	[-1…15]	ISO > 50 µm	[0…25]
Kanal 5	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
Kanal 6	Analog 1 (bei Firmware Version ≥ 4.00)					
Kanal 7	Analog 2 (bei Firmware Version ≥ 4.00)					

CS 21xx

	Partikelzahlen		NAS/SAE Klasse		ISO Code	
Kanal 1	2-5µm	[0…20484000]	NAS 2-5 µm	[-1…15]	ISO > 2 µm	[0…25]
Kanal 2	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO > 5 µm	[0…25]
Kanal 3	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO > 15 µm	[0…25]
Kanal 4	>25 µm	[0…129600]	NAS >25 µm	[-1…15]	ISO > 25 µm	[0…25]
Kanal 5	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
Kanal 6	Analog 1 (bei Firmware Version ≥ 4.00)					
Kanal 7	Analog 2 (bei Firmware Version ≥ 4.00)					

CS 22xx

	Partikelzahlen		NAS/SAE Klasse		ISO Code	
Kanal 1	> 4 µm	[0…3200000]	SAE A	[-2…15]	ISO > 4 µm	[0…25]
Kanal 2	> 6 µm	[0…1250000]	SAE B	[-2…15]	ISO > 6 µm	[0…25]
Kanal 3	> 14 µm	[0…222000]	SAE C	[-2…15]	ISO > 14 µm	[0…25]
Kanal 4	> 21 µm	[0…39200]	SAE D	[-2…15]	ISO > 21 µm	[0…25]
Kanal 7	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
Kanal 6	Analog 1 (bei Firmware Version ≥ 4.00)					
Kanal 7	Analog 2 (bei Firmware Version ≥ 4.00)					

HYDAC FILTER SYSTEMS

HYDAC FILTER SYSTEMS GMBH

Industriegebiet
66280 Sulzbach/Saar
Deutschland

Postfach 1251
66273 Sulzbach/Saar
Deutschland

Tel: +49 (0) 6897 509 01Zentrale

Fax: +49 (0) 6897 509 9046 Technik

Fax: +49 (0) 6897 509 577 Verkauf

Internet: www.hydac.com

E-Mail: filtersystems@hydac.com