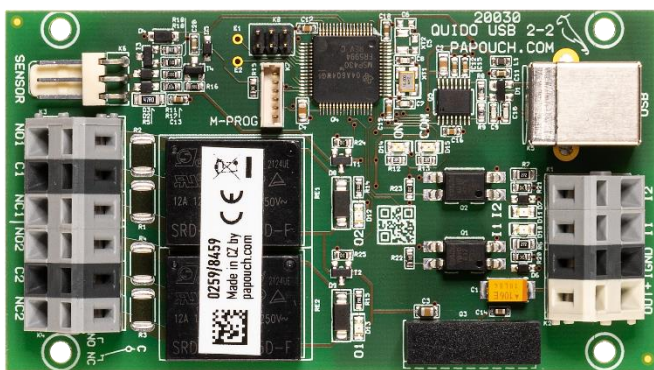




Quido Spinel

Complete description of the Quido I/O
modules communication protocol



Quido Spinel

Datasheet

Created: 23.11.2005

Last update: 18.10.2023 11:18

Number of pages: 48

This document has been designed using resources from Flaticon.com

© 2023 Papouch s.r.o.

Papouch s.r.o.

Address:

**Strasnicka 3164/1a
102 00 Praha 10**

Phone:

+420 267 314 267

Internet:

en.papouch.com

Mail:

papouch@papouch.com



TABLE OF CONTENTS

Changelog	4
Quido communication parameters	4
About Spinel	6
Online Spinel parser	6
Node-RED	6
Spinel.NET C# library	6
Spinel terminal	7
Configuration jumpers	7
Text-based communication format 66	9
How to control Quido easily	9
Closing the relay contact	9
Opening relay contact	9
Read input status	10
Address change	10
Description of text format 66	11
List of format 66 instructions	12
Binary communication format 97	14
Format 97 instructions	16
Inputs	16
Inputs states – 0x31	16
Change Notification: All inputs – 0x10/0x11	17
Change Notification: One input – 0x15/0x16	19
Sampling rate – 0x62/0x63	19
Counter reading – 0x60	20
Subtracting from counter – 0x61	21
Counter mode – 0x6A/0x6B	22
Outputs	23
Output control – 0x20	23
Reading outputs – 0x30	23
Pulse on output – 0x23/0x33	24
Pulse on output: Separate triggering – 0x26/0x36/0x25	25
Output mode check – 0x38	26
Linking input and output – 0x40/0x41	27
Temperature measurement and monitoring	28
Temperature measuring – 0x51	28
Temperature measuring: int, float, string – 0x58	29
Temperature unit – 0x1C/0x1D	30
Temperature limits – 0x13/0x14	30
Thermostat – 0x1A/0x1B	32
Communication port and address	34
Enable configuration – 0xE4	34
Address and baudrate – 0x0E/0x0F	34
Address setting by serial number – 0xEB	35
Others	36
Name and version – 0xF3	36
Manufacturing data – 0xFA	37
User data – 0xE2/0xF2	38
Input names – 0x2B/0x3B	38
Output names – 0x2A/0x3A	39
Status and Run time – 0xE1/0xF1	40
Errors in communication – 0xF4	40
Checksum – 0xEE/0xFE	41
Communication timeout – 0xE5/0xF5	41

Reset – 0xE3	42
Default settings – 0x8F	42
Communication protocol switching – 0xED	42
Appendix 1: Thermostat	43
Mode 1	43
Mode 2	43
Mode 3	43
Mode 4	44
Mode 5	44
Mode 6	44
Mode 7	44
Mode 8	45

Changelog

Version 4.52 ¹

- Set/Reset option added to [the input/output link function](#).
- Two more temperature monitoring modes have been added to [the Thermostat function](#). See modes 7 and 8 in [Appendix 1](#).

Version 4.50

- Document revision.
- Added the option to check the elapsed time since the device restart (run time) to the [read status instruction](#).
- Option to machine read the [number of inputs and outputs](#).
- [Input and output link function](#) can respond to connection failure.

A description of older changes is available only in the Czech language version of this document.

Quido communication parameters

RS232 and RS485 interfaces

Communication speed selectable 300 Bd to 230400 Bd
Default communication speed..... 9600 Bd
 Data bits..... 8
 Parity..... no parity
 Stop bits..... 1

USB interface

Communication speed 115200 Bd (not changeable)
 Data bits..... 8
 Parity..... no parity
 Stop bits..... 1

Ethernet interface

Communication speed 115200 Bd (not changeable)

¹ By version is meant *[hw-version].[sw-version]* as specified in the Name and version – 0xF3. I.e. a device with the identification *Quido USB 4/4; v0253.04.38; f66 97; t1* is version **4.38**.

Data bits8

Parityno parity

Stop bits.....1

How do I find out current communication parameters?

You can find out what communication parameters Quido has set by using one of the shorting jumpers on the Quido board. The procedure is shown on page 7.

ABOUT SPINEL

This document describes Quido I/O Modules communication protocol. Inputs and output counts, communication speeds and interface description is described in a separate document of a particular Quido I/O module. Any exceptions are described at each particular instruction.

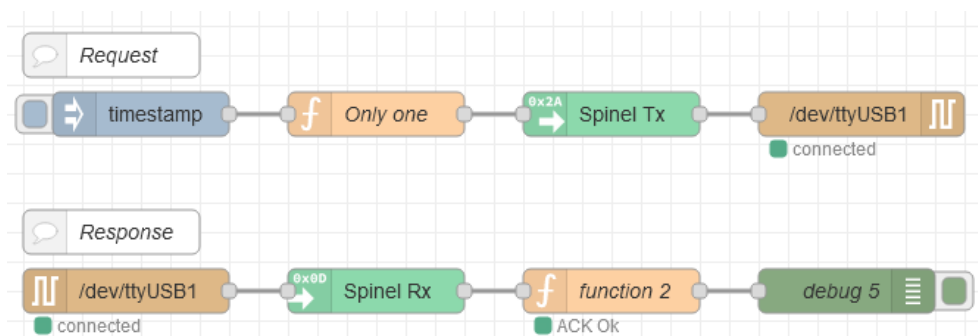
Online Spinel parser

On en.papouch.com/free-binary-serial-protocol-online-parser is a universal tool for creating and parsing Spinel packets.

- PRE: 2AH FRM: 61H NUM: 0006H ADR: 01H SIG: 02H INST: 20H DATA: [82H] SUM: C9H CR: 0DH [link copy](#)
- PRE: 2AH FRM: 61H NUM: 0005H ADR: 01H SIG: 02H ACK: 00H DATA: [] SUM: 6CH CR: 0DH [link copy](#)
- PRE: 2AH FRM: 61H NUM: 0005H ADR: 01H SIG: 02H ACK: 00H DATA: [] SUM: 66H CR: 0DH [link copy](#)

Node-RED

In [the article on papouch.com](#) (in Czech only) you will find a detailed description of working with Spinel in Node-RED.



Spinel.NET C# library

Spinel protocol is also used by [Spinel.NET](#) library for .NET, which is free to download from GitHub. (This library has documentation only in Czech.)

Example of switching OUT 3 for 5 sec ([docs](#)):

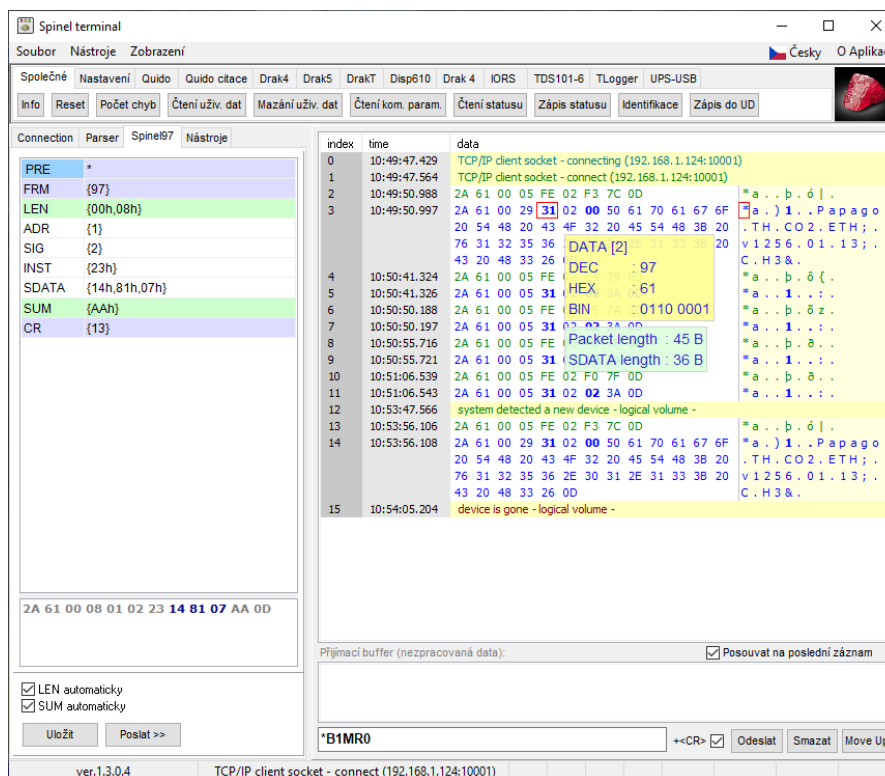
```
if (MyQuido.CmdSetOutput(3, true, 10))
    Console.WriteLine("Ok");
else
    Console.WriteLine("Error");
```

Example of reading number of units from counter on IN 2 input ([docs](#)):

```
if (MyQuido.CmdGetCounter(2, out int counter))
    Console.WriteLine($"Counter value is {counter}.");
```

Spinel terminal

For easier debugging of Spinel devices, the Spinel terminal program is available for free download at en.papouch.com/spinel. It allows communication via serial ports and Ethernet, using the Spinel binary protocol (format 97).



Configuration jumpers

As we make hardware revisions of Quido, we are gradually adding configuration jumpers that make it easier to use Quido in some typical situations.

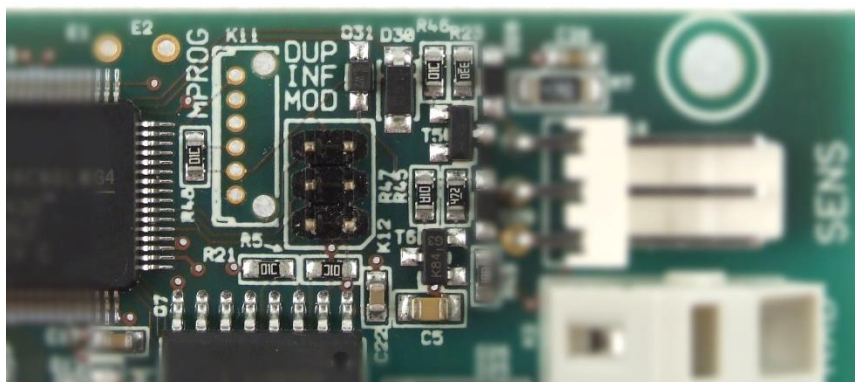


fig. 1 - Example of jumpers on Quido RS 2/2

There are three jumpers – Duplex, Info and Modbus. (As you can see in the picture, sometimes the label in the print is slightly abbreviated.)

