PSUP

PyroScience Unified Protocol

REFERENCE MANUAL
VALID FOR FIRMWARE REVISIONS 4.10 - 4.19

TABLE OF CONTENT

INTRODUCTION

PyroScience offers a broad product range of instruments and sensor heads based on its Redflash technology for optical oxygen, pH, and temperature measurements. This document provides details on the communication interface of all devices based on the firmware generation 4, which was introduced in the year 2020. This firmware generation uses the "PyroScience Unified Protocol" (PSUP), which provides cross-platform compatibility between all different devices from PyroScience. So, a programmer used to e.g. the laboratory device FireSting-PRO can easily switch to basic OEM modules or even deep-sea devices.

The following devices (item nr.) are currently covered by this document:

- FireSting-O2 with firmware revision 4 (FSO2-C1, FSO2-C2, FSO2-C4)
  Note: The old generation of FireSting-O2 (FSO2-1, FSO2-2, FSO2-4) with firmware revision 3 is not supported by this manual.
- FireSting-PRO (FSPRO-1, FSPRO-2, FSPRO-4)
- AquapHOx Logger (APHOX-LX, APHOX-L-O2, APHOX-L-PH)
- AquapHOx Transmitter (APHOX-TX, APHOX-T-O2, APHOX-T-PH)
- AquapHOx Sensor (APHOX-S-O2)
- PICO-O2, PICO-PH, PICO-T, PICO-EXT
- FD-OEM-O2, FD-OEM-PH, FD-EXT

SUPPORTED INTERFACES

1.1 Serial Interface (UART) – PyroScience Protocol

All PyroScience devices possess a central microcontroller which handles all the different task the device has to fulfill. This microcontroller communicates via a UART interface with the outer world. A UART interface is a classical serial interface operated at low voltage levels, which can be directly connected to UART interfaces of other microcontroller boards (e.g. Raspberry Pi). The standard connector of basic OEM modules (e.g. PICO-O2, FD-OEM-PH) provides a UART interface. But also laboratory devices like the FireSting-PRO offer this UART interface at the extension port.

Note, several PyroScience devices do not provide a UART interface accessible for the user. They offer either a USB interface (1.2) or a RS485 interface (1.3), which are internally converted and connected to the UART interface of the microcontroller.

The UART interface of PyroScience devices is operated with 3.0 V or 3.3 V levels (3.0–5.0 V tolerant). The configuration is

19200 or 115200 baud, 8 data bit, 1 stop bit, no parity, no handshake

Please refer to the data sheet of the respective module about the actual baud rate.

Communication via the UART interface is always based on the proprietary PyroScience protocol (refer to 2.1).

Note: The serial interface of this module is not a RS232 interface. However, the UART interface can be made compatible to RS232 by third-party UART-RS232-adapters.

1.2 USB Interface – PyroScience Protocol

Many PyroScience devices possess a USB interface for easy operation e.g. at Windows PCs, All laboratory devices (e.g. FireSting-PRO) and several underwater devices (e.g. AquapHOx-LX) provide such a built-in USB port. Further, USB adapter cables are available for most basic OEM modules. These USB adapter cables are especially useful for evaluation purposes at Windows PC. PyroScience offers powerful software packages like the Pyro Workbench or the Pyro DeveloperTool. Please refer to the homepage of PyroScience for more details. Communication via the USB interface is always based on the proprietary PyroScience protocol (refer to 2.1.).

All USB ports and all USB adapter cables from PyroScience are based on solutions from the company FTDI. using the so called d2xx drivers, which are automatically installed

together with the PyroScience software packages. The d2xx driver can be used in two different ways: (a) via generic USB communication, or (b) via virtual COM ports. All PyroScience software uses the generic USB communication. But the virtual COM ports are very useful for custom made software, because most programming languages support virtual COM ports.

For advanced users: the USB String Descriptors of the FTDI USB interface is coded for all PyroScience devices and adapter cables as follows:

- FTDI Manufacturer = "Pyro"
- FTDI Product Description = either "Pyro_115200" or "Pyro_19200"

The number in the Product Description gives the default baud rate of the specific device.

1.3　RS485 Interface - Modbus Protocol

PyroScience offers devices with integrated RS485 interface (e.g. AquapHOx-TX), or it offers RS485 adapter modules for the UART interface of OEM modules (e.g. PICO-EXT in combination with PICO-O2). The default configuration is

19200 baud, 8 data bit, 1 stop bit, even parity, no handshake

The default communication protocol is the standardized Modbus RTU protocol. Please refer to the chapter 3.1 and following for more details. Note, all RS485 devices can be switched into a so called "transparent mode", where it is again possible to use the PyroScience protocol (2.1).

The advantages of RS485 in combination with the Modbus RTU protocol are:

- cable lengths up to several 100m are supported
- up to 247 devices can be connected to a 4-wire RS485 bus
- Modbus RTU is a popular communication protocol

1.4　Analog Output

There are several solutions available which offer analog outputs. Please refer to the respective data sheets for more details on the analog outputs. The general configuration of the analog outputs is described in sections 2.10 or 3.3.4.

PYROSCIENCE PROTOCOL

2.1 General Structure

The following figure shows the general structure of some representative PyroScience devices which use the proprietary PyroScience communication protocol:
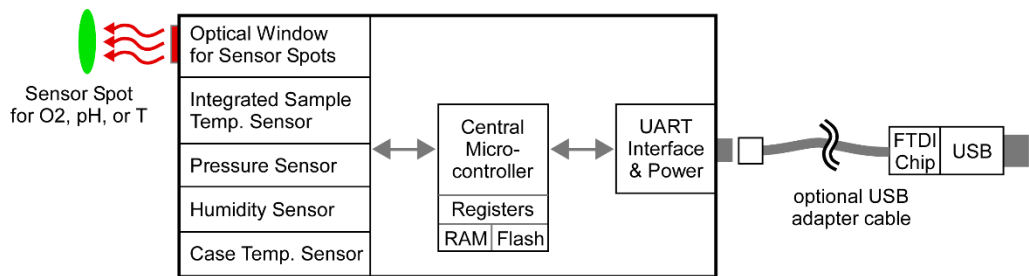
OEM Device FD-OEM-x
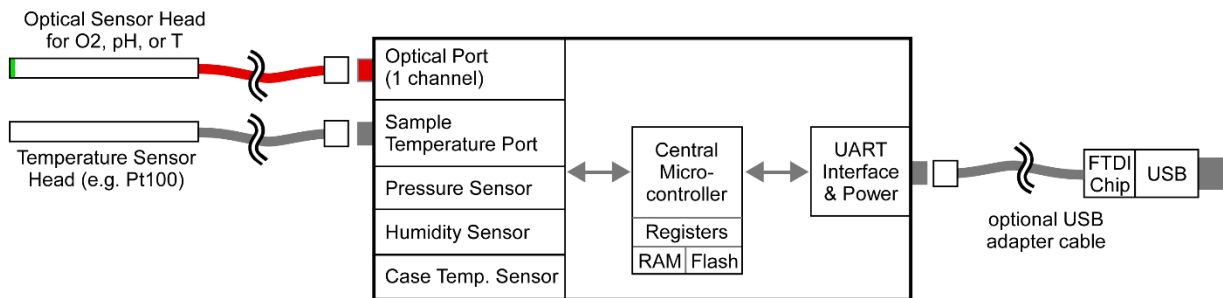
OEM Device PICO-x
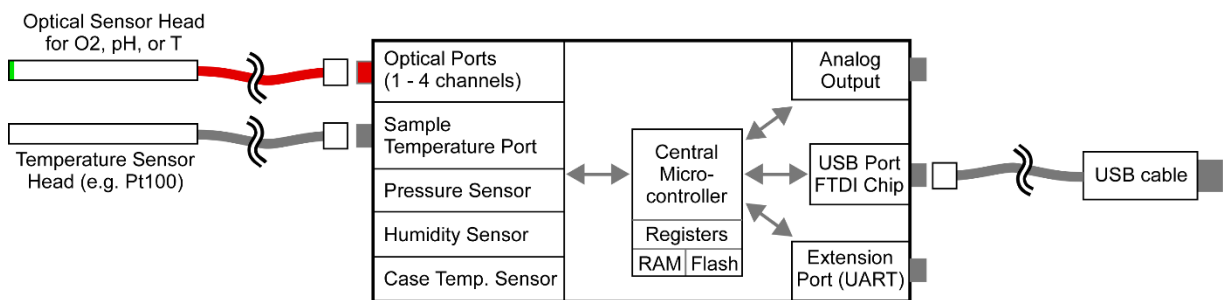
Laboratory Device FireSting
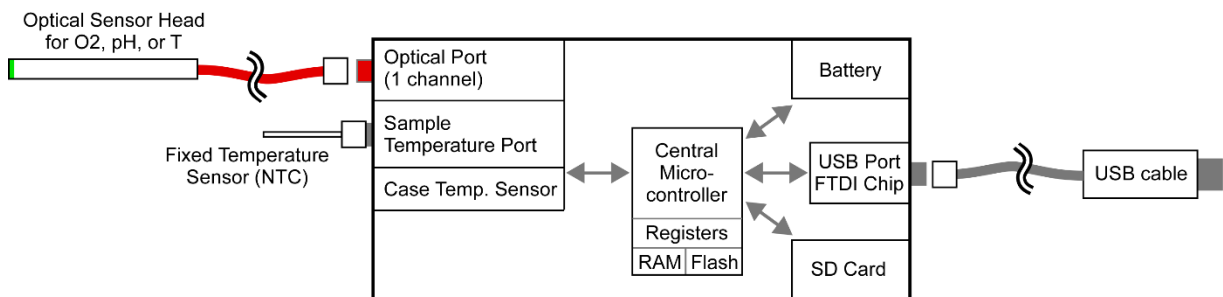
Underwater Device AquapHOx Logger

All devices possess a central microcontroller which communicates via its UART interface with the outer world. The UART interface is the standard interface for several OEM devices from PyroScience. There are optional USB adapter cables available, which convert the UART interface into a standard USB port. These adapter cables have integrated FTDI chips, which is a popular chipset for USB solutions.

Many end-user devices (e.g. laboratory or underwater devices) are equipped with a USB port, which is internally converted by a FTDI chip to a UART interface connected to the central microcontroller. Some end-user devices offer additionally an extension port giving direct access to the UART interface of the central microcontroller, which might be used by OEM applications.

A USB solution is always needed, if you want to use the powerful Windows software packages Pyro DeveloperTool or Pyro Workbench, provided for free download on the PyroScience homepage. Especially the Pyro DeveloperTool is strongly recommended for OEM developers for initial evaluation purposes. The Labview source code of the Pyro DeveloperTool is also available, providing a general reference how to implement PyroScience devices into other software or firmware projects.

The basic building blocks of the PyroScience protocol are outlined in the following.

2.1.1   General Command Structure

The commands provide the communication between the master (e.g. Windows PC) and the device (e.g. FireSting-PRO from PyroScience). Some commands trigger actions in the device, for example acquiring a data point with an optical oxygen sensor. Other commands are used for writing or reading registers within the device. A command comprises always the following sequence:

- the master sends the command string to the device
- the device executes the command
- the device sends the response string to the master
- the device is immediately ready to receive the next command

A command string always starts with a specific command header (e.g. MEA, #VERS, #LOGO) optionally followed by several input parameters. Input parameters are given as human readable decimal numbers, separated by spaces from each other. Each command must be terminated by a carriage return.

The response string starts always with an copy of the original command string including all input parameters, optionally followed by space separated output parameters, and again terminated by a carriage return.

There are two classes of commands: (a) device commands which affect the complete device (2.2), and (b) channel commands which trigger actions related to a specific optical channel (2.3).

If the internal processing of the received command causes an error within the module, the response will be the error header #ERRO followed by an error code "#ERRO" (2.4).

Style Conventions for Commands	
MEA **#VERS** **#LOGO**	Command headers are written in bold. They are transmitted unchanged as ASCII strings.
C S R	Single letters represent placeholders for parameters transmitted as human readable ASCII strings of decimal numbers. For example S might be replaced by 47, so the ASCII code for "4" followed by the ASCII code for "7" would be transmitted.
␣	Space (ASCII code 0x20), separator between parameters
↵	Carriage return (ASCII code 0x0D), final character of all messages.

