



Alpha-Log

Communication protocols



Document Alpha-Log – Communication protocols

Pages 38

Revisions list

<i>Issue</i>	<i>Data</i>	<i>Description of changes</i>
Origin	27/11/2017	
a	12/04/2021	Removed Pluvi-ONE; added Alpha-Log and new command of Modbus RTU and Modbus TCP parts
b	15/11/2021	Added MQTT protocol; formatting of some tables
c	04/07/2022	Added examples of use with modpoll program
5 (d)	12/09/2024	Added SDI-12, SAP and http protocols with REST API

Notes about this manual

The information contained in this manual may be changed without prior notification. No part of this manual may be reproduced, neither electronically or mechanically, under any circumstance, without the prior written permission of LSI LASTEM.

LSI LASTEM reserves the right to carry out changes to this product without timely updating of this document.

Copyright 2017-2024 LSI LASTEM. All rights reserved.

Table of contents

1	Introduction.....	5
2	Modbus RTU protocol.....	5
2.1	Messages format.....	5
2.1.1	Network address.....	5
2.1.2	Function code	6
2.1.3	CRC16.....	6
2.2	Communication lines.....	6
2.3	Supported functions.....	6
2.3.1	Read Coils.....	6
2.3.2	Read Holding Registers	8
2.3.3	Read Input Registers	8
2.3.4	Write Single Coil.....	13
2.3.5	Write Multiple Coils	14
2.3.6	Write Multiple Registers	15
2.3.7	Read Device Identification	17
2.4	Exception codes	18
2.5	Check with modpoll program.....	18
3	Modbus TCP protocol	21
3.1	Physical level	21
3.2	Addressing.....	21
3.3	Supported functions.....	21
3.4	Check with modpoll program.....	21
4	Protocollo MQTT	23
4.1	MQTT on Alpha-Log	24
4.2	Data organization	24
4.2.1	Instantaneous data	25
4.2.2	Processed data.....	25
4.2.3	Configuration of instantaneous and processed data.....	26
4.2.4	Diagnostic data	28
5	SDI-12 Protocol.....	31
5.1	Request comands for measurement values.....	31
6	SAP Protocol	33
6.1	Message format	33
6.2	Commands	35

6.2.1	Access commands to data produced by measurements.....	36
6.3	Data format in the payload	36
7	Http protocol with REST API	37
7.1	Configuring Alpha-Log.....	37
7.2	Using the API	37

1 Introduction

Alpha-Log supports some different communication protocols. In addition to the native *SAP* protocol, it supports *Modbus RTU* and *TCP*, *SDI-12*, *MQTT*, *TCP/IP* and others.

This manual describes the *Modbus RTU* and *Modbus TCP* protocol in slave mode, *MQTT* and *SDI-12*.

2 Modbus RTU protocol

Modbus RTU is a serial communication protocol widely used in industry to enable communication between a *master* (usually a PC or a SCADA system) and one or more *slaves* (measuring instruments, control or PLC), connected to the same communication bus. *Modbus* defines how the *master* and *slaves* devices establish and interrupt the communication, how messages are exchanged and how errors are detected. Only the master can initiate the communication.

Each device connected to the bus is assigned a unique address. A *Modbus* command contains the address of the destination device which the message is direct to. Only the addressed device will respond to the command, although other instruments are receiving the same message. All transmitted *Modbus* messages contain control information, that allow the receiver to evaluate their correct reception.

2.1 Messages format

Master/slave devices use following messages format:

Field name	Size	Description
Address	1 byte	The address of the slave device with which the master must communicate (instrument network address); broadcast message (ID = 0) is not supported
Function code	1 byte	Data field (payload) of variable length
Data	n bytes	Data payload
CRC16	2 bytes	Message control code calculated according to the CRC16 algorithm

After receiving the message, the slave device analyzes the correspondence between its own network address and the one contained in the message; it also evaluates the control code against the one calculated independently. If both conditions are verified, the slave device responds to the received message, otherwise it discards the message without transmitting any reply (§2.3.4).

2.1.1 Network address

The address is used to identify the message's receiver: it includes the numeric address of selected slave device. It can have values from 1 to 247. Use 3DOM application to change the *network address*.

Broadcast messages (address = 0) are not supported.

2.1.2 Function code

Function code detects the command to execute or the manages command by the slave.

Instrument supports following function codes:

Function code	Function name	Description
01	Read Coils	It reads the status of the actuators and the operation state of the instrument
03	Read Holding Registers	It reads last acquired values by the instrument (digital and analogue measures)
04	Read Input Registers	Same as <i>Read Holding Registers</i>
05	Write Single Coil	It sets up an instrument's actuator state
0F	Write Multiple Coils	It resets the instrument's operation errors
10	Write Multiple Registers	It sets the system date time and some measurement configuration parameters
2B	Read Device Identification	It reads the instrument information

The instrument does not consider any function codes not included into the table, and in case of unmanaged command, it does not replay any errors (exception).

2.1.3 CRC16

CRC16 includes the cyclic redundancy code (*Cyclic Redundancy Check*) that has been calculated with CRC16 algorithm (polynomial $X^{16} + X^{15} + X^2 + 1$). For calculation of these two characters, the message is considered like one continuous binary number and its most important bit (MSB) is broadcast first.

2.2 Communication lines

Alpha-Log supports Modbus RTU on the RS485 (called *Com3*) and RS232 (called *Com2*) communication lines. Refer to the instrument manual to find the electrical connection diagram.

2.3 Supported functions

2.3.1 Read Coils

Use *Read Coils* to read instrument's actuators state and possible operation errors.

Note: the command does not provide the status of the actuators available on the ALIEM units possibly connected to Alpha-Log.

Following table sums up the meaning.

Coils	Value	Meaning
Status of actuators	0	Actuation output switched off
	1	Actuation output switched on
Operating status	0	Not in error
	1	Error

Address (Hex)	Coil	Meaning
Status of actuators		
0x00	01	Status of actuator no. 1
	02	Status of actuator no. 2
	03	Status of actuator no. 3
	04	Status of actuator no. 4
	05	Not used
	06	Not used
	07	Not used
	08	Not used
Operating status (errors)		
0x01	09	Error Read After Write
	10	Saved configuration not-valid
	11	Search error of storage page
	12	Max loop number exceeded during stored data search
	13	Overrun acceptance queue acquisition request
	14	Overrun storage queue acquisition results
	15	Not used
	16	Failed page writing in store
0x02	17	One or more pages of processing data lost
	18	Failed deleting store sector
	19	Timeout about waiting operation stop in store
	20	Store device not supported
	21	Wrong received message (check code not valid)
	22	Message repeated up to 3 times max.
	23	Message has been deleted owing to max repeat
	24	Error deleted during writing/reading of EEPROM slave inside
0x03	25	Slave cannot operate because configuration hasn't been programmed
	26	Overflow reception of single message
	27	Messages queue full in slave
	28	All CISS errors about syntax
	29	CISS Error not specified (Unspecified)
	30	CISS Error (BadCommandCode)
	31	CISS Error (BadParameter)
	32	CISS Error (ParameterOutOfRange)
0x04	33	CISS Error (UnrecognizedCDV)
	34	CISS Error (BeyondMaxClassLevel)
	35	CISS Error (ParameterIndexOutOfRange)
	36	CISS Error (ClassIndexOutOfRange)
	37	CISS Error (RequestNotPermitted)
	38	Not used
	39	Not used
	40	Not used

Request

Function code	1 byte	0x01
Start address	2 bytes	from 0x00 to 0x27
Coils number	2 bytes	from 1 to 40

Answer

Function code	1 byte	0x01
Byte number	1 byte	1
Actuator/error status	1 byte	1=On, 0=Off

Error

Function code	1 byte	0x81
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see §2.4.

