HYDAC FILTER SYSTEMS

FluidMonitoring Toolkit

FluMoT

Version 1.3x

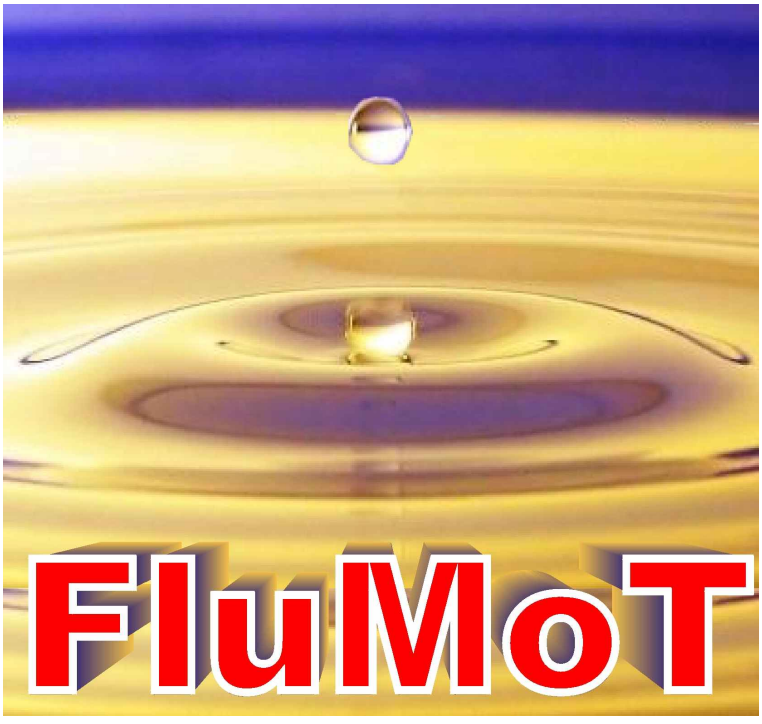
For:

- **CS 1000 / CS 2000 series**
- **AS 1000 series**
- **HLB 1000 series**
- **HMG 3000 series**
- **CMU 1000 series**
- **FCU 1000 / 2000 / 8000 series**
- **CSM 1000 / 2000 series**
- **FMM series**

Operating Manual

English (translation of original instructions)

Document No.: 3377564a



Trademarks

The trademarks of other companies are exclusively used for the products of those companies.

Copyright © 2011 by HYDAC Filter Systems GmbH all rights reserved

Exclusion of Liability

We made every endeavor to ensure the accuracy of the contents of this document. However, errors cannot be ruled out. Consequently, no liability is assumed for any errors or deficiencies inherent in this document, including consequential damage or loss. The content of the manual is checked regularly. Any corrections required will be incorporated in subsequent editions. We welcome any suggestions for improvements.

All details are subject to technical modifications.

Technical specifications are subject to change without notice.

HYDAC FILTER SYSTEMS GMBH
Postfach 12 51
66273 Sulzbach / Saar

Germany

Documentation Representative

Mr. Günter Harge

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

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

Contents

Filling out the FluMoT registration card

Registration
FluMoT Version 1.3x

By opening the sealed package containing the disk(s), alternatively by installing the software, you declare that you accept the terms of use set forth in the software license agreement.

Please complete this registration card and return it to us in order to be registered as an authorized user of the program.

Take advantage of the benefits that registration offers:

- Free email support: filtersysteme_support@hydac.com
- Software News
- Information about updates to this software

We assure you that we will not pass this information on to any third parties.

FluMoT Registration key

Company

Street

Postal code, city

User name

E-mail address

City, date, signature

Please send the completed registration card via mail or fax to:

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

Without registration, software support is refused by HYDAC!

FluMoT features

The FluidMonitoring Toolkit FluMoT is a driver package which is used to connect HYDAC fluid sensors or new generation sensors with an HSI interface to the customer´s own PC software.

FluMoT replaces all previous drivers for connecting ContaminationSensors and FluidControl Units (e.g. for Windows, LabVIEW).

The driver package FluMoT consists of the following components:

- dll
  - → HSI/HeCom
  - → HSITP
  - → DinMessBus
- Example programs*
  - → Delphi
  - → LabVIEW
  - → VB/VBA
  - → C/C++
  - → OPC-Server

This driver package provides application developers with a number of standard functions and interfaces.

Included are complete solutions and interfaces needed for software development.

The FluMoT can be used to poll the following devices:

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

Please refer to the respective installation instructions for information on connecting the sensors and units.

Safety information and instructions

We assume you are familiar with the operation of WINDOWS XP, Vista or 7 and the design and installation of typical Windows programs.

Explanation of Symbols and Warnings

The following designations and symbols are used in this manual:

Important **information** is summarized under this symbol.

This symbol designates **tips for use** and other particularly useful information.

This symbol provides important **tips** for the proper handling and operation of the product. The non-observance of these instructions can lead to false use or malfunction of the product.

If you have any questions, suggestions, or encounter any problems of a technical nature concerning the **FluMoT**, please contact:

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

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

System requirements for hardware

System requirements for hardware

- Pentium processor 200 MHz or higher
- 64 MB RAM memory
- VGA graphic card (800x600 min.)
- Hard disk with a minimum of 30 MB free disk space
- A free serial port (RS232 / USB) port:
  1. Not occupied by a connector
  2. Not being used by the operating system
  3. Not being used by any other program
     (e.g. terminal, modem or network software).
- Microsoft Windows-compatible mouse

Software

- WINDOWS 98, 2000, ME, XP, Server 2003, Windows Vista (32bit), Windows 7
- Microsoft Internet Explorer 4.0 or higher
- Administrator rights to software installation

Installing FluMoT

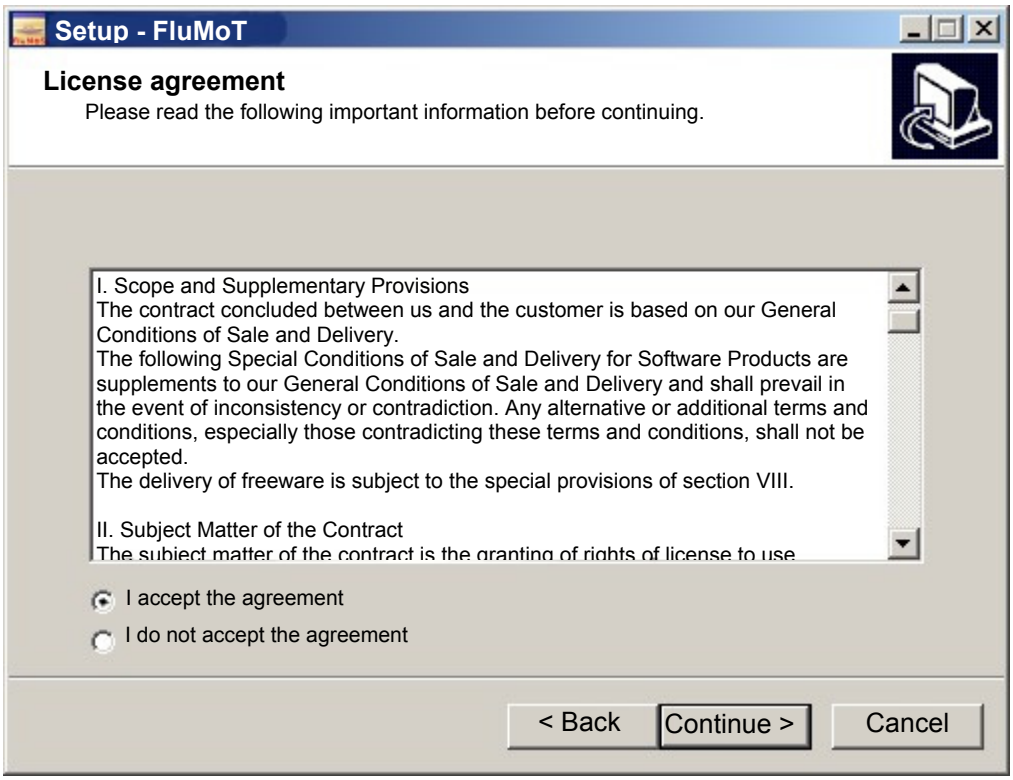
Deinstall all older versions of FluMoT prior to the installation.

You can take the details for the connection of the device/sensor from the respective operating instructions.

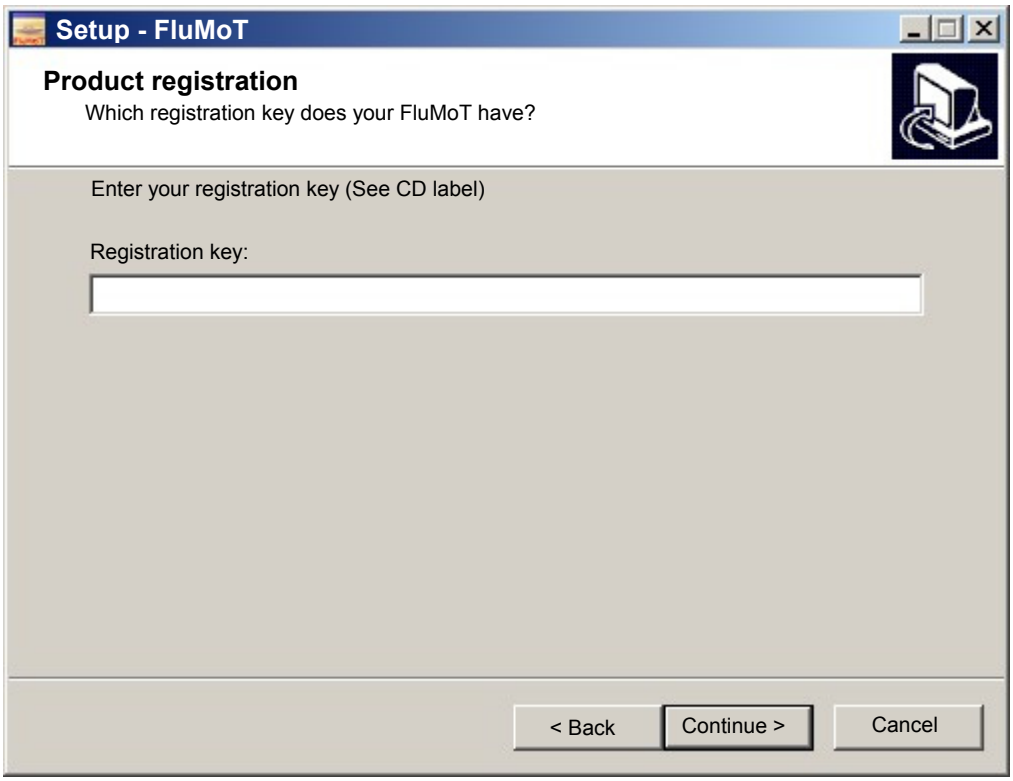
To install FluMoT, start the program SETUP_FLUMOT_Vxxx.EXE on the CD.

The Setup wizard will guide you through the installation process. To continue, click on Next.

To continue the installation, carefully read through the licence agreement in the next window and then click "I accept the agreement."



