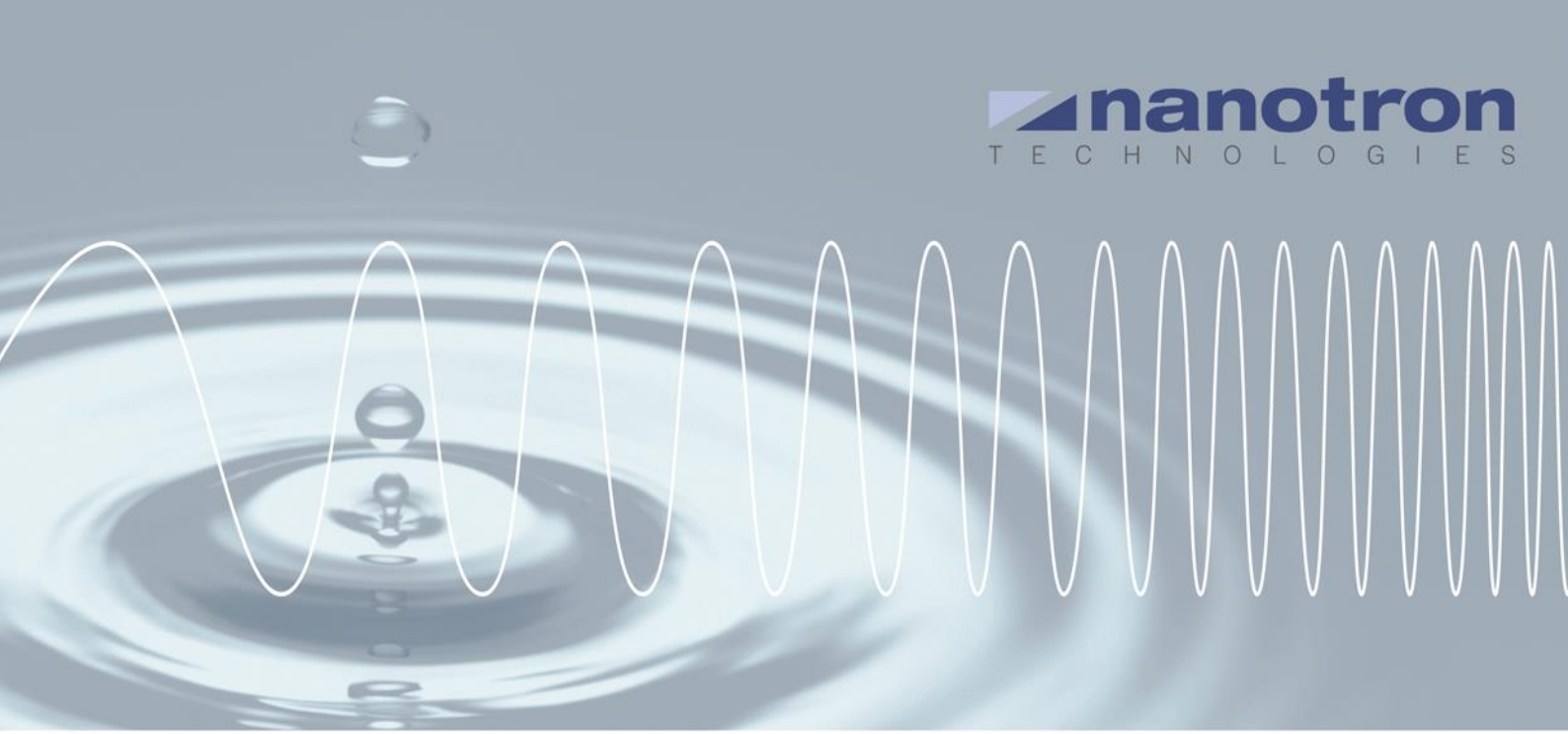

AN0602 - RTLS tag back channel

1.1

NA-16-0263-0050

Document Information

Document Title:	AN0602 - RTLS tag back channel
Document Version:	1.1
Current Date:	2016-09-28
Print Date:	2016-09-28
Document ID:	NA-16-0263-0050
Document Author:	nanotron Technologies

Contents

1. Introduction

When using the RTLS, the communication is normally in the direction from the tags to the anchors. The tag blinks and the anchors detect the blink; the tag acts as a transmitter and the anchors as receivers.