Example: reading request about instrument's actuators state with ID equal to 01:

Answer to request with following values: 0, 0, 1, 0, 0, 0, 0, 0:

Request	
Field name	(Hex)
Device address	01
Function code	01
Start address (Hi)	00
Start address (Lo)	00
Actuators number (Hi)	00
Actuators number (Lo)	08
CRC16 (Hi)	3D
CRC16 (Lo)	CC

Answer	
Field name	(Hex)
Device address	01
Function code	01
Byte number	01
Value	04
CRC16 (Hi)	50
CRC16 (Lo)	4B

2.3.2 Read Holding Registers

Read Holding Registers and *Read Input Registers* are similar. In the Request use 0x03 function code for the first case and 0x04 for the second.

For more information, see *Read Input Registers* explanation command.

2.3.3 Read Input Registers

Use the function *Read Input Registers* to read last values acquired by the instrument, both analogue and digital measures, and the system date time. Acquired values can be read in numeric floating-point format by requesting them from the address 0 (0x0000) and in integer format from the address 1000 (0x03E8) while the date time from the address 2000 (0x07D0).

The address to set for each request is as follows:

Address (Hex)	Number of registers	Meaning
IEEE754 float values		
0x0000	2	Measure value 1
0x0002	2	Measure value 2
...
0x00C4	2	Measure value 99
Integer values (WORD)		
0x03E8	1	Measure value 1
0x03E9	1	Measure value 2
...
0x044B	1	Measure value 99
System date time (yy MM dd hh mm ss)		
0x07D0	1	yy MM
0x07D1	1	dd hh
0x07D2	1	mm ss

2.3.3.1 Values expressed in floating point format

Each measure value transmitted in floating point format required 4 byte, so it uses 2 Modbus registers. Floating point values are expressed as indicated in the IEEE754 standard.

Through program 3DOM it's possible select the data format between *Big Endian* and *Little Endian*. Data transmission, with option set to *false*, starts from the most significant byte (MSB - Most Significant Byte) while with option set to *true* it starts from the least significant byte (LSB). i.e. the value 11,0 is stored from the address 0x20 like follow:

			Memory address (Hex)	(Hex) value in Big Endian format	(Hex) value in Little Endian format
			...		
Value 11,0	Register 1	Byte 1	0x20 (Hi)	00	30
		Byte 2	0x20 (Lo)	00	41
	Register 2	Byte 1	0x21 (Hi)	41	00
		Byte 2	0x21 (Lo)	30	00
			...		

As the frame consists of 255 characters, it is possible to transmit max 60 measures for each request message. It is necessary arrange two requests to obtain all 99 measures.

The value -999999 (0xF02374C9), unless otherwise specified in the instrument configuration (parameter set via 3DOM), corresponds to *measure in error*.

Request

Function code	1 byte	0x04
Start address	2 bytes	from 0x0000 to 0x00C4
Number of registers	2 bytes	from 2 to 198 (max 120)

Answer

Function code	1 byte	0x04
Number of bytes	1 byte	2 x N*
Value	2 bytes x N*	

*N = number of measures.

Error

Function code	1 byte	0x84
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see chapter 2.4.

Data area

#Measure	1	2	3	4	5	6	...	99
Address (hex)	0x00	0x02	0x04	0x06	0x08	0x0A	...	0x0C4

Example: request for reading, in the float format, of measures 3 and 4 of instrument with ID equal to 01:

First measure that has to be read is number 3, so set up the start address 0x04, like specified in *Data area*. The output number (registers) is 0x04 because each measure consists of 2 bytes as specified in the request.

Answer to request with values 99.0 for measure 3 (output value 1) and 101.0 for measure 4 (output value 2):

Request	
Field name	(Hex)
Device address	01
Function code	04
Start address (Hi)	00
Start address (Lo)	04
Output number (Hi)	00
Output number (Lo)	04
CRC16 (Hi)	B0
CRC16 (Lo)	08

Answer	
Field name	(Hex)
Device address	01
Function code	04
Byte number	08
Output value 1 (byte 1)	00
Output value 1 (byte 2)	00
Output value 1 (byte 3)	42
Output value 1 (byte 4)	C6
Output value 2 (byte 1)	00
Output value 2 (byte 2)	00
Output value 2 (byte 3)	42
Output value 2 (byte 4)	C4
CRC16 (Hi)	13
CRC16 (Lo)	C9

2.3.3.2 Values expressed in integer binary format

In the integer binary format each measure consists of 2 bytes that is one Modbus register. Data format is *Little Endian*. Data storage starts from less significant byte (LSB). So, the value 1149 (0x0545) is stored from the address 0x03E8 like follow:

			Memory address (Hex)	Value (Hex) in Little Endian format
			...	
Value 1149	Register 1	Byte 1	0x03E8 (Hi)	05
		Byte 2	0x03E8 (Lo)	45

The instrument uses the setting of the precision of each specific measurement (number of digits after decimal) to calculate the integer value transmitted by Modbus. For example, the value 23.45 of a measure set with two decimal values becomes 2345, while the value 68.4 of a measure set with a decimal value becomes 684.

Unlike to the floating-point format, for the integer type it is possible to obtain all the 99 measures, which the data logger is capable, in a single request.

A value of -1 (0xFFFF), unless otherwise specified in the instrument configuration (parameter set via 3DOM), corresponds to *measure in error*.

Request

Function code	1 byte	0x04
Start address	2 bytes	da 0x03E8 a 0x044B
Number of registers	2 bytes	da 1 a 99

Answer

Function code	1 byte	0x04
Number of bytes	1 byte	2 x N*
Value	2 bytes x N*	

*N = number of measures.

Error

Function code	1 byte	0x84
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see chapter 2.4.

Data area

# Measure	1	2	3	4	...	99
Address (hex)	0x03E8	0x03E9	0x03EA	0x03EB	...	0x044B

Example: request for reading, in the integer format, of measure 3 of instrument with ID equal to 01:

The measure that has to be read is number 3, so set up the start address 0x03EA, like specified in Data area. The output number (register) is 0x01 because the measure consists of 1 byte like specified in the request.

Request	
Field name	(Hex)
Device address	01
Function code	04
Start address (Hi)	03
Start address (Lo)	EA
Output number (Hi)	00
Output number (Lo)	01
CRC16 (Hi)	A5
CRC16 (Lo)	BA

Answer to request with value 1343:

Answer	
Field name	(Hex)
Device address	01
Function code	04
Byte number	02
Output value 1 (byte 1)	05
Output value 1 (byte 2)	3F
CRC16 (Hi)	FB
CRC16 (Lo)	04

The measure that has to be read is number 3, so set up the start address 0x03EA, like specified in *Data area*. The output number (register) is 0x01 because the measure consists of 1 byte like specified in the request.

2.3.3.3 System date time

Request

Function code	1 byte	0x04
Start address	2 bytes	from 0x07D0 to 0x07D2
Number of registers	2 bytes	from 1 to 3

Answer

Function code	1 byte	0x04
Number of bytes	1 byte	2 x N*
Value	2 bytes x N*	

*N = number of registers.

The sequence of date/time fields starting from address 0x07D0 is: yy MM dd hh mm ss

Error

Function code	1 byte	0x84
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see chapter 2.4.