Jumper Modbus

If this jumper is shorted when the power is turned on, Quido communicates using the Modbus protocol regardless of the software configuration.

Jumper Info

If this jumper is momentary short-circuited when power is connected, Quido sends the current communication parameters to a serial line.

This information is always sent in the Spinel protocol. The RS version of the device sends the information at 9600 Bd, the USB and ETH versions at 115.2 kBd. Quido sends first the response to instruction *Name and version* – 0xF3 and then a packet with address, speed and protocol in ASCII format. Example:

```
*a?"4N?Address:34 Speed:6 Protocol:1ü?
```

The address is hexadecimal, the baud rate is the code according to instruction *Address and baudrate* – 0x0E/0x0F, and the protocol is the protocol number according to instruction *Communication protocol switching* – 0xED.

Jumper Info must not be short-circuited on startup or reboot!

Jumper Duplex

The jumper activates the two-way transfer mode of input and output states between two Quids 4/4 or 8/8. This set can be ordered as [QuidoDuplexRS](#) or [QuidoDuplexETH](#). The manuals for these sets provide further information on how to configure this mode.

TEXT-BASED COMMUNICATION FORMAT 66

How to control Quido easily

The examples are written for simplicity in "format 66", which is suitable for learning, debugging and communication via the terminal.

For control using your application **in normal operation, we recommend using format 97**, which contains a checksum, and which is described in more detail in the section beginning on page 14.

The following examples assume communication with the module in its default settings. First, send the string specified in the Request column. Between each character there cannot be a delay of more than 5 sec. If all is well, the module will respond as described in the Response column.

Closing the relay contact

The following example switches relay no. 2 on the module with address 1.

Request	Response	Explanation
*B10S2H↵	*B	Prefix
		Address
	1	You can also use \$ as the address. This character is a universal address and only works if one module is connected.
	0S	Instruction code for changing the output state
	2	Output number
	H	Code for switching on (High)
	↵	Terminating character (enter, 0x0D)
	*B10↵	*B Prefix
	1	Device address
	0	Acknowledge code
	↵	Terminating character (enter, 0x0D)

Opening relay contact

The following example opens relay no. 4 on the module with address D.

Request	Response	Explanation
*BD0S4L↵	*B	Prefix
		Address
	D	You can also use \$ as the address. This character is a universal address and only works if one module is connected.
	0S	Instruction code for output state change (Output Set)
	4	Output number
	L	Code for switching off (Low)
	↵	Terminating character (enter, 0x0D)
	*BD0↵	*B Prefix
	D	Device address
	0	Acknowledge code

↵ Terminating character (enter, 0x0D)

Read input status

Example of reading input 3 status when only one device is connected to the serial link (universal address is used).

Request	Response	Explanation
*B\$IR3↵	*B	Prefix
	\$	Universal address
	IR	Instruction code for reading input state
	3	Input number
	↵	Terminating character (enter, 0x0D)
*B10H↵	*B	Prefix
	1	Device address
	0	Acknowledge code
	H	Input is active (High state)
	↵	Terminating character (enter, 0x0D)

Address change

Example of changing device address from **f** to **5**.

Request	Response	Explanation
First you must enable configuration with special instruction. This instruction authorizes a setting change for one subsequent instruction. After any subsequent instruction, configuration is again disabled.		
*BfE↵	*B	Prefix
	f	Address
	E	Instruction code to enable configuration
	↵	Terminating character (enter, 0x0D)
*Bf0↵	*B	Prefix
	f	Device address
	0	Acknowledge code
	↵	Terminating character (enter, 0x0D)

We now have configuration enabled. So, we can change the address.

*BfAS5↵	*B	Prefix
	f	Old address
	AS	Instruction code for address change
	5	New address
	↵	Terminating character (enter, 0x0D)
*Bf0↵	*B	Prefix
	f	Old address
	0	Acknowledge code
	↵	Terminating character (enter, 0x0D)

Description of text format 66

Format 66 uses only decadic variables or characters that can be typed on the keyboard. This format is useful for debugging applications with Spinel. There must be no more than a 5 sec delay between characters. Communication is based on a request-response style:

Packet structure

Request:	PRE	FRM	ADR	INST	[DATA]	CR
----------	-----	-----	-----	------	--------	----

Response:	PRE	FRM	ADR	ACK	[DATA]	CR
-----------	-----	-----	-----	-----	--------	----

- PRE** Prefix, character *
- FRM** Identification of format 66 (character B).
- ADR** Device address to which the request is sent or which sends the response. (One ASCII character.)
- INST** Instruction code – one ASCII character from range A to Z, a to z and digits 0 to 9.
- ACK** Acknowledge whether and how the query was executed. ACKs are from the interval 0 to 9 and A to F (one ASCII character).
- DATA** Transferred data. None, one or more ASCII characters. We recommend to transfer data in common form and units. It must not contain a prefix (*) or terminating character (CR).
- CR** Terminating character (suffix) ²

Explanations

*	B	1	TR	↵
Prefix	Format 66	Address	Instruction code	Suffix ²

ADR – Address

Address is a single character that uniquely identifies a specific device among others on a single communication line. A device always uses this number to identify itself in responses to requests from the parent system. The address may be the following ASCII characters: digits '0' to '9', lower case 'a' to 'z' and upper case 'A' to 'Z'. The address shall not be the same as a prefix or CR.

The address '%' is reserved for 'broadcast'. If the address '%' is used in the query, the device behaves as if its address is specified. No response is returned to queries with this address.

The '\$' address is a universal address. If the address '\$' is queried, the device behaves as if its address is specified. In the response, the device gives the actual address just set. The universal address is used when only one device is connected.

INST – Instruction code

The instruction code of a specific device.

If a valid instruction is received (ADR match) and the received message flag is set, the device must already respond to this instruction.

ACK – Acknowledge

ACK informs the parent device how it processed the received instruction. Acknowledgement codes:

² Carriage Return character with ASCII code 13 (decadic) or 0x0D (hexadecimal). In the examples, it is replaced by the character ↵

- 0.....ALL RIGHT
Instruction was well received and completely executed.
- 1.....GENERAL ERROR
- 2.....INVALID INSTRUCTION CODE
- 3.....INVALID DATA
The data does not have a valid length or contains an invalid value.
- 4.....NOT PERMITTED FOR ANY OF THE FOLLOWING REASONS
 - Attempting to change settings without a previous Configuration Enable instruction.
 - Attempt to write to inaccessible memory.
 - Required device function cannot be performed because the conditions for this are not met. For example, a higher communication speed is required.
 - Access to password-protected memory.
- 5.....FAILURE
 - Device needs service.
 - Device internal memory or settings memory error.
 - Internal device peripheral error.
 - Any other error affecting the proper functioning of the device.
- 6.....NO DATA AVAILABLE
- DAUTOMATICALLY SENT RESPONSE – STATE CHANGE DETECTED ON DIGITAL INPUT

DATA

| Data transferred by this instruction.

List of format 66 instructions

Description	Structure 1. Request 2. Response	Example (addr. is always 1 in examples)
Input status reading	*B[address]IR[input] ³ ↓ *B[address]0[status] ⁴ ↓	*B1IR2↓ *B10H↓
Output status reading	*B[address]OR[output] ⁵ ↓ *B[address]0[status] ⁶ ↓	*B10R4↓ *B10L↓
Output control	*B[address]OS[output] ⁵ [status] ⁶ ↓ *B[address]0↓	*B10S3H↓ *B10↓
Output control for a specified duration	*B[address]OT[output] ⁵ [status] ⁶ [duration] ⁷ ↓ *B[address]0↓	*B10T1H20↓ ⁸ *B10↓
Device name and type	*B[address]?↓ *B[address]0Quido [line] ⁹ [I/O] ¹⁰ ; v[VC] ¹¹ ; F66 97↓	

³ Zero, to get the status of all inputs, or input number (indexed from 1).

⁴ L - inactive input; H - active input

⁵ Output number (indexed from 1).

⁶ L - open contact; H - closed contact.

⁷ On/off length of the chosen output. It is possible to enter the number 1 - 255. The step is 0.5 sec. It is therefore possible to set the time from 0.5 sec to 127.5 sec.

⁸ Switch off output 1 for 10 sec (10 sec = 20 * 0.5).

⁹ Identification of communication interface (USB, ETH or RS).

¹⁰ Number of inputs/number of outputs (e.g., 10/1 for a version with ten inputs and one output).

¹¹ Device version - for example 0227.02.03: product 227, hardware revision 2, firmware version 3.

Enable configuration ¹²	*B[address]E↵	*B1E↵
	*B[address]0↵	*B10↵
Address setting ¹³	*B[old address]AS[new address]↵	*B1AS5↵
	*B[old address]0↵	*B10↵

¹² The universal address \$ cannot be used for this instruction.

¹³ This instruction must be preceded by Enable Configuration.

BINARY COMMUNICATION FORMAT 97

The device communicates in binary – this method is called "format 97" (the initial character has the decade code 97). For communication, binary 8-bit characters are used (decadic range 0 to 255, hexadecimal range 0x00 to 0xFF).

Spinel debugging developer tools are listed on page 7.

Below are two typical examples of request and response structures. The first line contains the names of the individual bytes or groups and the second line contains a specific example of a request or response.

→ Request:

PRE	FRM	NUM	NUM	ADR	SIG	INST	[DATA]	SUM	CR
2A	61	00	07	31	02	61	38, E6	BB	0D

← Response:

PRE	FRM	NUM	NUM	ADR	SIG	ACK	[DATA]	SUM	CR
2A	61	00	05	31	02	00		3C	0D

<i>item</i>	<i>length in bytes</i>	<i>description</i>
PRE	1	Prefix. Always 0x2A, character *, decadic 42.
FRM	1	Number of format. Always 0x61 (hexadecimal), character a, decadic 97.
NUM	2	Number of bytes in the instruction from the next byte to the end of the message (i.e. ADR to CR).
ADR	1	Device address to which the request is sent or which sends the response.
SIG	1	Message Signature. The SIG sent in the request will be returned in the response. By SIG, the query and response can be paired.
INST	1	Instruction code (0x10 to 0xFF).
ACK	1	Request acknowledge (0x00 to 0x0F). Determines if and how the request was executed. A list of standard ACKs is given below.
DATA	x	The length of the data and its content varies according to the specific instruction. ¹⁴
SUM	1	Checksum. Calculation formula: $\text{SUM} = 0xFF - ((\text{PRE} + \text{FRM} + ((\text{NUM} \& 0xFF00) \gg 8) + (\text{NUM} \& 0xFF) + \text{ADR} + \text{SIG} + \text{ACK/INST}^{15} + \text{DATA}) \& 0xFF)$
CR	1	Ending character (Carriage Return, 0x0D, \r, decadic 13).