2.1.2   General Register Structure

The registers comprise sets of integer numbers stored within the device. They are used for configuring the device, or for read access of the measurement results. A register always contains a 32bit integer number. Each optical channel has its own set of registers. The registers are organized in several "register blocks":

Register Block	Register Block Nr.	Number of Registers	Type	Comment
Settings	0	20	read / write	
Calibration	1	30	read / write	register definitions depend on the analyte detected by the sensor
Results	3	18	read only	
AnalogOutput	4	12	read / write	shared by all optical channels

Each register block consists of a defined number of registers containing signed 32bit-integer numbers. The first register address is always the number 0. The commands RMR (2.3.8) and WTM (2.3.9) are used for reading and writing registers, respectively.

The content of the read/write registers are kept at two locations: in the volatile RAM memory and in the flash memory of the devices. At power-up of the device, the register values from the flash memory are automatically loaded into the corresponding registers in the RAM memory. All further command actions are then based on the values in the RAM memory. The command SVS (2.3.10) can be used for saving the RAM memory content into the flash memory, i.e. the current register configuration is retained even after a power cycle.

Note, the register block *AnalogOutput* is shared by all optical channels. In this case it does not matter which optical channel is selected e.g. by the first parameter of the RMR-command.

Details on the register structure and the register definitions are given in chapter 0 and following.

Style Conventions for Registers	
Settings *Calibration* *Results* *AnalogOutput*	The name of register blocks are printed in italics with an initial capital letter.
intensity *pressure* *analyte* *tempSample*	The name of single register are printed in italics with an initial lowercase letter.
Settings.intensity *Calibration.pressure* *Results.tempSample*	A single register can be also printed in combination with the register block it belongs to. In this case, first the register block is printed before the register name, separated by a period character.

2.1.3   Broadcast Mode

A special feature is the optional broadcast mode, which can be configured and activated in the *Settings* registers (0). An enabled broadcast mode means, that the master does not need to trigger the measurements. Instead, the device triggers itself periodic measurements, or measurements are triggered by a digital input at the extension port of the device (if available). For more details refer to chapter 0.

2.1.4   Detecting Communication Errors and Optional CRC

The protocol supports two levels of communication error handling. The first level is realized simply by the fact, that the device always echoes the complete command as it was received from the master. This way the master can compare the echo with the originally sent request.

However, the echoing of the command does not allow detecting potential communication errors within the output parameters. For this purpose, it is possible to enable a cyclic redundancy check (CRC) for all responses from the device to the master. Refer to chapter 2.5.2 for more details. The reminder of this document shows only examples with disabled CRC.

2.1.5   Checklist for Programmers and Available Software Libraries

The following software libraries are available from PyroScience with an implementation of the PyroScience protocol:

- PyroScience Labview Library
- PyroScience Python Library

The following checklist contains the most important issues to be considered for an OEM implementation:

- ✓ Configure sensor specific constants registers marked with ⓣ. PyroScience offers many different sensor types (e.g. microsensors, flow-through cells, sensor spots) which require a specific configuration of many Settings and Calibration registers. It is adviced to configure the device initially with the software package Pyro DeveloperTool or Pyro Workbench. Based on the entered sensor code the software will adjust all sensor specific constants. This must be only done once for a specific sensor type (and not for each individual sensor head). Study for this the sections 0, 2.6, 2.7, and 2.8.
- ✓ Configure the sensor code registers marked with ⓢ. The respective information is given on the label attached to the sensor head. This must be configured for each individual sensor head. Study for this the sections 2.6, 2.7, and 2.8.
- ✓ Configure the Settings registers mode, intensity, and amp in order to find the best compromise between low long-term drift and high signal-to-noise ratio (2.5.2).
- ✓ Calibrate the sensor with the commands CHI (2.3.2) and CLO (2.3.3) for oxygen sensors, COT (2.3.4) for optical temperature sensors, and CPH (2.3.5) for pH sensors.
- ✓ Store the configuration permanently in the internal flash memory by using the command SVS (2.3.10).
- ✓ Use the command MEA for triggering measurements (2.3.1).
- ✓ Optionally, enable the broadcast mode, so the device might trigger itself periodic measurements (2.5.2).
- ✓ Optionally, configure the analog outputs in the *analogOutput* registers (2.10).

2.2 Device Commands

Device commands affect the complete device (i.e. they are not specific for one of the optical channels). Their command header always start with the character "#" followed by 4 letters.

2.2.1 #VERS – Get Device Information

This command returns general information about the device.

Command: **#VERS**↵
Response: **#VERS**␣D␣N␣R␣S␣B␣F↵

Output Parameters:

D   Device ID, identifies the specific device type.

Device ID	Description	Item Nrs.

0	FireSting-O2	FSO2-C1, FSO2-C2, FSO2-C4
1	FireSting-PRO	FSPRO-1, FSPRO-2, FSPRO-4
2-3	reserved	
4	Pico-x	PICO-O2, PICO-PH, PICO-T
5-7	Reserved	
8	FD-OEM-x	FD-OEM-O2, FD-OEM-PH
9-11	reserved	
12	AquapHOx Logger	APHOX-LX, APHOX-L-O2, APHOX-L-PH
13	AquapHOx Transmitter	APHOX-TX, APHOX-T-O2, APHOX-T-PH
14	AquapHOx Sensor	APHOX-S-O2

N   Number of optical channels.

R   Firmware version, e.g. R=410 designates firmware version 4.10

S   Bit field about available sensor types and supported optical analytes as follows:

Bit 0-7: Available Sensor Types	Bit 8-15: Supported Optical Analytes
Bit 0: optical channel	Bit 8: oxygen
Bit 1: sample temperature (typ. Pt100)	Bit 9: optical temperature
Bit 2: pressure	Bit 10: pH
Bit 3: humidity	Bit 11: CO2
Bit 4: analog in	Bit 12: reserved
Bit 5: case temperature	Bit 13: reserved
Bit 6: reserved	Bit 14: reserved
Bit 7: reserved	Bit 15: reserved

Example: S = 1 + 2 + 4 + 8 + 32 + 256 = 303 means, that the device supports an optical channel as well as sample and case temperature, pressure, and humidity sensors, and the optical channel supports the analytes oxygen.

B   Firmware build number starting at 1 for each firmware version (reflects minor firmware revisions)

F   Bit field about available features as follows:

Bit 0: analog out 1
Bit 1: analog out 2
Bit 2: analog out 3
Bit 3: analog out 4
Bit 4: user interface (display, buttons)
Bit 5: battery
Bit 6: stand-alone logging
Bit 7: sequence commands
Bit 8: user memory
Bit 9: ready for calibration free pH
Bit 10-31: reserved

Example: F = 1 + 2 + 4 + 8 + 256 = 271 means that 4 analog outputs are supported and the module possesses a user memory.

Example Communication:

Command: #VERS↵

Response: #VERS␣1␣4␣410␣1071␣2␣271↵

2.2.2   #IDNR – Get Unique ID Number

This command returns the unique identification number of the respective device.

Command: **#IDNR**↵

Response: **#IDNR␣**N↵

Output Parameters:

N   Unique ID number. Note, this parameter is given as an unsigned 64 bit integer!

Returns the unique identification number of the device (does NOT correspond to the serial number of the device).

Example Communication:

Command: #IDNR↵
Response: #IDNR_2296536137892833272↵

2.2.3   #LOGO – Flash Status LED

This command lets the status LED flash for 4 times within ca. 1 s.

Command: #LOGO↵
Response: #LOGO↵

This command can be used to check proper communication with the device. Or it might be helpful in setups with more than one device, in order to identify which port is connected to which device.

2.2.4   #PDWN – Power Down Sensor Circuits

This command switches off the power supply of the sensor circuits.

Command: #PDWN↵
Response: #PDWN↵

This command can be used for some power saving during idle operation periods. Note, that the sensor circuits are automatically powered up again, if the module receives any command (e.g. MEA) requiring a sensor measurement. This is also the case if a broadcast measurement takes place.

2.2.5   #PWUP – Power Up Sensor Circuits

This command switches on the power supply of the sensor circuits.

Command: #PWUP↵
Response: #PWUP↵

The wake-up duration is up to 250 ms.

2.2.6   #STOP – Enter Deep Sleep Mode

This command is only available on dedicated OEM modules (e.g. Pico-x or FD-OEM-x). It puts the device into a deep sleep mode with very low power consumption

Command: #STOP↵
Response: #STOP↵

During deep sleep mode the device has very low power consumption. No standard communication via USB/UART is possible. The deep sleep mode can be only exit by sending a <CR> (0x0D) to the device. The device will respond then also with a single <CR> indicating that it is ready to receive new commands. The wake-up duration can be up to 250 ms.

2.2.7   #RSET – Reset Device

This command triggers a reset of the device.

Command: #RSET↵

Response: #RSET↵

Triggers a reset of the device, as if the device experienced a power cycle. All current configurations in the RAM registers are overwritten with the values in the flash memory. This is not a factory reset.

2.2.8   #RDUM – Read User Memory

This command is only available on dedicated OEM modules (e.g. Pico-x or FD-OEM-x). It reads values from the user memory registers.

Command: #RDUM_R_N↵

Response: #RDUM_R_N_Y_0…Y_N↵

Input Parameters:

R   Address of first register to be read from the user memory (0…63)

N   Total number of registers to be read (1…64)

Output Parameters:

Y_0…Y_N Content of the requested user memory registers (signed 32bit integers).

The device offers a user memory of altogether 64 signed 32bit integer numbers (range -2147483648 to 2147483647) which is located in the flash memory and is therefore retained even after power cycles. This read command returns N (N=1…64) consecutive values Y_1 … Y_N from the user memory starting at the user memory address R (R=0…63). Note, that N+R must be <=64. The content of the user memory has no influence on the module itself. It can be used for any user specific purpose.

Example Communication:

Command: #RDUM_12_4↵

Response: #RDUM_12_4_-40323_23421071_0_-555↵

This example shows a command which requests the values of 4 consecutive beginning with the user memory address 12.

2.2.9 #WRUM – Write User Memory

This command is only available on dedicated OEM modules (e.g. Pico-x or FD-OEM-x). It writes values into the user memory registers.

Command: `#WRUM_R_N_Y₀…Y_N↵`
Response: `#WRUM_R_N_Y₀…Y_N↵`

Input Parameters:

R   Address of first user memory register to be written (0…63)

N   Total number of registers to be written (1…64)

$Y_0…Y_N$ Values to be written to the user memory registers (signed 32bit integers).

This command writes N (N=1…64) values $Y_1 … Y_N$ consecutively starting at the user memory address R (R=0…63). Note, that $N+R$ must be <=64.

This command must be used economically, because the flash memory is designed for typ. max. 20,000 flash cycles. Each time this command is executed, it will trigger a flash cycle.

NOTE, the user memory registers are located in a separate flash memory region than the configuration registers. This way calling the command #WRUM will not influence the flash memory of these registers.

Example Communication:

Command: `#WRUM_0_2_-16_777↵`
Response: `#WRUM_0_2_-16_777↵`

This example shows a command which writes the value -16 into the memory address 0, and the value 777 into the memory address 1.

2.3 Channel Commands

Channel commands trigger actions related to a specific optical channel. Their command header always consists of 3 letters. The first parameter is always the selected optical channel.

2.3.1 MEA – Trigger Measurement

This command triggers a measurement and returns the results.

Command: `MEA_C_S↵`
Response: `MEA_C_S_R₀_R₁…R₁₇↵`

Input Parameters:

C   Optical channel number (C=1 for one channel devices).

S   If in doubt, then set S=47!
           This parameter defines the enabled sensor types,
           given as decimal representation of the following bit field:

Bit 0 (add 1):	optical channel
Bit 1 (add 2)	sample temperature (typ. the external Pt100-sensor)
Bit 2 (add 4)	ambient air pressure
Bit 3 (add 8)	relative humidity within the module
Bit 4 (add 16)	reserved
Bit 5 (add 32)	case temperature (temperature within the module)

Example: S = 1 + 2 + 4 + 8 + 32 = 47 means, that the command will trigger the following measurements: optical sensor (oxygen, pH, or temperature), sample temperature, case temperature, ambient air pressure, and relative humidity within the module housing.

Output Parameters:

$R_0...R_{17}$ The results of the measurement given as the integer values of the 18 *Results* registers (refer to 0).

This command is the essential command for triggering measurements.

IMPORTANT: If automatic temperature compensation is enabled for the optical sensor, it is mandatory to enable Bit1 of the input parameter S!

Example Communication:

Command: MEA_1_3↵

Response: MEA_1_3_0_30120_270013_210211_98007_20135_0_87016_11788_0_0_123022_ 20980_0_0_0_0_0↵

This example command triggers the measurement of the sample temperature (e.g. external Pt100) and of the optical channel (in this example configured for oxygen sensors). The highlighted output parameters of the response are interpreted as follows:

status = 0 → No error or warning occurred; the measurement is valid!