Example: request for reading the date time of instrument with ID equal to 01:

Answer to request with date time 01/02/21
10:36:42:

Request	
Field name	(Hex)
Device address	01
Function code	04
Start address (Hi)	07
Start address (Lo)	D0
Output number (Hi)	00
Output number (Lo)	03
CRC16 (Hi)	B0
CRC16 (Lo)	86

Answer	
Field name	(Hex)
Device address	01
Function code	04
Byte number	06
Output value 1 (byte 1)	15
Output value 1 (byte 2)	02
Output value 2 (byte 1)	01
Output value 2 (byte 2)	0A
Output value 3 (byte 1)	24
Output value 3 (byte 2)	2A
CRC16 (Hi)	A4
CRC16 (Lo)	D7

2.3.4 Write Single Coil

Use *Write Single Coil* function to set up the state of instrument's actuators (digital outputs).

The value equal to *0x0000* sets up the actuator's output to *0*; *0xFF00* sets up the actuator's output to *1*. State *0* usually indicates actuator switched-off, and state *1* indicates actuator switched-on.

Request

Function code	1 byte	0x05
Start address	2 bytes	from 0x0000 to 0x0003
Value	2 bytes	0x0000 or 0xFF00

Answer

Function code	1 byte	0x01
Start address	2 bytes	from 0x0000 to 0x0007
Value	2 bytes	0x0000 or 0xFF00

Error

Function code	1 byte	0x85
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see §2.4.

Example: request to set up to 0 the state of actuator number 3 of the instrument with ID equal to 01:

Request	
Field name	(Hex)
Device address	01
Function code	05
Start address (Hi)	00
Start address (Lo)	02
Value (Hi)	00
Value (Lo)	00
CRC16 (Hi)	6C
CRC16 (Lo)	0A

Answer	
Field name	(Hex)
Device address	01
Function code	05
Start address (Hi)	00
Start address (Lo)	02
Value (Hi)	00
Value (Lo)	00
CRC16 (Hi)	6C
CRC16 (Lo)	0A

2.3.5 Write Multiple Coils

Use *Write Multiple Coils* function to reset possible operation errors detected through *Read Coils* function. This command executes on all bit together; it is not possible to reset only one or a group of errors. Following command reset all detected errors.

Request

Function code	1 byte	0x0F
Start Code	2 bytes	0x0000
Coil Number	2 bytes	32
Byte number	1 byte	4
Value	4 bytes	0x0000

Answer

Function code	1 byte	0x0F
Start Code	2 bytes	0x0000
Coil Number	2 bytes	32

Error

Function code	1 byte	0x8F
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see §2.4.

Example: request to set to 0 the operation errors of instrument with ID equal to 01:

Request	
Field name	(Hex)
Device address	01
Function code	0F
Start address (Hi)	00
Start address (Lo)	00
Coils number (Hi)	00
Coils number (Lo)	20
Byte number	04
Value 1 (Hi)	00
Value 1 (Lo)	00
Value 2 (Hi)	00
Value 2 (Lo)	00
CRC16 (Hi)	C4
CRC16 (Lo)	88

Answer	
Field name	(Hex)
Device address	01
Function code	0F
Start address (Hi)	00
Start address (Lo)	00
Coils number (Hi)	00
Coils number (Lo)	20
CRC16 (Hi)	21
CRC16 (Lo)	79

2.3.6 Write Multiple Registers

Use *Write Multiple Registers* function to set the system date time and some measures configuration parameters.

The address to set for each request is as follows:

Address (Hex)	Number of registers	Meaning
System date time (yy mm dd hh mm ss)		
0x07D0	3	yy mm dd hh mm ss
Measure configuration		
0x07DA	5	Value used for measurement error in the decimal binary and floating-point type, measure enable (up to max 32)

The configuration parameters of the measure that can be changed are:

- Value to assign to the measure when it is in error in the WORD format (2 bytes = 1 register);
- Value to assign to the measure when it is in error in the float format (4 bytes = 2 registers); use *Little Endian* format;
- Measure enabled status (one bit for each activation status of the measure for a maximum of 32 measures (4 bytes = 2 registers); the measure not enabled will be considered in error.

These parameters are stored permanently until they are reset or until the next reconfiguration of the instrument by 3DOM. In this case they are set with default values: -1 for binary decimal values, -999999 for floating point values, all measures enabled.

Example 1: request to set the system date time to 09/06/21 16:03:05 (yy/mm/dd hh:mm:ss format):

Request	
Field name	(Hex)
Device address	01
Function code	10
Start address (Hi)	07
Start address (Lo)	D0
Registers number (Hi)	00
Registers number (Lo)	03
Byte number	06
Value 1 (Hi)	15
Value 1 (Lo)	06
Value 2 (Hi)	09
Value 2 (Lo)	10
Value 3 (Hi)	03
Value 3 (Lo)	05
CRC16 (Hi)	B0
CRC16 (Lo)	32

Answer	
Field name	(Hex)
Device address	01
Function code	10
Start address (Hi)	07
Start address (Lo)	D0
Coils number (Hi)	00
Coils number (Lo)	03
CRC16 (Hi)	80
CRC16 (Lo)	85

Example 2: request to set the measures parameters with the following values: -12345 (0xCFC7) for measure in error for binary decimal type, -12345678 (0x4E613CCB) for measure in error for float type and all measures disabled except the first 3:

Request	
Field name	(Hex)
Device address	01
Function code	10
Start address (Hi)	07
Start address (Lo)	DA
Registers number (Hi)	00
Registers number (Lo)	05
Byte number	0A
Value 1 (Hi)	C7
Value 1 (Lo)	CF
Value 2 (Hi)	4E
Value 2 (Lo)	61
Value 3 (Hi)	3C
Value 3 (Lo)	CB
Value 4 (Hi)	07
Value 4 (Lo)	00
Value 5 (Hi)	00
Value 5 (Lo)	00
CRC16 (Hi)	6C
CRC16 (Lo)	11

Answer	
Field name	(Hex)
Device address	01
Function code	10
Start address (Hi)	07
Start address (Lo)	DA
Coils number (Hi)	00
Coils number (Lo)	05
CRC16 (Hi)	20
CRC16 (Lo)	85

2.3.7 Read Device Identification

Use *Read Device Identification* to get information about instrument, i.e.: producing company name, type, code, serial number and instrument model.

Request

Function code	1 byte	0x2B
MEI type	1 byte	0x0E
Id code device reading	1 byte	01
Id object	1 byte	0x00

Answer

Function code	1 byte	0x2B
MEI*type	1 byte	0x0E
Id code device reading	1 byte	01
Compliance level	1 byte	0x01
Follow	1 byte	0
Next object Id	1 byte	Object Id number
Objects number	1 byte	3
Object 1 Id	1 byte	0
Object 1 Length	1 byte	26
Object 1 Value	object 1 length	“LSI-Lastem - Milan (Italy)”
Object 2 Id	1 byte	1
Object 2 Length	1 byte	23
Object 2 Value	object 2 length	Type, Code, Serial number and user name
Object 3 Id	1 byte	2
Object 3 Length	1 byte	7
Object 3 Value	object 3 length	Instrument version

Error

Function code	1 byte	Function code + 0x80
Exception code	1 byte	01 or 02 or 03 or 04

For detailed information on *Exception code* see §2.4.