At some moments, however, it can be interesting to be able of dynamically change certain parameters of the tags, like, for instance, the blink rate. In addition that should be done dynamically, while the application is running and without the need of detaching the tag from the place where it is installed.

To handle this kind of situations the RTLS includes the feature 'tag backchannel'; which can be used to communicate with nanotags (all varieties) or the swarm bee. The backchannel support the communication from the system to the tags via the anchors.

2. What is the back channel?

The back channel is a service provided by the nanoANQs and used by nanoLES to send data in the direction from the server to the tag. This is normally used to dynamically change the settings of a tag device in the field, remote tag. For this, the user or user application sends a message to the server containing the instructions that the remote tag should follow. The server will retransmit that instruction to an anchor, and the anchor to the remote tag.

2.1. Tag backchannel with nanoLES 2

By default nanoLES 2 provides the backchannel service at the management port, port 3456. This means that all messages should be sent through that port. The message should contain, not only in the instruction for the tag, but also the ID of the remote tag and the ID and port of the anchor that should proceed with the communication to the tag. The format of the message is:

```
tagbc msg <anchor_ID> <anchor_port> <tag_ID> <instruction>
```

The anchor port is always the port 4646 (<anchor_port> = 4646). The <instruction> filed will contain the command that the remote tag should follow, and it will be explain in detail in section 3.

Although communication is from server to the tag, there are acknowledgements (ack) sent in opposite direction. The first one is received from the anchor to indicate the instruction was sent to the remote tag. The second ack is received from the tag indicating the tag received the message and successfully applied the instruction, or not.

2.2. Tag backchannel with nanoLES 3

nanoLES 3 has an specific server port for the backchannel, by default the port 3461.

As in nanoLES 2 the user can indicate to the server what anchor should be used to retransmit the message to the remote tag. However, this is not necessary anymore; the user can also leave this option open so that nanoLES 3 will select the anchor that is the best suited for the operation. To do so, nanoLES 3 keeps a list where it saves, for all the tag broadcasting blinks in the area, the tag ID and the anchor that received its last blink with the highest strength (RSSI). Thus when nanoLES 3 needs to select an anchor for the backchannel it only needs to look it up in the table.

The format of the message when the anchor to be used is specified is:

```
tagbc msg <tag_ID> "<instruction>" <anchor_ID> <anchor_port>
```

(again <anchor_port> = 4646)

When no anchor is specified the message format should be:

```
tagbc msg <tag_ID> "<instruction>"
```

Note that whether the first or the second option is chosen, the instruction field goes always in between " ".

Similarly to nanoLES 3, after sending a backchannel message, the server should receive two ack's, the first one from the anchor retransmitting the message and the second one from the remote tag itself.

3. Instruction sent over the backchannel

In the previous section we have seen the format of the backchannel messages that should be sent to the server. However we do not know yet what data should include the field <instruction>. This field follows the structure below:

`0x08 0x12 0x55 0x54 <protocol version> <command type> <reserved> <command length> <command code> <command parameter>`

Protocol version: indicates whether the message is send to or received from a nanoTAG or a swarm bee device. The possible values are:

`<Protocol Version>` `0x01` for a nanoTAG

`0x02` for a swarm bee device

The command type: indicates whether the message is to set a new value for a certain parameter (set command), to read out the value of a certain parameter (get command) or it is the tag response to a received command (response)

`<Command Type>` `0x54 Get command`

`0x55 Set command`

`0x56 Response to a get command`

`0x57 Response to a set command`

`0x60 Error message from the tag` (only when using swarm bee)

Reserved: it is normally 0x00

Command length: indicates the length of the command code plus the required parameters; it also indicates how many more bytes are included in the message after this field. This value depends on the command that is send to the tag.

Command code and Command parameter: each kind of device, nanoTAG or swarm bee, has a set of commands with their corresponding codes and parameters. It is explain in sections 3.1 and 3.2.

3.1. nanoTAG

As explained before the protocol version used by the nanoTAG devices is 0x01.

Table 3-1 shows the available command codes. The columns Get, Response Get, Set and Response Set indicate the length of the command code plus the parameters (if required) depending on the type of message. An empty box indicates that it is not possible to use that command as such a message type. For instance, the hardware version can be read out but it cannot be set; it cannot be changed by software as it depends on the hardware.