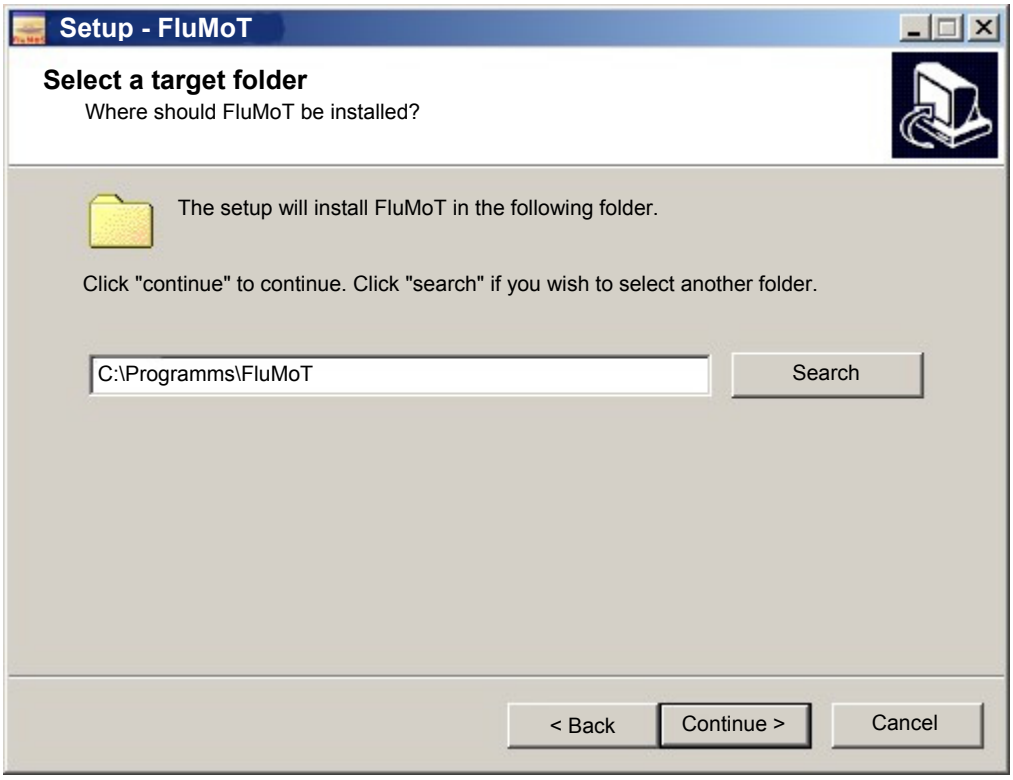
To activate your FluMoT software, enter your registration key from the FluMoT CD.



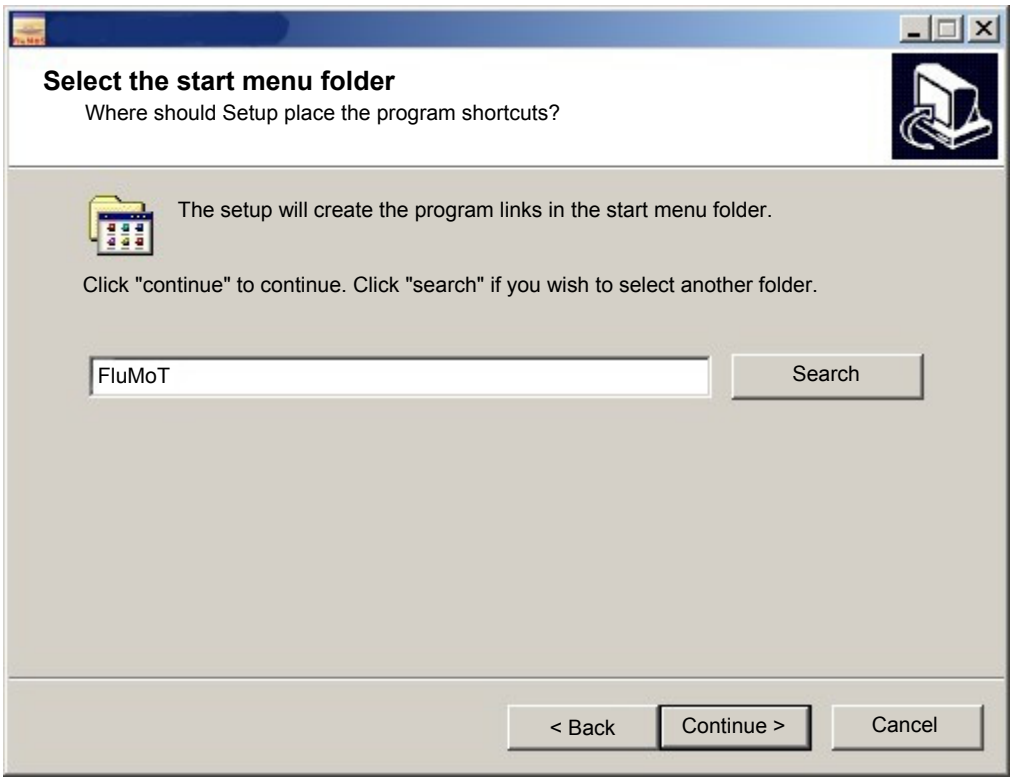
Program files are copied into the installation directory during the installation.

You then determine the installation directory.

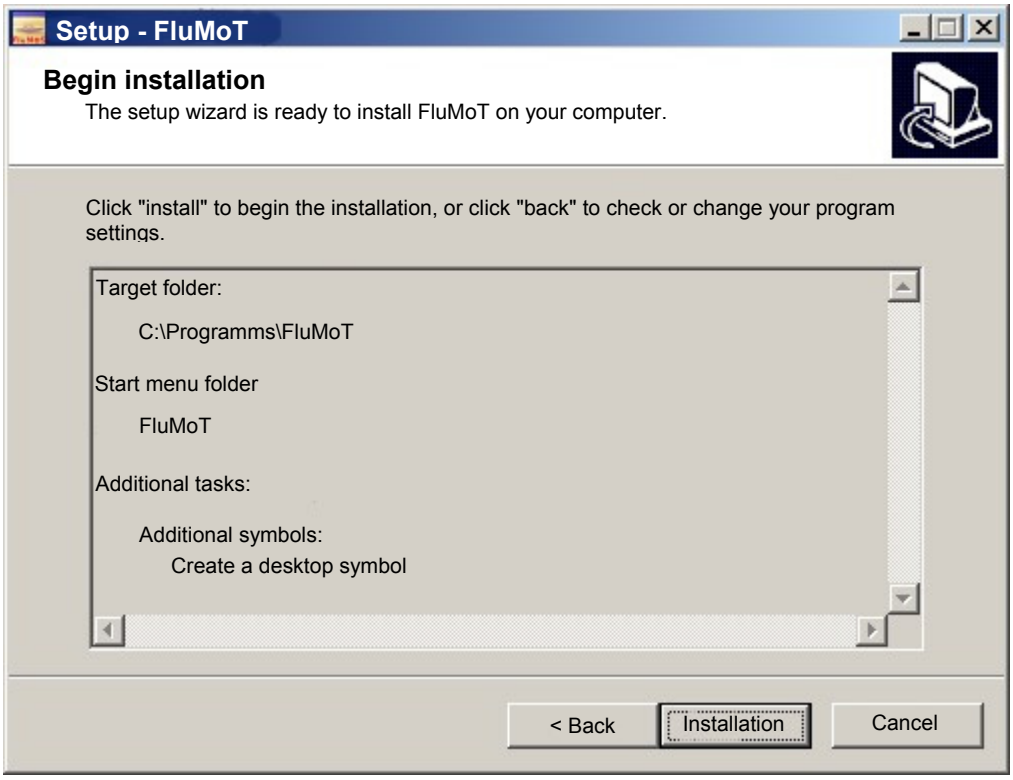
If the installation already exists, you must decide whether they have to be overwritten.



The start menu folder is then created.



The installation process is started after confirmation by clicking Continue >.



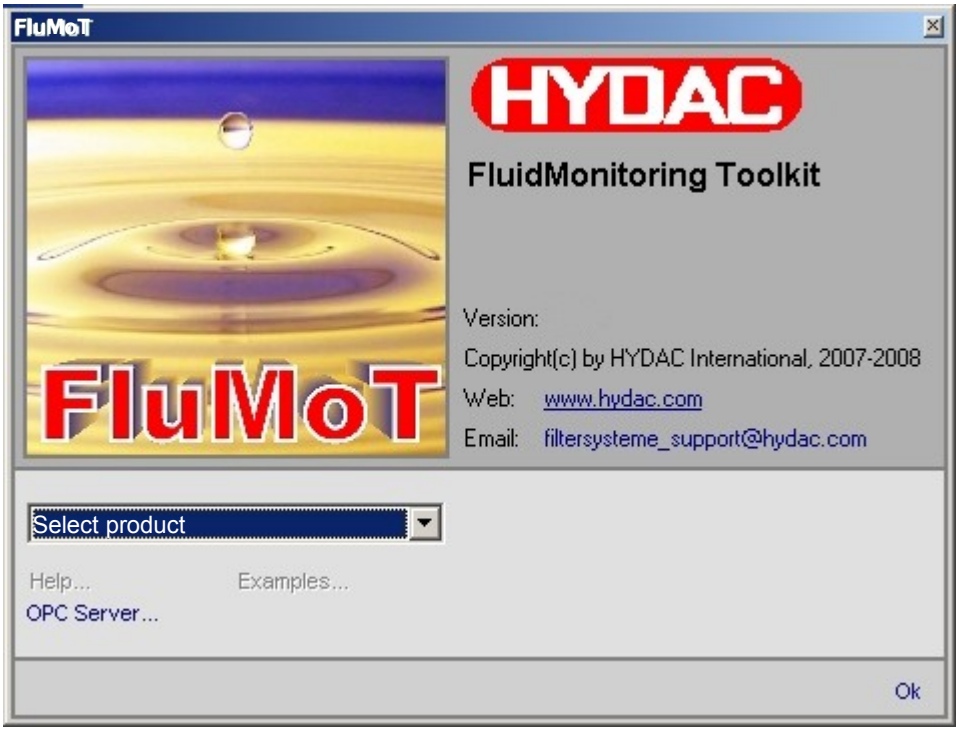
Click the "finish" button to close the setup wizard.