Example: here below the example with Alpha-Log type ALP-001, serial number 18020267, firmware version 2.01.00 with ID set to 01:

Request		Answer	
Field name	(Hex)	Field name	(Hex)
Device address	01	Device address	01
Function code	2B	Function code	2B
MEI type	0E	MEI* type	0E
Id Code device reading	01	Id Code device reading	01
Id object	00	Compliance level	01
CRC16 (Hi)	70	follow	00
CRC16 (Lo)	77	Id next object	00
		Objects number	03
		Object 1 Id	00
		Object 1 length	1A
		Object 1 value *	"LSI-Lastem - Milan (Italy)"
		Object 2 Id	01
		Object 2 length	17
		Object 2 value *	"ALP-001; Serial18020267"
		Object 3 Id	02
		Object 3 length	07
		Object 3 value *	"2.01.00"
		CRC16 (Hi)	27
		CRC16 (Lo)	3B

*Hexadecimal value for:

- Object 1 value: [4C][53][49][2D][4C][61][73][74][65][6D][20][2D][20][4D][69][6C][61][6E][20][28][49][74][61][6C][79][29]
- Object 2 value: [41][4C][50][2D][30][30][31][3B][20][53][65][72][69][61][6C][31][38][30][32][30][32][36][37]
- Object 3 value: [32][2E][30][31][2E][30][30]

2.4 Exception codes

The exception codes are transmitted when the command sent to slave cannot be executed, even if it's correct. The returned exception codes are the following:

Code	Name	Meaning
01	ILLEGAL FUNCTION (not supported at the moment)	The function code doesn't correspond with one function supported by the slave device
02	ILLEGAL DATA ADDRESS	The register address specified is not valid
03	ILLEGAL DATA VALUE	The value to assign isn't valid for specified address
04	SLAVE DEVICE FAILURE	Error detected during the command execution

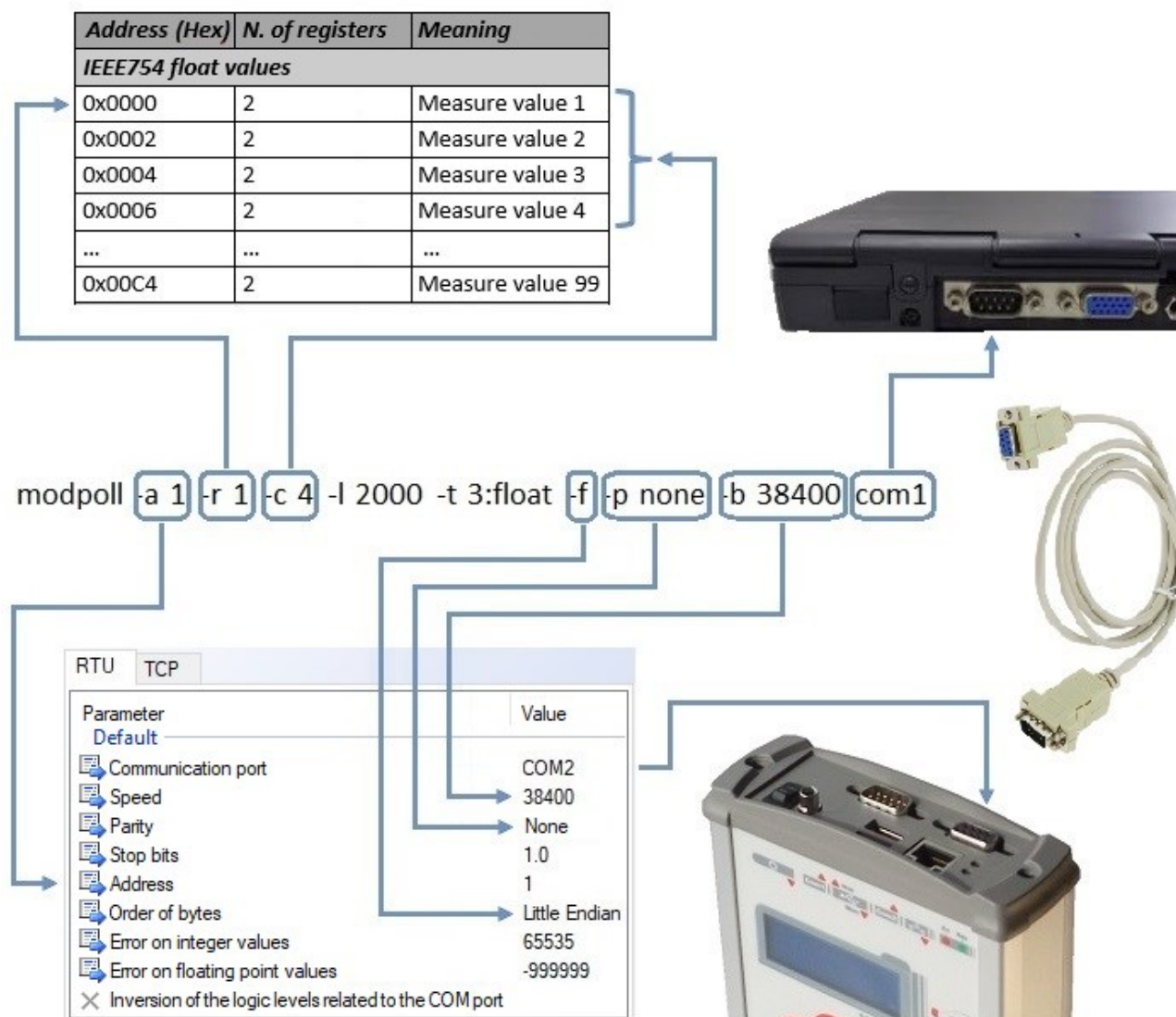
2.5 Check with modpoll program

The function codes 0x03 (Read Holding Registers) and 0x04 (Read Input Registers) can be checked with modpoll.

Modpoll is a command line-based Modbus master simulation program. It can be downloaded for free from the website <https://www.modbusdriver.com/modpoll.html>.

With the modpoll -h command you get the list of parameters that can be used.

As an example, the “construction” of the *Read Input Register* command to request the values of the first 4 measurements of the Alpha-Log configured in Modbus Slave mode on the Com2 serial port is shown.



Following the result:

modpoll 3.1 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2011 proconX Pty Ltd
Visit <http://www.modbusdriver.com> for Modbus libraries and tools.

Protocol configuration: Modbus RTU
Slave configuration....: address = 1, start reference = 1, count = 4
Communication.....: com1, 38400, 8, 1, none, t/o 1.00 s, poll rate 2000 ms
Data type.....: 32-bit float, input register table
Word swapping.....: Slave configured as big-endian float machine

-- Polling slave... (Ctrl-C to stop)

[1]: 22.604630

[3]: 12.187080

[5]: 1002.520020

[7]: 1002.234985

-- Polling slave... (Ctrl-C to stop)

[1]: 22.608200

[3]: 12.187080

[5]: 1003.099976

[7]: 1002.523254

-- Polling slave... (Ctrl-C to stop)

[1]: 22.608200

[3]: 12.187080

[5]: 1003.099976

[7]: 1002.523254

-- Polling slave... (Ctrl-C to stop)

[1]: 22.608200

[3]: 12.187080

[5]: 1003.099976

[7]: 1002.523254

...

3 Modbus TCP protocol

Modbus TCP is an extension of the Modbus RTU protocol. It uses the TCP interface in an Ethernet network. Error checking is not performed via CRC but at the IP frame level. The addressing of the devices is managed via IP.

3.1 Physical level

Alpha-Log can be reached via Modbus TCP if connected to the network via the Ethernet port or via a USB Wi-Fi stick. For more information, refer to the user manual of the instrument.

3.2 Addressing

The addressing of the devices is IP-based. Master and slave, or better, Client and Server, must belong to the same IP network in order to communicate with each other.

Use the 3DOM program to set the IP address.

Broadcast messages (with an address equal to 0) are not supported.

3.3 Supported functions

Modbus TCP supports the *Read Holding Registers* and *Read Input Registers* functions.