μmolar = 270.013 μmol/L   (dissolved oxygen concentration)

mbar = 210.211 mbar = 210.211 hPa   (partial pressure oxygen)

airSat = 98.007 % air sat.   (% air saturation)

tempSample = 20.135 °C

signalIntensity = 87.016 mV

ambientLight = 11.788 mV

percentO2 = 20.980 %O$_2$ (volume fraction oxygen)

2.3.2   CHI – Calibrate Oxygen Sensor at ambient air

This command is used for calibrating the upper calibration point of the oxygen sensor at ambient air.

Command: **CHI‿C‿T‿P‿H**↵
Response: **CHI‿C‿T‿P‿H**↵

Input Parameters:

C Optical channel number (C=1 for one channel devices).

T Temperature of the calibration standard in units of 10^{-3} °C (e.g. 20000 means 20°C)

P Ambient air pressure in units of 10-3 mbar (e.g. 1013000 means 1013 mbar)

H Relative humidity of the ambient air in units of 10-3 %RH (e.g. 50000 means 50%RH) Set H=100000 (=100%RH) for calibrations in air saturated water.

This command performs 16 repeated optical measurements, and uses the average for the calibration. The total duration for this procedure varies between ca. 3s and ca. 6s depending on the configuration of the module.

Optional Factory Calibration: This upper calibration point might be skipped by applying a rough factory calibration given by the sensor code attached to the respective sensor head. Please refer to section 2.6.2 for more details.

For standard range oxygen sensors this command is by default configured for ambient air acting as the calibration gas with 20.95 %O2. Please refer to section 2.6.3 how to adapt the *Calibration* register *percentO2* in order to use other calibration gases with e.g. 15 %O2 or 100 %O2.

In order to keep the calibration permanently even after a power cycle, the command SVS (2.3.10) must be executed afterwards.

2.3.3 CLO – Calibrate Oxygen Sensor at 0% (anoxic)

This command is used for calibrating the oxygen sensor at 0% O2.

Command: **CLO‿C‿T**↵
Response: **CLO‿C‿T**↵

Input Parameters:

C Optical channel number (C=1 for one channel devices).

T Temperature of the calibration standard in units of 10^{-3} °C (e.g. 20000 means 20°C)

This command performs 16 repeated optical measurements, and uses the average for the calibration. The total duration for this procedure varies between ca. 3s and ca. 6s depending on the configuration of the module.

Optional Factory Calibration: This 0% O_2 calibration point might be skipped by applying a rough factory calibration given by the sensor code attached to the respective sensor head. Please refer to section 2.6.2 for more details.

In order to keep the calibration permanently even after a power cycle, the command SVS (2.3.10) must be executed afterwards.

2.3.4   COT – Calibrate Optical Temperature Sensor

This command performs a 1-point calibration of the optical temperature sensor.

Command: **COT␣C␣T↵**
Response: **COT␣C␣T↵**
Input Parameters:
C   Optical channel number (C=1 for one channel devices).
T   Temperature of the calibration standard in units of 10^{-3} °C (e.g. 20000 means 20°C)

This command performs 16 repeated optical measurements, and uses the average for the calibration. The total duration for this procedure varies between ca. 3s and ca. 6s depending on the configuration of the module.

In order to keep the calibration permanently even after a power cycle, the command SVS (2.3.10) **must be executed afterwards.**

2.3.5   CPH – Calibrate pH Sensor

This command is used for calibrating the pH sensor at up to 3 different calibration points. The sensor should be calibrated at 2 points ("low pH" and "high pH"). In contrast to electrochemical pH sensors, optical pH sensors are calibrated outside of the dynamic range where the pH indicator is either fully protonated or fully deprotonated. The first calibration point is called "low pH calibration" and is performed using a highly acidic buffer (pH 2, item no. **PHCAL2**). A second calibration point, "high pH calibration" is performed using a highly basic buffer (pH 11.00, item no. **PHCAL11**). After these two calibration points, the sensor is ready to use. It is **advised to read the pH sensor manual** for more details how to perform proper pH calibrations regarding equilibration time, salinity, which buffers must not be used, etc.

Optional Factory Calibration: The "high pH calibration" might be skipped by applying a rough factory calibration given by the sensor code attached to the respective sensor head. Please refer to section 2.8.2 for more details.

The third calibration point ("offset calibration") is optional and only recommended for advanced users. It adds an offset to the pH output. This offset has to be determined using a well-known buffer at a pH value close to the pKa of the sensor (e.g. a buffer with pH 8 could be used for -PK8 sensors).

Command:	**CPH_C_N_P_T_S**↵
Response:	**CPH_C_N_P_T_S**↵

Input Parameters:

C	Optical channel number (C=1 for one channel devices).
N	Calibration point type.
	N=0: low pH calibration point
	N=1: high pH calibration point
	N=2: offset calibration point (optional calibration point for advanced users)
P	pH value of calibration standard in units of 10^{-3} pH (e.g. 3000 means pH 3
T	Temperature of the calibration standard in units of 10^{-3} °C (e.g. 20000 means 20°C)
S	Salinity of the calibration standard in units of 10^{-3} g/L (e.g. 1000 means 1 g/L)

This command performs 16 repeated optical measurements, and uses the average for the calibration. The total duration for this procedure varies between ca. 3s and ca. 6s depending on the configuration of the module.

In order to keep the calibration permanently even after a power cycle, the command SVS (2.3.10) must be executed afterwards.

2.3.6   BGC – Perform Background Compensation

ONLY FOR ADVANCED APPLICATIONS with contactless sensors (e.g. sensor spots, flow-through cells) in order to compensate for possible background luminescence of the fiber-optics. Such a background compensation is recommended for the optical measurements, if *Results.signalIntensity* is below ca. 50mV for *Settings.amp* = 400x.

Command:	**BGC_C**↵
Response:	**BGC_C**↵

Input Parameters:

C	Optical channel number (C=1 for one channel devices).

Before calling this command, the fiber-optics must be separated from the actual optical sensor spot or flow-through cell. This command performs 16 repeated measurements of the background luminescence of the fiber-optics. It calculates the average and saves the normalized background compensation values in the corresponding *Calibration* registers *bkgdAmpl* and *bkgdDphi*. From this on, all optical measurements are automatically corrected for the background luminescence. The background compensation must always be executed before calibrating the sensor.

In order to keep the background compensation even after a power cycle, the command SVS (2.3.10) must be executed afterwards.

Example Communication:
Command:　BGC␣2↵
Response:　BGC␣2↵

.

2.3.7　BCL – Clear Background Compensation

ONLY FOR ADVANCED APPLICATIONS with contactless sensors. Clears (disables) the background compensation by writing zero to the *Calibration* registers *bkgdAmpl* and *bkgdDphi*.

Command:　**BCL␣C**↵
Response:　**BCL␣C**↵

Input Parameters:
C　　　　Optical channel number (C=1 for one channel devices).

In order to keep the background compensation cleared after a power cycle, the command SVS (2.3.10) must be executed afterwards.

Example Communication:
Command:　BCL␣4↵
Response:　BCL␣4↵

2.3.8　RMR – Read Multiple Registers

This command is used for reading out configuration registers (refer to 0).

Command:　**RMR␣C␣T␣R␣N**↵
Response:　**RMR␣C␣T␣R␣N␣Y_1␣Y_2␣Y_3...␣Y_N**↵

Input Parameters:
C　　　　Optical channel number (C=1 for one channel devices).
T　　　　Register block number:
　　　　　T=0: *Settings* registers
　　　　　T=1: *Calibration* registers
　　　　　T=3: *Results* registers
　　　　　T=4: *AnalogOutput* registers
R　　　　Start register number (R=0 for starting with the first register)
N　　　　Number of registers to read.

Output Parameters:
Y_1...Y_N　　Read register values.

Example Communication

Command: RMR_1_0_2_3↵

Response: RMR_1_0_2_3_0_5_2↵

This example reads out the 3rd, 4th, and 5th *Settings* register of the first optical channel, returning *salinity*=0, *duration*=5, and *intensity*=2.

2.3.9 WTM – Write Multiple Registers

This command is used for writing configuration registers (refer to 0). Note, this command changes the register values only in the volatile memory of the device.

In order to keep the changed register valuesn after a power cycle, the command SVS (2.3.10) must be executed afterwards.

Command: WTM_C_T_R_N_Y$_1$_Y$_2$_Y$_3$..._Y$_N$↵

Response: WTM_C_T_R_N_Y$_1$_Y$_2$_Y$_3$..._Y$_N$↵

Input Parameters:

C	Optical channel number (C=1 for one channel devices).
T	Register block number.
	T=0: *Settings* registers
	T=1: *Calibration* registers
	T=3: *Results* registers
	T=4: *AnalogOutput* registers
R	Start register number (R=0 for starting with the first register)
N	Number of registers to write.
Y$_1$...Y$_N$	Register values to be written.

Example Communication:

Command: WTM_2_1_2_4_-5000_12000_976000_50000↵

Response: WTM_2_1_2_4_-5000_12000_976000_50000↵

This example writes the 2nd, 3rd, 4th, and 5th *Calibration* register of the second optical channel, adjusting the registers to *temp0*=-5000 (i.e. -5°C), *temp100*=12000 (i.e. 12°C), *pressure*=976000 (i.e. 976 mbar), and *humidity*=50000 (i.e. 50%RH).

2.3.10 SVS – Save Registers in Flash Memory

Saves the actual RAM register configuration as the new default values into the internal flash memory. These default values are automatically loaded after a power cycle.:

Command: **SVS_C**↵

Response: **SVS_C**↵

Input Parameters:

C	Optical channel number (set always C=1).

Important: This command should be used economically, as the hardware is specified for up to 20,000 flash cycles.

Note: This command always stores the register configuration of all optical channels into the internal flash memory. So the parameter C can be fixed to 1.

Example Communication:
Command: SVS␣1↵
Response: SVS␣1↵

2.3.11 LDS –Load Registers from Flash Memory

Loads all RAM register values from the internal flash memory, i.e. the configuration is restored, which was given when the SVS command was executed last. Note, after a power cycle this action is automatically performed.

Command: **LDS␣C**↵
Response: **LDS␣C**↵

Input Parameters:
C   Optical channel number (set always C=1).

Note: This command always loads the register configuration of all optical channels from the flash memory into the RAM memory. So the parameter C can be fixed to 1.

Example Communication:
Command: LDS␣1↵
Response: LDS␣1↵

2.4   Error Responses

If an error occurred while processing a received command, the device will give the following response:

Command: *any command*
Response: **#ERRO␣C** ↵

This error response is mostly given, if the master did not send the command with the correct communication syntax. The output parameter C represents the general PyroScience error types as given by the following table.

Note: Warnings and errors directly related to the sensor measurements (e.g. a broken Pt100 temperature sensor, or a "worn out" optical oxygen sensor) will not result in such an #ERRO response. Instead, such warning and errors are given in the register *Results.status* (refer to 0).

C	Error Type	Description
-1	*General*	A non-specific error occurred.
-2	*Channel*	The requested optical channel does not exist.
-11	*Memory Access*	Memory access violation either caused by a not existing requested register, or by an out of range address of the requested value.
-12	*Memory Lock*	The requested memory is locked (system register) and a write access was requested.
-13	*Memory Flash*	An error occurred while saving the registers permanently. The SVS request should be repeated to ensure a correct permanent memory.
-14	*Memory Erase*	An error occurred while erasing the permanent memory region for the registers. The SVS request should be repeated.
-15	*Memory Inconsistent*	The registers in RAM are inconsistent with the permanently stored registers after processing SVS. The SVS request should be repeated.
-21	*UART Parse*	An error occurred while parsing the command string. The last command should be repeated.
-22	*UART Rx*	The command string was not received correctly (e.g. device was not ready, last request was not terminated correctly). Repeat the last command.
-23	*UART Header*	The command header could not be interpreted correctly (must contain only characters from A-Z). Repeat the last command.
-24	*UART Overflow*	The command string could not be processed fast enough to prevent an overflow of the internal receiving buffer
-26	*UART Request*	The command header does not match any of the supported commands.
-28	*UART Range*	One or more parameters of the command are out of range.
-30	*I2C Transfer*	There was an error transferring data on the I2C bus.
-40	*Temp Ext*	The communication with the sample temperature sensor was not successful.
-41	*Periphery No Power*	The power supply of the device periphery (sensors, SD card) is not switched on.