Uninstalling FluMoT

To uninstall FluMoT, run the UNINS000.EXE file located in the installation directory or launch uninstallation from the Start menu:

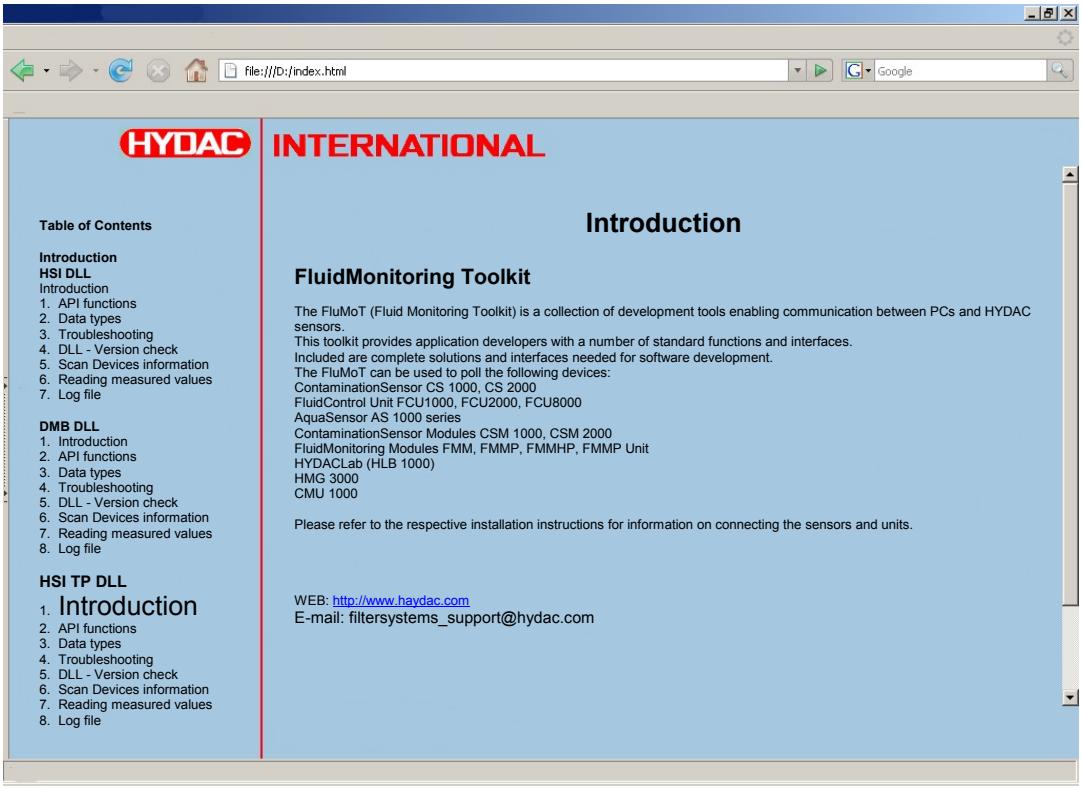
Using the FluMoT

The Toolkit starting window appears after the installation of **FluMoT**. Here one will find the selection of tools which are available in **FluMoT**. They are sorted in accordance with the HYDAC device families. All links in the program will be automatically adjusted accordingly. An entry to the starting window will appear in the Windows Start menu.



The starting window contains the links to the **FluMoT** HTML Help feature and to the corresponding folders, with examples on the hard disk.

The **FluMoT** HTML help provides an extensive description of all Toolkit components. An Internet browser (Internet Explorer ≥ 4.0) is required to open the online help.



One component of the **FluMoT** is the program libraries.

These DLLs provide the interfaces for communication with different HYDAC sensors. These sensors can be addressed with various protocols.

FluMoT contains the following three DLLs:

DLL	Description in Chapter	Page
dinmessbus32.dll	DIN Measurement Bus - DLL	18
hecom32.dll	HSI– DLL	34
hsitp32.dll	HSITP - DLL	54

To facilitate high-level language programming, simple examples are supplied as small projects in Delphi7, LabView 7 and Excel -Macros (VBA 6) in the source code. They can be found in the installation directory of **FluMoT**.

The same data types are used in all of FluMoT's program libraries. They are listed in the following table.

Model	*Description*	*Delphi*	*C/C++*	*VB/VBA*	*Labview*
Integer	Range: -2147483648 … 2147483647 Format: 32-bit, with digit sign	Integer	Integer	Long	Long
Double	Range: 5.0 x 10^-324 … 1.7 x 10^308 Format: 8 bytes	Double	Float	Double	Double
String	Represents a pointer to a Char value. The end of the string is terminated by a zero character. Format: 1 byte per character	PChar	PChar*	String	String (C String Pointer)

Sometimes several pieces of information are transferred with a single string. In this case a control character is used. It is a carriage return (ASCII code 13).
This character is marked as <CR> in the description.

All DLL functions in this manual are described in Pascal syntax. The following table provides an overview of the functions in dinmessbus32.dll.

DIN Measurement Bus - DLL

The Dinmessbus32.dll file contains interfaces designed to facilitate communication between a PC and the following HYDAC sensors:

- FCU2000 Series starting with Index G
- FCU8000 Series starting with Index G
- CS 2000 series

This program library contains the entire protocol layer in accordance with the criteria laid out in DIN 66348 part 2. It transforms the communication mechanisms of the DIN measurement bus into simplified function interfaces.

Several FCUs/CSs can be connected to a single BUS. For this reason, assign a separate address to each sensor, where it can be addressed. The address range according to DIN 66348 stretches from 1 to 31. Make sure each address is unique in the bus.

Consequently the maximum number of sensors per COM port is 31.

API Functions

All DLL functions in this manual are described in Pascal syntax. The following table provides an overview of the functions in *dinmessbus32.dll*.

Function	***Brief description***
GetDLLVersion_DMB	DLL - Version as number
GetDLLVersionText_DMB	DLL – Version as text
SearchBusDevice_DMB	Read SensorID (also in a bus system)
GetDeviceSerialNumber_DMB	Read serial number of device
GetDeviceSensorNumber_DMB	Read sensor number of device
GetDeviceCalibrationDate_DMB	Read date of last calibration
GetDeviceChannelCount_DMB	Read number of measurement channels
GetDeviceChannelInfo_DMB	Read measurement channel properties
SetBusAddress_DMB	Set bus address
SetMeasuringState_DMB	Start/stop measurement
GetDeviceState_DMB	Get device status
GetDeviceMeasuringValues_DMB	Reading measured values
GetDeviceLogListDataBlock_DMB	Read file folder from device
GetDeviceLogDataBlock_DMB	Read file
EraseDeviceLog_DMB	Delete file

Troubleshooting / error code

A number of DLL functions supply an error code in the event of an error. This error code can contain the following values.

General

State code	*Status text.*	*Description*
0	no error	No error. Device is ready for operation.
1	new measuring is done (no error!)	(new measured values are present)
2	filter contaminated	Filter contaminated
3	battery voltage too low	Battery charge too low
4	EXIN	Analog input error
5	Water warning	LED current at upper limit, cloudiness due to water, air etc. The LED may be defective.
6	Memory is full	Memory is full
7	BSU error	Signal from flow rate sensor is available

Communication with the device

State code	*Status text.*	*Description*
10	transmit error	Error while transferring data to device.
11	receive error	Error during transfer of data from device.
12	invalid mode	Wrong mode setting
13	invalid bus address	Wrong bus address.
14	invalid device model	Unknown device series
15	invalid channel index	Wrong channel number
16	no device found	No device found
17	protocol error	DIN measurement bus protocol error
18	com port error	COM Port is blocked
19	tx completed (no error!)	Transfer successfully completed
20	invalid fileID	FileID wrong
21	invalid file part	FilePart wrong
22	no channel active	No measurement channel active

Error messages from FCU / CS

State code	*Status text.*	*Description*
30	calibration values incorrect, fatal	Invalid calibration factors
31	constant parameter incorrect: serial no., sensor no., …	Parameter constant wrong (e.g. serial number)
32	normal parameter incorrect	Parameter variable wrong
33	error I²C - bus handling	I²C bus error
34	checksum in EEPROM incorrect	Wrong checksum in EEPROM
35	error in bus command: syntax	Syntax error in command
36	error in bus command: semantic	Semantics error in command
37	log memory incorrect	Log damaged
38	error in transmission log	Error in transfer protocol
39	error flow rate	Flow error
40	error ±VDD	Error ±VDD
41	error supply current particle sensor	Error, LED current, particle sensor
42	error power supply voltage	Battery charge too low

Status value in the log file

State code	*Status text.*	*Description*
50	error flow rate	Flow error
51	no flow	No flow
52	M3: limit reached	M3: limit reached
52	M4: limit reached	M3: limit reached
54	M4: measuring started	M4: measuring started
55	M4: test cycle time started	M4: measuring cycle running

Status Check

GetErrorStateText_DMB()

This function enables a matching English status message to be outputted on the basis of a status code.

Syntax:	**function** GetErrorStateText_DMB (State: **Integer**): **String**;
Parameters:	*State* – communication status.
Return value	Status message of sensor (English)
Response:	16: no device found

Version Check

GetDLLVersion_DMB()

This function can establish the library version (double value).

Syntax:	**function** GetDLLVersion_DMB(): **double**;
Return value	The version number is returned as a double-digit number.
Response:	1.1

GetDLLVersionText_DMB()

This function can establish the library version (string value).

Syntax:	**function** GetDLLVersionText_DMB(): **String**;
Return value	The version number and publication date are returned as text.
Response:	v1.01 09.11.2007

Serial interface

The DIN measurement bus system is based on the EIA RS 485 interface. The RS 485 is a serial interface. A so-called COM port must be opened in order to make transfer possible through this interface. Each subsequent function first opens a COM port, runs its routine and then closes the COM port. If one device is subjected to several queries, these circumstances will take a certain amount of time. That amount can be reduced if one carries out a one-time opening of a COM port (e.g. at the time of program start), runs all queries and then closes the COM port (e.g. when closing the program). The following 3 functions are used for working with one/several COM ports:

Syntax:	**procedure** OpenPort_DMB(PortNumber: **Integer**; **var** State: **Integer**);
	procedure ClosePort_DMB(PortNumber: **Integer**; **var** State: **Integer**);
	procedure CloseAllPorts_DMB();
Parameters:	*PortNumber* – The number of the COM port. *State* – Reference to the communication status variable.
Remark:	The Baud rate is permanently defined and is 9600 Baud.

Device search and device information

SearchBusDevice_DMB()

This function is used to search for a device at a specific COM port with a specific bus address.

Syntax:	**function** SearchBusDevice_DMB (PortNumber: **Integer**; Address: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	SensorID if search is successful.
Response:	CS2200 V04.01
Remark:	the number 4.01 descries the firmware version of the device

GetDeviceSerialNumber_DMB()

This function supplies the serial number of a device as a string.

Syntax:	**function** GetDeviceSerialNumber_DMB (PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	Serial number of device as string variable
Response:	406C120456

GetDeviceSensorNumber_DMB()

This function supplies the sensor number of a device as a string.

Syntax:	**function** GetDeviceSensorNumber_DMB(PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	Sensor number of the device as a string variable.
Response:	120456
Remark:	the sensor number is also contained in the serial number

GetDeviceCalibrationDate_DMB()

This function supplies the date of last calibration of a device.

Syntax:	**function** GetDeviceCalibrationDate_DMB(PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	Date of last calibration
Response:	2005-02-03

GetDeviceChannelCount_DMB()

This function supplies the number of measurement channels of a device.

Syntax:	**function** GetDeviceChannelCount_DMB(PortNumber, Address: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	Number of measurement channels
Response:	***5***
Remark:	From CS 2000: 4 channels with particle counts or contamination classifications and flow rate

GetDeviceChannelInfo_DMB()

This function supplies the measurement channel properties of a device. (e.g. channel name, unit of measurement, etc.)

Syntax: **function** GetDeviceChannelInfo_DMB(PortNumber, Address, ChNumber, Mode: **Integer**; **var** State: **Integer**): **String**;

Parameters: *PortNumber* – The number of the COM port.
Address – Bus address of the device as integer from 1 to 31.
ChNumber – Channel number (beginning from 0) State - reference to communication status variable.

Mode – Measurement data units (also used in GetDeviceMeasuringValues_DMB)

0 – particle counts (differential)
1 – NAS/SAE classes
2 – ISO code
3 – particle counts (cumulative)

The FCUs do not supply the ISO code as measured values in Mode 2 in all measurement channels (see overview of devices in Chapter Messkanal Übersicht)

Return value The answer consists of 5 sub-lines, divided with a separator. The structure of this type of answer is shown in the table below:

Line number	***Parameters:***	***Note***
1	Name	Channel name
2	Unit	Measuring range, units
3	Decimals	Decimal places All numbers are indicated as whole numbers. The parameter Decimals indicates the number of places after the decimal point in the figure. For example: LowerRange = -250, UpperRange = 1000 und Decimals = 1 one measurement range of –25.0 to 100.0.
4	LowerRange	Measurement range, lower limit
5	LowerRange	Measurement range, upper limit

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

SetBusAddress_DMB()

The following function causes a new bus address to be set in the device. If successful, the function outputs a new bus address as a whole number, otherwise it outputs the old bus address.