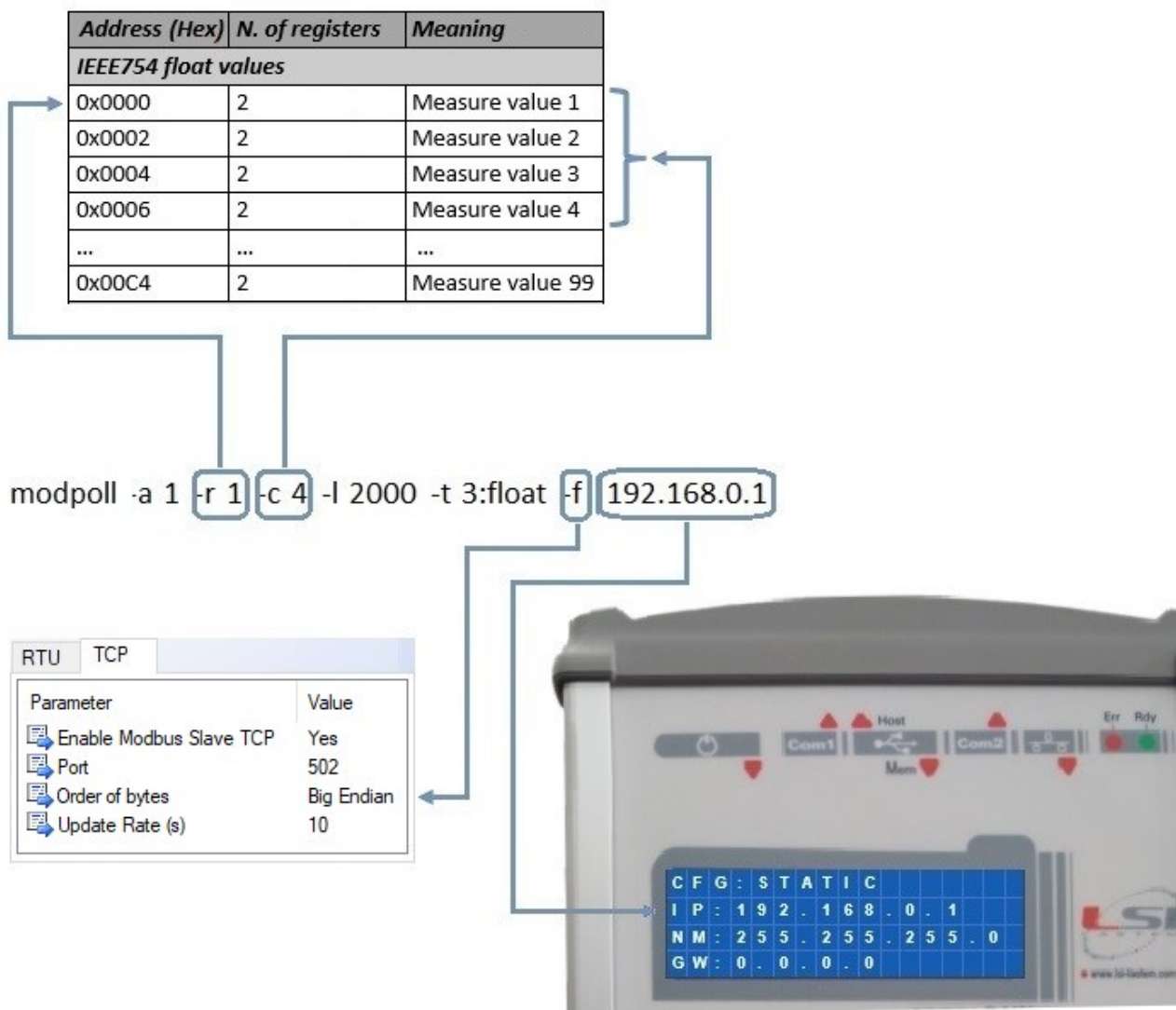
Function code	Function name	Description
03	Read Holding Registers	It reads last acquired values by the instrument (digital and analogue measures)
04	Read Input Registers	Same as <i>Read Holding Registers</i>

Responses to these commands are produced by Alpha-Log in the same way as in Modbus RTU. The functions are implemented exactly as in Modbus RTU. For more information refer to §2.3.2 and §2.3.3.

3.4 Check with modpoll program

As for Modbus RTU, also the function codes 0x03 (Read Holding Registers) and 0x04 (Read Input Registers) of Modbus TCP can be verified with the modpoll simulation program. The parameters to be used are not those of the serial port but those of the TCP connection.

As an example, the “construction” of the Read Input Register command to request the values of the first 4 measurements of the Alpha-Log configured in Modbus Slave TCP is shown.



Below is the result:

```
modpoll 3.1 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2011 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration....: address = 1, start reference = 1, count = 4
Communication.....: 192.168.0.1, port 502, t/o 1.00 s, poll rate 2000 ms
Data type.....: 32-bit float, input register table
Word swapping.....: Slave configured as big-endian float machine

-- Polling slave... (Ctrl-C to stop)
[1]: 22.599998
[3]: 12.200000
[5]: 1002.700012
[7]: 1002.400024
-- Polling slave... (Ctrl-C to stop)
[1]: 22.599998
[3]: 12.200000
[5]: 1002.700012
[7]: 1002.400024
-- Polling slave... (Ctrl-C to stop)
[1]: 22.599998
[3]: 12.200000...
```

4 Protocollo MQTT

Alpha-Log supports the use of the MQTT protocol to send information on measured and elaborations, diagnostics and configurations.

MQTT is a TCP/IP-based protocol and respects the *publish-subscribe pattern* whereby one or more clients connect to a central server called a *broker* and, through it, send and receive data on specific *topics*. The exchange of data takes place through two main actions:

- *publish*: the client requests the broker to publish the data on a related topic;
- *subscribe*: the client asks the broker to obtain all information relating to a topic of interest.

Each client can send information and subscribe to as many topics as they want without any limitation and the broker will simply act as a "card feeder" forwarding the messages identified by a certain topic to all clients currently connected and subscribed to that topic.

Each message sent, in addition to topic and payload, is also associated with a level of *QoS* and a flag called *retain*.

The *QoS* tells the broker with what certainty you want to deliver the message:

- 0 = At most once
- 1 = At least once
- 2 = Exactly once

The *retain* flag indicates whether the message, for the assigned topic, should be kept in memory by the broker. In this case the message will be distributed to the clients subscribed to the topic of the message not only in real time but also to a new subscription of the client, after its publication. The last message with retain sent on each topic will overwrite the message previously saved with retain on the same topic, thus always keeping the last message arrived (with retain) for each topic.

The payload of an MQTT message can have any format; Alpha-Log uses the JSON text format.

4.1 MQTT on Alpha-Log

Alpha-Log is able to send data via MQTT protocol to a configurable broker.

The configuration of this service allows the choice of different parameters:

- the broker's hostname/IP address[mandatory]
- port, default=1883.
- username, default=<empty>.
- password, default=<empty>.
- flags for sending different types of data:
 - instant.
 - processed.
 - diagnostic.
 - alarms.
- If you choose to send instant data, you can choose a sending rate (in seconds)

The topic of the messages is composed of a fixed and unmodifiable base, followed by a specific part for the type of message:

device/<model>/<serial>/<action>

where:

- *<model>*: is the technical code of the product (ALP001, ALP002, ALPxxx depending on the generation).
- *<serial>*: is the device's factory serial or user serial, if specified in the configuration.
- *<action>*: is the specific topic part for each type of data (for example, instantaneous data, metrics/inst).

An example topic for a second generation Alpha-Log device that sends instant data and with serial 21030052 could be:

device/ALP002/21030052/metrics/inst.

