HYDAC INTERNATIONAL

FluidMonitoring Toolkit

FluMoT

Version 1.2x

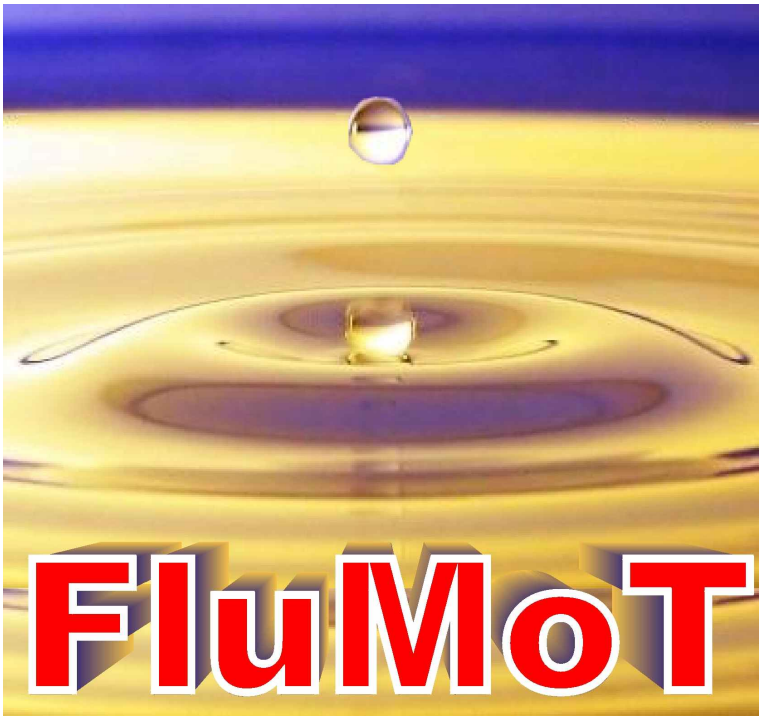
Für:

- **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**

Bedienungsanleitung

Deutsch (Originalanleitung)

Dokumentation Nr.: 3377564



Warenzeichen

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

Copyright © 2008 by HYDAC Filtertechnik 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 Filtertechnik GmbH

Servicetechnik / Filtersysteme

Industriegebiet

D-66280 Sulzbach / Saar

Germany

Tel.: ++49 (0) 6897 / 509 - 01

Fax: ++49 (0) 6897 / 509 - 846

Inhalt

Warenzeichen 2

Inhalt 3

Registrierkarte 6

Einführung 7

Allgemeines 7
Zum Gebrauch dieser Bedienungsanleitung 7
Symbol- und Hinweiserklärung 7

Soft- und Hardware installieren 8

Systemvorrausetzungen 8
Hardware 8
Software 8
Installation vorbereiten 8
FluMoT installieren 8
FluMoT deinstallieren 13

Arbeiten mit FluMoT 14

DIN Messbus - DLL 17

API – Funktionen 17
Fehlerbehandlung 18
Allgemein 18
Kommunikation mit Gerät 18
Fehlermeldungen von FCU/CS 19
Statuswert in Protokolldatei 19
Statuskontrolle 20
GetErrorStateText_DMB() 20
Versionskontrolle 20
GetDLLVersion_DMB() 20
GetDLLVersionText_DMB() 20
Serielle Schnittstelle 21
Gerätesuche und Geräteinformation 21
SearchBusDevice_DMB() 21
GetDeviceSerialNumber_DMB() 21
GetDeviceSensorNumber_DMB() 22
GetDeviceCalibrationDate_DMB() 22
GetDeviceChannelCount_DMB() 22
GetDeviceChannelInfo_DMB() 23

SetBusAddress_DMB() 24
Messwerte lesen 25
SetMeasuringState_DMB() 25
GetDeviceState_DMB() 25
GetDeviceMeasuringValues_DMB() 26
Dateien aus dem Gerät lesen 26
GetDeviceLogDirectory_DMB() 27
GetDeviceLogHeader_DMB() 28
GetDeviceLogDataBlock_DMB() 29
Dateien im Gerät löschen 30
EraseDeviceLog_DMB () 30
HSI - DLL 31
API - Funktionen 32
Fehlerbehandlung 33
Statuskontrolle 34
GetErrorStateText_HSI() 34
Versionskontrolle - DLL 34
GetDLLVersion_HSI() 34
GetDLLVersionText_HSI() 34
Serielle Schnittstelle 34
Gerätesuche und Geräteinformation 35
SearchOneDevice_HSI() 35
SearchBusDevice_HSI() 35
GetDeviceChannelCount_HSI() 35
GetDeviceSerialNumber_HSI() 36
GetDeviceChannelInfo_HSI() 36
Busaddressen verwalten 37
GetBusAddress_HSI() 37
SetBusAddress_HSI() 37
Messwerte lesen 38
GetDeviceChannelsMask_HSI() 38
GetDeviceMeasuringValues_HSI() 39
Beispiele zur Messwert Interpretation 40
GetDeviceState_HSI() 41
Dateien aus dem Gerät lesen 42
GetDeviceLogDirectoryBlock_HSI() 43
GetDeviceLogHeaderBlock_HSI() 44
GetDeviceLogDataBlock_HSI() 46
Dateien im Gerät löschen 48
EraseDeviceLog_HSI () 48
HSITP - DLL 49

API - Funktionen 49
Fehlerbehandlung 49
Statuskontrolle 50
GetErrorStateText_HTP() 50
DLL – Versionskontrolle 50
GetDLLVersion_HTP() 50
GetDLLVersionText_HSI() 50
Ethernet Schnittstelle 50
Gerätesuche und Geräteinformation 51
SearchOneDevice_HTP() 51
GetDeviceChannelCount_HTP() 51
GetDeviceSerialNumber_HTP() 51
GetDeviceChannelInfo_HTP() 51
Messwerte lesen 53
GetDeviceChannelsMask_HTP() 53
GetDeviceMeasuringValues_HTP() 54
GetDeviceState_HTP() 55
Beispiele 56

Der OPC - Server 57

Messkanal Übersicht 60

FCU 2000 Serie 60
FCU 8000 Serie 61
CS 2000 Serie 62

Registrierkarte

Registrierung
FluMoT Version 1.2x

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 Filtertechnik GmbH, Servicetechnik / Filtersysteme,
Industriestraße, Werk 6, D-66280 Sulzbach / Saar,
Fax: ++49 (0) 6897 / 509-846, E-Mail: filtersysteme_support@hydac.com

Ohne Registrierung wird der Support durch HYDAC verweigert!