Syntax:	**function** SetBusAddress_DMB (PortNumber, Address, NewAddress: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *NewAddress* – Desired new bus address for a device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	New bus address of the device as integer in the interval [1…31].
Example:	30 (new bus address)

In some devices, a reset or reboot is required (power supply on/off).

Reading measured values

SetMeasuringState_DMB()

This command is used to start or stop a measurement. You can also this command to reset the error status of the device, provided no fatal/serious error is present.

Syntax: **function** SetMeasuringState_DMB(PortNumber, Address, Mode: **Integer**; **var** State**: Integer): Integer**;

Parameters:
PortNumber – The number of the COM port.
Address – Bus address of the device as an integer from 1 to 31.
Mode – Mode designates the following actions:
0 – Start measurement
1 – Stop measurement
2 – Reset error status
State – Reference to the communication status variable.

Return value
Operating status
The first digit shows the measurement mode number:
1x --> M1, 2x --> M2, etc.
The second digit defines the exact status:
The following applies to M1 (Measure), M2 (Measure and switch), M3 (Filter to):
x0 Measurement off
x1 Wait for correct flow rate
x2 Measurement currently in progress

The following applies to M4 (Filter from to):
40 Measurement off
41 Wait for correct flow rate
42 Measurement currently in progress, testing for lower limit
43 Waiting time running
44 Waiting time timed out, wait for correct flow rate
45 Measurement currently in progress, testing for upper limit

Response: 20 (M2 measurement mode, measurement off)

GetDeviceState_DMB()

This command enables the current device status to be determined.

Syntax:	**function** GetDeviceState_DMB(PortNumber, Address: **Integer**; **var** State**: Integer): Integer**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as an integer from 1 to 31. *State* – Reference to the communication status variable.
Return value	Device status. Possible values: 0 – No error 1 – New measurement present 2 – Filter clogged 4 – Battery charge too low
Response:	1 (new measured values are present)

GetDeviceMeasuringValues_DMB()

This command enables the measured values to be requested and transferred. The measured value view has to correspond to the names of the measurement channels. (see GetDeviceChannelInfo_DMB) The return value is a string with measured values which is interpreted on the basis of channel properties.

Syntax:	**function** GetDeviceMeasuringValues_DMB(PortNumber, Address, Mode: **Integer**; const DeviceID: **String**; **var** State**: Integer): String**;
Parameters:	*PortNumber* – the number of the COM port. Address – Bus address of the device as an integer from 1 to 31. Mode – Measurement data units (also used in GetDeviceChannelInfo_DMB function) DeviceID – Result of the function „SerachBusDevice_DMB“ (for example: „CS2200 V04.01“) 0 – particle counts (differential) 1 – NAS/SAE classes 2 – ISO code 3 – particle counts (cumulative) *State* - reference for communication status variable.
Return value	Measured values
Remark:	In the mode 1 means the measuring value '-1' the NAS class '00'. The SAE class '00' and '000' also with the help of negative figures marked.
Response:	7680<CR>1860<CR>5<CR>0<CR>122<CR>0<CR>0<CR>

Reading out files

We use block commands to read out large amounts of data from the sensor system memory. In doing so, all information to be transmitted is distributed among smaller individual packages. Such individual packages are transmitted by the sensor and saved in a buffer. The corresponding functions usually supply only part of the entire information and include the word "Block" in their names.

A string variable is used as a buffer in example programs. It must however be taken into account that hte the amount of information to be transferred may be very large. It is for that reason that the program must tak into consideration the maximum string size of the respective programming language in order to avoid an overflow.

One sensor file is always made up of 2 parts: heading data and measurement data.

The heading data contain the structure and the size of the measurement data. The actual measurement data are as a rule very large in size and are evaluated on the basis of the heading data. They are always transferred explicitly.

The operations with files are not supported by all HYDAC sensors. Detailed information can be found in the respective operating instructions for the devices/sensors.

GetDeviceLogDirectory_DMB()

The file folder of the sensor is read with this function.

Syntax:	**GetDeviceLogDirectory_DMB**(PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *State* - reference to the communication status variable.
Return value	Pointer to a character variable that contains the file folder of the sensor.
Example:	Response (in case only one protocol is available in the memory): „1<CR>HYDAC FCU 8110<CR>5<CR>14.12.2006 10:01<CR>“ The meaning of the individual entries will be explained in greater detail in the following Table.

Line number	***Parameters:***	***Note***
1	FileID	Identifies a file unambiguously. This ID will be used for the later file operations
2	Measuring point	Measuring point
3	RecordCount	Number of data records in the file
4	FileDate	Create date of file

GetDeviceLogHeader_DMB()

This function is used to read recording information (file heading). This involves the structure of the data in the sensor. This information is considered a prerequisite for the correct evaluation of measurement data.

Syntax:	**GetDeviceLogHeader_DMB** (PortNumber, Address, FileID, Mode: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – the number of the COM port *Address* – bus address of the device in ASCII code FileID – File – Identificator (see GetDeviceLogDirectory_DMB) *Mode – Measurement data units (also used in "GetDeviceChannelInfo_DMB" function)* 0 – Particle numbers (differential) 1 – NAS/SAE classes 2 – ISO code 3 – particle counts (cumulative) *State* - reference for communication status variable.
Return value	Pointer to a character variable with file heading information.
Example:	Return value „7<CR><CR>1<CR>3<CR>2006-12-14 14:36:00<CR>0<CR> The meaning of the individual entries will be explained in greater detail in the following Table.

Line number	***Parameters:***	***Note***
1	ChannelCount	Number of measurement channels
2	HasTimeStamps	Timing of the measured values (0/1), change of minutes with 1)
3	RecordCount	Number of data records
4	StartDate	Timestamp: Start measurement or 0
5	StopDate	Timestamp: Stop measurement or 0
(for measurement channel 1)		
6	Name	Channel name
7	Unit	Measuring unit
8	Decimals	Numbers of decimal places
9	LowerRange	lower measurement limit
10	LowerRange	upper measurement limit
(possibly for measurement channel 2)		
17	Name	
18	Unit	
…	…	
…	…	
24	…	

GetDeviceLogDataBlock_DMB()

The measurement data are read with this function. The file heading must first always be read in order to know the structure of these data.

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

Parameters: *PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign.
PortNumber – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign.
FileID – File – Identificator (see GetDeviceLogDirectory_DMB)
Mode – Measurement data units (also used in "GetDeviceChannelInfo_DMB" function)
0 – Particle numbers (differential)
1 – NAS/SAE classes
2 – ISO code
3 – particle counts (cumulative)
Offset – reference to a variable that contains the current position in the sensor memory. The content of this variable equals 0 at the start and will be used internally in the DLL.
State - reference for communication status variable.

Return value Pointer to a character variable. A part of the measurment data from a log file. The following configuration always applies within a data record.

		Channel 1	***Channel 2***	***…***
[status]	[Timestamp]	Measured value 1	Measured value 2	…

The time stamp is optional. This value always equals 0 or 1. In this case, one means change very minute. All status codes can be found on page 18.

Example: Answer (possibly after several runs):

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

The meaning of the individual entries will be explained in greater detail in the following Table. This string will be evaluated on the basis of the file heading information.

Examples for the evaluation of the log data:

Log data set:

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

The following information can be obtained from the file heading, for example:

Parameters:	***Value***
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

The values are to be interpreted as follows for this reason:

	Value
Status	0
Timestamp	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

Deleting files

EraseDeviceLog_DMB ()

This function removes one log file from the device memory.

Syntax:	**function** EraseDeviceLog_DMB (PortNumber, Address, FileID: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign. FileID – File – Identificator (see GetDeviceLogDirectory_DMB) *State* – Reference to the communication status variable.
Return value	1 when the file is deleted successfully or 0 if there is an error
Response:	0 (error: see content of "State" variable)

HSI – DLL

Communication between digital sensors / devices and the analysis devices from HYDAC is implemented with the **H**YDAC **S**ensor **I**nterface (HSI).

The following sensors/evaluation devices are involved:

Sensor		Evaluation device
AquaSensor AS 1000 series	HSI	HMG 3000 series
HYDACLab HLB 1000 series		
ContaminationSensor CS 1000 series		CMU 1000 series
FluidControl Unit FCU 1000 series		

HSI is a 1-wire digital interface enabling sensors, measurement instrumentation and PCs to be linked. A sensor / device sends measured values over this interface to a connected evaluating processor unit (for example: a PC). The manner in which the data is packed is referred as the HeCom protocol..

The address range according to HECOM stretches from 97 to 122 (ASCII code of the characters: ′a′ … ′z′). This results in a maximum number of devices of 26 per COM interface. Each sensor must be assigned a unique address (′a′ … ′z′) to ensure the sensor can be addressed in the BUS.

If only one sensor is connected, the character '+' (ASCII code 43) can be used.

API functions

The following functions are provided by *hecom32.dll*:

Function	***Brief description***
GetDLLVersion_HSI	DLL - Version as number
GetDLLVersionText_HSI	DLL – Version as text
SearchOneDevice_HSI	Establish SensorID If no bus system exists
SearchBusDevice_HSI	Establish SensorID in a bus system
GetDeviceChannelCount_HSI	Establish number of measurement channels
GetDeviceSerialNumber_HSI	Establish serial number
GetDeviceSerialNumber_HSI	Measurement channel - establish characteristics
GetBusAddress_HSI	Establish bus address
SetBusAddress_HSI	Set bus address
GetDeviceChannelsMask_HSI	Read out structure of measured values
GetDeviceMeasuringValues_HSI	Read out measured values
GetDeviceState_HSI	Establish sensor status
GetDeviceLogDirectoryBlock_HSI	Read folder log
GetDeviceLogHeaderBlock_HSI	Read heading information log
GetDeviceLogDataBlock_HSI	Read contents log (measured values)
EraseDeviceLog_HSI	Remove log from sensor memory

Troubleshooting

Nearly all functions display an error code in the case of a malfunction. This error code can have the following values:

State code	*Status text.*	*Description*
0	no error	No error. Device is ready for operation.
1	transmit error	Error during transmission of data to the device
2	receive error	Error during data transmission from the device
3	too much devices	Too many devices found
4	search error	Error in sensor ID of device
5	no channels	No measurement channel active
6	invalid channel index	Incorrect channel number
7	invalid checksum	Wrong check sum
8	com port blocked	COM Port is blocked
9	invalid channels mask	Wrong "Device mask"
10	no device found	No device found
11	protocol error	HSI protocol error
12	invalid device	Wrong device
13	multipacket tx not supported	Multipacket transfer is not supported
14	no logs supported	Files are nor supported
15	tx completed	Transfer successfully ended
16	no logs found	No file found
17	invalid FileID	FileID wrong
18	invalid FilePart (0 -> fileheader, 1 -> measurement data)	FilePart wrong
19	no smart sensor	No Smart - Sensor
20	invalid log mask	File mask wrong

Status Check

GetErrorStateText_HSI()

This function enables a matching English status message to be outputted on the basis of a status code.

Syntax:	**function** GetErrorStateText_HSI(State: **Integer**): **String**;
Parameters:	*State* – communication status.
Return value	Status message of sensor (English)
Example:	10: no device

DLL - Version check

GetDLLVersion_HSI()

This function can establish the library version.