- **NUM:** Number of bytes from ADR (inclusive) up to and including CR. It is two bytes, so the NUM can be up to 65535. The minimum is 5, which corresponds to an instruction with no data. If the

¹⁴ **Brackets in parameter descriptions:** if a parameter is in square brackets [], it is an optional parameter. If one or more parameters are in round brackets (), the delimited group may be repeated. **The | character** between two parameters or parameter groups means OR - only one of the two parameters (or parameter groups) can be entered.

¹⁵ ACK/INST means ACK or INST.

NUM is less than 5, the packet is not valid. The upper byte is the MSB, the lower byte is the LSB. If the number of bytes is less than 256, the upper byte is null.

- **ADR:** The device address can be in the range 0x00 to 0xFD (253). The following addresses are reserved for special use:
 - 0xFF (255) is broadcast. This means that if the device receives a message with this address, the device will execute the instruction but will not send any response. Configuration using this address is not possible.
 - 0xFE (254) is universal address. If the device receives a message with this address, the device acts as if it were its own address, executes the instruction, and sends a response. The universal address can only be used if there is only one device on the communication link. Configuration cannot be done with this address. Configuration using this address is not possible.
- **ACK** in the same place as the INST byte in the query. It is in the range 0x00 to 0x0F. This byte is used to inform the device about the status of the last received instruction. The reserved ACKs are as follows:
 - 0x00 Ok: The instruction was received and executed.
 - 0x01 General error: Unspecified error.
 - 0x02 Unknown instruction code: Device does not know instruction code.
 - 0x03 Data error: DATA has an unexpected length or contains an unexpected value.
 - 0x04 Not permitted for any of the following reasons:
 - Attempting to change settings without a previous Configuration Enable instruction.
 - Attempt to write to inaccessible memory.
 - Required device function cannot be performed because the conditions for this are not met. For example, a higher communication speed is required.
 - Access to password-protected memory.
 - 0x05 Failure:
 - Device needs service.
 - Device internal memory or settings memory error.
 - Internal device peripheral error.
 - Any other error affecting the proper functioning of the device.
 - 0x06 No data available: For example, shortly after powering up the device, readings from external sensors may not yet be available, etc.
 - 0x0A to 0x0F are messages that the device has sent automatically without any request from the parent system. For example, notifications of changes in the input, periodic measurements, logs, etc.
- **SUM** is the checksum. A message with an incorrect checksum is not responded to. The CR byte is waited for even if an incorrect checksum is received.

Examples

In the descriptions of specific instructions on the following pages, examples are given as follows:

Example 1:

```
→ 2A 61 00 08 31 02 40 00 01 87 71 0D
← 2A 61 00 05 31 02 00 3C 0D
```

- Examples are in hexadecimal format.¹⁶
- The arrow → in the examples means the question, the arrow ← means the answer.
- If no example response is given for the instructions on the following pages, this means a standard query acknowledgement with ACK 00 as shown in Example 1 above.

FORMAT 97 INSTRUCTIONS

Inputs

If the device has no inputs, it responds with ACK 0x02 (invalid instruction).

Inputs states – 0x31

The instruction reads the current inputs states.

Parameters

data	1-13 bytes	<p>Each bit represents one input. So the number of bytes depends on the number of inputs (8 inputs → 1 byte, 10 inputs → 2 bytes, 16 inputs → 2 bytes, ...)</p> <p>The lowest bit in each byte is the input with the lowest sequence number (MSb). Non-existing inputs always have the value 0.</p> <ul style="list-style-type: none"> • Up to eight inputs: one byte with bits in order 87654321 (8 is MSb) • Up to sixteen inputs - two bytes: [¹⁶₁₅¹⁴₁₃¹²₁₁¹⁰₉][⁸₇⁶₅⁴₃²₁] • ... • One hundred inputs in thirteen bytes: [¹⁰⁴₁₀₃¹⁰²₁₀₁¹⁰⁰₉₉⁹⁸₉₇][12B][11B][10B][9B][8B][7B][6B][5B][4B][3B][2B][⁸₇⁶₅⁴₃²₁]
------	------------	---

Read

Structure:	→ 0x31 ← data
Example 1: <i>Quido 8/8</i>	<p>→ 2A 61 00 05 01 02 31 3B 0D</p> <p>← 2A 61 00 06 01 02 00 C2 A9 0D</p> <ul style="list-style-type: none"> • 0xC2, binary 11000010 → inputs 8, 7 and 2 are active (i.e. log 1, voltage connected, switched on, etc. depending on the input type). • By the data length you can tell that it is a Quido with eight inputs.
Example 2: <i>Quido 10/1</i>	<p>→ 2A 61 00 05 01 02 31 3B 0D</p> <p>← 2A 61 00 07 01 02 00 02 C2 A6 0D</p> <ul style="list-style-type: none"> • 0x02C2, binary 0000 0010 1100 0010 → inputs 10, 8, 7 and 2 are active • By the data length you can tell that it is a Quido with a maximum of 16 inputs.

¹⁶ (Unless explicitly stated otherwise.) Hexadecimal means that the decadic number 142 is given as 8E, the number 11 as 0B. For more information on hexadecimal representation of numbers, see, for example, the [Wikipedia article Hexadecimal](#).

Read 66:

→ *B1IR29↵

- 29: Reading the status of input 29.

← *B10L↵

- L: Input is in the log. 0 (inactive). If the input was active, there would be an H.

Change Notification: All inputs – 0x10/0x11

This feature allows you to automatically inform the parent system about input status changes. The response always includes the status of all inputs. This feature eliminates the need to periodically check inputs status. For rapid input changes, it must be taken into account that the device must have sufficient time to send the change information. (Functions 0x10/0x11 are not related to functions [0x15/0x16](#).)

- Quido RS with RS485 line: notifications can only be used if there is one device on RS485! By default, sending notifications is disabled.
- Quido USB has notifications turned off by default.
- Quido ETH has notifications turned off by default.

The automatic notification (message) is sent in the same Spinel protocol format as the format of the instruction that enabled the notification.

The *mask* parameter (only in binary format 97) can be used to specify which inputs will cause the notification to be sent and which will not.

Parameters

enable	1 byte	<p>In request:</p> <ul style="list-style-type: none"> • 0x01: Enable notifications <ul style="list-style-type: none"> ○ Notifications must first be switched off before switching back on or changing <i>mask</i> parameter! ○ When notifications are enabled without the <i>mask</i> parameter, notifications are set internally from all inputs. • 0x00: Disable notifications <ul style="list-style-type: none"> ○ Turning it off also resets the <i>mask</i> parameter. <p>In response:</p> <ul style="list-style-type: none"> • 0x00: Notifications are disabled • 0x42: Notifications were turned on by the instruction in format 66 • 0x61: Notifications were turned on by the instruction in format 97
mask	1-13 byte	<p>The bit mask (one bit for each input) determines whether a change in the input triggers a notification (1) or not (0). This parameter is optional. Method of writing bits for each input is identical with function Inputs states – 0x31 on page 16.</p>

data	1-13 byte	<p>Každý bit představuje jeden výstup. Počet byte tedy záleží na počtu výstupů. (< 8 outputs → 1 byte, < 16 outputs → 2 byte, ...)</p> <p>The lowest bit in each byte is the output with the lowest sequence number (MSb). Non-existing outputs is always 0.</p> <ul style="list-style-type: none"> Up to eight inputs: one byte with bits in order 87654321 (8 is MSb) Up to sixteen inputs - two bytes: $[^{16}_{15} \ ^{14}_{13} \ ^{12}_{11} \ ^{10}_9][^8_7 \ ^6_5 \ ^4_3 \ ^2_1]$... One hundred inputs in thirteen bytes: $[^{104}_{103} \ ^{102}_{101} \ ^{100}_{99} \ ^{98}_{97}][12B][11B][10B][9B][8B][7B][6B][5B][4B][3B][2B][^8_7 \ ^6_5 \ ^4_3 \ ^2_1]$
------	-----------	---

Write

Structure:	→ 0x10 , enable, [mask] ¹⁴
Example:	<p>→ 2A 61 00 08 31 02 10 01 1C 03 09 0D</p> <ul style="list-style-type: none"> 0x01: Enable notifications. 0x1C03, binary 0001 1100 0000 0011: Status of all inputs will be sent only after changing the status of inputs 1, 2, 11, 12 or 13.

Read

Structure:	<p>→ 0x11</p> <p>← enable, mask</p>
Example:	<p>→ 2A 61 00 05 31 02 11 2B 0D</p> <p>← 2A 61 00 07 31 02 00 61 03 D6 0D</p> <ul style="list-style-type: none"> 0x61: Notifications are enabled with format 97 (i.e. 0x61) 0x03, i.e. 00000011: Notification of the status of all inputs will only be sent if there is a change on inputs 1 or 2.

Notification

Structure:	← 0x0D , data
Example:	<p>← 2A 61 00 06 31 02 0D 10 1E 0D</p> <ul style="list-style-type: none"> 0x0D: ACK Automatic notification of the status of all inputs. 0x10, i.e. 00010000: Input IN6 is active (i.e. in log.1, switched, etc.)

Write 66:

- *B1IS1.↓
- 1: Turn automatic notifications on (0 to turn off).

← *B10.↓

Read 66:

→ *B1IX.↓

← *B10B.↓

- B: Automatic notifications have been enabled with format 66 (B). Alternatively, „0“ if notifications are disabled, or „a“ if they have been enabled with format 97.

Automatic notification 66:

Example notification from Quido with eight inputs:

← *B1D LLLLL LHL↓

- D: Auto-notification marking.
- LLLLL LHL: Only input 7 is activated. H means active input, L means inactive. A space is inserted after every five inputs for better readability.

Change Notification: One input – 0x15/0x16

This feature allows you to automatically notify the parent system about input status changes. Unlike the previous function, notification includes only the status of the single input that has changed. (Functions 0x15/0x16 are not related to functions [0x10/0x11](#).)

With this function, it is not necessary to periodically check the status of the inputs. For fast changes on inputs, the sending speed must also be taken into account to make sending information about changes technically possible.

Parameters

enable	1 byte	Enable (0x01) or disable (0x00) notifications. It always applies to all inputs at once.
in	1 byte	Input number. First is 0x01.
value	1 byte	The input is active (0x01) or not (0x00).

Write

Structure:	→ 0x15 , enable
Example:	→ 2A 61 00 06 31 02 15 01 25 0D <ul style="list-style-type: none"> • 0x01: Enable notifications.

Read

Structure:	→ 0x16 ← enable
Example:	→ 2A 61 00 05 31 02 16 26 0D ← 2A 61 00 06 31 02 00 00 3B 0D <ul style="list-style-type: none"> • 0x00: Notifications are disabled.