Einführung

Allgemeines

FluMoT (**FluidMonitoring Toolkit**) ist eine Sammlung von unterschiedlichen Entwicklungswerkzeuge, die Kommunikation zwischen PC und HYDAC - Sensoren ermöglichen.

Dieses Toolkit 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 FMM, FMMP, FMMHP, FMMP Unit
- HYDACLab (HLB 1000)
- HMG 3000
- CMU 1000

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

Zum Gebrauch dieser Bedienungsanleitung

Im folgenden wird vorausgesetzt, dass Sie mit der Bedienung von WINDOWS 98, 2000, ME, XP, dem Aufbau und der Installation Windows typischer Programme vertraut sind!

Symbol- und Hinweiserklärung

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

Unter diesem Symbol werden die wichtigen **Informationen** zusammengefasst.

Mit diesem Symbol werden die **Anwendungstipps** und besonders nützliche Informationen bezeichnet.

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

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

HYDAC FILTERTECHNIK GmbH
Servicetechnik / Filtersysteme
Postfach 12 51
D-66273 Sulzbach / Saar - Deutschland

E-Mail: filtersysteme@hydac.com
Fax.: ++49 (0) 6897 509 - 846

Soft- und Hardware installieren

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)
- Microsoft Internet Explorer 4.0 oder höher
- Administrator Rechte zur Softwareinstallation

Installation vorbereiten

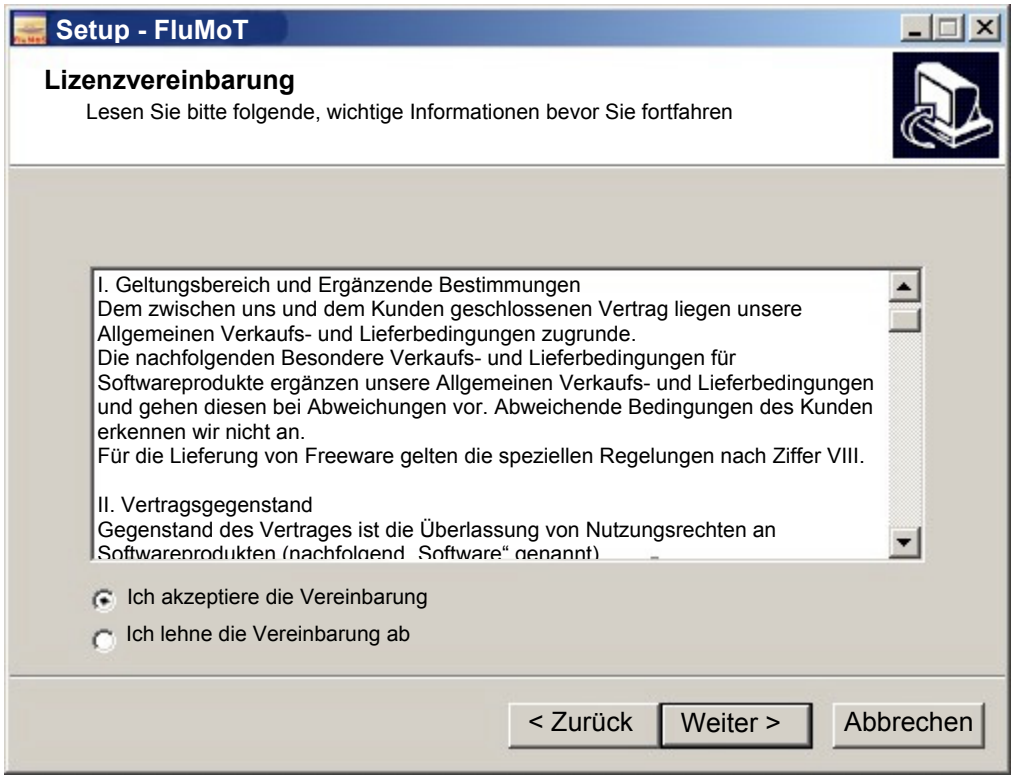
- Um die Funktion zu gewährleisten, empfehlen wir ältere Versionen von **FluMoT** zu deinstallieren.
- Wie die Sensoren bzw. Geräte angeschossen werden, entnehmen Sie bitte der jeweiligen Bedienungsanleitung.