All messages are transmitted with QoS=1 that cannot be modified by the user.

4.2 Data organization

The data produced by the tool are organized into:

- instantaneous data: the readings of the single sampling of the single variable (e.g. the current value of the ambient temperature);
- processed data: the statistical processing of an instantaneous data in a time interval (e.g. maximum value of the ambient temperature in the 5 minutes from 16:00:00 to 16:04:59).

The instant data is not saved in the instrument and can only be sent in real time, without the possibility of re-reading. The processed data are instead kept in the memory of the instrument and can be downloaded several times even after their creation. Please note, however, that in the current implementation of MQTT, the processed data are only published by the data logger to the broker according to the set transmission rate.

After their publication, the data, although still resident in the memory, will not be published any further, even if a possible programme subscribed to at the broker for the topic of interest had not been able to receive them (e.g. if the programme had not been running at the time of their publication). In this case, the data are retrievable from Alpha-Log via another protocol (e.g. SFTP). To cope with this limitation, a suitably reliable storage architecture should be implemented, typically server-side and parallel to the MQTT broker.

The processed data produced are grouped by processing instalment: each processing instalment defines a processing basis. The processing bases are identified by a zero-based index decided automatically during configuration. For example, for three processing rates (60", 300", 600") you will have 3 processing bases (in order: 0, 1, 2).

4.2.1 Instantaneous data

The instant data message is a message containing the values of the last data taken from the device, without any statistical processing but with the necessary corrections (engineering, linearisation with polynomial, etc.). The message is sent at a configurable range and cannot be retransmitted.

TOPIC	<code>device/<model>/<serial>/metrics/inst</code>
PAYLOAD	<pre>{ "inst": [1022.4, 14.1, 0.0, 287.48], "time": "2021-10-28T10:04:21" }</pre>

where:

- *inst*: is an array of floating-point numbers containing the instantaneous values of the measurements at that time, in case of measurement in error the transmitted value is *null*.
- *time*: is the timestamp in UTC of the time of reading data from the device.

4.2.2 Processed data

The processed data are derived from a statistical processing over time (average, minimum, maximum, standard deviation, etc.). This type of message is sent at a specified interval during configuration, and does not necessarily coincide with the processing rate.

For each processing base a message is sent containing all the variables processed at the processing rate associated with the base.

TOPIC	<code>device/<model>/<serial>/metrics/elabs</code>
-------	--

PAYLOAD	<pre> { "elab_config_time": "2021-10-27T15:16:33", "last_elab_time": "2021-10-28T01:20:00", "first_elab_time": "2021-10-28T01:11:00", "base": 0, "original_filename": "M18050080-C20211027151633-B00-E20211028011100- L20211028012000.txt", "serial": "18050080", "elab": [{ "items": [45.376802, 9.398528, 200.0, 7200.0, 1014.59, 1014.62, 1014.67, 35.54, 35.55, 35.55], "time": "2021-10-28T01:11:00" }] }</pre>
---------	--

where:

- *elab_config_time*: is the data configuration timestamp.
- *[first/last]_elab_time*: represent the timestamps of the first and last data of the current message.
- *base*: is the processing base to which the data belong.
- *original_filename*: is the name of the message source file, as well as searchable on the device file system in the data folder (*/var/local/ssb/data/history*).
- *serial*: is the factory device serial.
- *elab*: is an array of json objects composed of an items field (the double values resulting from statistical processing).
- *Time*: the reference timestamp for processing.

The message is sent by Alpha-Log only once and with a programmable instalment. This means that if the data logger is programmed to produce data with a single processing base every 10 minutes and the sending instalment is programmed every hour, a single message will be produced every hour containing 6 processing groups.

4.2.3 Configuration of instantaneous and processed data

Instantaneous data and processed data, in order to be correlated with the measure that produced them and the type of statistical processing they represent, are appropriately associated with a descriptive message that enables this association. The instantaneous/processed data, to be recognized, need a descriptive message that allows it to be mapped between double and variable arrays. This type of message, sent with a processing rate specified in the configuration phase, contains information about the measures transmitted.

TOPIC	<i>device/<model>/<serial>/config/metrics</i>
-------	---

PAYLOAD	<pre> { "MsgUpdateRate": 10, "Measures": [{ "MeasKey": "ATM", "Name": "ATM", "Unit": "", "UpdateRate": 5, "Prec": 2, "ElabSingleGuid": 65535, "Elabs": [{ "Rate": 60, "Type": "ScalarStat", "Elements": "Min, Ave, Max", "CoupledGuid": 65535 }], "ProbeSerial": "", "ProbeCert": "", "WMOCode": "", "WMOLevel": "" }, { "MeasKey": "TEMP", "Name": "TEMP", "Unit": "", "UpdateRate": 10, "Prec": 2, "ElabSingleGuid": 65535, "Elabs": [{ "Rate": 60, "Type": "ScalarStat", "Elements": "Min, Ave, Max", "CoupledGuid": 65535 }, { "Rate": 300, "Type": "ScalarStat", "Elements": "Min, Ave, Max", "CoupledGuid": 65535 }], "ProbeSerial": "", "ProbeCert": "", "WMOCode": "", "WMOLevel": "" }] } </pre>	
---------	---	--

where:

- *MsgUpdateRate*: is the rate of sending instant data.
- *Measures*: is an array containing the data of the configured measurements (name, type, id, unit of measure, rate of sampling) and the elaborations configured for each of them. Among these parameters the most important are:
 - o *Name*: measure name (free textual parameter)
 - o *Unit*: measurement unit (free textual parameter)

- *UpdateRate*: sampling rate (whole, in seconds, if =0 the measurement does not carry out periodic sampling but communicates the new measurement value spontaneously)
- *Elabs*: measurement processing arrays, grouped by type (vector/scale) and by processing rate. Each element of the array contains several parameters, the most important of which are:
 - *Rate*: processing rate, in seconds
 - *Type*: type of processing (scalar or vector)
 - *Elements*: elaborations calculated on this measure in the time interval expressed by *Rate* (Inst, Min, Ave, Max, StdDev, Tot, Duration, TimeMin, TimeMax, PrevDir, RisDir, RisVel, StdDevDir, CalmPerc)

In the *Measures* array the measurements are in the same order as the instantaneous data, while to map the processed data you need to verify the sequence of information related to the processing of each measurement.

4.2.3.1 Configuration example

In the example *config/metrics* message above you can find two ATM and TEMP measurements that will reflect only the two values present in the array of instantaneous measurements. Each measurement is sampled every 5 and 10 seconds respectively.

For the ATM measurement, the minimum, average and maximum minute processing of the measured values in that time interval has been configured.

Two processing bases have been configured for the TEMP measurement:

- processing base 0: 60 seconds, with minimum, average and maximum
- processing base 1: 300 seconds, with minimum, average and maximum

The following values will then be produced:

- every 60 seconds: ATM[min], ATM[avg], ATM[max], TEMP[min], TEMP[avg], TEMP[max]
- every 300 seconds: TEMP[min], TEMP[avg], TEMP[max]

Assuming that the rate of data transmission is every 10 minutes (600"), the following *config/metrics* messages will be sent when data is sent:

- a message related to processing base 0, where in the *elab* array there will be 10 elements, each with 6 values within the *items* array.
- a message related to processing base 1, where in the *elab* array there will be 2 elements, each with 3 values within the *items* array.

In case of failure to send, the data will be put in a queue and then re-sent together with any new data created.

4.2.4 Diagnostic data

Diagnostic data is the data obtained by extracting information from different components of the system (Linux OS, data logging and sampling, communications, clocks, etc.)