2.5 Settings Registers

These read/write registers contain settings for the optical measurement. The register block number is 0. They can be manipulated by using the commands RMR (2.3.8), WTM (2.3.9), and SVS (2.3.10).

Example Communication (reading the first 13 registers)

Command: RMR_1_0_0_13↵

Response: RMR_1_0_0_13_20000_1013000_0_5_1_6_4000_0_0_3_0_1_2 ↵

Example Communication (writing the first 3 registers)

Command: WTM_1_0_0_3_-30000_-1_12↵

Response: WTM_1_0_0_3_-30000_-1_12 ↵

The following table provides an overview. The registers can be grouped into 3 different categories is indicated by the circles in the first column.

Reg. Nr.	Label	Unit	Range	Description
(E) 0	*temp*	0.001 °C	-300096 – 300000	Temperature in the sample. Set to -300000 for automatic temperature compensation using the sample temperature sensor (e.g. Pt100). Set to -300000 - N for automatic temperature compensation using an optical temperature sensor connected to channel N.
(E) 1	*pressure*	0.001 mbar	-1 – 10000000	Ambient air pressure in the sample (set to -1 for automatic pressure compensation)
(E) 2	*salinity*	0.001 g/l	0 – 1000000	Salinity in the sample
(M)(T) 3	*duration*	-	1 – 8	Determines the duration of the red flash used for the optical measurement. The values 1-8 correspond to a flash duration of 1, 2, 4, 8, 16, 32, 64, and 128ms, respectively.
(M)(S) 4	*intensity**	-	0 – 7	Determines the intensity of the red flash use for the optical measurement. The values 0-7 correspond to 10%, 15%, 20%, 30%, 40%, 60%, 80%, and 100% of the maximum intensity.
(M)(S) 5	*amp**	-	4 – 6	Determines the amplification level of the optical detector used for reading out the optical sensor. The values 4-6 correspond to 80x, 200x, and 400x amplification.
(T) 6	*frequency*	Hz	1 – 32000	Sensor type specific constant
(M) 7	*crcEnable*	-	0 – 1	0: no cyclic redundancy check (CRC) 1: enables CRC of the communication protocol (only used for channel=1, see comment below)
8	Reserved			
(T) 9	*options*	-	0 – 7	Sensor type specific constant bit0 (add 1): automaticFlashDuration is enabled bit1 (add 2): automaticAmpLevel is enabled bit2 (add 4): 1000xOxygen is enabled
(M) 10	*broadcast*	-		See below
(T) 11	*analyte*	-	0 – 4	Sensor type specific constant Detected analyte by the optical sensor: 0: no sensor, 1: oxygen, 2: temperature, 3: pH
(T) 12	*fiberType*	-	0 – 2	Sensor type specific constant Diameter of the used optical fiber 0 = 230µm, 1 = 430µm, 2 = 1mm

| ⓣ 13 | *fiber length* | mm | 0 - 5000 | Fiber length in mm. Only needed for fiberType = 2. Fiber length can be found on the PyroScience Label next to the sensor code. |
| 14-19 | Reserved | | | |

* Note: When changing these values, a recalibration of the optical sensor is recommended.

2.5.1 Ⓔ Environmental Condition Registers

Before triggering an optical measurement, the registers *temp*, *pressure* and *salinity* must be adjusted by the user in order to define the environmental conditions in the sample at the very position where the sensitive part of the optical sensor is located. This is needed by the device for calculating the correct result for the optical sensor measurement. Refer to the table below, which environmental settings are needed for the different analytes. The special setting *temp* = -300000 activates the automatic temperature compensation of the optical measurement, which means that the result of the sample temperature measurement (e.g. external Pt100) is used for calculating the results.

Devices with more than one optical channel can use optical temperature sensors for compensating optical sensors connected to other channels. For example, if an oxygen, a pH, and an optical temperature sensor are connected to the optical channels 1, 2, and 3, respectively. Then *Settings.temp* should be adjusted to -300003 for the channels 1 and 2. This way, the oxygen and the pH sensor will be automatically temperature compensated by the results of the last optical temperature measurement at channel 3. In this scenario it is important to trigger first the measurement at channel 3, followed by measurements at channels 1 and 2.

The special setting *pressure* = -1 activates the automatic pressure compensation of the optical measurement, which means that the result of ambient air pressure measurement is used for calculating the results.

Analyte	Temperature	Pressure	Salinity
Oxygen	Required	Required only for oxygen units: μmolar (mg/L), %air sat., %O2	Required only for oxygen units: μmolar (mg/L)
Optical Temperature	--	--	--
pH	Required	--	Required

2.5.2 Ⓜ Measurement Mode Registers

Optical Measurement Parameters. All optical sensors from PyroScience are based on proprietary luminescent dyes, which are excited with LED light. The excited dyes emit then light in the near infrared (NIR), which is detected by a light detector. The measured lifetime of the emitted NIR light is then used to calculate the final oxygen, pH, or temperature values (all done within the firmware). The user can adjust (i) the *intensity* of the LED light, (ii) the *duration* of the emitted LED flash, and the (iii) the amplification level *amp* of the detector.

The registers *intensity* and *amp* should be adjusted together. Suggested procedure: set *amp* = 6 and increase *intensity* stepwise from 0 to 7 until the signal intensity

(*Results.signalIntensity*) is within the typical range of 100-400mV. If *Results.signalIntensity* is then still too high, decrease stepwise *amp* to 5 or 4 and choose again the best value for *intensity*. But *amp* must be kept in the range of 4-6. Note that the chosen value of *intensity* is a compromise between lower long-term drift (*intensity*=1) and lower signal-to-noise ratio (*intensity*=8).

The registers *intensity* and *amp* should be always adjusted properly if a new sensor head is connected to the device. After the sensor has been calibrated, these two registers should not be changed again.

Please refer also to section 2.5.3 describing an alternative way how to adjust the registers *intensity* and *amp* based on the sensor code provided with each individual sensor head.

The register *duration* determines the length of the LED flash as indicated in the table. The chosen value is a compromise between lower long-term drift (*duration*=1) and lower signal-to-noise ratio (*duration*=8). If in doubt set *duration*=5. This register has no effect on the calibration, so it can be changed anytime.

Broadcast Mode. The register *broadcast* can be used to enable and configure the broadcast mode for the specific optical channel. During broadcast mode, the device triggers itself periodic measurements as defined in the *broadcast* register (see details in the table below). Optionally it is also possible to trigger broadcast measurements via the digital input TRIGIN at the extension port of the device (if available). The results can be given at the analog output (if available), and the results can be transmitted via the UART/USB interface. In the latter case, the results are transmitted following the syntax of the MEA command (2.3.1), with the only difference, that the message starts with the character "
>" (ASCII code 0x3E).

The broadcast mode of different optical channels can be configured independently. For example, one channel can measure every 10s, the next one every 0.5s.

Note, the minimum realizable broadcast interval depends on the hardware. For laboratory (e.g. FireSting) and underwater devices (e.g. AquapHOx Transmitter) it can be as low as 25 ms, while for most OEM devices (e.g. PICO-x) the minimum realizable broadcast interval is 1000 ms. The reason is, that these OEM devices are optimized for low energy and not for high speed operation. The maximum possible broadcast interval is 65000 ms = 65 s.

Note, the master can anytime send standard commands to the device, even if the broadcast mode is enabled. If the device receives a command from the master, while it is executing a broadcast measurement, the device will (1) complete the broadcast measurement, (2) send the broadcast message, (3) thereafter it will immediately execute the requested command, and (4) send then the response to the master.

Broadcast Mode with Deep Sleep. For devices which support the #STOP command (2.2.6), the broadcast mode does also work in combination with the deep sleep mode in order to reduce the power consumption significantly. This mode is enabled by sending

the command #STOP or it is enabled directly after power up if the *enableDeepSleep* bit is set to 1. In this case the device automatically wakes up from deep sleep for each required broadcast measurement. When the measurement is finished it goes back into deep sleep mode.

broadcast Register	Label	Description
Bit 0..15	*Interval* (ms)	interval = 0: Interval broadcast mode is disabled interval = 1-65535: Interval broadcast mode is enabled. The module triggers itself periodic measurements, whereby interval defines the broadcast interval in units of ms. The max. possible interval is 65 s. The minimum is in the range of 25-1000ms depending on the specific hardware and the configuration of the measurement mode registers.
Bit 16..23 Bit 16 (add 65536) Bit 17 (add 131072) Bit 18 (add 262144) Bit 19 (add 524288) Bit 20 (add 1048576) Bit 21 (add 2097152)	*enabledSensors* optical channel sample temperature ambient air pressure relative humidity reserved case temperature	Bitfield defining which sensors are enabled during the broadcast mode as given by the parameter S in the command MEA.
Bit 24 (add 16777216)	*enableUartTransmit*	If set to 1, then the broadcast mode will send after each measurement via the UART (USB) interface the character '>' followed by the answer string as given by the command MEA.
Bit 25 (add 33554432)	*enableTrigin*	If set to 1, then a broadcast measurement can be triggered by the TRIGIN pin at the extension port (if available).
Bit 26 (add 67108864)	*enableDeepSleep*	If set to 1, then the device will go after power up directly into deep sleep mode (only if device supports the #STOP command). It will wake up periodically for each required broadcast measurement, and then go back to deep sleep.

Cyclic Redundancy Check (CRC). Setting the register *Settings.crcEnable*=1 will enable a cyclic redundancy check (CRC) for all messages (i.e. command responses and broadcast messages) transmitted from the device to the control hardware (e.g. PC). Now every answer from the device to the master is terminate by the string "**:␣C↵**" where C is the CRC16/Modbus checksum represented as a decimal ASCII-string (human readable). The CRC is calculated for all ASCII-characters (unsigned bytes) from the very beginning of the returned string until the character just before the "**:**". Note, also spaces (ASCII code 32) are included in the CRC calculation. See also 2.1.4.

Note, crcEnable is only used for channel=1. The value of crcEnable for channel 1 enables/disables the CRC for the whole device!

2.5.3 ⓢ Sensor Code Settings

Section 2.5.2 described how to configure the registers *intensity* and *amp.* This procedure can be optionally omitted by utilizing information given by the "sensor code" provided with each individual sensor head from PyroScience. This sensor code contains the factory calibration (see section 2.6.2) and recommended measurement settings for *intensity* and *amp.* The last two digits of the first block of the sensor code contain the recommended settings. For example, in the sensor-code "XB7-547-213" "B" encodes the *intensity* setting (the letters A to H correspond to 10%, 15%, 20%, 30%, 40%, 60%, 80%,

and 100% of the maximum intensity) and "7" encodes the *amp* setting (5, 6 and 7 correspond to 80x, 200x, and 400x amplification). These values are only recommendations for typical applications for the specific sensor head. For optimal performance please follow the instruction given in section 2.5.2.

2.5.4 ⓣ Sensor Type Specific Constants

These registers comprise constants which are specific for the used sensor type (and not for each individual sensor head). For example: if the user intents to use always oxygen microsensor (item nr. OXR50), then these registers need to be configured only one time. It is recommended to adjust the proper values by using the Windows software tools "Pyro DeveloperTool" or "Pyro Workbench". These software tools will configure these registers based on the sensor code attached to the sensor head. Standard users might skip the following paragraphs of this section, as these registers are adjusted by the mentioned software tools.

Automatic Flash Duration. Setting bit 0 (*automaticFlashDuration*) of *Settings.options* to 1 will enable the automatic flash duration. Now the flash duration is dynamically adjusted within the range of 1-128 ms depending on the last value of *Results.signalIntensity*. In this case *duration* defines the flash duration if *Results.signalIntensity* equals 400mV. For higher signal intensities the duration is automatically reduced, while for lower ones it is increased. This option is especially useful for oxygen sensors, because their *signalIntensity* depends strongly on the actual oxygen level. This way the *duration* is dynamical adjusted in order to assure a pretty constant signal-to-noise level while ensuring minimal drift (caused by photobleaching).

Automatic Amplification Level. If bit 1 (*automaticAmpLevel*) of *Settings.options* is set to 1, then the amplification level of the detector is reduced automatically in case of an oversaturation of the detector. It is recommended for precision measurements to adjust *intensity* and *amp* in such a way, that *Results.status* never indicates "sensor signal or ambient light too high". But an enabled automatic amplification level is anyhow recommended, because the optical sensor will still give reasonable values if e.g. increased ambient light might saturate the amplifier. In this case the optical measurement is automatically repeated with stepwise decreased amplification levels, until the amplifier is not anymore saturated. If such an automatic amplification reduction has taken place, then *Results.status* shows "Warning - automatic amplification level active". Note, that the content of the register *amp* is not changed by this procedure.