FluMoT installieren

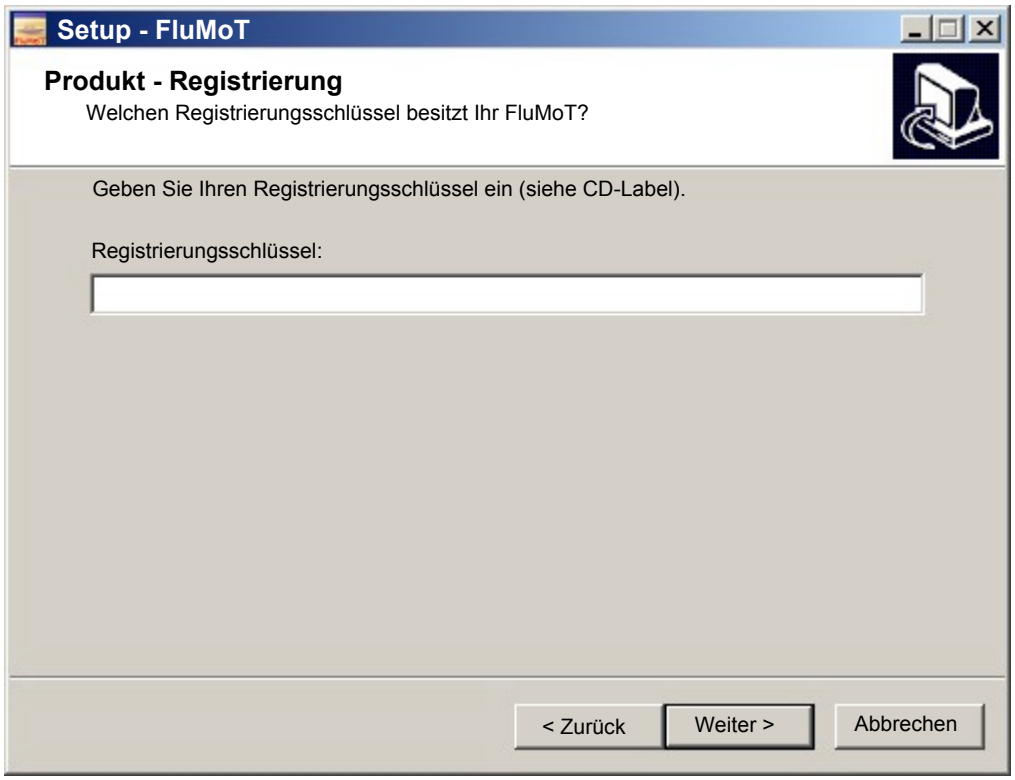
Zur Installation von FluMoS, starten Sie das Programm SETUP_FLUMOT_Vxxx.EXE auf der CD.

Der Setup-Assistent führt Sie durch die gesamte Installation. Zum Fortfahren drücken Sie „Weiter“.

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



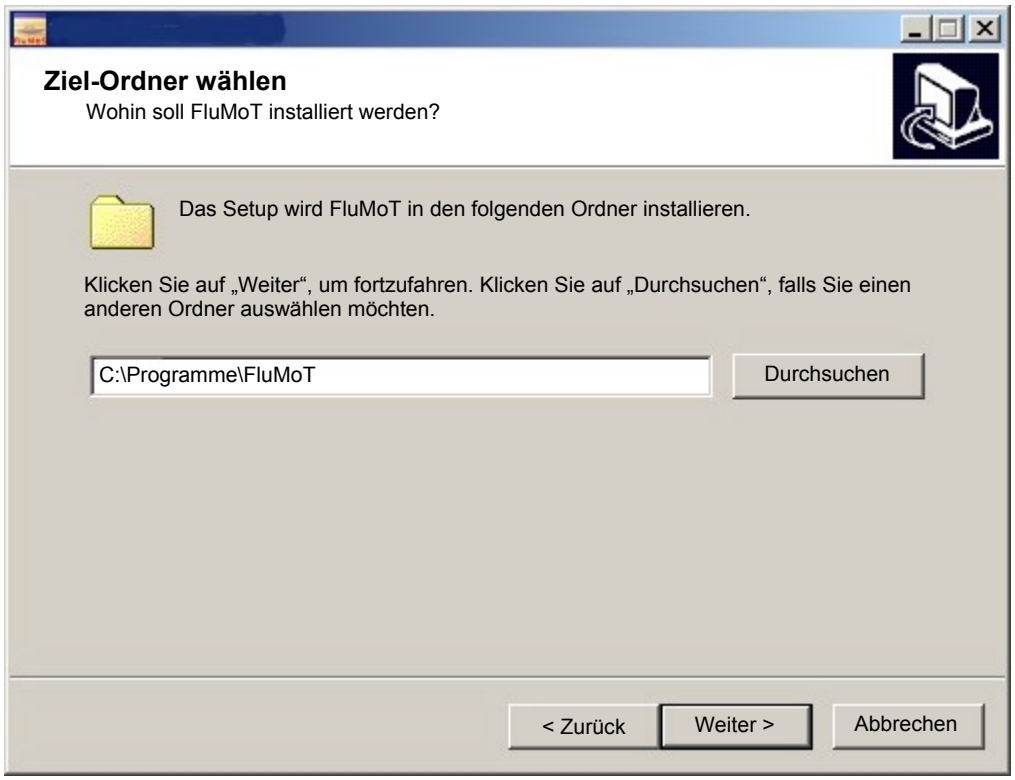
Um Ihre FluMoT Software zu aktivieren, tragen Sie den Registrierungsschlüssel von der FluMoT-CD ein.



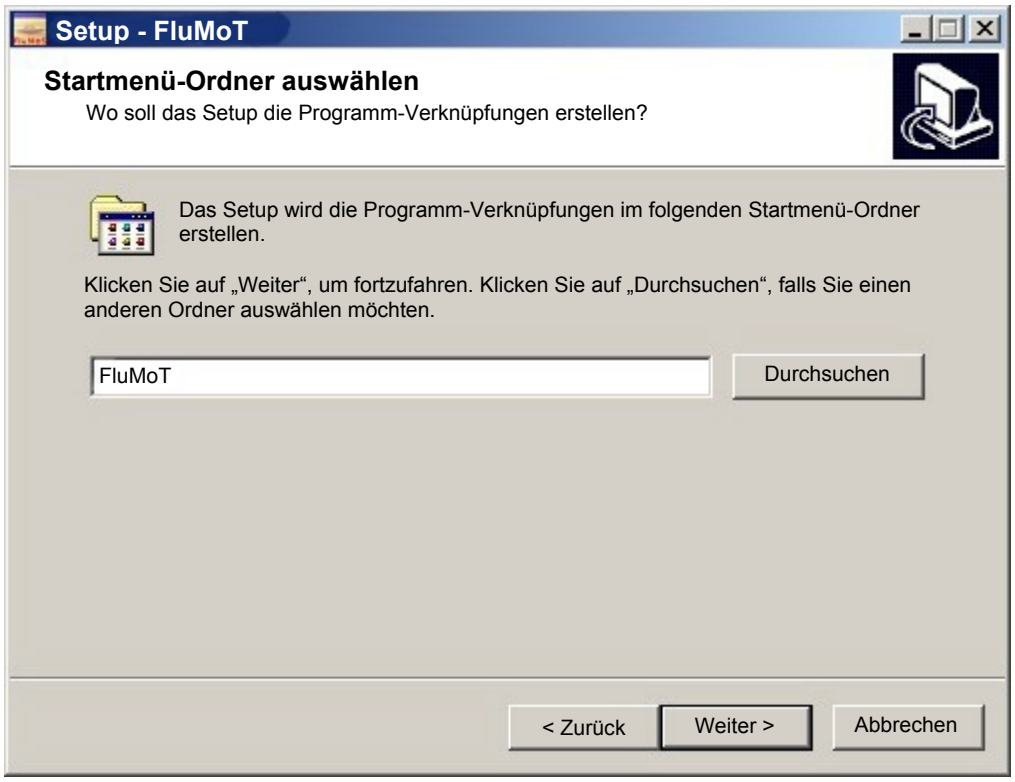
Bei der Installation werden Programmdateien in das Installationsverzeichnis übertragen.