Automatic notification

Structure:	← 0x0C , in, value
Example:	← 2A 61 00 07 31 03 0C 05 01 27 0D <ul style="list-style-type: none"> • 0x0C: Automatic change message on a single input. • 0x05: Input IN5. • 0x01: The input is active (log 1, switched on).

Sampling rate – 0x62/0x63

The device samples its inputs every millisecond. If the specified number of consecutive samples is the same, this is evaluated as a change in the input state. More samples result in higher jitter immunity, but longer reaction time to state change. The default value is 20 ms.

Parameters

samples	1 byte	Number of samples.
---------	--------	--------------------

Write

Structure:	→ 0x62 , samples	
Example:	→ 2A 61 00 06 31 02 62 0A CF 0D <ul style="list-style-type: none"> • 0x0A, i.e. decadically 10 samples. 	

Read

Structure:	→ 0x63 ← samples	
Example:	→ 2A 61 00 05 31 02 63 D9 0D ← 2A 61 00 06 31 02 00 0A 31 0D <ul style="list-style-type: none"> • 0x0A, i.e. decadically 10 samples. 	

Counter reading – 0x60

Getting the state of the change counters on the inputs. The counter allows to count individual changes of the input state. A change is considered to be a change of a logical state (i.e. the state of a connected contact). Each of the first sixty inputs has its own counter. [Only when the type of change is set](#) (change from 1 to 0; change from 0 to 1; or both changes), a one is added to the counter value. The status of the counters is not retained after disconnection from the power supply or after reset!

Parameters

counter	1 byte	A bit-oriented byte in the form CXnnnnnn: <ul style="list-style-type: none"> • C: Read only without changing the counter value (0) or reset after reading (1). The loss of pulses (changes) in the short time between counter read and counter reset can be eliminated by the procedure described for the Subtracting from counter – 0x61 instruction. • nnnnnn: Number 0 to 60¹⁷: <ul style="list-style-type: none"> ○ 0: Read all counters. ○ A number greater than 0 means only a particular counter was read, in which case the <i>counter</i> parameter may appear more than once in the query.
bits	1 byte	Counter resolution in bits. (Usually 16 bits.)
value	X byte	Current counter value. Two or more bytes according to the counter resolution (for 16bit counters two bytes in MSB:LSB order). There are as many <i>value</i> parameters in the response as there are counter states requested in the query. The order corresponds to the query. If all states are sent, the counter on the IN1 input is sent first.

Read

Structure:	→ 0x60 , (counter) ¹⁴ ← bits, (value)	
------------	--	--

¹⁷ Quido with more than 60 inputs has counters only on the first 60 inputs.

Example:	→ 2A 61 00 06 31 02 60 00 DB 0D
	• 0x00: We ask for the values of all counters, without zeroing the values.
	← 2A 61 00 1A 31 02 00 10 01 23 00 00 00 AC 00 00 70 00 00 31 AA 00 00 00 00 00 00 00 FC 0D
	• 0x10, i.e. decadicly 16: 16bit counters • 0x0123, i.e. decadicly 291: IN1 counter value • 0x00AC: IN3 counter value.

Read 66:

→ *B1CR15↓

- 1: Zero the counter after reading (or 0 for reading without zeroing).
- 5: Counter number.

← *B10230↓

- 230: Counter value is 230.

Subtracting from counter – 0x61

The instruction subtracts the specified value from the current counter value. This eliminates the loss of pulses (changes) in the short moment between reading and counter zeroing, which can occur in principle when using the [0x60 instruction](#).

The procedure to avoid the loss of pulses:

- 1) Use the Counter reading – 0x60 instruction to read the counter value (without zeroing, i.e. bit C=0).
- 2) Subtract the read value with this instruction (0x61) from the current counter value. Due to this procedure, no change in the input will be lost.

It is not possible to subtract a number greater than the instantaneous counter value.

Parameters

If one parameter *counter* and one *value*, both 0, are specified, all counters are deleted at once. There can be more *counter+value* pairs in the query, up to a maximum of 12.

counter	1 byte	Counter number. The counter on IN1 has the number 0x01. ¹⁷
value	X byte	Value to be subtracted. Two or more bytes depending on the counter resolution (for 16-bit counters, two bytes in MSB:LSB order).

WriteStructure: → **0x61**, (counter, value)¹⁴Example: → 2A 61 00 08 31 02 **61 02 00 01** D5 0D

- 0x02: Counter on IN2.
- 0x0001: Subtract one.

Write 66:

→ *B1CD021↓

- 02: Counter on IN2. The input number here is always two digits.
- 1: Subtract one.

← *B10↓

Counter mode – 0x6A/0x6B

This function sets what input changes are counted by the counter.

Parameters

counter	1 byte	<p>A bit-oriented byte in the form CCnnnnnn:</p> <ul style="list-style-type: none"> CC: Counter mode: <ul style="list-style-type: none"> 00: Counter turned off. 10: Adding one when changing state from log. 0 to 1. 01: Adding one when changing state from log. 1 to 0. 11: Adding one for any change of state. nnnnnn: Number from 0 to 60¹⁷: <ul style="list-style-type: none"> 0: Set all counters at once. A number greater than 0 indicates the setting of a specific counter, in which case the <i>counter</i> parameter may be queried multiple times.
input	1 byte	Input number. Value 0 for all counters, or value 1-60 for a specific counter (in which case there may be multiple parameters <i>input</i> at once in the query).

Write

Structure:	→ 0x6A , (counter)
Example:	<p>→ 2A 61 00 06 31 02 6A 80 51 0D</p> <ul style="list-style-type: none"> 0x80, i.e. 10000000: Adding one on state change from 1 to 0 (CC=10) for all counters (n=000000).

Read

Structure:	<p>→ 0x6B, (input)¹⁴</p> <p>← (counter)</p>
Example:	<p>→ 2A 61 00 09 31 02 6B 01 05 07 09 B7 0D</p> <ul style="list-style-type: none"> We ask about the mode of counters on inputs 1, 5, 7 and 9. <p>← 2A 61 00 09 31 02 00 81 C5 47 49 62 0D</p> <ul style="list-style-type: none"> 0x81: Counter IN1 counts the changes from log 1 to 0. 0xC5: Counter IN5 counts all changes. 0x47: Counter IN7 counts changes from log. 0 to 1.

Write 66:

→ *B1C015↓

- 1: Counts changes from 0 to 1. You can also specify 0 (counter off), 2 (counts changes from 1 to 0) or 3 (counts all changes).
- 5: Input (counter) number.

← *B10↓

Read 66:

→ *B1CX3↓

- 3: We are asking about the IN3 counter settings.

← *B11↓

- 1: Changes from 0 to 1 are counted. (For more options, see Write.)

Outputs

If the device has no outputs, the response is ACK 0x02 (invalid instruction)

Output control – 0x20

Basic instructions for controlling outputs - i.e. immediate switching on or off.

Parameters

set	1 byte	A bit-wise oriented byte in the format Sooooooo: <ul style="list-style-type: none"> • S: Set output – ON (1) or unset output – OFF (0). • ooooooo: Output number from 1 to 127.
-----	--------	---

Write

Structure:	→ 0x20 , (set) ¹⁴
Example:	→ 2A 61 00 06 01 02 20 82 C9 0D <ul style="list-style-type: none"> • 0x82, i.e. 10000010: Switch-on (bit 7=1) output 2.

Read 66:

- *B10S25H↓
- 25: Output number 25.
 - H: On (L for Off).

← *B10↓

Reading outputs – 0x30

This instruction reads the current state of outputs (mostly relays or outputs with open collector).

Parameters

data	1-13 bytes	Each bit represents one output. So the number of bytes depends on the number of outputs (8 outputs → 1 byte, 10 outputs → 2 bytes, 16 outputs → 2 bytes, ...) The lowest bit in each byte is the input with the lowest sequence number (MSb). Non-existing inputs always have the value 0. <ul style="list-style-type: none"> • Up to eight inputs: one byte with bits in order 87654321 (8 is MSb) • Up to sixteen inputs - two bytes: [¹⁶₁₅¹⁴₁₃¹²₁₁¹⁰₉][⁸₇⁶₅⁴₃²₁] • ... • One hundred inputs in thirteen bytes: [¹⁰⁴₁₀₃¹⁰²₁₀₁¹⁰⁰₉₉⁹⁸₉₇][12B][11B][10B][9B][8B][7B][6B][5B][4B][3B][2B][⁸₇⁶₅⁴₃²₁]
------	------------	---

Read

Structure:	→ 0x30 ← data
Example:	→ 2A 61 00 05 01 02 30 3C 0D ← 2A 61 00 06 01 02 00 11 5A 0D

	<ul style="list-style-type: none"> • 0x11, binary 00010001 → outputs 5 and 1 are connected
--	---

Read 66:

→ *B10R14.↓

- 14: Reading the status of output number 14.

← *B10H.↓

- H: The output is switched on. If the output were open, there would be an L.

Pulse on output – 0x23/0x33

Switches the outputs on or off for the specified time. Pulse parameters are entered when the pulse is initiated. The pulse starts immediately upon receipt of this instruction.

It is possible to restart a pulse when the previous one has not yet finished. Thus, pulse restart can be used, for example, in situations where it is not desirable to keep the output switched after a connection failure. For example, if a pulse is triggered on the output every second for 5 sec, the relay will open after 5 sec after a connection failure at the latest.

The time set here is not related to the pulse length stored by the „Pulse on output: Separate triggering – 0x26/0x36“ on page 25.

Parameters

time	1 byte	<p><u>In request:</u> The time for which the outputs listed below in the <i>out</i> parameters are to be set. Range 1 to 255, unit is 0.5 sec.</p> <p><u>In response:</u> Remaining pulse time at the output. Zero means the time has expired.</p>
set	1 byte	<p>A bit-oriented byte in the form Sooooooo:</p> <ul style="list-style-type: none"> • S: Set output (1) or reset output (0). • ooooooo: Output number from range 1 to 127.
out	1 byte	<p>Output number:</p> <ul style="list-style-type: none"> • 1 – 255: Output number. • 0: Request all outputs on the device at once.

Write

A maximum of twelve *set* parameters can be sent in one instruction.

Structure:	→ 0x23, time, (set) ¹⁴
Example:	<p>→ 2A 61 00 08 35 02 23 04 81 84 09 0D</p> <ul style="list-style-type: none"> • 0x04: Time 2 sec (4 × 0,5 sec) • 0x81, eq. 10000001: Set output 1. • 0x84, eq. 10001000: Set output 4.

Read

There are as many sequences (*set,time*) in the response as there were queried outputs, or as many as there are outputs on the device (if 0 was specified in the query).

Structure:	→ 0x33, out ← (set, time)
Example:	→ 2A 61 00 06 31 02 33 00 08 0D

	<ul style="list-style-type: none"> • 0x00: Request all outputs. <p>← 2A 61 00 0B 31 02 00 81 1B 02 00 83 09 0C 0D</p> <ul style="list-style-type: none"> • 0x811B: 0x81 (10000001) Output 1 will be ON for 13.5 (0x1B × 0,5) sec. • 0x201B: 0x20 (00000010) Output 2 is OFF and has no time set. • 0x8309: 0x83 (10000011) Output 3 will be set for 4.5 (9 × 0,5) sec.
--	---

Write 66:

→ *B10ST5H20↓

- 5: Output 5.
- H: Set (L for reset).
- 20: Time in multiples of 0.5 sec. A range of 1 to 255 means 0.5 to 127.5 sec.