Syntax:	**function** GetDLLVersion_HSI(): **double**;
Return value	The version number is returned as a double-digit number.
Example:	1.03

GetDLLVersionText_HSI()

This function can establish the library version.

Syntax:	**function** GetDLLVersionText_HSI(): **String**;
Return value	The version number and publication date are returned as text.
Example:	v.1,03 03.07.2007

Serial interface

A so-called COM port must first be opened when HSI is operated through the RS 485 interface. Each subsequent function first opens a COM port, runs its routine and then closes the COM port. If one device is subjected to several queries, these circumstances will take a certain amount of time. That amount can be reduced if one carries out a one-time opening of a COM port (e.g. at the time of program start), runs all of one's queries and then closes the COM port (e.g. when closing the program). The following 4 functions are used for working with one/several COM ports:

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();
Parameters:	*PortNumber* – The number of the COM port. *State* - reference to the communication status variable.
Remark:	The Baud rate of 9600 Baud will be used in the default settings.

Device search and device information

SearchOneDevice_HSI

This function is used to search for a device at a specific COM port without a BUS. It has to be ensured that **one, and only one**, device is connected to a COM port.

Syntax:	**function** SearchOneDevice_HSI(PortNumber: I**nteger**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – the number of the COM port. *State* – Reference to the communication status variable.
Return value	SensorID if search is successful.
Example:	A response can look like the following: "CS 1320 V02.21", in which the number 2.21 designates the firmware version of the sensor.

SearchBusDevice_HSI()

This function is used to search for a device at a specific COM port with a specific bus address.

Syntax:	**function** SearchBusDevice_HSI (PortNumber: **Integer**; Address: **Integer**; **var** State: Integer): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *State* - reference to the communication status variable.
Return value	SensorID if search is successful.
Example:	An answer can look like the following: "AS 1000 V02.00", in which the number 2.00 designates the firmware version of the sensor.

GetDeviceChannelCount_HSI()

This function shows the number of channels in a device.

Syntax:	**function** GetDeviceChannelCount_HSI (PortNumber, Address: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *State* - reference to the communication status variable.
Return value	Number of measurement channels
Response:	10 (from CS 1000)

GetDeviceSerialNumber_HSI()

This function shows the serial number of a device.

Syntax:	**function** GetDeviceSerialNumber_HSI(PortNumber, Address: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *State* - reference to the communication status variable.
Return value	Serial number of device as string variable
Example:	4711

GetDeviceChannelInfo_HSI()

This function establishes the channel characteristics in a device. (e.g. channel name, measurement units, etc.)

Syntax:	**function** GetDeviceChannelInfo_HSI(PortNumber, Address, ChNumber: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign. ChNumber - channel number. (beginning from 0) State - reference for communication status - variable.
Return value	The answer consists of 5 sub-lines, divided with a separator. The structure of this type of answer is shown in the table below:

Line number	***Parameters:***	***Notice***
1	Name	Channel name
2	Unit	Measuring range, units
3	Decimals	Decimal places All figures are represented by whole numbers. The parameter Decimals indicates the number of places after the decimal point in the figure. For example, the following figure means: LowerRange = -250, UpperRange = 1000 and Decimals = 1 a measurement range of –-25.0 to 100.0.
4	LowerRange	Measurement range, lower limit
5	LowerRange	Measurement range, upper limit

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

Administrating bus addresses

Not all HSI sensors support the following two commands.

GetBusAddress_HSI()

This function shows the bus address of a device.

Only one device may be connected to the COM port.

Syntax:	**function** GetBusAddress_HSI(PortNumber: I**nteger**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – the number of the COM port. *State* – Reference to the communication status variable.
Return value	Bus address of the device as ASCII - code of the sign.
Example:	97 (character ‘a’)

SetBusAddress_HSI()

Sets a new bus address in a sensor.

Syntax:	**function** SetBusAddress_HSI (PortNumber, Address, NewAddress: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *NewAddress* – New bus address desired of the device as the ASCII code of the character. *State* - reference to the communication status variable.
Return value	New bus address of the device as the ASCII code of the character. If successful, the new address is returned, otherwise the old one is returned.
Example:	98 (new bus address is 'b')

Reading out measured values

GetDeviceChannelsMask_HSI()

This command is used to determine how the measured values are composed, e.g. whether they are 8-bit or 16-bit numbers. Execute this command only once to allow the "device mask" to be reused.

Syntax:	**function** GetDeviceChannelsMask_HSI(PortNumber, Address: **Integer**; **var** State**: Integer): String**;
Parameters:	*PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign. *State* – Reference to the communication status variable.
Return value	Pointer to a character variable, referred to as a "device mask"
Response:	***3 7*** 0 0 2 4 2

The structure of a "device mask" is shown in the following table:

Line number	***Parameters:***	***Notice***
1	ChannelCount	Number of measurement channels
2	ActivityMask	A bit is assigned to each channel to show whether or not the the channel is active
3	MinMask	A bit is assigned to each channel to show whether or not the channel possesses min. values
4	MaxMask	A bit is assigned to each channel to show whether or not the channel possesses max. values
5	DataSize, Channel 1	Data size in first measurement range
6	DataSize, Channel 2	Data size in second measuremetn range
7	…	… (for every channel)

The parameter "DataSize" can only have the values 1, 2 or 4. These values correspond to 8, 16 or 32-bit values.

The "ActivityMask" shows which channels are actually active. No data is transmitted from non-active channels during the transfer of measured data. Bit 0 shows whether channel 0 is active, bit 1 whether channel 1 is active, etc.

MinMask and MaxMask specify whether there is also a min. value and/or max. value for the respective measured value. Here, too, bit 0 corresponds to channel 0. If a channel is not active, the min. and max. values are not active either. This means that a minimum or maximum value may not occur when there is no current measured value.

GetDeviceMeasuringValues_HSI()

This command is used to request and transmit the measured values. Determine the composition of the measured values first with the "GetDeviceChannelsMask_HSI" command. You receive the device mask as response. You have to confirm it each time with the "GetDeviceMeasuringValues_HSI" command. The reason for this is the option to adapt the structure of the measured values dynamically during operation.

The structure of the "device mask" for GetDeviceChannelsMask_HSI is described above.

Syntax:	**function** GetDeviceMeasuringValues_HSI(PortNumber, Address: **Integer**; **const** DeviceChannelsMask: **String**; **var** State**: Integer): String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *DeviceChannelsMask* - "Device masks" *State* - Reference to communication status variable.
Return value	Measured values
Response:	127<CR>104<CR>80<CR>20<CR>21<CR>22<CR>19<CR>246<CR>100<CR>0<CR>

Interpreting measured values (examples)

1) Measured values from the device: 135<CR>47<CR>7<CR>

Device mask: 3<CR>7<CR>0<CR>0<CR>2<CR>4<CR>2<CR>, thus:

Parameters:	***Value***		
ChannelCount	3		
	Bit 0 Channel 0	Bit 1 Channel 1	Bit 2 Channel 2
ActivityMask	1	1	1
MinMask	0	0	0
MaxMask	0	0	0
DataSize, Channel 1	2		
DataSize, Channel 2		4	
DataSize, Channel 3			2

The values are to be evaluated as follows:

	Value
Channel 1	135
Channel 2	47
Channel 3	7

2) Measured values from the device: 135<CR>47<CR>7<CR>

Device mask: 3<CR>6<CR>2<CR>0<CR>2<CR>4<CR>2<CR>, thus:

Parameters:	***Value***		
ChannelCount	3		
	Bit 0 Channel 0	Bit 1 Channel 1	Bit 2 Channel 2
ActivityMask	1	1	0
MinMask	0	1	0
MaxMask	0	0	0
DataSize, Channel 1	2		
DataSize, Channel 2		4	
DataSize, Channel 3			2

The values are to be evaluated as follows:

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

GetDeviceState_HSI()

The sensor status is used to determine whether the connected device is operational, or has entered into a fault state. The sensor status has the following structure:

- 8-bit status byte
- 16-bit status code or error code (with leading sign)
- Optional status text

The *status byte* shows the current status of the device. The individual statuses can be specified with the following status code.

The following values are defined for the status byte:

0:	Ready for operation	No active fault present, device is operational.
1:	Standby	No active fault is present, however the device is currently not operational. Individual device functions may have been switched off or the device is in a startup phase, etc.
2:	Minor error	A minor fault is present which can be acknowledged.
3:	Moderate error	A moderate fault is present which may be remedied by switching the unit on/off.
4:	Serious error	A serious fault is present; the unit must be sent in to the manufacturer.

The *status code* specifies the current status. It is a 16-bit value. The exact meaning varies from device to device. The user should consult the manual for more information on the status code.

The *status text* is optional and consists of a maximum of 32 characters. It is used to enable a control device to display the status of a sensor in plain text.

Syntax:	**function** GetDeviceState_HSI(PortNumber, Address: **Integer**; **var** StateByte, StateCode, State: **Integer**): **String**;
Parameters:	*PortNumber* – The number of the COM port. *Address* – Bus address of the device as the ASCII code of the character. *StateByte* – Status byte. *StateCode* – Status code. *State* - reference to the communication status variable.
Return value	Status text.
Response:	ASIC-CRC-Error, *StateByte* = 3, *StateCode* =17

Exporting files

Due to the large data volumes, what are referred to as block commands are used to read out the sensor system memory. All the information to be transmitted is distributed among smaller individual packages. Such individual packages are transmitted by the sensor and saved in a buffer. The corresponding functions usually supply only part of the entire information and include the word "Block" in their names.

The operations with files are not supported by all HYDAC sensors. Detailed information can be taken from the respective operating instructions.

A string variable is used as a buffer in example programs. It must however be taken into account that hte the amount of information to be transferred may be very large. It is for that reason that the program must tak into consideration the maximum string size of the respective programming language in order to avoid an overflow.

One sensor file is always made up of 2 parts: heading data and measurement data. The heading data contain the structure and the size of the measurement data. The actual measurement data are as a rule very large in size and are evaluated on the basis of the heading data. They are always transferred explicitly.

GetDeviceLogDirectoryBlock_HSI()

The file folder of the sensor is read with this function.

Syntax:	**GetDeviceLogDirectoryBlock_HSI**(PortNumber, Address: **Integer**; **var** Offset, State: **Integer**): **String**;
Parameters:	*PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign. *Offset* – reference to a variable that contains the current position in the sensor memory. The contents of this variable equals 0 at the start and will be used internally in the DLL. *State* – Reference to the communication status variable.
Return value	Pointer to a character variable. A part of the folder list in the sensor.
Example:	Answer (possibly after several runs): 1<CR>0<CR>HLB1000 - LOGDAT<CR>1<CR>0<CR> The meaning of the individual entries will be explained in greater detail in the following Table.

Line number	Parameters	Remark
1	FileID	Identifies a file unambiguously. This ID will be used for the later file operations
2	FileType	Data type: 0 = Log – File 1 = Configuration file:
3	FileName	FileName (max. 32 characters)
4	FileNumber	Number of the measurement in the sensor
5	FileDate	Create date of file

GetDeviceLogHeaderBlock_HSI()

This function is used to read recording information (file heading). This involves the structure of the data in the sensor. This information is considered a prerequisite for the correct evaluation of measurement data.