Trace Oxygen Option. If bit 2 (*1000xOxygen*) of *Settings.options* is set to 1, the integer values given in the *Results* registers *umolar*, *mbar*, *airSat*, and *percentO2* are multiplied by a factor of 1000. This means, that after conversion to floating point values, the user will obtain 3 more digits of precision for oxygen measurements. This option should be only used in combination with special "trace oxygen sensors" intended for measurements close to 0%O2. If the *1000xOxygen* option is enabled, then *Results.status* will always show the warning "WARNING – 1000xOxygen enabled".

Analyte. The detected analyte of the attached optical sensor must be adjusted to the register *analyte*, which determines how the *RawData* registers are internally processed to the outcomes in the Result registers.

Fiber Type and Fiber Length. The sensor type defines the fiber type and the standard fiber length. Customized fiber lengths are printed on the **PyroScience** label next to the sensor code. The fiber length is only relevant for plastic fibers (i.e. *fiberType* = 2)

The sensor type specific constants for the settings registers are available on request.

2.6   Calibration Registers (Oxygen Sensors)

These read/write registers contain all information about the sensor calibration. The register block number is 1. They can be manipulated by using the commands RMR (2.3.8), WTM (2.3.9), and SVS (2.3.10).

Example Communication (reading the first 6 registers)
Command: RMR_1_1_0_6↵
Response: RMR_1_1_0_6_53212_20123_20212_21209_1024089_100000 ↵

Example Communication (writing the first 2 registers)
Command: WTM_1_1_0_2_53000_20000↵
Response: WTM_1_1_0_2_53000_20000 ↵

Note, the specific definition of the *Calibration* registers depends on the analyte of the connected sensor head. This chapter describes the definition of the *Calibration* registers, if an optical oxygen sensor is connected. Important, do not forget to adjust the *Settings.analyte* to 1 in this case. This ensures, that the device applies the calibration registers accordingly.

The *Calibration* registers provide all information needed for converting the raw value *Results.dphi* into the final oxygen units in the *Results* registers. The following table provides an overview. The registers can be grouped into 2 different categories as indicated by the circles in the first column.

Reg. Nr.	Label	Unit	Configured by	Description
Ⓒ Ⓢ 0	dphi0	0.001°	command CLO (or sensor code)	Phase shift at 0%O2.
Ⓒ Ⓢ 1	dphi100	0.001°	command CHI (or sensor code)	Phase shift at upper calibration point.
Ⓒ Ⓢ 2	temp0	0.001°C	command CLO (or sensor code)	Calibration temperature at 0%O2.
Ⓒ Ⓢ 3	temp100	0.001°C	command CHI (or sensor code)	Calibration temperature at upper calibration point.
Ⓒ Ⓢ 4	pressure	0.001 mbar	command CHI (or sensor code)	Ambient air pressure during calibration at upper calibration point.
Ⓒ Ⓢ 5	humidity	0.001 %RH	command CHI (or sensor code)	Relative humidity during calibration at upper calibration point. Set to 100%RH for calibrations in liquid samples.

(T)	6	f	0.001	software tools	Sensor type specific constant
(T)	7	m	0.001	software tools	Sensor type specific constant
(T)	8	calFreq	Hz	Software tools	Sensor type specific constant
(T)	9	tt	10^{-5}/K	software tools	Sensor type specific constant
(T)	10	kt	10^{-5}/K	software tools	Sensor type specific constant
(T)	11	bkgdAmpl	0.001 mV	software tools / adv. users	Sensor type specific constant
(T)	12	bkgdDphi	0.001°	software tools / adv. users	Sensor type specific constant
(T)	13	useKsv		software tools	Sensor type specific constant
(T)	14	ksv	10^{-6}/mbar	software tools	Sensor type specific constant
(T)	15	ft	10^{-6}/K	software tools	Sensor type specific constant
(T)	16	mt	10^{-6}/K	software tools	Sensor type specific constant
	17	Reserved			
(T)	18	percentO2	0.001 %O$_2$	software tools / adv. users	Sensor type specific constant. Oxygen level at upper calibration point
	19-29	Reserved			

2.6.1 Ⓒ User Calibration

The calibration data registers contain all information about the last sensor calibration. It is only necessary to write these registers if the factory calibration of a sensor head should be used. During a calibration they are automatically adjusted by the oxygen calibration commands CHI (2.3.2) and CLO (2.3.3). The command CHI will adjust the registers *dphi100, temp100, pressure,* and *humidity*. The command CLO will adjust the registers *dphi0* and *temp0*.

2.6.2 Ⓢ Factory Calibration Taken from Sensor Code

The user calibration is generally advised for highest accuracy. But it is possible to replace either calibration point by a rough factory calibration, which is given by the so called "Sensor Code" attached to each oxygen sensor. This is especially interesting, if the aimed application will not measure oxygen values close to one of the calibration points. For example, a respiration measurement might measure always around oxygen levels of 18-21 %O2. Then it is possible to take the factory calibration for the 0 %O2 calibration point, without losing too much accuracy.

How to deduct the factory calibration from the Sensor Code is best explained by an example, where we assume the following Sensor Code: XB7-547-213

A 0 %O2 factory calibration is encoded by the numbers 547 which give *dphi0* = 54,7°. The user has to write the following registers, in order to apply this calibration:

Register	Value	int32 value
dphi0	54.7°	54700
temp0	20.0°C	20000

An ambient air factory calibration is encoded by the numbers 213 which give *dphi100* = 21,3°. The user has to write the following registers, in order to apply this calibration:

Register	Value	int32 value
dphi100	21.3°	21300
temp100	20.0°C	20000
pressure	1013 mbar	1013000
humidity	0 %RH	0

2.6.3 ⓣ Sensor Type Specific Constants

These registers comprise constants which are specific for the used sensor type (and not for each individual sensor head). For example: if the user intents to use always oxygen microsensors (item nr. OXR50), then these registers must be configured only one time. It is recommended to adjust the proper values by using the Windows software tools "Pyro DeveloperTool" or "Pyro Workbench". These software tools will configure these registers based on the sensor code provided with the sensor head.

Advanced users might change the register *percentO2*. For most standard range oxygen sensors this register is fixed to 20.95 %O2. This defines, that the command CHI (2.3.2) can be used for calibrations at ambient oxygen levels. If you want to apply a calibration gas of e.g. 15 %O2 or 100 %O2, then you have to write this value into the register *percentO2* **before** using the command CHI.

Advanced users applying contactless sensor solutions like sensor spots or flow-through cells might adjust the registers *bkgdAmpl* and *bkdgDphi* additionally by using the commands BGC or BCL. Refer to section 2.3.6 for more details. Standard users should use the described software tools for configuring these registers. For sensors using a 1 mm plastic fiber *bkgdAmpl* can be estimated as:

$$bkgdAmpl = 0.234 \times length(m) + 0.343 \qquad bkgdDphi = 0 \qquad \text{(eq. 1)}$$

If the user intends to switch sensor type between measurements and a reconfiguration with the "Pyro DeveloperTool" or "Pyro Workbench" is not possible, the registers can be adjusted manually. The sensor type specific constants are available on request.

2.7 Calibration Registers (Optical Temperature Sensors)

These read/write registers contain all information about the sensor calibration. The register block number is 1. They can be manipulated by using the commands RMR (2.3.8), WTM (2.3.9), and SVS (2.3.10).

Example Communication (reading the first 2 registers)
Command: `RMR_1_1_0_2↵`
Response: `RMR_1_1_0_2_343_223↵`

Example Communication (writing the 10ᵗʰ register _Tofs_)
Command: `WTM_1_1_9_1_-1023↵`
Response: `WTM_1_1_9_1_-1023↵`

Note, the specific definition of the _Calibration_ registers depends on the analyte of the connected sensor head. This chapter describes the definition of the _Calibration_ registers, if an optical temperature sensor is connected. Important, do not forget to adjust the _Settings.analyte_ to 2 in this case. This ensures, that the device applies the calibration registers accordingly.

These registers provide all information needed for converting the raw value _Results.dphi_ into the temperature given in _Results.tempOptical_. The following table provides an overview. The registers can be grouped into 3 different categories as indicated by the circles in the first column.

Reg. Nr.	Label	Unit	Configured by	Description
(S) 0	_M_	1	sensor code	Sensor code constant
(S) 1	_N_	1	sensor code	Sensor code constant
2-5	Reserved			
(T) 6	_C_	0.001	software tools	Sensor type specific constant
7-8	Reserved			
(C) 9	_Tofs_	0.001 K	command COT	Offset temperature used for single point calibration (e.g. -1230 corresponds to an offset of -1.23K)
10	Reserved			
(T) 11	_bkgdAmpl_	0.001 mV	software tools / adv. users	Sensor type specific constant
(T) 12	_bkgdDphi_	0.001°	software tools / adv. users	Sensor type specific constant
13-29	Reserved			

2.7.1 ⓒ User Calibration

The register _Tofs_ contains a simple offset user calibration. The command COT (2.3.4) will adjust this register.

2.7.2 ⓢ Sensor Code Constants

These registers comprise constants which must be adjusted for each individual sensor head. These constants are part of the sensor code printed on the label attached to the sensor head. An example explains how to deduct these constants from the sensor code:

We assume the following Sensor Code: CD6-303-407
Then the register M must be adjusted to M = 303.
And the register N must be adjusted to N = 407.

2.7.3 Ⓣ Sensor Type Specific Constants

These registers comprise constants which are specific for the used sensor type (and not for each individual sensor head). It is recommended to determine the proper values by using the Windows software tools "Pyro DeveloperTool" or "Pyro Workbench" initially as a reference. These software tools will configure these registers based on the sensor code attached to the sensor head.

Advanced users applying contactless sensor solutions like sensor spots or flow-through cells might adjust the registers *bkgdAmpl* and *bkdgDphi* additionally by using the commands BGC or BCL. Refer to section 2.3.6 for more details. Standard users should use the described software tools for configuring these registers. For sensors using a 1 mm plastic fiber *bkgdAmpl* and *bkgdDphi* can be estimated with the equation given in section 2.6.3.

If the user intends to switch sensor type between measurements and a reconfiguration with the "Pyro DeveloperTool" or "Pyro Workbench" is not possible, the register C can be adjusted manually. The sensor type specific constants are available on request.

2.8 Calibration Registers (pH Sensors)

These read/write registers contain all information about the sensor calibration. The register block number is 1. They can be manipulated by using the commands RMR (2.3.8), WTM (2.3.9), and SVS (2.3.10).

Example Communication (reading the 14th register)
Command: RMR_1_1_13_1↵
Response: RMR_1_1_13_1_154↵

Example Communication (writing the 1st register)
Command: WTM_1_1_0_1_7013↵
Response: WTM_1_1_0_1_7013↵

Note, the specific definition of the *Calibration* registers depends on the analyte of the connected sensor head. This chapter describes the definition of the *Calibration* registers, if an optical pH sensor is connected. Important, do not forget to adjust the *Settings.analyte* to 3 in this case. This ensures, that the device applies the calibration registers accordingly!

These read/write registers provide all information needed for converting the raw value *Results.dphi* into the final pH value given in the *Results.ph*. The following table provides an overview on all *calibration* registers. These registers can be grouped into 3 different categories is indicated by the circles in the first column.