← *B10↓

Read 66:

→ *B10RT3↓

- 3: Request the remaining pulse time at output 3.

← *B10H9↓

- H: Set (L for reset).
- 9: The output will still be switched for 4.5 sec (9 × 0.5 sec).

Pulse on output: Separate triggering – 0x26/0x36/0x25

Switches the outputs on or off for the specified time. Pulse parameters are set separately. The pulse is triggered by a separate instruction. Pulse parameters are stored in non-volatile memory.

The time set here is not related to the pulse length stored by the „Pulse on output – 0x23/0x33“ on page 25.

Parameters

out	1 byte	Output number from the range of 1 to 255. (0 can be entered when reading, which means all output settings are read.)
type	1 byte	Output pulse type: <ul style="list-style-type: none"> • 0x00: None • 0x02: Positive pulse (L → H → L) • 0x03: „Negative“ pulse (H → L → H)
time	1 byte	Pulse length as number 1 to 255, unit 0.5 sec. When reading the setting, the set time is sent here, not the remaining pulse time. (The remaining time of the current pulse is read using instruction 0x33.)

Write (set settings)

An instruction can contain multiple *out*, *type*, *time* sequences (the order does not matter). A maximum of twelve of these sequences can be sent in one instruction.

Structure:	→ 0x26 , (out, type, time)
Example:	<p>→ 2A 61 00 08 31 02 26 04 02 04 09 0D</p> <ul style="list-style-type: none"> • 0x04: Output 4. • 0x02: Positive pulse.

	<ul style="list-style-type: none"> 0x04: Time 2 sec (4 × 0x,5 sec)
--	---

Read (read settings)

The instruction can contain multiple *type*, *time* sequences, depending on the number of queried inputs.

Structure:	→ 0x36 , (out) ← (type, time)
Example:	→ 2A 61 00 06 31 02 36 00 05 0D <ul style="list-style-type: none"> 0x00: query to all outputs. ← 2A 61 00 0D 31 02 00 03 14 02 14 00 00 02 04 01 0D <ul style="list-style-type: none"> 0x0314: First in „negative pulse“ (0x03), time 10 sec (0x14 × 0,5). 0x0214: Second in „positive pulse“ mode (0x02), time 10 sec (0x14 × 0,5). 0x0000: Third output without pulse setting. 0x0204: Fourth in „positive pulse“ mode (0x02), time 2 sec (0x04 × 0,5).

Start Pulse

The pulse on the output is only executed if [the thermostat function](#) is not activated on the output. If a thermostat is activated on the output, the pulse can only be triggered when the temperature is between *tempx* and *tempy*. Otherwise, Quido will respond with ACK 0x03.

An instruction can contain multiple sequences of *out*, *type*, *time* (the order does not matter). A maximum of twelve of these sequences can be sent in one instruction.

Structure:	→ 0x25 , (out)
Example:	→ 2A 61 00 07 31 02 25 02 04 0F 0D <ul style="list-style-type: none"> 0x02, 0x04: Start pulse on outputs 2 and 4.

Output mode check – 0x38

This instruction allows you to see what mode the output is currently operating in.

Parameters

out	1 byte	Output number from 1 to 255 or 0 for all outputs.																																																												
flags	1 byte	Bit-oriented byte (bit 7 = MSB); the meaning of each bit is as follows:																																																												
		<table border="1"> <thead> <tr> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> <th>Bit</th> <th>Description</th> </tr> <tr> <th>A</th> <th>M1</th> <th>M0</th> <th></th> <th>S3</th> <th>S2</th> <th>S1</th> <th>S0</th> <th>Bit name</th> <th></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>S</td> <td>S</td> <td>K</td> <td></td> <td>The output is in thermostat mode. The SSK bits correspond to the SSK bits from the thermostat function (page 32).</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>The output is in manual mode and the length of the positive pulse is set.</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>The output is in manual mode and the negative pulse length is set.</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>The output is in manual mode and no pulse is set.</td> </tr> </tbody> </table>	7	6	5	4	3	2	1	0	Bit	Description	A	M1	M0		S3	S2	S1	S0	Bit name		1	0	1	0	0	S	S	K		The output is in thermostat mode. The SSK bits correspond to the SSK bits from the thermostat function (page 32).	0	0	0	0	0	0	0	1	0	The output is in manual mode and the length of the positive pulse is set.	0	0	0	0	0	0	0	1	1	The output is in manual mode and the negative pulse length is set.	0	0	0	0	0	0	0	0	0	The output is in manual mode and no pulse is set.
		7	6	5	4	3	2	1	0	Bit	Description																																																			
		A	M1	M0		S3	S2	S1	S0	Bit name																																																				
		1	0	1	0	0	S	S	K		The output is in thermostat mode. The SSK bits correspond to the SSK bits from the thermostat function (page 32).																																																			
0	0	0	0	0	0	0	1	0	The output is in manual mode and the length of the positive pulse is set.																																																					
0	0	0	0	0	0	0	1	1	The output is in manual mode and the negative pulse length is set.																																																					
0	0	0	0	0	0	0	0	0	The output is in manual mode and no pulse is set.																																																					

Read

An instruction can contain multiple *out* parameters. Similarly, multiple flags parameters are included in the response.

Structure:	→ 0x38 , (out) ← (flags)
Example:	→ 2A 61 00 06 31 02 38 00 03 0D <ul style="list-style-type: none"> • 0x00: Requesting all output. ← 2A 61 00 09 31 02 00 A0 02 03 A0 F3 0D <ul style="list-style-type: none"> • OUT1: static mode • OUT2: positive pulse • OUT3: negative pulse • OUT4: temperature monitoring (bits SSK = 0)

Linking input and output – 0x40/0x41

Allows you to set the linking of inputs with outputs. Activate an input to trigger an output etc. This function is only available for Quido 2/2, 4/4 and 8/8.

Parameters

in	1 byte	Input number. The first input has the number 0x01.
out	1 byte	Output number. The first output has the number 0x01.
fn	1 byte	Link type: <ul style="list-style-type: none"> • 0x00: <u>None</u>. • 0x01: <u>Mirroring</u>: the input state is copied to the output active input = switched output. • 0x02: <u>Pulse trigger</u>: The edge on the input (moment of change) triggers a pulse on the output. • 0x03: <u>State change by edge</u>: It works as a divider by two, i.e. the first edge on the input switches the output on, the second one switches the output off. • 0x04: <u>Response to connection failure</u>: Output switching on/off. • 0x05: <u>Set/Reset</u>: The edge on the input will switch the output on or off. One input can switch or un-switch one output, not both.
edge	1 byte	<u>Edge type</u> (only for $fn=2$, $fn=3$ and $fn=5$): <ul style="list-style-type: none"> • 0x00: Rising edge. • 0x01: Falling edge. <u>Length of connection failure</u> (only for $fn=4$): <ul style="list-style-type: none"> • A number in the range 0x01 to 0x03 as time in seconds.

param	1 byte	<p><u>Invert</u> (only for $fn=1$): Allows to invert (0x01) the input level. Disconnecting the input disconnects the output and vice versa.</p> <p><u>Pulse extension</u> (only for $fn=2$):</p> <ul style="list-style-type: none"> • 0x00: Repeating an edge on the input during a pulse on the output has no effect on the pulse duration. • 0x01: Repeating an edge on the input extends the duration of the output switching. <p><u>Response type to connection failure</u> (only for $fn=4$ and $fn=5$):</p> <ul style="list-style-type: none"> • 0x00: Reset output. • 0x01: Set output.
-------	--------	--

Write

Structure:	→ 0x40, in, out, fn, edge, param
Example:	<p>→ 2A 61 00 0A 31 02 40 02 02 03 00 00 F0 0D</p> <ul style="list-style-type: none"> • 0x02: Input IN2. • 0x02: Output OUT2. • 0x03: State change by edge. • 0x00: Response to rising edge.

Read

Structure:	<p>→ 0x41, in</p> <p>← in, out, fn, edge, param</p>
Example:	<p>→ 2A 61 00 06 31 02 41 03 F7 0D</p> <ul style="list-style-type: none"> • 0x03: Query to third input. <p>← 2A 61 00 0A 31 02 00 03 04 01 00 01 2E 0D</p> <ul style="list-style-type: none"> • 0x03: Input IN3. • 0x04: Output OUT4. • 0x01: Mirroring state. • 0x01: Invert.

Temperature measurement and monitoring

Note: If the device does not allow a thermometer connection, it responds to the following instruction ACK 0x02 (invalid instruction).

Temperature measuring – 0x51

Returns the temperature measured by the connected thermometer. The temperature is returned in the set temperature unit.

Parameters

id	1 byte	Thermometer number (the first one is 0x01). 0x00 means a query for all thermometers. The parameter can be queried a maximum of twelve times.
----	--------	--

int	2 byte	Measured temperature. MSB:LSB value in signed int format. The temperature is obtained from the value by dividing by ten. ¹⁸
-----	--------	--

Read

If the temperature is out of range, ACK 0x05 (device fault) is responded to. In the event of a sensor fault, ACK 0x05 is declared after approximately ten seconds of error duration.

Structure:	→ 0x51 , (id) ¹⁴ ← (id, value)
Example:	→ 2A 61 00 06 31 02 51 01 E9 0D <ul style="list-style-type: none"> • 0x01: Query to first thermometer. ← 2A 61 00 08 31 02 00 01 00 F6 42 0D <ul style="list-style-type: none"> • 0x01: First thermometer. • 0x00F6 is decadicly 246, i.e. temperature 24.6°

Read 66:

→ *B1TR1.↓

- 1: One thermometer number to be read.

← *B10+029.1C.↓

- +029.1C: Temperature as an ASCII string (always 7 characters right justified). Unused characters are filled with zero (0x30). A period (0x2E) is used as decimal separator.
- If the thermometer is out of range or the temperature cannot be read, ACK 5 (device error) is answered.

Temperature measuring: int, float, string – 0x58

It returns the temperature measured by the connected thermometer (1) as an integer multiplied by ten, (2) as a floating-point number, and (3) as an ASCII string.