Syntax:	**GetDeviceLogHeaderBlock_HSI** (PortNumber, Address, FileID: **Integer**; **var** Offset, State: **Integer**): **String;**
Parameters:	*PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign. *FileID* – Datei – Identifikator (see GetDeviceLogDirectoryBlock_HSI) *Offset* – reference to a variable that contains the current position in the sensor memory. The contents of this variable equals 0 at the start and will be used internally in the DLL. *State* – Reference to the communication status variable.
Return value	Pointer to a character variable. One part of the file heading in the sensor.
Example:	Answer (possibly after several runs):

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>

The meaning of the individual entries will be explained in greater detail in the following Table.

Line number	Parameters	Remark
1	ChannelCount	Number of measurement channels
2	HasTimeStamps	Timing of the measured values (0/1)
3	HasStates	Status to the measured value (0/1)
4	HasMinMax	Min/Max – value(0/1)
5	RecordCount	Numbers of records
6	Sample rate	Scanning rate (as a multiple of 100 µs) or 0
7	StatusCodeId	Coding of the status specification
8	PreTriggerCountr	How many data records are positioned before a trigger event. The trigger result is always shown on the time axis as 0.
(for measurement channel 1)		
9	LowerRange	lower measurement limit
10	LowerRange	upper measurement limit
11	LowerRawRange	May have to be converted
12	UpperRawRange	possible conversion
13	Name	Channel name
14	Unit	Measuring unit
15	Decimals	Numbers of decimal places
16	DataSize	Datasize
(possibly for measurement channel 2)		
17	LowerRange	lower measurement limit
18	LowerRange	upper measurement limit
19-24	…	
(etc. for all measurement channels)		
8 + 8 * ChannelCount + 1	InfoText	Comment

GetDeviceLogDataBlock_HSI()

The measurement data are read with this function. The file heading must first always be read in order to know the structure of these data.

Syntax: **GetDeviceLogDataBlock_HSI** ()PortNumber, Address, FileID: **Integer**; **const** LogChannelsMask: **String**;**var** Offset, State: **Integer**): **String**;

Parameters: *PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign.
FileID – file identifier (see GetDeviceLogDirectoryBlock_HSI)
LogChannelsMask – a string which describes the internal data structure. This mask can be either compiled from file heading information or empty. (Then the execution of the command will take a correspondingly longer time). The structure of this kind of a mask can be obtained from the following Table.
Offset – reference to a variable that contains the current position in the sensor memory. The contents of this variable equals 0 at the start and will be used internally in the DLL.

State – Reference to the communication status variable.

Return value: Pointer to a character variable. A part of the measurment data from a log file. The following configuration always applies within a data record.

		Channel 1		***Channel 2***		***…***	
[Timestamp]	[status]	Measured value 1	[min. value 1]	[Maxwert 1]	Measured value 2	[min. value 2]	[max. value 2]

The entries in brackets in this table are optional (see LogChannelsMask).

Example: Answer (possibly after several runs):

„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>“

The meaning of the individual entries will be explained in greater detail in the following Table. This string will be evaluated on the basis of the mask.

Line number	Parameters	Remark
1	ChannelCount	Number of measurement channels
2	HasTimeStamps	Timing of the measured values (0/1)
3	HasStates	This flag shows whether or not the current recording for each data record still contains a status number (0/1).
4	HasMinMax	This flag shows whether or not the minimum AND maximum (always together) measured values have also been saved separately to each channel (0/1).
5	DataSize1	Datasize measurement channel 1
6	DataSize2	Datasize measurement channel 2
7	…	(for each measurement channel)

The "DataSize" parameter can only be assigned the values 1, 2 or 4. These values correspond to 8, 16 or 32-bit values. They are intended only for internal use in the DLL.

Examples of the evaluation of log data:

1) Log data set: 0<CR>47<CR>7<CR>

Device mask: 2<CR>0<CR>1<CR>0<CR>2<CR>2<CR> (i.e. one data set = 2 numbers), thus:

Parameters	Value
ChannelCount	2
HasTimeStamps	0
HasStates	1
HasMinMax	0
DataSize, Channel 1	2
DataSize, Channel 2	2

Evaluate the responses as follows:

	Value
Status	0
Channel 1	47
Channel 2	7

2) Log data set: 13556<CR>47<CR>7<CR>56<CR>6<CR>1<CR>100<CR>

Device mask: 2<CR>6<CR>2<CR>0<CR>2<CR>4<CR>2<CR> (i.e. one data set = 7 numbers), thus:

Parameters	Value
ChannelCount	2
HasTimeStamps	1
HasStates	0
HasMinMax	1
DataSize, Channel 1	2
DataSize, Channel 2	2

Evaluate the responses as follows:

	Value
Timestamp	13556
Channel 1	47
Minimum Channel 1	7
Maximum Channel 1	56
Channel 2	6
Minimum Channel 2	1
Maximum Channel 2	100

Deleting files

EraseDeviceLog_HSI ()

This function removes one log file from the device memory.

Syntax:	**function** EraseDeviceLog_HSI (PortNumber, Address, FileID: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*PortNumber* – the number of the COM port. *Address* - bus address of the device as ASCII code of the sign. *FileID* – file identifier (see GetDeviceLogDirectoryBlock_HSI) *State* – Reference to the communication status variable.
Return value	FileID of the deleted file or -1 in case of error
Example:	Response: 10 (file 10 was deleted successfully)

HSITP - DLL

Communication between HYDAC digital sensors / devices and the respective evaluation units via modem und TCP-IP connection is effected with the aid of the HSI Text Protocol /HSI-TP). The protocol uses a purely peer-to-peer procedure (no bus addresses exist). It is a master-slave protocol, which means the master sends a packet and the slave answers with a packet.

API functions

All DLL functions in this manual are described in Pascal syntax. The following functions are provided by *hsitp32.dll*:

Functioning	***Brief description***
GetDLLVersion_HTP	DLL - Version as number
GetDLLVersionText_HTP	DLL – Version as text
OpenConnection_HTP	Set up connection with device
CloseConnection_HTP	Close connection
CloseAllConnections_HTP	Close all connections
SearchOneDevice_HTP	Establish designation of the unit
GetDeviceChannelCount_HTP	Establish number of measurement channels
GetDeviceSerialNumber_HTP	Establish serial number
GetDeviceChannelInfo_HTP	Measurement channel - establish characteristics
GetDeviceChannelsMask_HTP	Read structure of measured values
GetDeviceMeasuringValues_HTP	Reading measured values
GetDeviceState_HTP	Establish sensor status

Troubleshooting

Nearly all functions display an error code in the case of a malfunction. This error code can have the following values:

State code	Status text.	Description
0	no error	Device is ready
1	invalid IP address	IP – Address wrong
2	invalid port number	Port number wrong
3	no connection	No connection exists
4	invalid checksum	Checksum incorrect
5	no device found	No device found
6	protocol error	HSI – TP protocol error
7	invalid channel mask	Wrong "Device mask"
8	invalid sensor info	Sensor information inconsistent

Status Check

GetErrorStateText_HTP()

This function enables a matching English status message to be outputted on the basis of a status code.

Syntax:	**function** GetErrorStateText_HTP(State: Integer): PChar;
Parameters:	*State* – communication status.
Return value	Status message of sensor (English)
Response:	10: no device

DLL - Version check

GetDLLVersion_HTP()

This function can establish the library version.

Syntax:	**function** GetDLLVersion_HTP(): **double**;
Return value	The version number is returned as a double-digit number.
Response:	1.03

GetDLLVersionText_HSI()

This function can establish the library version.

Syntax:	**function** GetDLLVersionText_HTP(): **String**;
Return value	The version number and publication date are returned as text.
Response:	v.1,03 03.07.2007

Connecting the Ethernet interface

Before you start transmitting, first establish a connection to the interface. Each following function first opens the connection with host, performs its routine and closes the connection. Several requests of a device take a certain amount of time.

You can reduce the time by opening the connection to the device once (e.g. when starting the program) and executing all requests. The connection is then closed, for example when closing FluMoT.

The following three functions are used to establish/close a connection in the 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();
Parameters:	IpAddress – IP address of the device *PortNumber* – the number of the port. *State* – Reference to the communication status variable..
Remark:	The port number 5000 is required for communications with the CMU1000.

Device search and device information

SearchOneDevice_HTP()

This function searches for a device with a particular IP address.

Syntax:	**function** SearchOneDevice_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameters:	IpAddress – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255) *PortNumber* – the number of the port. *State* – Reference to the communication status variable.
Return value	SensorID if search is successful.
Example:	An answer can look like the following: "CMU1000 V00.20", in which the number 0.20 designates the firmware version of the sensor.

GetDeviceChannelCount_HTP()

This function shows the number of channels in a device.

Syntax:	**function** GetDeviceChannelCount_HTP (**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*IpAddress* – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255) *PortNumber* – the number of the port. *State* – Reference to the communication status variable.
Return value	Number of measurement channels
Response:	10 (from CS 1000)

GetDeviceSerialNumber_HTP()

This function shows the serial number of a device.

Syntax:	**function** GetDeviceSerialNumber_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameters:	IpAddress – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255) *PortNumber* – the number of the port. *State* – Reference to the communication status variable.
Return value	Serial number of device as string variable
Response:	4711

GetDeviceChannelInfo_HTP()

This function establishes the channel characteristics in a device. (e.g. channel name, measurement units, etc.)

Syntax: **function** GetDeviceChannelInfo_HTP(**const** IpAddress: **String**; PortNumber, ChNumber: **Integer**; **var** State: **Integer**): **String**;

Parameters: IpAddress – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255)
PortNumber – the number of the port.
ChNumber - channel number. (beginning from 0)
State - reference for communication status variable.

Return value: The answer consists of 5 sub-lines, divided with a separator. The structure of this type of answer is shown in the table below:

Line number	Parameters	Remark
1	Name	Channel name
2	Unit	Measuring range, units
3	Decimals	Decimal places All figures are represented by whole numbers. The parameter Decimals indicates the number of places after the decimal point in the figure. For example, the following figure means: LowerRange = -250, UpperRange = 1000 and Decimals = 1 a measurement range of –25.0 to 100.0.
4	LowerRange	Measurement range, lower limit
5	LowerRange	Measurement range, upper limit

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

Reading out measured values

GetDeviceChannelsMask_HTP()

This command enables you to determine the composition of the measured values, e.g. whether 8-bit or 16-bit figures are involved. This command should be run only once so that the "device mask" can continued to be used.

Syntax:	**function** GetDeviceChannelsMask_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State**: Integer): String**;
Parameters:	IpAddress – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255) *PortNumber* – the number of the port. *State* – Reference to the communication status variable..
Return value	Pointer to a character variable, the "device mask."
Response:	3<CR>7<CR>0<CR>0<CR>2<CR>4<CR>2<CR>

The structure of a "device mask" is shown in the following table:

Line number	***Parameters:***	***Notice***
1	ChannelCount	Number of measurement channels
2	ActivityMask	A bit is assigned to each channel to show whether or not the the channel is active
3	MinMask	A bit is assigned to each channel to show whether or not the channel possesses min. values
4	MaxMask	A bit is assigned to each channel to show whether or not the channel possesses max. values
5	DataSize, Channel 1	Data size in first measurement range
6	DataSize, Channel 2	Data size in second measurement range
7	…	for each channel

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.

The "ActivityMask" shows which channels are actually active. Inactive channels are not transferred as part of the measured values transfer. Bit 0 in the mask indicates whether channel 0 is active, bit 1 whether channel 1 is active, etc.

MinMask and MaxMask specify whether there is also a min. value and/or max. value for the respective measured value. Here, too, bit 0 corresponds to channel 0. If a channel is not active, the min. and max. values are not active either. This means that a minimum or maximum value may not occur when there is no current measured value.