Reg. Nr.	Label	Unit	Configured by	Description
Ⓒ Ⓢ 0	*R1*	10^{-6}	Command CPH (or sensor code)	Sensor code constant
Ⓒ 1	*pH1*	0.001 pH	Command CPH	
Ⓒ 2	*temp1*	0.001°C	Command CPH	
Ⓒ 3	*salinity1*	0.001 g/L	Command CPH	
Ⓒ Ⓢ 4	*R2*	10^{-6}	Command CPH (or sensor code)	Sensor code constant
Ⓒ 5	*pH2*	0.001 pH	Command CPH	
Ⓒ 6	*temp2*	0.001°C	Command CPH	
Ⓒ 7	*salinity2*	0.001 g/L	Command CPH	
Ⓒ 8	*offset*	0.001 pH	Command CPH	
Ⓣ 9	*dphi_ref*	0.001°	software tools	Sensor type specific constant
Ⓣ 10	*attenuation_coefficient*	10^{-6} 1/m	software tools	Sensor type specific constant
Ⓣ 11	*bkgdAmpl*	0.001 mV	software tools / adv. users	Sensor type specific constant
Ⓣ 12	*bkgdDphi*	0.001°	software tools / adv. users	Sensor type specific constant
Ⓣ 13	*dsf_dye*	10^{-6}	software tools	Sensor type specific constant
Ⓣ 14	*dtf_dye*	10^{-6}	software tools	Sensor type specific constant
Ⓢ 15	*pKa*	0.001 pH	sensor code	Sensor code constant
Ⓣ 16	*slope*	10^{-6}	software tools	Sensor type specific constant
Ⓣ 17	*bottom_t*	10^{-6} 1/K	software tools	Sensor type specific constant
Ⓣ 18	*top_t*	10^{-6} 1/K	software tools	Sensor type specific constant
Ⓣ 19	*slope_t*	10^{-6} 1/K	software tools	Sensor type specific constant
Ⓣ 20	*pKa_t*	10^{-6} pH/K	software tools	Sensor type specific constant
Ⓣ 21	*pKa_is1*	0.001	software tools	Sensor type specific constant
Ⓣ 22	*pKa_is2*	0.001	software tools	Sensor type specific constant
23-29	Reserved			

2.8.1 Ⓒ User Calibration

These registers are adjusted by applying the command CPH (2.3.5). When the sensor head is exchanged the offset *register* must be reset to 0.

2.8.2 Ⓢ Sensor Code Constants

The register *pka* is a constant which must be adjusted for each individual sensor head. This constant is printed on the label attached to the sensor head.

The registers *R1* and *R2* are intermediate values for the calculation of pH and must be adjusted for each individual sensor head. These values are part of the sensor code and printed on the label attached to the sensor head. These values are important for a calibration free use of the pH sensor heads. Alternatively, *R1* and *R2* can be determined via a two-point calibration by the user (see section 2.3.5)

2.8.3 ⓣ Sensor Type Specific Constants

These registers comprise constants which are specific for the used sensor type (and not for each individual sensor head). It is recommended to adjust the proper values by using the Windows software tools "Pyro DeveloperTool" or "Pyro Workbench". These software tools will configure these registers based on the sensor code attached to the sensor head.

Advanced users applying contactless sensor solutions like sensor spots or flow-through cells might adjust the registers *bkgdAmpl* and *bkdgDphi* additionally by using the commands BGC or BCL. Refer to section 2.3.6 for more details. Standard users should use the described software tools for configuring these registers. For sensors using a 1 mm plastic fiber *bkgdAmpl* and *bkgdDphi* can be estimated with the equation given in section 2.6.3.

If the user intends to switch sensor type between measurements and a reconfiguration with the "Pyro DeveloperTool" or "Pyro Workbench" is not possible, the registers can be adjusted manually. The sensor type specific constants are available on request.

2.9 Results Registers

The read-only *Results* registers provide the status and the results of the last measurement. This register block is not saved within the flash memory, it is only kept in the volatile RAM memory. The register block number is 3. They can be read by using the command RMR (2.3.8).

Example Communication (reading the first 8 registers)

Command: RMR␣1␣3␣0␣15↵

Response: RMR␣1␣3␣0␣15␣0␣21099␣210837␣203987␣97876␣23656␣21065␣234098↵

Interpreted as: status = 0, dphi = 21.099°, umolar = 210.837 µmol/L, mbar = 203.987 hPa, airSat = 97.876 %air sat., tempSample = 23.656°C, tempCase = 21.065°C, signalIntensity = 234.098 mV

Note, the MEA command (2.3.1) returns by default the content of the *Result* registers in the command response. So, in standard applications it is not necessary to use the RMR command.

Reg. Nr.	Label	Unit	Range	Description
0	*status*		int32	ERROR / Warning Flags of the last measurement (refer to separate table below)
1	*dphi*	0.001°	int32	Phase shift of optical measurement (raw data)
2	*umolar***	0.001 µmol/L	int32	Dissolved oxygen concentration (valid only in water)
3	*mbar***	0.001 mbar = 0.001 hPa	int32	Partial pressure oxygen (hPa=mbar) (valid in gases and in liquids)
4	*airSat***	0.001 %air sat.	int32	Dissolved oxygen given as % air saturation (valid only in liquids)
5	*tempSample*	0.001°C	int32	Sample temperature (typ. Pt100 sensor)
6	*tempCase*	0.001°C	int32	Case temperature within the device
7	*signalIntensity**	0.001 mV	int32	Signal intensity of optical measurement
8	*ambientLight*	0.001 mV	int32	Ambient light entering the sensor
9	*pressure*	0.001 mbar	int32	Ambient air pressure
10	*humidity*	0.001 %RH	int32	Relative humidity within the device
11	*resistorTemp*	0.001 Ohm	int32	Resistance of the sample temperature sensor
12	*percentO2***	0.001 %O_2	int32	Oxygen volume fraction (valid only in gases)
13	*tempOptical*	0.001 °C	int32	Temperature measured with optical temperature sensor
14	*ph*	0.001	int32	pH value measured by optical pH sensors
15	*R*	10^{-6}	int32	R-value (intermediate value of pH calculation)
16-17	Reserved			

* If *automaticAmpLevel* is enabled in *settings.options*, then *signalIntensity* is given as it would be for the actual amplification level adjusted in *settings.amp*! For example, if *settings.amp*=7 (=400x) and the measurement was due to *automaticAmpLevel* performed with 80x amplification and a real photodetector signal intensity of 800mV, then *signalIntensity* will have the value 800mV x 400 / 80 = 4000mV.

** If *1000xOxygen* is enabled in *settings.options*, then these registers are internally multiplied by 1000, giving 3 more digits of precision after conversion to floating point values.

Status Register. The *status* register contains several bit flags indicating errors or warnings of the last measured data point.

It is on the user's authority to check the *status* register after each measurement for possible Warnings or ERRORS. Especially in case of ERRORS, the given results might be not valid.

The user has to distinguish between warnings and errors. A warning indicates, that the measurement results are in principle still valid, but their precision and/or accuracy might be deteriorated. An error means, that the respective measurement result is not at all valid.

status Bit	ERROR / Warning Description
Bit 0 (add 1)	Warning - automatic amplification level active
Bit 1 (add 2)	Warning - sensor signal intensity low
Bit 2 (add 4)	ERROR - optical detector saturated
Bit 3 (add 8)	Warning - reference signal intensity too low
Bit 4 (add 16)	ERROR - reference signal too high
Bit 5 (add 32)	ERROR - failure of sample temperature sensor (e.g. Pt100)
Bit 6 (add 64)	Warning – 1000xOxygen enabled
Bit 7 (add 128)	Warning - high humidity (>90%RH) within the module
Bit 8 (add 256)	ERROR - failure of case temperature sensor

Bit 9 (add 512)	ERROR - failure of pressure sensor
Bit 10 (add 1024)	ERROR - failure of humidity sensor

Example: *status* = 34 = 2 + 32 means, that there is a warning about low signal intensity of the optical sensor, and that the external temperature sensor (Pt100) had a failure.

Results of optical oxygen sensors are given in the registers *umolar*, *mbar*, *airSat*, and *percentO2*. Refer to the table above for the details on the different oxygen units. These registers show only results if *Settings.analyte* = 1. Otherwise, they are set to 0.

Results of optical temperature sensors are given in the register *tempOptical*. This register shows only a result if *Settings.analyte* = 2. Otherwise, it is set to 0.

Results of optical pH sensors are given in the register *ph* and *R*. These registers show only a result if *Settings.analyte* = 3. Otherwise, they are set to 0.

Results of sample temperature sensors (typ. Pt100), case temperature sensors, internal pressure sensors and internal humidity sensors are given in *tempSample*, *tempCase*, *resistorTemp*, *pressure* and *humidity*, respectively.

Invalid Results. If for any reason the measurement produces an invalid result (e.g. the signal intensity of an optical sensor is much too low, or the reference LED measurement had a failure etc.), then the result register contains the integer value -300000. Any higher programming language should interpret this as "not a number" (NaN).

Raw data of the optical measurement. The raw data given by the optical phase-shift measurement are the "phase shift" *dphi*, the *signalIntensity*, and *ambientLight*.The phase shift *dphi* is the essential value used for calculating the value of the sensor analyte (see below). *signalIntensity* represents the measured luminescent signal intensity of the optical sensor. *signalIntensity* can be also used for checking the quality of the sensor (e.g. a broken sensor tip will result in a rapid decrease of the intensity). Please note, that the signal intensity is highly dependent e.g. on the actual oxygen concentration and/or on the temperature. Therefore, measured signal intensities can be only compared with each other, if they have been measured at similar analyte levels and temperatures. Immediately before each optical measurement, the device measures also the ambient light entering the optical detector and stores it in *ambientLight*. Considerable amounts of ambient light might enter the optoelectronics, if the sensor is exposed e.g. to direct sun light. This might lead to a saturation of the optical detector and to an invalid optical measurement. As a rule of thumb, the sum of *ambientLight* and *signalIntensity* should not exceed 2000mV. If it exceeds this value, the amplification level in the *settings.amp* should be lowered. Or the sensor tip should be shaded from the external light source. Or *automaticAmpLevel* should be enabled in *settings.options*.

2.10 AnalogOutput Registers

These read/write registers contain the configuration of the analog outputs (if available). The register block number is 4. They can be manipulated by using the commands RMR (2.3.8), WTM (2.3.9), and SVS (2.3.10).

Example Communication (reading the first 4 registers)

Command: RMR␣1␣4␣0␣4↵

Response: RMR␣1␣4␣0␣4␣260␣516␣1028␣2052↵

Example Communication (writing the 5ᵗʰ register)

Command: WTM␣1␣4␣4␣0↵

Response: WTM␣1␣4␣4␣0 ↵

The maximum number of supported analog outputs is 4. Any result register from any optical channel can be mapped on the analog outputs, whereby the linear scaling can be freely adjusted by min and max values. If the corresponding result register contains an invalid result (i.e. register value is -300000), then the analog output is set to 0.0V or 0mA.

Note, these registers can be comfortably configured with the Windows software tools Pyro DeveloperTool or Pyro Woekbench.

Reg. Nr.	Label	Unit	Range	Description
0	*aoSelectA*	bits 0-6: regNr bit 7: alarm** bits 8-15: C	regNr: 0..17 alarm: 0..1 C: 1..4	regNr is the register number of the *results*-register and C the optical channel number which is used for the analog output A. Use alarm=0 for standard analog output. Use alarm=1 for alarm output, see below *
1	*aoSelectB*	as above	see above	As *aoSelectA* but given for analog output B.
2	*aoSelectC*	as above	see above	As *aoSelectA* but given for analog output C.
3	*aoSelectD*	as above	see above	As *aoSelectA* but given for analog output D.
4	*aoMinA*	unit identical to the unit of the register number regNr in the *results*-registers	int32	Defines the *results*-register value (selected by *aoSelectA*), which corresponds to the minimum output level (0.0V or 4mA).
5	*aoMinB*	as above	int32	As *aoMinA* but given for analog output B.
6	*aoMinC*	as above	int32	As *aoMinA* but given for analog output C.
7	*aoMinD*	as above	int32	As *aoMinA* but given for analog output D.
8	*aoMaxA*	unit identical to the unit of the register	int32	Defines the *results*-register value (selected by *aoSelectA*), which

		number regNr in the *results*-registers			corresponds to the maximum output level (Umax or 20mA)
9	*aoMaxB*	as above	int32		As *aoMaxA* but given for analog output B.
10	*aoMaxC*	as above	int32		As *aoMaxA* but given for analog output C.
11	*aoMaxD*	as above	int32		As *aoMaxA* but given for analog output D.

* If alarm=1 then the analog output acts as a digital alarm output, giving 0.0V (or 4mA for current outputs) if the selected register is within the range defined by aoMinA and aoMaxA. If outside that range, the output is set to Umax (or 20mA). For invalid results (i.e. register value is -300000) it is set to Umax (or 0mA).

2.11 Resistive Temperature Sensor Registers

These read/write registers contain the configuration of the resistive temperature measurement (e.g. Pt100 temperature sensor connected to the sample temperature port. The register block number is 20. They can be manipulated by using the commands RMR (2.3.8), WTM (2.3.9), and SVS (2.3.10).

This register block contains 8 registers, but only the register *tempOffset* is interesting for developers. This register can be used to perform a simple offset user calibration of the attached sample temperature sensor. The following example communication show how to read and write this single register:

Example Communication for reading *tempOffset*

Command: RMR_1_20_6_1↵

Response: RMR_1_20_6_1_1200↵

I**n this example** the current temperature offset is +1.2 K.