Im nächsten Schritt wird das Installationsverzeichnis festgelegt.

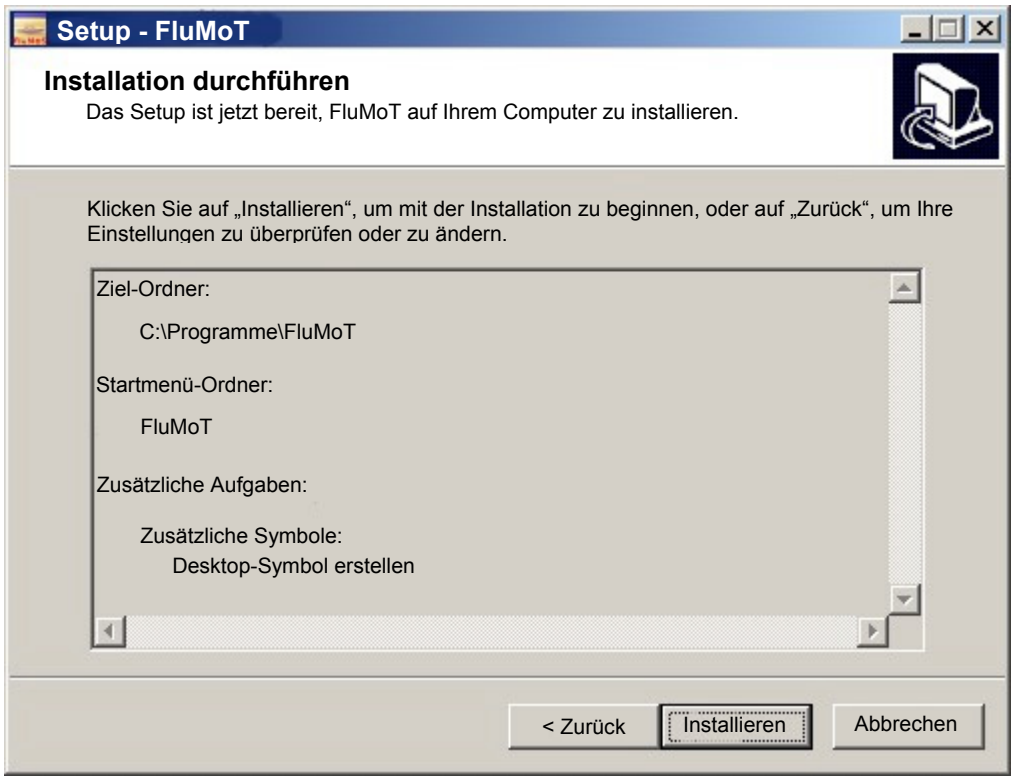
Sollte das Installationsverzeichnis bereits existieren, erfolgt die Frage, ob der Pfad überschrieben werden soll.



Danach wird ein Startmenü-Ordner erstellt.



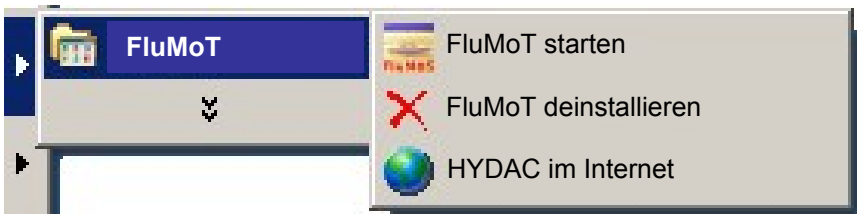
Nach der Bestätigung durch klicken auf Weiter > wird der Installationsprozess gestartet.