TOPIC	device/<model>/<serial>/status/diagnostic
-------	---

PAYLOAD	<pre> { "network": { "interfaces": { "eth0": { "tx_bytes": "241758", "ip": "192.168.1.69", "mac": "00:d0:69:4d:6f:de", "rx_bytes": "505146", "internet": { "status": true, "ip": "80.180.46.234" } } } }, "device_space": { "used": 2461060.0, "perc_used": 81.2, "free": 569740.0, "is_full": false, "perc_free": 18.8, "total": 3030800.0 }, "com_stats": { "up_time": 84150.0, "COM": [{ "tx_frames": 0, "name": "COM1", "rx_bytes": 0, "rx_frames": 0, "rx_failed": 0, "tx_bytes": 0 }, { "tx_frames": 0, "name": "COM2", "rx_bytes": 0, "rx_frames": 0, "rx_failed": 0, "tx_bytes": 0 }, { "tx_frames": 0, "name": "COM3", "rx_bytes": 0, "rx_frames": 0, "rx_failed": 0, "tx_bytes": 0 }, { "tx_frames": 0, "name": "COM4", "rx_bytes": 0, "rx_frames": 0, "rx_failed": 0, "tx_bytes": 0 }] } } </pre>
---------	--

(continues)	
PAYLOAD	<pre> { "tx_frames": 0, "name": "COM5", "rx_bytes": 0, "rx_frames": 0, "rx_failed": 0, "tx_bytes": 0 }, { "tx_frames": 11425, "name": "COM6", "rx_bytes": 133377, "rx_frames": 11426, "rx_failed": 0, "tx_bytes": 230795 }] }, "time": { "sbc": "2021-10-28T12:40:11", "sbc_uptime": "14:40:11 up 43 min, 1 user, load average: 0.25, 0.25, 0.32", "ssb": "2021-10-28T14:40:12" } } </pre>

where:

- *network.interfaces*: is a dictionary of network interfaces, where the key is the system name of the interface and the value are its properties, with the following fields:
 - o *tx_bytes, rx_bytes*: bytes transmitted and received.
 - o *ip*: local IP address.
 - o *mac*: MAC address.
 - o *internet.status* e *internet.ip*: [optional] indicate if the device is accessible from the Internet and, if so, with which IP.
- *device_space*: contains information on available space in the device, where:
 - o *is_full*: allows you to understand not only if the disk is full for too much data but also if there are too many files in the queue sending
 - o *used, free*: total number of bytes, in use and free
 - o *perc_used, perc_free*: percentage of space in use and free
- *com_stats*: contains information taken from the low-level sampling system: *uptime* is the number of seconds since the system is switched on while *COM* is the array containing the communication statistics of the various COM ports available. Each object related to the COM port consists of the following properties:
 - o *name*: name of the COM port (COM1, COM2, etc.)
 - o *tx_frames, rx_frames, rx_failed*: number of frames sent, received, error
 - o *tx_bytes, rx_bytes*: number of total bytes transmitted and received
- *time*: contains information on system clocks:
 - o *sbc*: is the UTC timestamp of the SBC card
 - o *ssb*: is the timestamp with SSB card timezone
 - o *sbc_uptime*: contains system boot information

5 SDI-12 Protocol

SDI (*Serial Digital Interface*) is a standard protocol for serial communication between a *master* device, typically a data logger or a SCADA system, and one or more *slave* devices, intelligent sensors based on microprocessor.

The sensors, each identified by a unique address, are powered at 12 V DC. Up to 62 sensors can be connected to the same SDI-12 bus. Communication occurs at 1200 bps with 7 data bits and 1 parity bit.

The peculiarity of the SDI-12 protocol is its low power consumption. The sensors remain in a low power state until they are awakened by the *master* through a command sent on the bus containing the address of the *slave* sensor with which it intends to communicate. The other sensors ignore the command and return to low power mode.

Alpha-Log is compatible with all versions of the SDI-12 protocol. As of the publication date of this document, the latest released version is 1.4 (Jan 2019). The *High Volume and Metadata* commands, introduced in version 1.4, are not currently supported.

5.1 Request commands for measurement values

Alpha-Log supports the following commands for requesting measurement values:

- *Measurement (Measurement - command 'M')*: it is the least efficient mode, but its operation is guaranteed for any SDI-12 sensor and any version of the protocol.
- *Continuous (Continuous Measurements - command 'R')*: it is the most efficient mode when using sensors with a response time of a few seconds.
- *Concurrent Measurements (Concurrent Measurements - command 'C')*: it is the most efficient mode when using sensors with relatively long response times.

The data logger allows the selection of a fourth command for the measurement mode called *Automatic*. In this case, the most efficient measurement mode is automatically set based on the selected protocol version during configuration. It is not guaranteed that the specific sensor supports the mode automatically selected. If the supported modes are not known, it is preferable to select the Measure mode.

In the command, it is possible to specify whether or not to use the CRC control code.

According to the SDI-12 protocol, the sensors provide one or more commands to read the acquired measurements. Each of these commands can return one or more measurements.

As an example, consider the commands described in the manual of an oxygen concentration sensor:

Command	Response	Response to 0D0!
aM! or aM0!	a0013<cr><lf>	Calibrated oxygen, sensor mV, sensor body temperature
aM1!	a0011<cr><lf>	Calibrated oxygen percent corrected for temperature

The table shows that the sensor implements two different measurement commands (aM0 and aM1). The first command returns the concentration of oxygen, a voltage value, and a temperature value. The second command, on the other hand, returns a temperature-corrected oxygen concentration value.

Alpha-Log refers to the measures returned by a specific command as "group of measures". In this case, there are two groups: 0 and 1.

The measures within a specific group have a fixed order.

Alpha-Log refers to the index of a measure within a group with the term "measure index" (or "measure position"). The first measure of a group has index 0.

Each measure is therefore read by specifying the group to which it belongs and the index within the group. In the example considered:

Correct oxygen concentration for the temperature: group 1, index 0

Body temperature of the sensor: group 0, index 2

To know the units of measurement, it is necessary to consult the technical documentation of the sensor.

6 SAP Protocol

SAP (*Simple ASCII Protocol*) is the proprietary communication protocol of Alpha-Log. It provides configuration services and the transfer of measured and processed data from the device, as well as direct control commands for system peripherals (actuation outputs, reading low-level channels, etc.).

The protocol is of ASCII type. Communication occurs via an RS-232 serial line. The system does not require control logics of the RTS/CTS or Xon/Xoff type; therefore, communication occurs with only three wires (Tx, Rx, Gnd) and is set by default with parameters 115200 bps 8N1; other communication parameters are still permitted.

6.1 Message format

Messages are transported by frames in which the beginning of the message is the character '!' or '\$' and the end is identified by the ASCII character CR (Carriage Return); the ASCII character LF (Line Feed) may optionally follow CR, for display reasons on the terminal, but is in any case ignored during reception; during transmission, every response is always terminated by the characters LF+CR.

The message start character '!' is used for easy communication via a terminal emulation program. When greater security is desired or when there is a bus with multiple devices, the message start character is '\$'; in this case, each message also consists of an address field and a validity control code (checksum). If an error condition is identified by the slave, it produces a response with an error identification code, or it does not respond at all when the packet is not fully decoded (e.g., the terminal part is missing); if the packet is received incorrectly by the master or not received at all within the expected time (timeout), the latter will resend a request command for retransmission to the slave; the part sending the retransmission command regulates the maximum number of attempts through which this operation is repeated; the receiving part does not impose a limit on the number of attempts received.