Parameters

id	1 byte	Thermometer number (the first one is 0x01). 0x00 means a query for all thermometers. The parameter can be queried a maximum of twelve times.
status	1 byte	Bit-oriented byte in the form: V00000HL (V is MSb) <ul style="list-style-type: none"> • V: Validity: <ul style="list-style-type: none"> ○ 1 = Temperature is valid ○ 0 = Temperature is not valid! • H: 1 = Temperature overflow • L: 1 = Temperature underflow
int	2 byte	Measured temperature. <u>Data validity is determined by status!</u> MSB:LSB value in the format signed int . The temperature is obtained from the value by dividing by ten.
float	4 byte	Measured temperature as a floating point reading according to IEEE 754 . <u>The validity of the data is determined by the status!</u>

¹⁸ The accuracy of your specific temperature sensor is specified in the documentation for your specific Quido module.

string	10 byte	Measured temperature as a string. For example, "24.6". <u>Validity is determined by the status!</u>
--------	---------	---

Read

When a sensor error occurs, the bit V is set to 0 in the status parameter and the temperature is set to -9999 after about ten seconds of the error.

Structure:	→ 0x58 , (id) ← (id, status, int, float, string)
Example:	→ 2A 61 00 06 B1 02 58 00 63 0D ← 2A 61 00 17 B1 02 00 01 80 01 10 41 DA 00 00 20 20 20 20 20 20 32 37 2E 32 74 0D <ul style="list-style-type: none"> • 0x01: First thermometer • 0x80: Measured temperature is valid and in range. • 0x0110, i.e. decadically 272: Temperature 27,2° • 0x41DA0000: 2.725e+1 • 0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x32,0x37,0x2E,0x32, i.e. „ 27,2“

Temperature unit – 0x1C/0x1D

The temperature unit with which the device operates.

Parameters

id	1 byte	Always 0x00 in the request and 0x01 in the response.
unit	1 byte	Unit code: <ul style="list-style-type: none"> • 0x00: degrees Celsius • 0x01: degrees Fahrenheit • 0x02: Kelvin

Write

Structure:	→ 0x1C , id, unit
Example:	→ 2A 61 00 07 B1 02 1C 00 01 9D 0D <ul style="list-style-type: none"> • 0x01: Fahrenheit setting

Read

Structure:	→ 0x1D ← id, unit
Example:	→ 2A 61 00 05 B1 02 1D 9F 0D ← 2A 61 00 07 B1 02 00 01 01 B8 0D <ul style="list-style-type: none"> • 0x01: Fahrenheit settings have been made

Temperature limits – 0x13/0x14

If the measured temperature leaves a pair of set temperature limits, the device sends an automatic message. When communicating via RS485, this function can only be used if there is only one device on the line!

Parameters

Parameters that have a constant before the length do not have to be listed in the instruction all at once. In the data, there is one byte with the constant first, followed by the parameter. The constant is used by the device to uniquely identify which parameter it is.

tid	1 byte	Thermometer number from the interval 0x01 to 0x08.
enable	0x01 + 1 byte	Temperature monitoring ON (0x01) or OFF (0x00).
highInt	0x02 + 2 bytes	Upper temperature limit. MSB:LSB value in signed int . The temperature is obtained from the value by dividing by ten.
lowInt	0x03 + 2 bytes	Lower temperature limit. MSB:LSB value in signed int . The temperature is obtained from the value by dividing by ten.
period	0x04 + 2 bytes	If the temperature is out of limits, an automatic message will be sent as often as set here. The value is in seconds.
highStr	0x05 + X bytes	Upper temperature limit as a string. For example, "24.6".
lowStr	0x06 + X bytes	Lower temperature limit as a string. For example, "-21.7".
status	1 byte	Bit-oriented byte in the form: V0000HL (V is MSb) <ul style="list-style-type: none"> • V: Validity: <ul style="list-style-type: none"> ○ 1 = Temperature is valid ○ 0 = Temperature is not valid! • H: 1 = Temperature above upper limit • L: 1 = Temperature below lower limit

Write

Structure:	→ 0x13 , tid, enable, highInt, lowInt, period, highStr, lowStr
Example:	→ 2A 61 00 11 31 02 13 01 01 01 02 01 36 03 00 FA 04 00 01 DF 0D <ul style="list-style-type: none"> • 0x01: First thermometer. • 0x01, 0x01: Temperature monitoring ON. • 0x02, 0x0136: Upper limit 310, i.e. 31,0° • 0x03, 0x00FA: Lower limit 250, i.e. 25,0° • 0x04, 0x0001: Period 1 sec.

Read

Structure:	→ 0x14 , tid ← tid, enable, highInt, lowInt, period, highStr, lowStr
Example:	→ 2A 61 00 06 31 02 14 01 26 0D <ul style="list-style-type: none"> • 0x01: Request for first thermometer. ← 2A 61 00 27 31 02 00 01 01 01 02 01 36 03 00 FA 04 00 01 05 20 20 20 20 20 20 33 31 2E 30 06 20 20 20 20 20 20 32 35 2E 30 CA 0D <ul style="list-style-type: none"> • 0x01: First thermometer. • 0x01, 0x01: Temperature monitoring ON. • 0x02, 0x0136: Upper limit 310, i.e. 31,0° • 0x03, 0x00FA: Lower limit 250, i.e. 25,0° • 0x04, 0x0001: Period 1 sec.

Automatic message

Structure:	← (0x0158, 0x02, tid, 0x03, status, 0x04, int, float, string)
Example:	<p>← 2A 61 00 1C 31 C0 0F 01 58 02 01 03 82 04 01 39 41 FB 00 00 20 20 20 20 20 33 31 2E 33 78 0D</p> <ul style="list-style-type: none"> • 0x0158: Constant. • 0x0F is an ACK indicating an automatic message. • 0x01: First thermometer. • 0x82: Valid value, temperature above upper limit. • 0x0139: The value 313 (signed int) means 31.3° • 0x41FB0000: Float format (IEEE 754). • Other bytes are ASCII string with temperature.

Thermostat – 0x1A/0x1B

Each output can automatically respond to the temperature measured by the temperature sensor. For a detailed description of the thermostat's function, see [Appendix 1](#) on page 43 of this document.

Parameters

out	1 byte	Output number. The first one has the number 0x01. The value 0x00 is invalid.
flags	1 byte	<p>Bit-oriented byte in the form: FSSKSTTT (F is MSb)</p> <ul style="list-style-type: none"> • F: Output <i>out</i> controlled by thermostat: <ul style="list-style-type: none"> ○ 1 = yes ○ 0 = no • SSxS: Output action to be performed at the set temperature: <ul style="list-style-type: none"> ○ 00x0 = switch on ○ 01x0 = switch off ○ 10x0 = switch on for <i>time</i> ("positive pulse") ○ 11x0 = switch off for <i>time</i> ("positive pulse") ○ 00x1 = switch on in limits <i>tempx</i> and <i>tempy</i> ○ 01x1 = switch on outside limits <i>tempx</i> and <i>tempy</i> • K: Critical temperature trend (only applies to SSxS=10x0 and SSxS=11x0): <ul style="list-style-type: none"> ○ 0 = temperature rise ○ 1 = temperature drop • TTT: Temperature sensor number (usually 1).
tempx	2 bytes	Higher temperature. MSB:LSB value in signed int . The temperature is obtained from the value by dividing by ten.
tempy	2 bytes	Lower temperature. MSB:LSB value in signed int . The temperature is obtained from the value by dividing by ten.
time	1 byte	For how long you want the output state to change (applies only to SSxS=10x0 and SSxS=11x0). Value in seconds.

error	1 byte	How to react to a thermometer error (interrupted/disconnected sensor): <ul style="list-style-type: none"> • 0: no response • 1: switch off • 2: switch on
-------	--------	--

Write

A sequence of parameters in parentheses () can be specified a maximum of twelve times in a query. Therefore, the thermostat can be set for up to twelve outputs at once.

Structure:	→ 0x1A , (out, flags, temp _x , temp _y , time, err) ¹⁴
Example:	→ 2A 61 00 0D 31 02 1A 01 81 01 0E 01 0E 05 00 75 0D <ul style="list-style-type: none"> • 0x01: First output. • 0x81 → binary 10000001: Thermostat enabled (F=1), Switch on action (SSxS=00x0), Thermometer 1 (TTT=1). • 0x010E → decadically 270: The higher temperature is 27.0 °C • 0x010E → decadically 270: The lower temperature is 27.0 °C • 0x05: time 5 sec (irrelevant due to SSxS=00x0) • 0x00: no response on thermometer error

Read

The *out* parameter is optional. It does not have to be in the query at all (all outputs will be in the response) or up to twelve times. This will correspond to the number of sequences () in the response.

Structure:	→ 0x1B , out ← (out, flags, temp _x , temp _y , time, err)
Example:	→ 2A 61 00 05 31 02 1B 21 0D <ul style="list-style-type: none"> • Request all outputs. ← 2A 61 00 15 31 02 00 01 81 01 0E 01 0E 05 00 02 00 27 0F D8 F1 00 00 86 0D <ul style="list-style-type: none"> • 0x01: First output. <ul style="list-style-type: none"> ○ 0x81 → binary 10000001: Thermostat enabled (F=1), Switch on action (xS=00), Thermometer 1 (TTT=1). • 0x02: Second output. <ul style="list-style-type: none"> ○ 0x00 → binary 00000000: Thermostat disabled (F=0).

Communication port and address

Enable configuration – 0xE4

Enables the execution of some important instructions. This is explicitly stated for these instructions. The configuration enable must immediately precede each of these instructions. The configuration enable applies to only one following instruction. Configuration enable cannot be used with a universal address.

Write

Structure:	→ 0xE4
Example:	→ 2A 61 00 05 31 02 E4 58 0D

Write 66:

→ *B1E↵

← *B10↵

Address and baudrate – 0x0E/0x0F

Parameters

addr	1 byte	Value 0x00 - 0xFD. If you also use the ascii format 66 for communication with the device, it is necessary to use only such addresses that can be expressed as a displayable ASCII character (i.e. characters with codes 0x20 to 0x7E).
baud	1 byte	Baud rate [Bd]: <ul style="list-style-type: none"> • 1200..... 0x03 • 2400..... 0x04 • 4800..... 0x05 • 9600..... 0x06 • 19200..... 0x07 • 38400..... 0x08 • 57600..... 0x09 • 115200..... 0x0A • 230400..... 0x0B <p>Quido ETH and Quido USB have a fixed speed of 115200 Bd. The baud parameter for these Quidos must always be set to 115200 Bd, otherwise the module responds with ACK 0x03 (invalid data).</p> <p>Quido RS have a default baud rate of 9600 Bd.</p>

Write

After changing the parameters, the device reboots and the counters and status are reset! The universal address 0xFE cannot be used for writing. If you do not know the address of the device, you can set a new address with the instruction Address setting by serial number (page 35). Immediately before writing, you must enable the configuration with the Enable configuration – 0xE4 instruction.

Structure:	→ 0xE0, addr, baud
------------	--------------------

Example:	→ 2A 61 00 07 01 02 E0 02 0A 7E 0D
	<ul style="list-style-type: none"> • 0x02: new address • 0x0A: baudrate code
	← 2A 61 00 05 01 02 00 6C 0D
	<ul style="list-style-type: none"> • The new address and communication speed will be set after the reply is sent.