GetDeviceMeasuringValues_HTP()

This command is used to request and transmit the measured values.

The composition of the measured values first has to be determined with the command GetDeviceChannelsMask_HTP. The response is the "device mask," which has to be confirmed by the command GetDeviceMeasuringValues_HTP each time. The reason for this is the possibility offered of dynamically adjusting the structure of the measured values during operation. The structure of the "device mask" for GetDeviceChannelsMask_HTP is described above.

Syntax:	**function** GetDeviceMeasuringValues_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **const** DeviceChannelsMask: **String**; **var** State**: Integer): String**;
Parameters:	*IpAddress* – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255) *PortNumber* – the number of the port. *DeviceChannelsMask* – "Device mask". *State* – Reference to the communication status variable..
Return value	Measured values
Response:	127<CR>104<CR>80<CR>20<CR>21<CR>22<CR>19<CR>24 6<CR>100<CR>0<CR>

GetDeviceState_HTP()

The sensor status query helps you to establish whether the connected device is ready for operation or in an error state.

The sensor status is designed as follows:

- 8-bit status byte
- 16-bit status code or error code (with leading sign)
- Optional status text

The *status byte* indicates the current status of the device. You can specify the individual statuses in detail with the following status code.

The following values are defined for the status byte:

0: Ready	No active error present. The device/sensor is ready for operation.
1: Standby	No active error present. However, the device/sensor is currently not ready for operation. Individual device functions may be switched off or the device may be in the starting phase etc.
2: Minor error	A minor error is present. Acknowledge the error.
3: Moderate error	A medium-sized error is present. Try to rectify the error by switching on/off the device/sensor.
4: Serious error	A serious error is present. Send the device/sensor to HYDAC.

The *status code* specifies the current status in detail. It is a 16-bit value. The exact meaning depends on the connected device/sensor. Further information in this respect can be taken from the instructions for the device/sensor.

The *status text* is optional and consists of a maximum of 32 characters. It is used to enable a control device to display the status of a sensor in plain text.

Syntax:	**function** GetDeviceState_HTP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** StateByte, StateCode, State: **Integer**): **String**;
Parameters:	IpAddress – IP Address of the device ("XXX.XXX.XXX.XXX", for which XXX is a number between 0 and 255) *PortNumber* – the number of the port. *StateByte* – StatusByte. *StateCode* – Statuscode. *State* – Reference to communication status variable.
Return value	Status text
Response:	ASIC-CRC-Error“, *StateByte* = 3, *StateCode* =17

Sensors of the CS 2000 series with Ethernet interface

The devices of the CS 2000 series are a special case with regard to communication. These devices can optionally have an Ethernet module. In this case, you can transmit data via the TCP/IP communication protocol. You require a few special commands to address the CS 2000 with Ethernet interface.

The GetDLLVersion_HTP and GetDLLVersionText_HTP commands can still be used for the DLL version check.

The connection to the device is controlled with the functions described above (OpenConnection_HTP, CloseConnection_HTP and CloseAllConnections_HTP). Use port number 49322 for communication with the CS 2000.

The following functions for communication with a CS 2000 are available in the file *hsitp32.dll*:

Function	***Brief description***
SearchOneDevice_CSTCP	Determine device designation
GetDeviceChannelCount_CSTCP	Determine number of measurement channels
GetDeviceSerialNumber_CSTCP	Determine serial number
GetDeviceChannelInfo_CSTCP	Determine measurement channel properties
SetMeasuringState_CSTCP	Start/stop measurement
GetDeviceMeasuringValues_CSTCP	Reading measured values
GetDeviceState_CSTCP	Determine sensor status

The Ethernet module is designed for the standardized cabling variant "10Base-T" and meets the criteria defined in IEEE 802.3 (bit rate: 10 Mbps, transmission medium: twisted pair (UTP/STP) cable).

Searching for devices and reading out device information

SearchOneDevice_CSTCP()

This function is used to search for a device with a certain IP address.

Syntax:	**function** SearchOneDevice_CSTCP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*IpAddress* – IP address of the device ("XXX.XXX.XXX.XXX", with XXX standing for a number between 0 and 255) *PortNumber* – the number of the port *State* – reference to communication status variable
Return value:	SensorID if search is successful.
Example:	A response may, for example, look like this: "CS2210 V03.25", with the number 03.25 designating the firmware version of the sensor.

GetDeviceChannelCount_CSTCP()

This function supplies the number of channels in a device.

Syntax:	**function** GetDeviceChannelCount_CSTCP(**const** IpAddress: **String**; PortNumber, Mode: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*IpAddress* – IP address of the device („XXX.XXX.XXX.XXX“, with XXX standing for a number between 0 and 255) *PortNumber* – the number of the port *Mode* – measurement data units (also used in the "GetDeviceMeasuringValues_CSTCP" function) 0 – Particle numbers (differential) 1 – NAS/SAE classes 2 – ISO – Code 3 – Particle numbers (accumulative) *State* – Reference to the communication status variable.
Return value:	Number of measurement channels
Response:	5

GetDeviceSerialNumber_CSTCP()

This function supplies the serial number of a device.

Syntax:	**function** GetDeviceSerialNumber_CSTCP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*IpAddress* – IP address of the device ("XXX.XXX.XXX.XXX", with XXX standing for a number between 0 and 255) *PortNumber* – the number of the port *State* – reference to communication status variable
Return value:	Serial number of the device as string variable
Response:	406C120456

GetDeviceChannelInfo_CSTCP()

This function is used to determine the channel properties in a device (e.g. channel name, measurement unit etc.).

Syntax:	**function** GetDeviceChannelInfo_CSTCP(**const** IpAddress: **String**; PortNumber, ChNumber, Mode: **Integer**; **var** State: **Integer**): **String**;
Parameters:	*IpAddress* – IP address of the device ("XXX.XXX.XXX.XXX", with XXX standing for a number between 0 and 255) *PortNumber* – the number of the port *ChNumber* – channel number (beginning with 0) *Mode* – measurement data units (also used in the "GetDeviceMeasuringValues_CSTCP" function) 0 – Particle numbers (differential) 1 – NAS/SAE classes 2 – ISO – Code 3 – Particle numbers (accumulative) *State* – Reference to the communication status variable.
Return value:	The response consists of five sublines, separated with a separator. The structure of such a response is displayed in the following table:

Line number	Parameter	Remark
1	Name	Channel name
2	Unit	Measurement range, unit
3	Decimals	Decimal places All numerical data are specified as integers. The Decimals parameter specifies the number of places behind the decimal point. For example, the numerical data: LowerRange = -250, UpperRange = 1000 and Decimals = 1 represent a measurement range of -25.0 … 100.0.
4	LowerRange	Measurement range, lower limit
5	UpperRange	Measurement range, upper limit
Example:	Temp<CR>°C<CR>2<CR>-2500<CR>10000<CR>	

Reading measured values

SetMeasuringState_CSTCP()

This command is used to start or stop a measurement. You can also use this command to reset the error status of the device, provided no serious/fatal error is present.

Syntax: **function** SetMeasuringState_CSTCP(**const** IpAddress: **String**; PortNumber, Mode: **Integer**; **var** State**: Integer): String**;

Parameters: *IpAddress* – IP address of the device ("XXX.XXX.XXX.XXX", with XXX standing for a number between 0 and 255)
PortNumber – the number of the port
Mode – mode, designates the following actions:

0 – Start measurement
1 – Stop measurement
2 – Reset error status

State – Reference to the communication status variable.

Return value: Operating state
The first place indicates the number of the measurement mode:
1x → M1, 2x → M2, etc.
The second place defines the exact status:
The following applies to M1 (Measure), M2 (Measure and switch), M3 (Filter to):
x0 Measurement off
x1 Wait for correct flow rate
x2 Measurement currently in progress

The following applies to M4 (filters from to):
40 Measurement off
41 Wait for correct flow rate
42 Measurement currently in progress, testing for lower limit
43 Waiting time running
44 Waiting time timed out, wait for correct flow rate
45 Measurement currently in progress, testing for upper limit

Response: Response: 20 (measurement mode M2, measurement off)

GetDeviceMeasuringValues_CSTCP()

This command is used to request and transmit the measured values.

Syntax:	**function** GetDeviceMeasuringValues_CSTCP(**const** IpAddress: **String**; PortNumber, Mode: **Integer**; **var** State**: Integer): String**;
Parameters:	*IpAddress* – IP address of the device ("XXX.XXX.XXX.XXX", with XXX standing for a number between 0 and 255) *PortNumber* – the number of the port *Mode* – measurement *data units* *0 – Particle numbers (differential)* *1 – NAS/SAE classes* *2 – ISO-Code* *3 – Particle numbers (accumulative)* *State* – Reference to the communication status variable.
Return value:	Measured values
Response:	127<CR>104<CR>80<CR>20<CR>100<CR>

GetDeviceState_CSTCP()

This command can be used to determine the current status of the device.

Syntax:	**function** GetDeviceState_CSTCP(**const** IpAddress: **String**; PortNumber: **Integer**; **var** State: **Integer**): **Integer**;
Parameters:	*IpAddress* – IP address of the device ("XXX.XXX.XXX.XXX", with XXX standing for a number between 0 and 255) *PortNumber* – the number of the port *State* – reference to communication status variable
Return value:	The possible error values can be taken from the chapter "DIN measurement bus DLL: troubleshooting"
Example:	Response: 1 (new measured values are present)

Examples

In order to facilitate high-level language programming with FluMoT DLLs, simple examples are supplied as small projects in Delphi7, LabView 7 and Excel macros (VBA 6) in source code.

These examples can be found in the following folder after installation:

[LW]:\…\FluMoT\Dlls\Examples

These are not finalized software products but rather small demo programs.

LabView example (EXE file) can only be executed if LabView Runtime Engine 7 is installed.

If no Runtime Engine 7 is installed, you can find a complete installation in the "Examples" directory.

OPC server interface

OPC (Openness, Productivity, Collaboration, previously OLE for Process Control) is a standardised interface for accessing process data. It is based on the Microsoft Standard DCOM and has been expanded for the requirements of automation data access. It is used primarily for reading measured values from the control unit. Clients are as a rule visualisations, programs for compiling operational data, etc. OPC servers are typically provided as drivers with with various field devices or sensors.

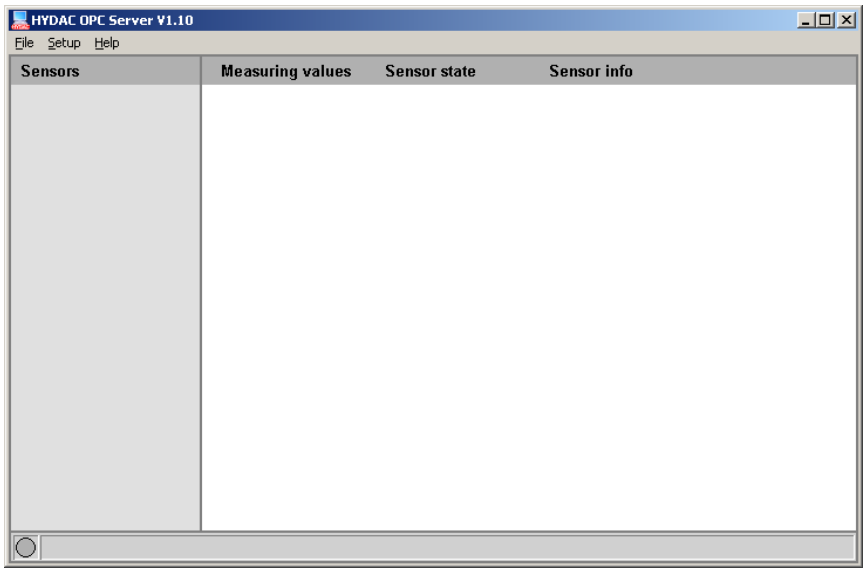
The OPC server is an executable program which is started when a connection is established between the client and sensor. The server collects all available measurement data of sensors and makes them available to the client as variables. Several clients can access these data simultaneously. The DCOM technology also enables access to a server running on a different PC. However, certain DCOM settings are required for this purpose. The DCOM safety settings are configured with the utility program "DCOMCNFG.EXE". Detailed information can be taken from the respective WINDOWS documentation.