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

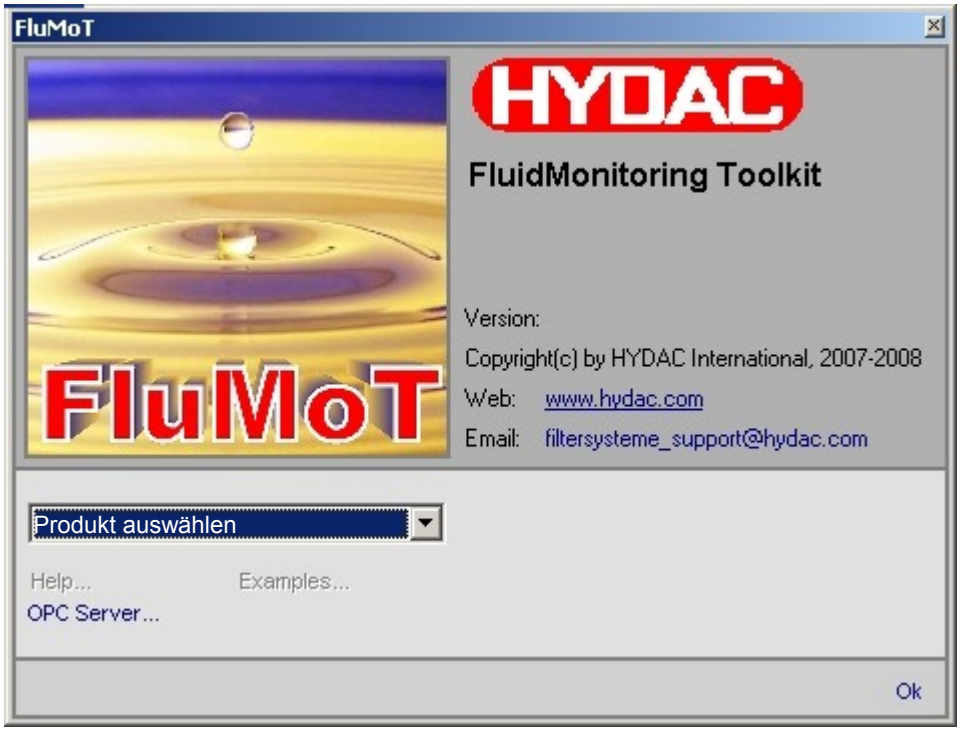
FluMoT deinstallieren

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



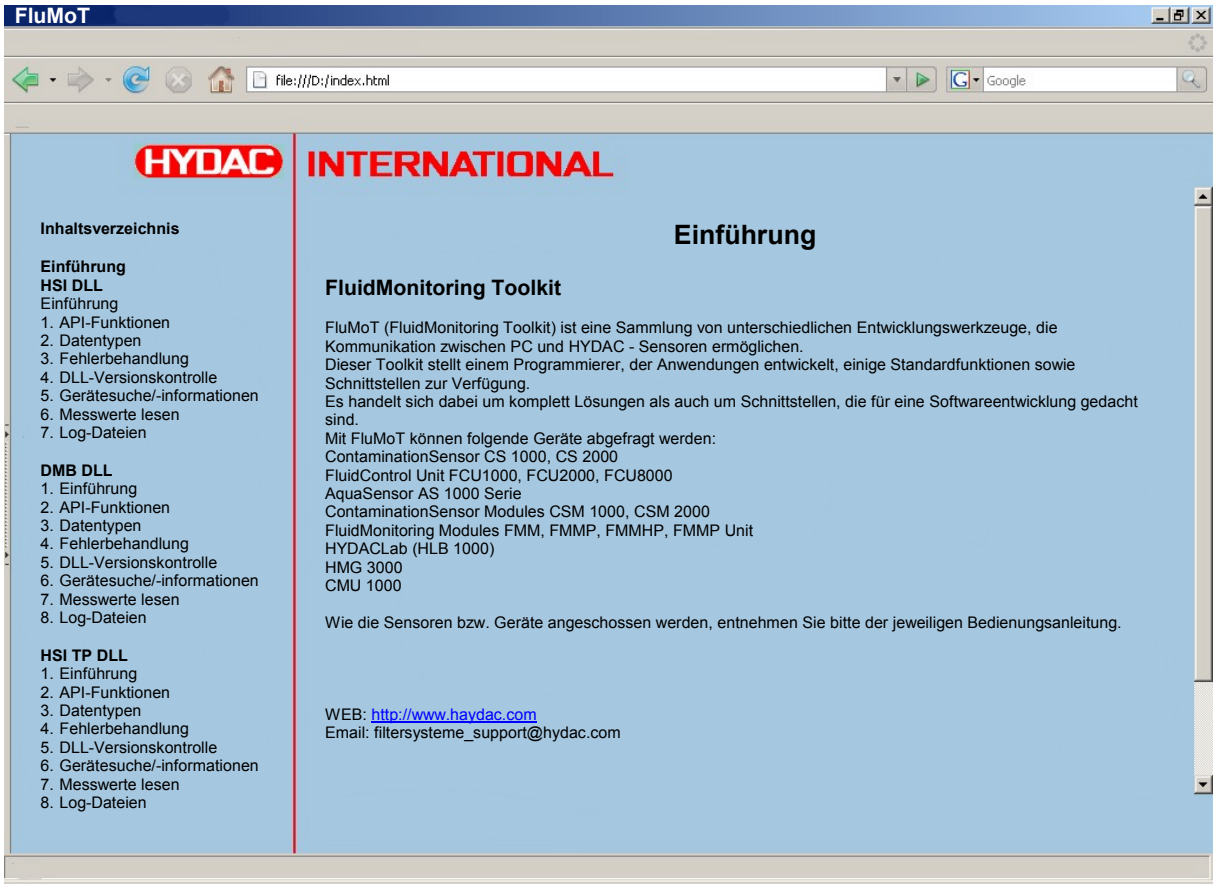
Arbeiten mit FluMoT

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 zur **FluMoT** HTML Hilfe und den entsprechenden Verzeichnissen mit Beispielen auf der Festplatte.

Die **FluMoT** HTML Hilfe stellt eine umfangsreiche Beschreibung aller Toolkit – Komponenten dar. Um die Online Hilfe zu öffnen, ist ein Internet Browser erforderlich (≥ IE 4.0).



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

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

FluMoT beinhaltet folgende drei DLLs:

DLL	Beschreibung in Kapitel	Seite
dinmessbus32.dll	DIN Messbus - DLL	17
hecom32.dll	HSI– DLL	31
hsitp32.dll	HSITP - DLL	49

Um die Hochsprachenprogrammierung zu erleichtern, werden einfache Beispiele als kleine Projekte in Delphi7, LabView 7 und Excel -Macros (VBA 6) im Sourcecode mitgeliefert. Diese befinden sich in 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 Format: 32 Bit, mit Vorzeichen	Integer	Integer	Long	Long
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.	Char	Char*	String	String (C String Pointer)

Mit einer String werden manchmal mehrere Informationen übertragen. In diesem Fall wird ein Steuerzeichchen 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.

DIN Messbus - DLL

Die Datei Dinmessbus32.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
- CS2000 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, deshalb muss jedem Sensor eine Adresse zugewiesen werden, unter welcher dieser angesprochen werden kann. Der Adressbereich nach DIN 66348 erstreckt sich von 1 bis 31. Jede Adresse darf nur einmal im Bus vorkommen.