Example Communication for writing *tempOffset*

Command: WTM_1_20_6_-3340↵

Response: WTM_1_20_6_-3340↵

This example adjusts the temperature offset to -3.34 K.

Note: The default value of *tempOffset* is 0. Resistive temperature sensors are generally quite precise and long-term stable. In many typical applications it is not required to adjust this offset.

Important: Changing the *tempOffset* register will have immediate impact on the calibration of any optical sensor, which is using the sample temperature sensor for automatic temperature compensation. Therefore, it is generally recommended to adjust *tempOffset* before the optical sensors are calibrated.

Warning: Never change any of the registers *reg0-reg5* and *reg7*. They contain the factory configuration of the internal circuits used for reading out the sample temperature sensors. Please contact PyroScience if you have changed some of these registers accidentally.

Reg. Nr.	Label	Unit	Range	Description
0	*reg0*			Never change!
1	*reg1*			Never change!
2	*reg2*			Never change!
3	*reg3*			Never change!
4	*reg4*			Never change!
5	*reg5*			Never change!
6	*tempOffset*	0.001 K	int32	Offset added to measured temperature
7	*reg7*			Never change!

MODBUS PROTOCOL

3.1   General Structure
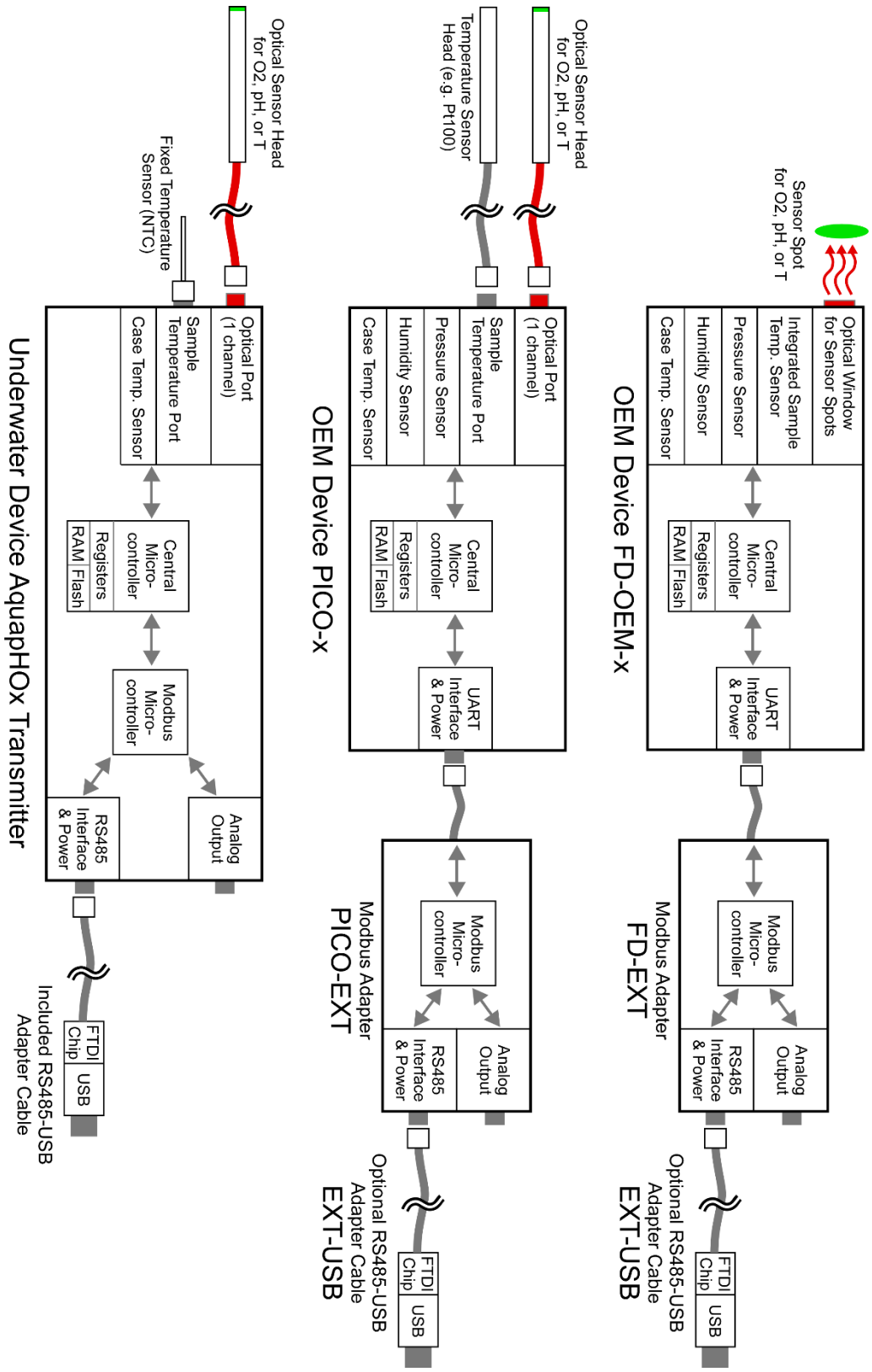
The figure above shows the general structure of some representative solutions from PyroScience offering a Modbus interface.

All PyroScience devices possess a central microcontroller which is communicating via a UART interface based on the proprietary PyroScience protocol (2.1). Devices with a RS485/Modbus interface possess a second "Modbus microcontroller" which acts as a translator: it communicates with the outer world via the RS485 interface based on the Modbus protocol, while it is internally communicating with the central microcontroller based on the PyroScience protocol.

3.1.1 Modbus RTU Standard

PyroScience devices offering a Modbus interface are compatible with the Modbus RTU protocol as described in the official documentation "Modbus over serial line specification and implementation guide V1.02" and "Modbus application protocol specification V1.1b" provided by the Modbus Organization inc. (https://modbus.org).

The advantages of RS485 in combination with the Modbus RTU protocol are:

- cable lengths up to several 100m are supported
- up to 247 devices can be connected to a 4-wire RS485 bus
- Modbus RTU is a popular communication protocol

3.1.2 Modbus Implementation in PyroScience Devices

The PyroScience protocol is based on (1) registers and on (2) commands. In contrast, the Modbus protocol is solely based on registers. It is the task of the Modbus microcontroller to map the functionality of the PyroScience registers and commands onto the Modbus register structure. PyroScience registers are simply mapped onto corresponding Modbus registers as outlined in the following chapters. The implementation of PyroScience commands into the Modbus protocol is a little bit more complex, it is realized by the special command register block as described in 3.3.6.

Note: If the content of Modbus registers is changed by standard Modbus write accesses, these changes are only kept until the next power cycle. Please refer to section 3.3.6 how to make such changes persistent after power cycles.

The enumeration of Modbus registers is always based on 16bit registers. PyroScience devices use by default always 32bit registers (signed integer), so two Modbus registers are always coupled to a 32bit register. Within this document a 32bit register is therefore indicated for example like "40001/40002", which refers to a signed 32bit integer located in the two 16bit Modbus registers 40001 and 40002. The used byte order is "CDAB", so in this example the register 40001 contains the least significant 16bit of the 32bit integer, and the register 40002 contains the most significant ones. This is also known as little-endian byte swap.

3.1.3   Modbus Slave Address

The default Modbus slave address is 1. Refer to 3.3.5 how to change this slave address.

3.1.4   Operation Principle of Periodic Measurements

The periodic measurements mode is activated by writing the desired sample interval in units of ms into the Modbus register *broadcast* located at the address 40021/40022 (3.3.2). The minimum realizable sample interval depends on the hardware. For underwater devices (e.g. AquapHOx) it can be as low as 50 ms, while for most OEM devices (e.g. PICO-x) the minimum realizable sample interval is 1000 ms. The reason is, that these OEM devices are optimized for low energy and not for high speed operation. The maximum possible broadcast interval is 65000 ms = 65 s. Writing a 0 to this register will disable periodic measurements.

The results of the latest periodic measurement can read from the *Results* registers (3.2.1). Or it can be read from the analog output.

3.1.5   Operation Principle of Triggered Measurements

Individual measurements can be triggered by applying the MEA command by using the special command registers. Please refer to 3.3.6 for more details.

The results of the measurement can read from the *Results* registers (3.2.1). Or it can be read from the analog output.

3.1.6   Transparent Mode and Evaluation Software

The Modbus microcontroller can be optionally switched into a transparent mode, where the RS485 communication is again based on the PyroScience protocol (i.e. the Modbus microcontroller is simply transmitting any received message without any changes). By using the optional RS485-USB adapter cables all PyroScience Modbus devices can be operated with the Windows software Pyro DeveloperTool (some devices also with the software Pyro Workbench). These software packages use the transparent mode for operating the devices.

IMPORTANT: During the transparent mode, the analog output is not operational! But the analog output can be configured.

The Modbus protocol can be used with up to 247 devices connected to the same RS485 bus. However, the transparent mode does not work with a bus system. During the transparent mode only a single device can be connected e.g. by a RS485-USB adapter cable to a computer.

3.2 Read-Only Modbus Registers

Read-only Modbus registers, also known as "input registers" are used for information which is readable, but not writeable by the Modbus master. Their entity numbers begin with 3 and they can be read from using function code 4.

3.2.1 Results Registers

The register block *Results* (0) of the PyroScience protocol is mapped to the following Modbus registers:

Modbus protocol				PyroScience protocol	
Entity Nr.	Input register address	Label	Range	Reg. Block	Reg. Nr.
30001/30002	0000/0001	*status*	int32	3	0
30003/30004	0002/0003	*dphi*	int32	3	1
30005/30006	0004/0005	*umolar*	int32	3	2
30007/30008	0006/0007	*mbar*	int32	3	3
30009/30010	0008/0009	*airSat*	int32	3	4
30011/30012	0010/0011	*tempSample*	int32	3	5
30013/30014	0012/0013	*tempCase*	int32	3	6
30015/30016	0014/0015	*signalIntensity*	int32	3	7
30017/30018	0016/0017	*ambientLight*	int32	3	8
30019/30020	0018/0019	*pressure*	int32	3	9
30021/30022	0020/0021	*humidity*	int32	3	10
30023/30024	0022/0023	*resistorTemp*	int32	3	11
30025/30026	0024/0025	*percentO2*	int32	3	12
30027/30028	0026/0027	*tempOptical*	int32	3	13
30029/30030	0028/0029	*ph*	int32	3	14
30031/30032	0030/0031	*ldev*	int32	3	15
30033/30034	0032/0033	-- reserved --	-	-	-
30035/30036	0034/0035	-- reserved --	-	-	-
30037/30038	0036/0037	data point counter	uint32	-	-

The registers 30001/30002 up to 30035/30036 are mapped onto the corresponding *Results* registers. Refer to 0 for details on the definitions of each register.

The registers 30037/30038 contain a 32 bit data point counter, which is incremented by one for each performed measurement. After a reset or a power cycle this counter starts at 0. This register has no counterpart within the PyroScience registers.

3.2.2 Device Info Registers

The device info registers have no counterpart within the PyroScience registers. Instead, they contain the information returned by the PyroScience commands #VERS and #IDNR

of the PyroScience protocol, and some specific information on the Modbus microcontroller.

Modbus protocol					PyroScience protocol	
Entity Nr.	Input register address	Label	Range		Command	Parameter Nr.
36001/36002	6000/6001	*PyroScience device ID*	uint32		#VERS	1
36003/36004	6002/6003	*Number of optical channels*	uint32		#VERS	2
36005/36006	6004/6005	*Firmware version of the central microcontroller*	uint32		#VERS	3
36007/36008	6006/6007	*Available sensor types and supported analytes*	uint32		#VERS	4
36009/36010	6008/6009	*Firmware build number*	uint32		#VERS	5
36011/36012	6010/6011	*Features*	uint32		#VERS	6
36013/36014	6012/6013	*MSW Unique ID number (most significant 32bit of 64bit UID)*	uint32		#IDNR	1 (MSW)
36015/36016	6014/6015	*LSW Unique ID number (least significant 32bit of 64bit UID)*	uint32		#IDNR	1 (LSW)
36017/36018	6016/6017	*Firmware version of the Modbus microcontroller.*	uint32		-	-
36019/36020	6018/6019	*Internal baud rate between the central and the Modbus microcontroller*	uint32		-	-

After power-up the Modbus microcontroller automatically sends the commands #VERS and #IDNR to the central microcontroller and writes the returned parameters into the following Modbus registers.