Read

Reading the communication parameters is useful if the address of the device is not known. The reading is performed with the universal address 0xFE. If the communication speed is not known, it is possible to use the [Configuration jumpers](#) or to try all communication speeds. [The Modbus Configurator](#) software can be used to find devices with unknown communication parameters.

Structure:	→ 0xF0 ← addr, baud
Example:	→ 2A 61 00 05 FE 02 F0 7F 0D <ul style="list-style-type: none"> • Query with universal address 0xFE. ← 2A 61 00 07 04 02 00 04 06 5D 0D <ul style="list-style-type: none"> • 0x04: address 0x04 • 0x06: baudrate 9600 Bd

Write 66:

In format 66, the address and speed are set by two separate instructions:

→ *B1**A**S4↵

- 4: Set the address to 4.

← *B1**0**↵

- The address will be changed after the reply has been sent.

→ *B1**S**S7↵

- 7: Setting the communication speed to 19200 Bd.

← *B1**0**↵

- The speed will be changed after the reply is sent.

Read 66:

→ *B1**C**P↵

← *B1**0**B7↵


- B: Address B.
- 7: Baud rate 19200 Bd.

Address setting by serial number – 0xEB

The instruction allows you to set the address by the unique serial number of the device. This instruction is useful in case the parent system or operator loses the address of a device that is on the same communication link with other devices.

Parameters

newaddr	1 byte	New address from range 0x00 – 0xFD.
---------	--------	-------------------------------------

sn	2+2 byte	<p>Serial number composed of product number and piece number. The S/N is on a label on the device and can also be read using the Manufacturing data – 0xFA instruction.</p>  <p style="text-align: center;">fig. 2 – device serial number</p>
----	----------	---

Write

Structure:	→ 0x24 , newaddr, sn
Example:	<p>→ 2A 61 00 0A FE 02 EB 32 01 3B 04 F9 14 0D</p> <ul style="list-style-type: none"> • 0x32: new address • 0x013B: product type number 315 • 0x04F9: piece number 1273 <p>← 2A 61 00 05 32 02 00 3B 0D</p> <ul style="list-style-type: none"> • Response with new address 0x32.

Others

Name and version – 0xF3

Depending on the type of query, it returns the number of inputs, outputs and thermometers, device name, internal software version and a list of possible communication formats. It can also find out the address using the serial number.

Parameters

getio	1 byte	If 0x01 is specified here, the response is not standard string, but the instruction returns an <i>ios</i> with a machine-readable I/O count in the response.
ios	3 byte	<p>I/O counts:</p> <ul style="list-style-type: none"> • 1. byte: Number of inputs • 2nd byte: Number of outputs • 3rd byte: Number of thermometers
sn	2+2 byte	Serial number composed of the serial and product number. The S/N is indicated on the label on the device (see page 36). If there are several devices with the same address on one communication link, one specific device can be addressed in this way using the universal address 0xFE and the serial number.
string	x byte	A string with device identification in the form: <i>[device-type]; v[device-number].[hw-version].[sw-version];</i>

Read

Structure:	<p>→ 0xF3, [getio] [sn] ¹⁴</p> <p>← string ios</p>
------------	--

Example 1: <i>Identification string</i>	<p>→ 2A 61 00 05 FE 02 F3 7C 0D</p> <ul style="list-style-type: none"> Request without parameters. <p>← 2A 61 00 2B 31 02 00 51 75 69 64 6F 20 55 53 42 20 34 2F 34 3B 20 76 30 32 35 33 2E 30 34 2E 34 38 3B 20 66 36 36 20 39 37 3B 20 74 31 CF 0D</p> <ul style="list-style-type: none"> Device identification string: <i>Quido USB 4/4; v0253.04.48; f66 97; t1</i>
Example 2: <i>I/O counts</i>	<p>→ 2A 61 00 06 FE 02 F3 01 7A 0D</p> <ul style="list-style-type: none"> 0x01: Request for the number of inputs, outputs and thermometers. <p>← 2A 61 00 08 31 02 00 04 04 01 30 0D</p> <ul style="list-style-type: none"> 0x040401: <ul style="list-style-type: none"> 0x04: Number of inputs. 0x04: Number of outputs. 0x01: Number of thermometers.
Example 3: <i>Addressing by S/N</i>	<p>→ 2A 61 00 09 FE 02 F3 00 FD 08 8F E4 0D</p> <ul style="list-style-type: none"> Using the universal address <u>0xFE</u>, we request the device with serial number 0253/2191 (0x00FD/0x088F). <p>← 2A 61 00 2B 31 02 00 51 75 69 64 6F 20 55 53 42 20 34 2F 34 3B 20 76 30 32 35 33 2E 30 34 2E 34 38 3B 20 66 36 36 20 39 37 3B 20 74 31 CF 0D</p> <ul style="list-style-type: none"> Device identification string: <i>Quido USB 4/4; v0253.04.48; f66 97; t1</i>

Read 66:

→ *B1?↓

← *B10Quido ETH 4/4; v0254.02.07; f66 97; t1↓

Manufacturing data – 0xFA

The instruction reads the manufacturing data from the device.

Parameters

type	2 byte	Product type number. From string 0254/0001 it is number 254.
item	2 byte	Piece number. From string 0254/0001 it is number 1.
factory	4 byte	Manufacturing data.

Read

Structure:	<p>→ 0xFA</p> <p>← type, item, factory</p>
Example:	<p>→ 2A 61 00 05 FE 02 FA 75 0D</p> <p>← 2A 61 00 0D 35 02 00 00 C7 00 65 20 05 09 23 B3 0D</p> <ul style="list-style-type: none"> 0x00C7: Type 199. 0x0065: Piece 101. 0x20050923: Manufacturing data.

User data – 0xE2/0xF2

The user data space is a memory where the user can store any data that the device will remember even after power off or reset. This space is useful for naming the measuring point, for example.

Parameters

offset	1 byte	The address of the memory where the data is to be stored. A number from the range 0x00 - 0x0F. In case of writing from offset e.g. 0x0C, a maximum of 4 bytes can be written.
data	1-16 byte	User data.

Write

Structure:	→ 0xE2 , offset, data
Example:	→ 2A 61 00 10 FE 02 E2 00 53 74 6F 72 61 67 65 20 34 32 27 0D <ul style="list-style-type: none"> • 0x00: Offset 0 - writing from the beginning. • String "Storage 42" (10 characters).

Read

Structure:	→ 0xF2 ← data
Example:	→ 2A 61 00 05 FE 02 F2 7D 0D ← 2A 61 00 16 31 02 00 53 74 6F 72 61 67 65 20 34 32 20 20 20 20 20 20 F0 0D <ul style="list-style-type: none"> • String "Storage 42 " (16 characters).

Write 66:

- *B1DW0Storage 42↵
- 0: The memory position to be written to. A character from the interval 0-9 or A-F.
 - Storage 42: 1 to 16 characters.

← *B10↵

Read 66:

→ *B1DR↵

← *B10Storage 42↵

Input names – 0x2B/0x3B

Allows you to store a unique character string for each input. This function is useful for naming inputs.

This memory space is used by the control software that comes free with Quido modules. It is also used in the Ethernet versions of Quido modules to store the names of inputs and outputs. For these reasons, we do not recommend manipulating this memory location when used with our standard software or when using the standard web interface in the Ethernet Quido modules.

If there is no input on the device, ACK 0x02 (invalid instruction) is answered.

Parameters

input	1 byte	Input number. The first input has the number 0x01.
-------	--------	--

data	21 bytes	Any user data.
------	----------	----------------

Write

Structure:	→ 0x2B , input, data
Example:	→ 2A 61 00 1B 31 02 2B 01 30 4B 6F 74 65 6C 6E 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FC 0D <ul style="list-style-type: none"> • Saving the name "0Kotelna" to input 1 (0x01). (Unused bytes are filled with zeros.)

Read

Structure:	→ 0x3B , input ← data
Example:	→ 2A 61 00 06 31 02 3B 01 FF 0D <ul style="list-style-type: none"> • 0x01: Query to first input. ← 2A 61 00 1A 31 02 00 30 4B 6F 74 65 6C 6E 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 29 0D <ul style="list-style-type: none"> • The text "0Kotelna" is stored in the data. (Unused bytes are filled with zeros.)

Output names – 0x2A/0x3A

Allows you to store a unique character string for each output. This function is useful for naming outputs.

This memory space is used by the control software that comes free with Quido modules. It is also used in the Ethernet versions of Quido modules to store the names of inputs and outputs. For these reasons, we do not recommend manipulating this memory location when used with our standard software or when using the standard web interface in the Ethernet Quido modules.

If there is no output on the device, the response is ACK 0x02 (invalid instruction).

Parameters

output	1 byte	Output number. The first output has the number 0x01.
data	21 bytes	Any user data.

Write

Structure:	→ 0x2A , output, data
Example:	→ 2A 61 00 1B 31 02 2A 04 30 53 69 72 65 6E 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 66 0D <ul style="list-style-type: none"> • Saving the name "0Sirena" to output 4 (0x04). (Unused bytes are filled with zeros.)

Read

Structure:	→ 0x3A , output ← data
Example:	→ 2A 61 00 06 31 02 3A 04 FD 0D <ul style="list-style-type: none"> • 0x04: Query to 4th output. ← 2A 61 00 1A 31 02 00 30 53 69 72 65 6E 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 95 0D

	<ul style="list-style-type: none"> The text "0Sirena" is stored in the data. (Unused bytes are filled with zeros.)
--	---

Status and Run time – 0xE1/0xF1

Status is a single byte of memory used to indicate the user's status of the device. It is automatically set to 0x00 when the device is powered on and after reset (even software reset).

In addition to the status, the time since the device was switched on in seconds can also be found here.

Parameters

status	1 byte	After power-up or reset, the value 0x00 is set automatically.
runtime	4 bytes	Time from device power-up or reset in seconds.

Write

Structure:	→ 0xE1 , status
Example:	→ 2A 61 00 06 01 02 E1 12 78 0D <ul style="list-style-type: none"> Setting status to 0x12.

Read

0x31 is an optional parameter that indicates a *runtime* read request.

Structure:	→ 0xF1 , [0x31] ¹⁴ ← status
Example 1: <i>Status only</i>	→ 2A 61 00 05 01 02 F1 7B 0D ← 2A 61 00 06 01 02 00 12 59 0D <ul style="list-style-type: none"> Status is 0x12.
Example 2: <i>Status + runtime</i>	→ 2A 61 00 06 FE 02 F1 31 4C 0D ← 2A 61 00 0A 31 02 00 12 00 00 00 DD 48 0D <ul style="list-style-type: none"> 0x12: The status is the decadic number 18. 0x000000DD: It's been 221 seconds since the reset (power on).

Write 66:

→ *B1SWA␣

- A: ASCII character from interval „space“ to „~“ (32 – 126)

← *B10␣

Read 66:

→ *B1SR␣

← *B10A

Errors in communication – 0xF4

The instruction returns the number of communication errors that have occurred since the device was turned on, or since the last communication error reading.