In summary, for manual communications from the terminal (or point to point), the message is composed as follows:

Field name	Description
!	Message start identifier
c	Data flow control
cmd	Specific code of the request or response command
param	Data related to the command, variable length
CR[+LF]	Message end identifier

While in the case of communication managed by a suitably programmed system (point-to-point or point-to-multipoint):

Field name	Description
\$	Message start identifier
dd	Address of the unit to which the message is addressed
ss	Address of the unit that generated the message
c	Data flow control

cmd	Specific code of the request or response command
param	Data related to the command, variable length
XXXX	Hexadecimal encoding in 4 ASCII characters of the control field
CR[+LF]	Message end identifier

The address fields *dd* and *ss* are numbers composed of two ASCII digits, allowing addressing up to 99 different units; the value "00" is intended as a response to the master unit, while the value "--" indicates a broadcast message, intended for any device connected to the master; no response follows the broadcast message from the receiving slave units.

The control byte *c* is used to manage data flow and can take the following values:

Symbol	Description
' '	(Space). First message in a series
'.'	(Dot). Single message or last message in a series
','	(Comma). Other messages to follow
'-'	(Minus). Request to retransmit the previous message (same data)
'+'	(Plus). Request to transmit the next message (subsequent data)

The control field (checksum) is calculated using the CCITT CRC16 algorithm (polynomial $X^{16} + X^{12} + X^5 + 1$) of the characters starting from the character immediately following the message header ('\$') and ending at the character immediately preceding the checksum field itself. The initial value of the calculation is zero. To perform the test of the CRC calculation system, it is possible to send the test command:

\$0100.MIVC3DD[CR] (CRC = 0xC3DD)

to which the instrument (ID = 01) responds with a message like this (in this case, the CRC is reported here as an example because it is also calculated based on the characters that represent the measured data):

\$0001.MIV1003.5;25.4;12.41A15[CR] (CRC = 0x1A15)

The command code *cmd* consists of three characters. It is not case sensitive, so for the tool, the commands MIV and miv are equivalent.

The transmission of data which, due to volume, cannot be included in a single message, is done by specifying the control byte *c* according to the following rules:

- Data transmitted in a single message: the control byte is a *dot* ('.');
 - Data transmitted in more than one message: the control byte can be a *comma* (',') or a *dot* ('.');
- upon receiving the message containing the control byte *comma*, the receiving part must send the message '+' to indicate to the transmitter the possibility of transmitting the next part of the data; upon receiving the message with control byte *dot*, the receiving part may refrain from responding (if the reception was correct), because sending a subsequent message '+' results in the return of a message containing the error code NoMoreData.

There is no specific limit imposed on the number of messages in which the data part is divided; for performance issues on certain communication lines, particularly slow ones or those at high risk of interference (radio), it is advisable that the data part transmitted in each message has relatively small sizes, so the entire set of data is, in this case, divided into a greater number of messages. The maximum size of the data transmitted in each message is a modifiable system parameter (SMS command).

The functions exposed by the communication protocol are:

- Management commands for adjustment;
- Management commands for configuration;
- Diagnostic commands;
- Commands for reading measured and processed data, both instantaneous and recorded in historical memory;
- System management commands;
- Specific commands for SBC (SBC is a card of Alpha-Log);
- Commands for accessing the user interface.

6.2 Commands

The letters used to identify the commands are indicated here in uppercase, but the protocol still considers them valid even if received in lowercase.

In the list of commands, the following conventions are used:

- **NP**: (not present) comando non presente;
- **PE**: (params empty) il comando non ha bisogno di parametri;
- Nella descrizione, per chiarezza, i parametri vengono indicati come **TAG=VALUE** ma nel protocollo reale va inserita solo la parte **VALUE**.
- **NP**: (not present) command not present;
- **PE**: (params empty) the command does not need parameters;
- In the description, for clarity, the parameters are indicated as **TAG=VALUE** but in the actual protocol, only the **VALUE** part should be entered.

This manual considers only the commands for accessing data produced by the measurements.

6.2.1 Access commands to data produced by measurements

Code	Read parameters	Write parameters	Response	Description
MEN	base=0/1/./n	NP	ER, xx	<i>Measures Elabs Num</i> : it requires the number of elaborations included in the specified base group
MIV	PE	NP	ER, xx.xx tt.tt yy.yy...	<i>Measures Instant Value</i> : it requires the instantaneous value of measurements
MRR	PE	NP	OK, ER	<i>Measures Recorded Reset</i> : it deletes all elaborated data from the measurements for the elaborating base specified in the command. Note: the current implementation does not require any parameters and therefore results in the deletion of data produced by all elaborating bases
MRV	base=0/1/./n dt=yyyy-mm-ddThh:mm:ss (o PE)	NP	ER, @yyyy-mm-ddThh:mm:ss;xx.xx;yy.yy;...	<i>Measures Recorded Values</i> : richiede i valori registrati ed elaborati specificati nel comando it requires the recorded and elaborated values specified in the command

6.3 Data format in the payload

During the formatting of the data, the following rules have been followed, when possible:





- Various parameters (both request and response) are separated by the space character; some responses, for clarity when the values are numerous and heterogeneous from a semantic point of view, use tags in the format tag:value;
- The date and time are expressed in ISO 8601 format; normally the tool expresses the time internally, in transmissions and files referred to GMT; durations are expressed in the format "ddd hh:mm:ss";
- Logical status:
 - "Y", "YES", "1", "TRUE", "ON" for true value;
 - "N", "NO", "0", "FALSE", "OFF" for false value.
- Integer numbers: decimal digits in a number depending on the number of bits allocated to the variable intended to hold the data;
- Floating-point values:
 - Decimal separator: dot ('.');
 - Decimal digits: dependent on the transmitted value; if necessary, scientific format is used (mantissaEexponent).

7 Http protocol with REST API





This chapter explains how to query Alpha-Log using the HTTP (*HyperText Transfer Protocol*) protocol and the REST APIs, the *Application Program Interfaces* that conform to the REST architectural style.

7.1 Configuring Alpha-Log

Using the 3DOM software, it is necessary to activate the Modbus TCP protocol and set the desired update rate (e.g., 10 seconds). Avoid rates lower than 10 seconds if the data logger produces processed data for sending to external systems (FTP, CLOUD, MQTT, ...). If the measurements have no processing, it is possible to set the update rate to up to one second.

Parameter	Value
 Enable Modbus Slave TCP	Yes
 Port	502
 Order of bytes	Big Endian
 Update Rate (s)	10

Next, enable the HTTP server.

Parameter	Value
 Enable internal server	Yes
 Port	9999
 User	ALuser
 Password	ALpwd123

7.2 Using the API

The APIs are based on JSON calls (excluding login). In all calls, a cookie called *data_integrator_session* is used. Once logged in, the cookie will allow access to all available APIs. Below is an example with the IP address of Alpha-Log equal to 192.168.1.138 and with Port, User, and Password corresponding to those indicated in the previous image.

Login

POST http://192.168.1.138:9999/di/login
Content-Type: application/x-www-form-urlencoded
POST DATA: username=ALuser&password=ALpwd

Instantaneous values

GET http://192.168.1.138:9999/di/dl_get_inst
REPNSE:
{
 "values": [
 12.5,
 1003.1,
 27.89
]
}

```
},  
"measures": [  
  "TensBATT (V)",  
  "Pcell (hPa)",  
  "t_int (C)"  
],  
"timestamp": "2024-05-29 10:10:31"  
}
```

The response indicates that the data logger is configured with the measurements *TensBATT*, *Pcell* and *t_int*, whose respective values are *12.5 V*, *1003.1 hPa* and *27.89 °C*.