The registers 36001/36002 up to 36011/36012 contain the information returned by the PyroScience command #VERS. Please refer to 2.2.1 for more details.

The registers 36013/36014 up to 36015/36016 contain the information returned by the PyroScience command #IDNR. Note, that the 64bit unique ID is splitted into two 32bit registers. Please refer to 2.2.2 for more details.

The register 36017/36018 contains the firmware version of the Modbus microcontroller. For example, a value of 114 corresponds to firmware version 1.14.

The register 36019/36020 contains baud rate used for the internal communication between the central microcontroller and the Modbus microcontroller (e.g. 19200 or 115200).

3.3 Read-Write Modbus Registers

Read-write Modbus registers are called "holding registers" are used for data which can be read or written. They use the address space 4xxxx starting with address 40001.

Read-Write Modbus registers, also known as "holding registers", are used for data which can be both read and written by the Modbus master. Their entity numbers begin with 4 and they can be read using function code 3 or written by using function codes 6 and 16.

3.3.1 Making Register Changes Persistent During Power Cycles

The values in the different Modbus registers can be changed by the standard Modbus write commands. It is important to understand, that such changes are only kept until the next power cycle.

In order to make any changes persistent even after a power cycle, the user has to trigger the special command SVS by writing the value 16 to the Modbus register 49001/49002 (refer also to 3.3.6. and 2.3.10).

Background: A Modbus write access will trigger a write to the corresponding PyroScience registers in the central microcontroller, which are kept in the volatile memory. If the special command SVS is triggered, the Modbus controller will send the PyroScience command SVS to the central microcontroller. This will save the current register configuration in the flash memory. After a power cycle, register configuration is automatically read from this flash memory and transferred into the corresponding Modbus registers.

3.3.2 Settings Registers

The register block *Settings* (0) of the PyroScience protocol is mapped to the following Modbus registers:

Modbus protocol				PyroScience protocol	
Entity Nr.	Holding register address	Label	Range	Reg. Block	Reg. Nr.
40001/40002	0000/0001	*temp*	int32	0	0
40003/40004	0002/0003	*pressure*	int32	0	1
40005/40006	0004/0005	*salinity*	int32	0	2
40007/40008	0006/0007	*duration*	int32	0	3
40009/40010	0008/0009	*intensity*	int32	0	4
40011/40012	0010/0011	*amp*	int32	0	5
40013/40014	0012/0013	*frequency*	int32	0	6
40015/40016	0014/0015	*crcEnable*	int32	0	7
40017/40018	0016/0017	Reserved	int32	0	8
40019/40020	0018/0019	*options*	int32	0	9
40021/40022	0020/0021	*broadcast*	int32	0	10

40023/40024	0022/0023	*analyte*	int32	0	11
40025/40026	0024/0025	*fiberType*	int32	0	12
40027/40028 - 40039/40040	0026/0027 – 0038/0039	-- reserved --	int32	0	13 - 19

Most of these Modbus registers are defined exactly as their corresponding PyroScience registers. Refer to section 2.5 for more details.

The only exception is the register *broadcast* (40021/40022) which is only mapped onto the lower 16 bit of the PyroScience register *Settings.broadcast*. These 16bit contain the broadcast interval giving a range of 0-60000ms (0-60s). The upper 16 bit of the PyroScience register *Settings.broadcast* are automatically configured by the Modbus microcontroller.

This *broadcast* register is very essential for operating the device in the periodic sampling mode. Refer for details to 3.1.4.

3.3.3 Calibration Registers

The register block *Calibration* of the PyroScience protocol is mapped to the following Modbus registers:

Modbus protocol						PyroScience protocol	
Entity Nr.	Holding register address	Label (Oxygen)	Label (Temperature)	Label (pH)	Range	Reg. Block	Reg. Nr.
40101/40102	0100/0101	*dphi0*	*M*	*R1*	int32	1	0
40103/40104	0102/0103	*dphi100*	*N*	*pH1*	int32	1	1
40105/40106	0104/0105	*temp0*	-- reserved --	*temp1*	int32	1	2
40107/40108	0106/0107	*temp100*	-- reserved --	*salinity1*	int32	1	3
40109/40110	0108/0109	*pressure*	-- reserved --	*R2*	int32	1	4
40111/40112	0110/0111	*humidity*	-- reserved --	*pH2*	int32	1	5
40113/40114	0112/0113	*f*	*C*	*temp2*	int32	1	6
40115/40116	0114/0115	*m*	-- reserved --	*salinity2*	int32	1	7
40117/40118	0116/0117	*calFreq*	-- reserved --	*offset*	int32	1	8
40119/40120	0118/0119	*tt*	*Tofs*	*dphi_ref*	int32	1	9
40121/40122	0120/0121	*kt*	-- reserved --	*attenuation_coefficient*	int32	1	10
40123/40124	0122/0123	*bkgdAmpl*	*bkgdAmpl*	*bkgdAmpl*	int32	1	11
40125/40126	0124/0125	*bkgdDphi*	*bkgdDphi*	*bkgdDphi*	int32	1	12
40127/40128	0126/0127	*useKsv*	-- reserved --	*dsf_dye*	int32	1	13
40129/40130	0128/0129	*ksv*	-- reserved --	*dtf_dye*	int32	1	14
40131/40132	0130/0131	*ft*	-- reserved --	*pKa*	int32	1	15
40133/40134	0132/0133	*mt*	-- reserved --	*slope*	int32	1	16
40135/40136	0134/0135	-reserved-	-- reserved --	*bottom_t*	int32	1	17
40137/40138	0136/0137	*percentO2*	-- reserved --	*top_t*	int32	1	18
40139/40140	0138/0139		-- reserved --	*slope_t*	int32	1	19

40141/40142	0140/0141		-- reserved --	*pKa_t*	int32	1	20
40143/40144	0142/0143	-reserved-	-- reserved --	*pKa_is1*	int32	1	21
40145/40146	0144/0145	-reserved-	-- reserved --	*pKa_is2*	int32	1	22
40147/40148 – 40159/40160	0152/0153 – 0158/0159	-reserved-	-- reserved --	-reserved-	int32	1	23-29

Note, the register definitions depend on the analyte of the connected sensor head. All these Modbus registers are exactly mapped onto the corresponding PyroScience register. Refer to the sections 2.6, 2.7, and 2.8 for more details.

3.3.4 Analog Output Registers

The register block *AnalogOutput* of the PyroScience protocol is mapped to the following Modbus registers:

Modbus protocol				PyroScience protocol	
Entity Nr.	Holding register address	Label	Range	Reg. Block	Reg. Nr.
40401/40402	0400/0401	*aoSelectA*	int32	4	0
40403/40404	0402/0403	*aoSelectB*	int32	4	1
40405/40406	0404/0405	*aoSelectC*	int32	4	2
40407/40408	0406/0407	*aoSelectD*	int32	4	3
40409/40410	0408/0409	*aoMinA*	int32	4	4
40411/40412	0410/0411	*aoMinB*	int32	4	5
40413/40414	0412/0413	*aoMinC*	int32	4	6
40415/40416	0414/0415	*aoMinD*	int32	4	7
40417/40418	0416/0417	*aoMaxA*	int32	4	8
40419/40420	0418/0419	*aoMaxB*	int32	4	9
40421/40422	0420/0421	*aoMaxC*	int32	4	10
40423/40424	0422/0423	*aoMaxD*	int32	4	11

All these Modbus registers are exactly mapped onto the corresponding PyroScience register. Refer to the sections 2.10 for more details.

3.3.5 Modbus Slave Address Register

A special Modbus register is used for defining the Modbus slave address:

Modbus protocol				PyroScience protocol	
Entity Nr.	Holding register address	Label	Range	Reg. Block	Reg. Nr.
43421/43422	3420/3421	*slaveAddress*	1-247	34	10

This register is mapped to a hidden PyroScience register, which is not documented within this document. The default value is 1. Note, that changes to this register come

effective after the next power cycle. After writing a new slave address to this register it is required to store the register configuration in the flash memory as described in section 3.3.1.

3.3.6   Command Registers

The command registers are used for executing special commands. Some commands affect the Modbus microcontroller directly; other commands are forwarded to the central microcontroller.

Entity Nr.	Holding register address	Description	
49001/49002	9000/9001	Command Register If any of the following numbers are written to this register, the module will first send the Modbus response confirming the writing of this module. Then one of the following actions is executed.	
		Register Content	Performed Action
		0	Sensor module is ready, the command register block can be written and a new command can be triggered by writing a number to this register here.
		1	Sensor module is busy. The command registers cannot be written. A write access will give the Modbus error "Slave Busy".
		10	Flash status LED of the device. (sends the #LOGO command to the central microcontroller)
		11	Trigger a single oxygen measurement. (sends the MEA command to the central microcontroller)
		12	Calibrate an oxygen sensor at zero oxygen level. (sends a CLO command to the central microcontroller)
		13	Calibrate an oxygen sensor at air oxygen level. (sends a CHI command to the central microcontroller)
		14	Calibrate an optical temperature sensor. (sends a COT command to the central microcontroller)
		15	Calibrate a pH sensor. (sends a CPH command to the central microcontroller)
		16	Store all registers in flash memory as new default values after a power cycle. (sends a SVS command to the central microcontroller)
		7777	Triggers a firmware reset.
		8888	Switch into a transparent mode.
49003/49004	9002/9003	Parameter Register 1 required by some commands.	
49005/49006	9004/9005	Parameter Register 2 required by some commands.	
49007/49008	9006/9007	Parameter Register 3 required by some commands.	
49009/49010	9008/9009	Parameter Register 4 required by some commands.	
49011/49012	9010/9011	Parameter Register 5 required by some commands.	

49013/49014	9012/9013	Parameter Register 6 required by some commands.
49015/49016	9014/9015	Parameter Register 7 required by some commands.
49017/49018	9016/9017	Parameter Register 8 required by some commands.

If a command requires parameters, it is necessary to write these into the parameter registers, before triggering the command by writing the corresponding number into 49001/49002.

Since PyroScience Modbus devices only support one optical channel, the first parameter register (49003/49004) refers to the first actual parameter necessary for each command, not the channel number.

Example to execute the MEA command, the "enabled sensor types" parameter s has to be written into registers 49003/49004 and then 11 has to be written into registers 49001/49002.

Writing 10 to the Command Register will trigger, that the Modbus controller sends the PyroScience command #LOGO to the central microcontroller. Please refer to 2.2.3 for more details on this command.

Writing 11 to the Command Register will trigger, that the Modbus controller sends the PyroScience command MEA to the central microcontroller. The required parameter must be written to Parameter Register 1. Please refer to 2.3.1 for more details on this command.

Writing 12 to the Command Register will trigger, that the Modbus controller sends the PyroScience command CLO to the central microcontroller. The required parameters must be written to the Parameter Registers. Please refer to 2.3.3 for more details on this command.

Writing 13 to the Command Register will trigger, that the Modbus controller sends the PyroScience command CHI to the central microcontroller. The required parameters must be written to the Parameter Registers. Please refer to 2.3.2 for more details on this command.

Writing 14 to the Command Register will trigger, that the Modbus controller sends the PyroScience command COT to the central microcontroller. The required parameters must be written to the Parameter Registers. Please refer to 2.3.4 for more details on this command.

Writing 15 to the Command Register will trigger, that the Modbus controller sends the PyroScience command CPH to the central microcontroller. The required parameters must be written to the Parameter Registers. Please refer to 2.3.5 for more details on this command.

Writing 16 to the Command Register will trigger, that the Modbus controller sends the PyroScience command SVS to the central microcontroller. Please refer to 2.3.10 for more details on this command.

Writing 7777 to the Command Register will trigger a firmware reset, equivalent to a power cycle of the complete device.

Writing 8888 to the Command Register will switch the Modbus microcontroller into a so called "transparent mode", in which the Modbus microcontroller will passively forward any message in either direction. This way the user can communicate directly with the central microcontroller based on the PyroScience protocol as given in section 2.1. The transparent mode is terminated by sending the following 3 chars: ##<CR>

IMPORTANT: During the transparent mode, the analog output is not operational! But the analog output can be configured.

The Modbus protocol can be used with up to 247 devices connected to the same RS485 bus. However, the transparent mode does not work with a bus system. During the transparent mode must be only a single device connected e.g. by a RS485-USB adapter cable to a computer.

CONTACT

PyroScience GmbH
Kackertstr. 11
52072 Aachen
Deutschland

Tel.: +49 (0)241 5183 2210
Fax: +49 (0)241 5183 2299
info@pyroscience.com
www.pyroscience.com