HYDAC OPC Server V1.30 based on OPC Data Access 2.0 (specification for the transmission of real time values via Microsoft Standard DCOM). The server is copied on the PC during the FluMoT installation and registered in the system. The HYDAC OPC server is also removed during the deinstallation of FluMoT. You can also register/deregister manually using the supplied files *reg_opc.bat* und *unreg_opc.bat*.

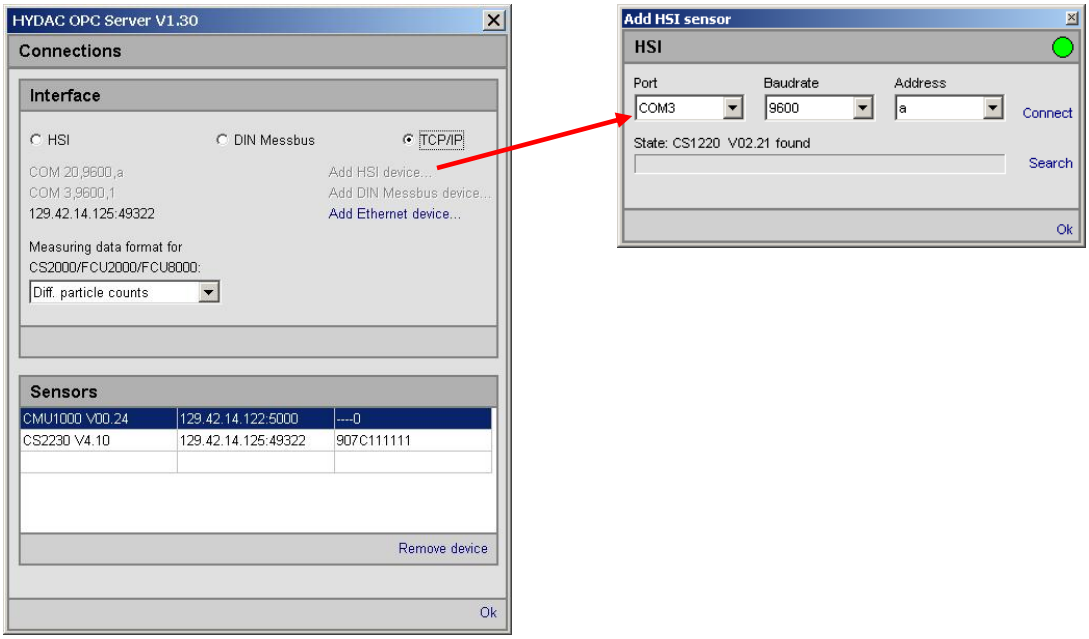
FluMoT communicates with the following devices/sensor types:

- **HSI Devices** (Serial communication by means of HSI protocol, different settings for Baud rate possible)
- **DIN Measurement Bus Devices** (Serial communication by means of DIN measurement bus)
- **HSITP Devices** (TCP/IP communication by means of HSI-TP protocol)
- **CS 2000 with an Ethernet interface**

Configure the HYDAC OPC server after the installation. In doing so, define the interfaces of all sensors to be queried in FluMoT.

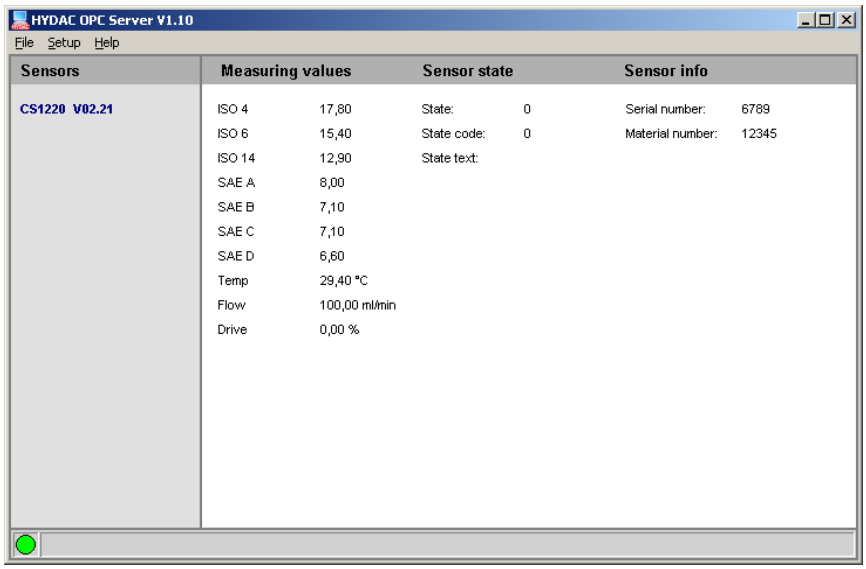


You can use the menu item SETUP -> CONNECTIONS in the main menu of the program to establish the connection to the individual sensors. The devices/sensors are added to a list. This list is saved by confirming with the OK key. The online measurement is then started.



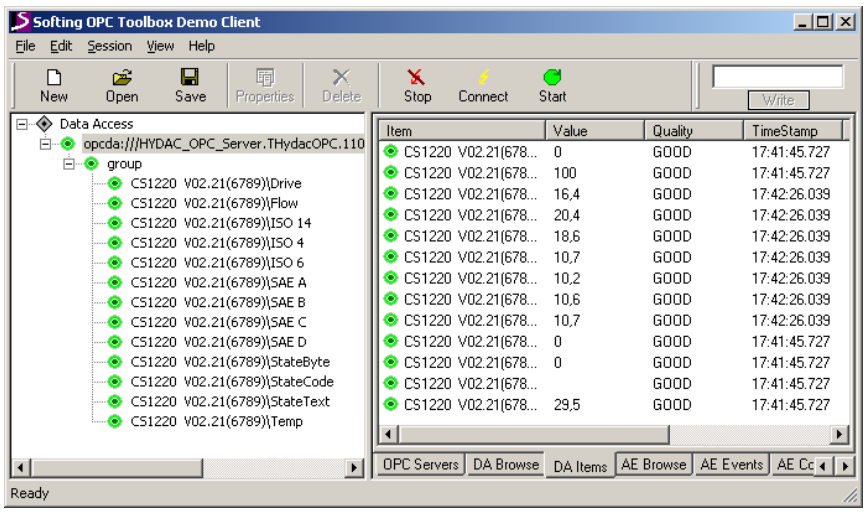
If a OPC client addresses the server, this connection is saved and always restored.

The number of sensors addressed in this way is unlimited. Observe that the transmission is slowed down considerably by communication with numerous devices/sensors.



The HYDAC OPC server can now communicate with an OPC client. Start one OPC Client. (e.g. Softing OPC Demo Client). Under the selection item "OPC Servers", select LOCAL DATA ACCESS V2 and create a connection to the "HYDAC OPC – Server V1.30". The OPC server will start automatically and begins to query all the devices from its list directly. This means that the measured values will be made available as quickly as possible. As soon as the measurement has been started, the program moves into the tray and runs in the background as a service.

The client application can now integrate the individual sensor channels (see the DA Browse registration tab in the Softing OPC Demo Client) and display/edit the corresponding measured values. (Registration tab DA Items)



Overview of Measurement Channels

The following table provides a listing of the measurement channels of various HYDAC sensors.

For the devices / sensors HLB1000, CMU1000 and HMG3000, refer to the overview of measurement channels in the respective operating instructions.

FCU 2000 measurement channel series

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

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

FCU 22xx						
	Particle counts		NAS/SAE class		ISO Code	
Channel 1	> 4 µm	[0…3200000]	SAE A	[-2…15]	ISO >4 µm	[0…25]
Channel 2	> 6 µm	[0…1250000]	SAE B	[-2…15]	ISO >6 µm	[0…25]
Channel 3	> 14 µm	[0…222000]	SAE C	[-2…15]	ISO >14 µm	[0…25]
Channel 4	> 21 µm	[0…39200]	SAE D	[-2…15]	ISO >21 µm	[0…25]
Channel 7	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
-	-	-	-	-	-	-

FCU 8000 measurement channel series

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

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

Messkanal CS 1000 Serie

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

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

AS 1000 measurement channel series

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

CS 2000 measurement channel series

CS 20xx

	Particle counts		NAS/SAE class		ISO Code	
Channel 1	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO > 5µm	[0…25]
Channel 2	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO > 15µm	[0…25]
Channel 3	25-50 µm	[0…129600]	NAS 25-50 µm	[-1…15]	ISO > 25µm	[0…25]
Channel 4	>50 µm	[0…23040]	NAS >50 µm	[-1…15]	ISO > 50µm	[0…25]
Channel 5	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
Channel 6	Analog 1 (with firmware version ≥ 4.00)					
Channel 7	Analog 2 (with firmware version ≥ 4.00)					

CS 21xx

	Particle counts		NAS/SAE class		ISO Code	
Channel 1	2-5µm	[0…20484000]	NAS 2-5 µm	[-1…15]	ISO > 2µm	[0…25]
Channel 2	5-15 µm	[0…4096000]	NAS 5-15 µm	[-1…15]	ISO > 5µm	[0…25]
Channel 3	15-25 µm	[0…729000]	NAS 15-25 µm	[-1…15]	ISO > 15µm	[0…25]
Channel 4	>25 µm	[0…129600]	NAS >25 µm	[-1…15]	ISO > 25µm	[0…25]
Channel 5	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
Channel 6	Analog 1 (with firmware version ≥ 4.00)					
Channel 7	Analog 2 (with firmware version ≥ 4.00)					

CS 22xx

	Particle counts		NAS/SAE class		ISO Code	
Channel 1	> 4 µm	[0…3200000]	SAE A	[-2…15]	ISO > 4µm	[0…25]
Channel 2	> 6 µm	[0…1250000]	SAE B	[-2…15]	ISO > 6µm	[0…25]
Channel 3	> 14 µm	[0…222000]	SAE C	[-2…15]	ISO > 14µm	[0…25]
Channel 4	> 21 µm	[0…39200]	SAE D	[-2…15]	ISO > 21µm	[0…25]
Channel 7	Flow ml/min	[0…800]	Flow ml/min	[0…800]	Flow ml/min	[0…800]
Channel 6	Analog 1 (with firmware version ≥ 4.00)					
Channel 7	Analog 2 (with firmware version ≥ 4.00)					

HYDAC FILTER SYSTEMS

HYDAC FILTER SYSTEMS GMBH

Industriegebiet
66280 Sulzbach/Saar
Germany

Postfach 1251
66273 Sulzbach/Saar
Germany

Phone:	+49 (0) 6897 509 01	Central
Fax:	+49 (0) 6897 509 846	Technical Department
Fax:	+49 (0) 6897 509 577	(Sales Department)
Internet:	www.hydac.com	
Email:	filtersystems@hydac.com	