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

Die meisten DLL-Funktionen liefern im Fehlerfall einen Fehlercode. Dieser Fehlercode kann folgende 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
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 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
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 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: no device found″

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″

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
1 – NAS/SAE - Klassen
2 – ISO – Code

Die FCU liefern nicht in allen Messkanälen den ISO-Code als Messwerte im Mode 2 (siehe Geräteübersicht in Kapitel Messkanal Übersicht)

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	Bezeichnung des Kanals
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	Untere Grenze des Messbereiches
5	UpperRange	Obere Grenze des Messbereiches

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)

Bei einigen Geräten ist ein Neustart oder Reset (Stromversorgung ein/aus) erforderlich.

Messwerte lesen

SetMeasuringState_DMB()

Mit diesem Befehl wird eine Messung gestartet oder gestoppt. Außerdem ist es möglich, mit diesem Befehl den Fehlerstatus des Gerätes rückzusetzen. (falls kein fataler 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: *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

Beispiel: 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

Beispiel: 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 1 – NAS/SAE - Klassen 2 – ISO-Code *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 (z.B.):	„7680<CR>1860<CR>5<CR>0<CR>122<CR>0<CR>0<CR>“

Dateien aus dem Gerät lesen

Aufgrund großer Datenmengen werden zum Auslesen der Sensorikspeicher 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.

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 sind den jeweiligen Bedienungsanleitungen zu entnehmen.

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 als ASCII - Code des Zeichens.
FileID – Datei – Identifikator (siehe GetDeviceLogDirectory_DMB)
Mode – Messdateneinheiten (wird auch in Funktion „GetDeviceChannelInfo_DMB“ verwendet)
0 – Partikelzahlen
1 – NAS/SAE - Klassen
2 – ISO – Code
State – Referenz auf Kommunikationsstatus - Variable.

Rückgabewert: Zeiger auf eine Zeichenvariable mit Dateikopf - Information.

Beispiel: Rückgabewert:

„7<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
1 – NAS/SAE - Klassen
2 – ISO – Code
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 mögliche Statuscodes finden Sie auf Seite 18.

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 Kanal 1	2
Decimals Kanal 2	2
Decimals Kanal 3	2
Decimals Kanal 4	2
Decimals Kanal 5	2
Decimals Kanal 6	2
Decimals Kanal 7	1

Deshalb sind die Werte folgendermaßen zu interpretieren:

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

Dateien im Gerät 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

Beispiel: Antwort: 0 (Fehler: siehe Inhalt der Variable „State“)

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	HSI	HMG 3000 Serie
ContaminationSensor CS 1000 Serie		CMU 1000 Serie
FluidControl Unit FCU 1000 Serie		

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 der Datei *hecom32.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 lesen
GetDeviceMeasuringValues_HSI	Messwerte lesen
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	Multipacket – Übertragung wird 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: Antwort: „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: Antwort: 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: Antwort: „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.

Beispiel: 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	Bezeichnung des Kanales
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	Untere Grenze des Messbereiches
5	UpperRange	Obere Grenze des Messbereiches

Beispiel: Antwort: „Temp**<CR>**°C**<CR>**2**<CR>**-2500**<CR>**10000**<CR>** “

Busadressen verwalten

Nicht alle HSI – Sensoren unterstützen die nachfolgenden zwei Befehle!

GetBusAddress_HSI()

Diese Funktion gibt die Busadresse eines Gerätes zurück.

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 lesen

GetDeviceChannelsMask_HSI()

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_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 sog. „Gerätemaske“.

Beipiel: „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, Kanal 1	Datengröße im 1. Messbereich
6	DataSize, Kanal 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.
Die Zusammensetzung der Messwerte muss vorher mit dem Befehl „GetDeviceChannelsMask_HSI“ festgestellt werden. Dabei bekommt man als Antwort sie sog. „Gerätemaske“, die jedes Mal mit dem Befehl „GetDeviceMeasuringValues_HSI“ bestätigt werden muss. 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

Beispiel: „127**<CR>**104**<CR>**80**<CR>**20**<CR>**21**<CR>**22**<CR>**19**<CR>**246**<CR>**100**<CR>**0**<CR>**“

Beispiele zur Messwert Interpretation

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, Kanal 1	2		
DataSize, Kanal 2		4	
DataSize, Kanal 3			2

Die Werte sind wie folgt auszuwerten:

	Wert
Kanal 1	135
Kanal 2	47
Kanal 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, Kanal 1	2		
DataSize, Kanal 2		4	
DataSize, Kanal 3			2

Die Werte sind wie folgt auszuwerten:

	Wert
Kanal 1	135
Kanal 2	47
Kanal 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 max. 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.
Beispiel:	Antwort: „ASIC-CRC-Error“, *StateByte* = 3, *StateCode* =17

Dateien aus dem Gerät lesen

Aufgrund großen Datenmengen werden zum Auslesen von Sensorikspeicher 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.

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

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>

--- Hardwareversion 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	
18	UpperRange	
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 (s. 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 Verbrach 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, Kanal 1	2
DataSize, Kanal 2	2

Deshalb sind die Werte folgendermaßen auszuwerten:

	Wert
Status	0
Kanal 1	47
Kanal 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, Kanal 1	2
DataSize, Kanal 2	2

Deshalb sind die Werte folgendermaßen auszuwerten:

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

Dateien im Gerät löschen

EraseDeviceLog_HSI ()

Diese Funktion entfernt eine Log – Datei aus dem Gerätspeicher.

Syntax: **function** 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 (s. GetDeviceLogDirectoryBlock_HSI)
State – Referenz auf Kommunikationsstatus - Variable.

Rückgabewert: FileID der gelöschten Datei oder -1 im Fehlerfall

Beispiel: Antwort: 10 (Datei 10 wurde erfolgreich gelöscht)

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).

Beispiel: 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.

Beispiel: 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.