Parameters

errors	1 byte	<p>The number of communication errors that have occurred since the device was switched on or since the last reading. The following events are considered communication errors:</p> <ul style="list-style-type: none"> • A prefix is expected, but another byte will come. • SUM checksum does not match. • The message is not complete - i.e. the communication timeout expires.
--------	--------	---

Read

Structure:	→ 0xF4 ← errors
Example:	<p>→ 2A 61 00 05 01 02 F4 78 0D ← 2A 61 00 06 01 02 00 05 66 0D</p> <ul style="list-style-type: none"> • 0x05: Five errors.

Checksum – 0xEE/0xFE

Checksum is a protection against data corruption during transmission. This function allows you to disable checksum checking and is intended for debugging applications only! When manually entering instructions via the terminal, it is not necessary to enter the checksum (penultimate byte) correctly. The check is enabled by default.

Parameters

state	1 byte	<ul style="list-style-type: none"> • 0x01: Enabled. • 0x00: Disabled.
-------	--------	---

Write

Structure:	→ 0xEE , state
Example:	<p>→ 2A 61 00 06 01 02 EE 01 7C 0D</p> <ul style="list-style-type: none"> • 0x01: enable checksum checking

Read

Structure:	→ 0xFE ← state
Example:	<p>→ 2A 61 00 05 01 02 FE 6E 0D ← 2A 61 00 06 01 02 00 01 6A 0D</p> <ul style="list-style-type: none"> • 0x01: checksum check is enabled

Communication timeout – 0xE5/0xF5

Allows you to set the communication timeout. The timeout is the time measured after each byte is received, after which the communication is considered interrupted (see Errors in communication – 0xF4).

Parameters

ms	1 byte	Time in tens of milliseconds, i.e. 10 - 2550 ms. The default time is 1 sec.
----	--------	---

Write

Structure:	→ 0xE5 , ms
Example:	→ 2A 61 00 06 B1 02 E5 20 B6 0D <ul style="list-style-type: none"> • 0x20: timeout setting 32 ms

Read

Structure:	→ 0xF5 ← ms
Example:	→ 2A 61 00 05 B1 02 F5 C7 0D ← 2A 61 00 06 B1 02 00 20 9B 0D <ul style="list-style-type: none"> • 0x20: timeout is 20 ms

Reset – 0xE3

Resets the device. The device will return to the same state as when the power was turned on. The reset is performed after the response is sent.

Write

Structure:	→ 0xE3
Example:	→ 2A 61 00 05 01 02 E3 89 0D

Write 66:

→ *B1RE↵
← *B10↵

Default settings – 0x8F

Sets all device parameters to the default settings. The instruction must be preceded by the Enable configuration instruction described on page 34.

Write

Structure:	→ 0x8F
Example:	→ 2A 61 00 05 B1 02 8F 2D 0D

Communication protocol switching – 0xED

The instruction must be preceded by the Enable configuration instruction described on page 34. You can use the [Modbus Configurator](#) software to switch the protocol.

Parameters

protocol	1 byte	Protocol identification number: <ul style="list-style-type: none"> • 0x01: Spinel, format 97 (binary) and 66 (ascii) • 0x02: Modbus RTU (only for Quido RS and Quido USB) • 0x0A: Spinel, format 97 only (binary)
----------	--------	--

Write

Structure:	→ 0x2A , protocol
Example:	→ 2A 61 00 06 31 02 ED 02 4C 0D

- Switching to Modbus RTU protocol.

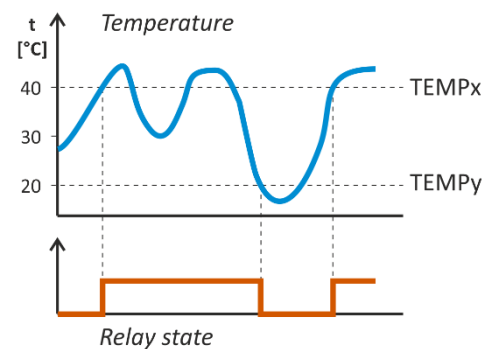
APPENDIX 1: THERMOSTAT

This appendix describes the thermostat modes in Quido modules with a connected temperature sensor. Temperature and flags parameter designations and other variables in the figures and text are identical to the parameter designations in the [thermostat setup instructions](#).

Mode 1

The output switches on when the TEMPx temperature is exceeded and switches off when the TEMPy temperature falls below.¹⁹

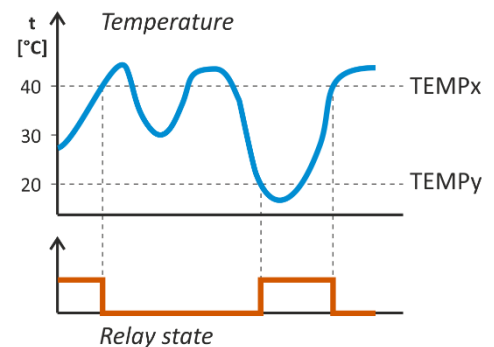
- Idle output state: OFF
- Bits in *flags* parameter:
 - SSxS: 00x0 – switch output on



Mode 2

The output switches off when the TEMPx temperature is exceeded and switches on when the temperature drops below TEMPy.¹⁹

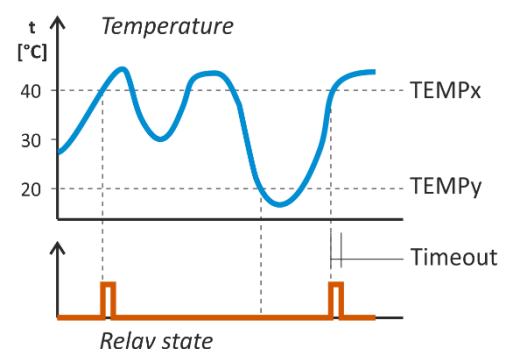
- Idle output state: ON
- Bits in *flags* parameter:
 - SSxS: 01x0 – switch output off



Mode 3

The output switches on for a set period of time when the TEMPx temperature is exceeded. It can only switch on again when the temperature drops below TEMPy and then rises again to TEMPx.¹⁹

- Idle output state: OFF
- Bits in *flags* parameter:
 - SSxS: 10x0 – switch on to *time*
 - K: 0 – temperature rising

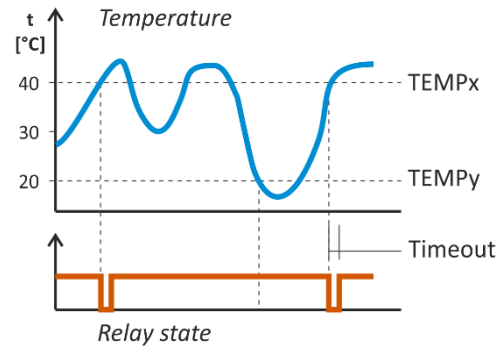


¹⁹ This introduces so-called hysteresis in the temperature control. It is possible to set both temperatures to the same value and thus cancel the hysteresis.

Mode 4

The output switches off for a set period of time when the TEMPx temperature is exceeded. It can open again only when the temperature drops below TEMPy and then rises again to TEMPx.¹⁹

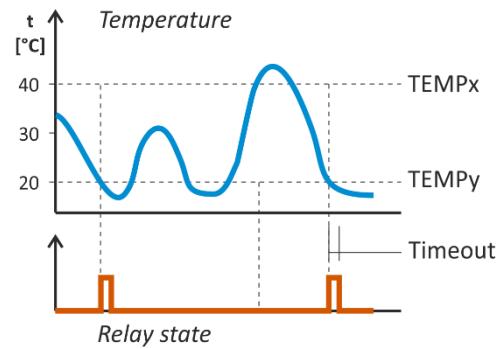
- Idle output state: ON
- Bits in *flags* parameter:
 - SSxS: 11x0 – switch off to *time*
 - K: 0 – temperature rising



Mode 5

The output switches on for a set period of time when the temperature drops below TEMPy. It can only switch on again when the temperature rises above TEMPx and then falls below TEMPy again.¹⁹

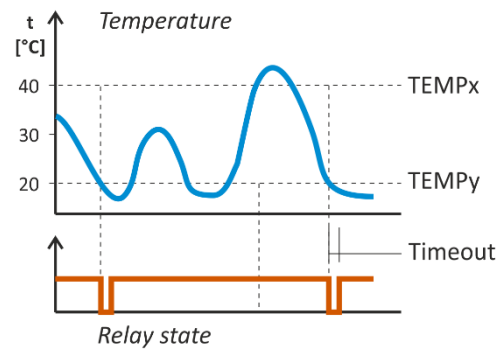
- Idle output state: OFF
- Bits in *flags* parameter:
 - SSxS: 10x0 – switch on to *time*
 - K: 1 – temperature drop



Mode 6

The output switches off for a set period of time when it drops below the TEMPy temperature. It can open again only when the temperature rises above TEMPx and then drops again to TEMPy.¹⁹

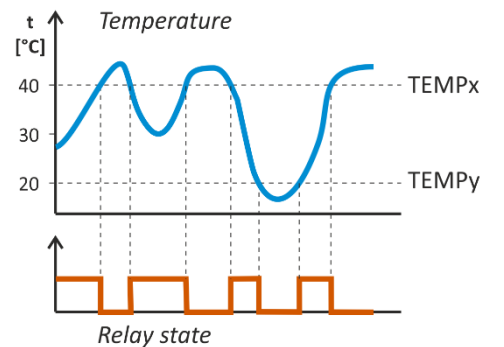
- Idle output state: ON
- Bits in *flags* parameter:
 - SSxS: 11x0 – switch off to *time*
 - K: 1 – temperature drop



Mode 7

The output is switched on if the temperature is between TEMPx and TEMPy.

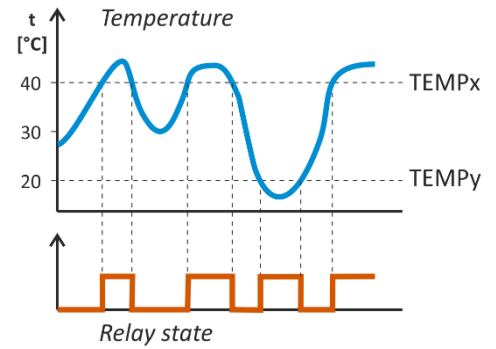
- Bits in *flags* parameter:
 - SSxS: 00x1 – switch on in limits



Mode 8

The output is switched on if the temperature is outside the TEMPx and TEMPy limits.

- Bits in *flags* parameter:
 - SSxS: 01x1 – switch off out of limits



Papouch s.r.o.

Industrial data transmission, line and protocol converters, RS232, RS485, RS422, USB, Bluetooth, Ethernet, LTE, WiFi, measurement modules, smart temperature sensors, I/O modules, custom development and manufacturing.

Address:

**Strasnicka 3164
102 00 Prague 10
Czech Republic**

Phone:

+420 267 314 267

Web:

en.papouch.com

Mail:

info@papouch.com