Beispiel: Antwort: „v.1,03 03.07.2007”

Ethernet Schnittstelle

Bevor man eine Übertragung startet, wird immer zuerst die Verbindung aufgebaut. Jede nachfolgende Funktion öffnet zuerst die Verbindung mit Host, führt seine Routine und schließt die Verbindung. Bei mehreren Anfragen eines Gerätes nimmt dieser Sachverhalt gewisse Zeit in Anspruch. Der Aufwand kann reduziert werden, indem man die Verbindung zum Gerät einmalig (z.B. beim Start des Programms) öffnet, führt alle Anfragen aus und schließt die Verbindung z.B. beim Schließen des Programms. 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.

Beispiel: 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.

Beispiel: 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	Bezeichnung des Kanals
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	Untere Grenze des Messbereiches
5	UpperRange	Obere Grenze des Messbereiches

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

Messwerte lesen

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“.

Beispiel: 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, Kanal 1	Datengröße im 1.Messbereich
6	DataSize, Kanal 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

Beispiel: Antwort:

„127**<CR>**104**<CR>**80**<CR>**20**<CR>**21**<CR>**22**<CR>**19**<CR>**246**<CR>**100**<CR>**0**<CR>**“

GetDeviceState_HTP()

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 max. 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.
Beispiel:	Antwort: „ASIC-CRC-Error“, *StateByte* = 3, *StateCode* =17

Beispiele

Um die Hochsprachenprogrammierung mit FluMoT DLLs zu erleichtern, werden einfache Beispiele als kleine Projekte in Delphi7, LabView 7 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.

LabView – Beispiel (EXE - Datei) ist nur ausführbar, wenn LabView Runtime Engine 7 installiert ist.
Sollte keine Runtime Engine 7 installiert sein, finden Sie im Verzeichnis „Examples“ eine vollständige Installation.

Der OPC - Server

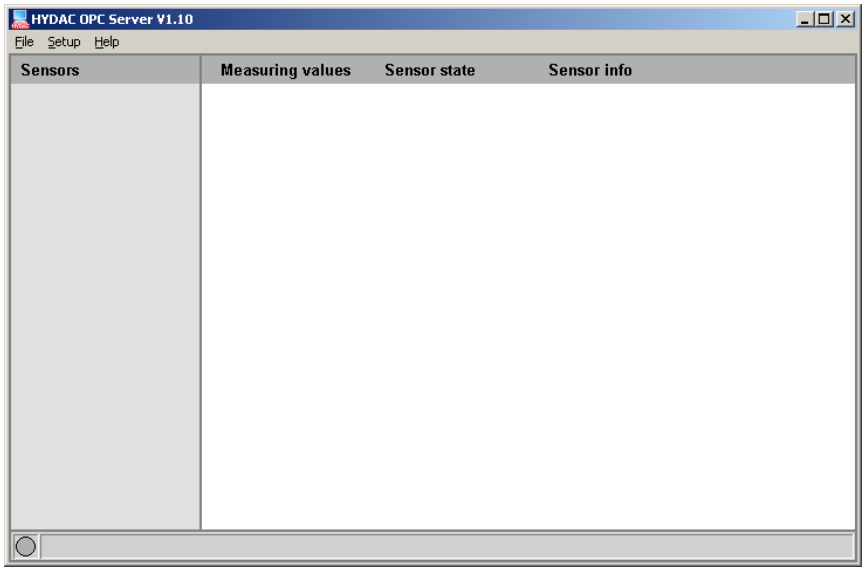
OPC (Openness, Productivity, Collaboration, früher OLE for Process Control) ist eine standardisierte Schnittstelle zum Zugriff auf Prozessdaten. Sie basiert auf dem Microsoft Standard DCOM und wurde für die Bedürfnisse des Datenzugriffs in der Automatisierung erweitert. Sie wird vorwiegend zum Lesen 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. DCOM - Technologie ermöglicht auch einen Zugriff auf Server, der auf einem anderen Rechner 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.

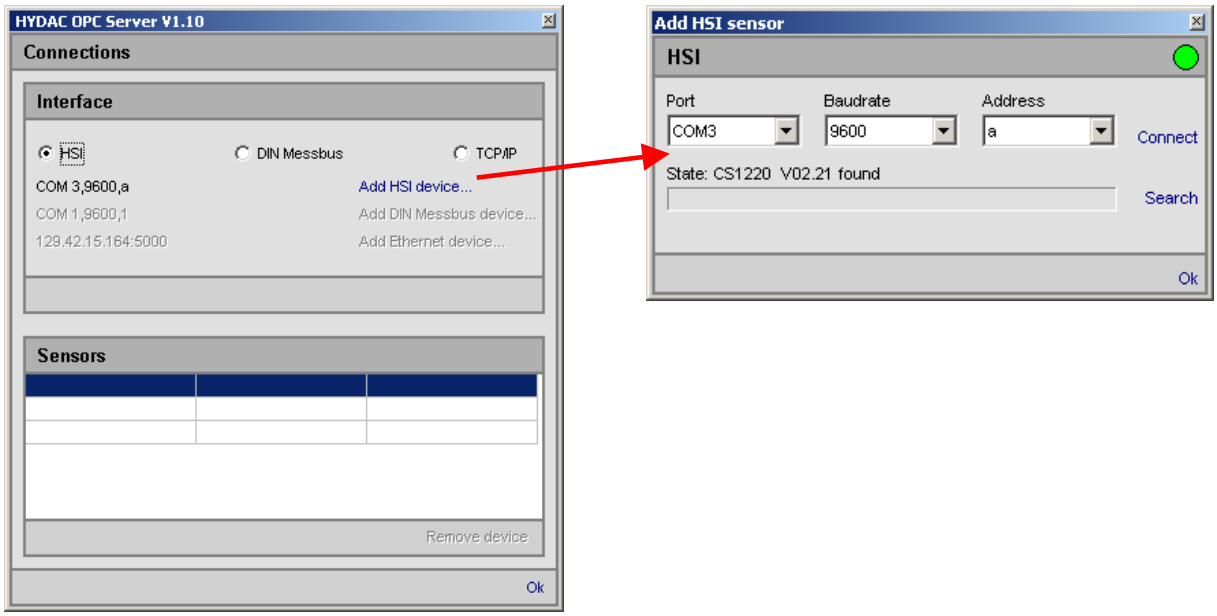
HYDAC OPC Server V1.10 basiert auf OPC Data Access 2.0 (Spezifikation zur Übertragung von Echtzeitwerten über Microsoft Standard DCOM). Der Server wird während FluMoT Installation auf dem Zielrechner kopiert und im System registriert. Bei Deinstallation von FluMoT wird auch HYDAC OPC Server entfernt. Man kann Registrierung / Deregistrierung mit Hilfe von beiliegenden Dateien *reg_opc.bat* und *unreg_opc.bat* auch manuell betätigen. Das Programm ist in der Lage mit folgenden Sensortypen zu kommunizieren:

- **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)

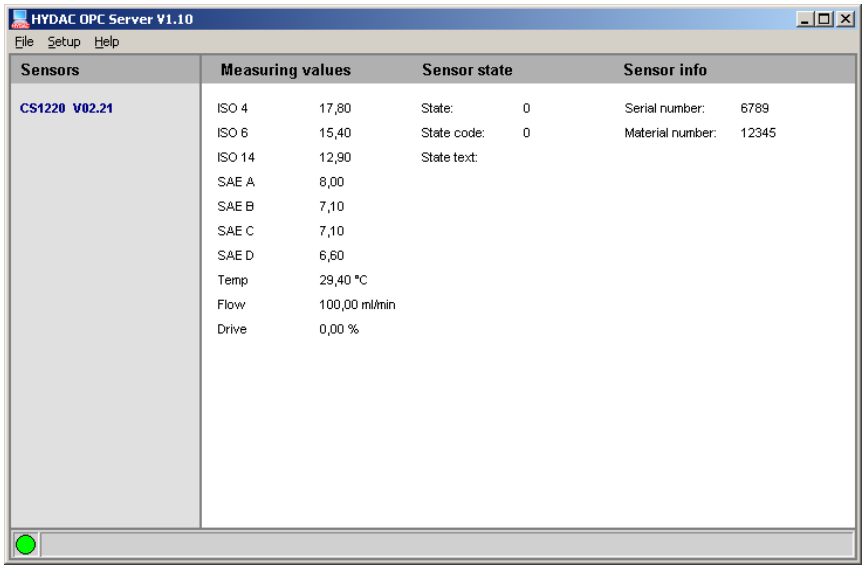
Nach der Installation vom HYDAC OPC Server wird das Programm zuerst konfiguriert. Die Schnittstellen aller abzufragenden Sensoren sind im Programm zu definieren.



Über den Menüpunkt SETUP → CONNECTIONS im Hauptmenü des Programms wird die Verbindung zu den einzelnen Sensoren hergestellt. So werden die Sensoren in eine Liste hinzugefügt und diese Liste wird bei der Bestätigung mit Ok Taste vom Programm gespeichert. Eine Online Messung wird anschließend gestartet.

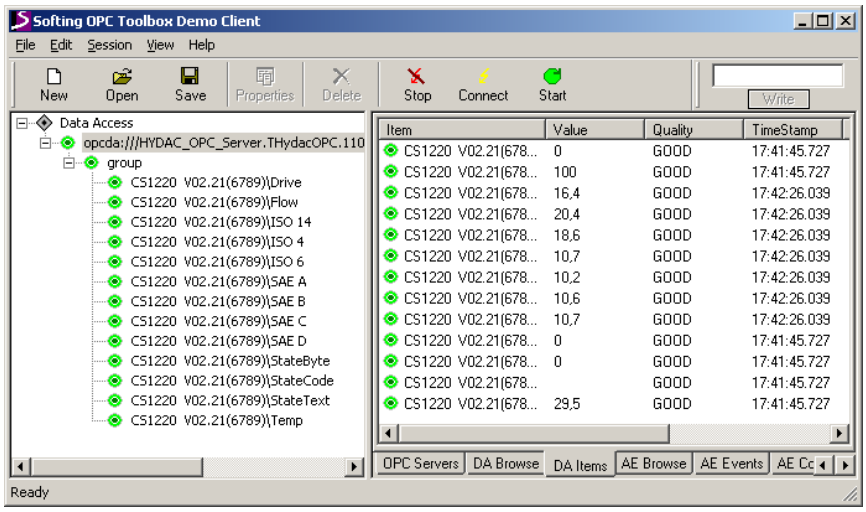


Jedes Mal wenn ein OPC – Client den Server anspricht, wird das Programm die so gespeicherten Verbindungen wiederherstellen. Die Anzahl von Sensoren, die auf diese Weise angesprochen werden, ist im Prinzip unbegrenzt. Jedoch sollte man beachten, dass die Kommunikation im Falle Großzahl von Sensoren deutlich langsamer ist.



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 V1.10”. 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)



Messkanal Übersicht

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

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	2-5µm	[0..20484000]	NAS 2-5µm	[-1..15]	ISO >2µm	[0..25]
Kanal 2	5-15µm	[0..4096000]	NAS 5-15µm	[-1..15]	ISO >5µm	[0..25]
Kanal 3	15-25µm	[0..729000]	NAS 15-25µm	[-1..15]	ISO >15µm	[0..25]
Kanal 4	>25 µm	[0..129600]	NAS >25µm	[-1..15]	ISO >25µm	[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 >4µm	[0..25]
Kanal 2	> 6 µm	[0..1250000]	SAE B	[-2..15]	ISO >6µm	[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]
-	-	-	-	-	-	-

FCU 8000 Serie

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

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	5-15µm	[0..4096000]	NAS 5-15 µm	[-1..15]	ISO > 5 µm	[0..25]
Kanal 3	15-25µm	[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)					

Notizen

HYDAC INTERNATIONAL

HYDAC Filtertechnik GmbH
Bereich Servicetechnik / Service Technology Division
Industriegebiet
66280 Sulzbach/Saar
Germany

Postfach 1251
66273 Sulzbach/Saar
Germany

Tel: +49 (0) 6897 509 01
Fax: +49 (0) 6897 509 846 (Technik)
Fax: +49 (0) 6897 509 577 (Verkauf)

Internet: www.hydac.com
email: filtersysteme@hydac.com