Table 3-1: Backchannel commands for the nanoTAG family

Code	Command	Length				Parameter
		Get	Resp. Get	Set	Resp. Set	
0x01	Hardware version	0x01	0x03	-	-	version
0x03	Temperature sensor present	0x01	0x02	-	-	0 = not present 1 = present
0x04	Acceleration sensor present	0x01	0x02	-	-	0 = not present 1 = present
0x05	Maximum transmitted power	0x01	0x02	0x02	0x02	0..63
0x06	Software version	0x01	0x05	-	-	version
0x09	Transmission mode	0x01	0x02	0x02	0x02	0 = 80/1 1 = 22/4 2 = 80/4

0x0A	Channel (22MHz mode)	0x01	0x02	0x02	0x02	0..13
0x0B	Sync word pattern	0x01	0x09	0x09	0x09	pattern
0x0C	Sync word	0x01	0x02	0x02	0x02	0..8
0x0D	RF oscillator calibration interval	0x01	0x03	0x03	0x03	1..600 seconds
0x11	CSMA	0x01	0x02	0x02	0x02	0: disable 1: enable
0x12	Transmit power	0x01	0x02	0x02	0x02	0..63
0x13	Acc. sensor threshold	0x01	0x02	0x02	0x02	0..255
0x14	Acc. sensor bandwidth	0x01	0x02	0x02	0x02	0…6 in FW v.12 8…15 in FW v.18
0x17	Battery level	0x01	0x03	0x03	0x03	0… 65535 dV
0x18	Unique tag ID	0x01	0x09	0x09	0x09	mac address
0x19	Save and update	-	-	0x01	0x01	(none)
0x1A	FEC	0x01	0x02	0x02	0x01	0: disable 1: enable
0x20	Reset EEPROM	-	-	0x01	0x01	
0x21	Allow power off via tag button	0x01	0x02	0x02	0x02	0 = no 1 = yes
0x23	Start streaming	0x01	0x03	0x03	0x03	1… 65535 ms
0x25	Stop streaming	-	-	0x01	0x01	(none)
0x30	Tag ID	0x01	0x04	-	-	e.i.: 000000000012
0x32	Blink interval (ms)	0x01	0x03	0x03	0x03	5… 65535 ms
0x35	Battery alarm threshold	0x01	0x03	0x03	0x03	
0x36	Tag backchannel duration	0x01	0x02	0x02	0x02	0..100 ms 0:backch disabled
0x38	Tag backchannel occurrence	0x01	0x02	0x02	0x02	1..250 ms
0x3A	Fall back time (ms)	0x01	0x05	0x05	0x05	0… 900000000 ms
0x3B	Fall back blink interval	0x01	0x05	0x05	0x05	0… 900000000 ms
0x3C	Fall back death time	0x01	0x05	0x05	0x05	0… 900000000 ms

In addition, the nanoTAG RX includes an optocoupler that can also be controlled through the air interface. The command is shown in Table 3-2

Table 3-2: backchannel command specific to nanoTAG RX

Code	Command	Length				Parameter format
		Get	Resp. Get	Set	Resp. Get	
0x27	Optocoupler	0x01	0x02	0x02	0x02	0 = off 1 = on

The length values shown in Table 3-1 and Table 3-2 correspond to nanoTAG's firmware version 18. In previous versions of the firmware the response message adds an extra byte at the end of the response. That extra byte is always 0x00.

Examples

Example 1: Via nanoLES 2 changing the blink interval of a tag with ID 00:00:09:2f:dd:70 to 1 second and verifying it.

a) First we need to set the blink interval. We will build the instruction that should be included in the backchannel message:

```
0x08 0x12 0x55 0x54 <protocol version> <command type> <reserved> <command length>
<command code> <command parameter>
```

Protocol version: 0x01
For this the required command type is 'set': 0x55
Packet ID counter is the sequence number of the packet.
Command code: 0x32
Command length: 0x03 (according to Table 3-1)
The parameter is 1000 ms, in hexadecimal with a length of 2 bytes: 0x03e8
(We know the length of the parameter because the total length is known and the command code is always 1 byte)

The instruction is then: `0x08 0x12 0x55 0x54 0x01 0x55 0x00 0x03 0x32 0x03 0xe8`

Finally this instruction should be added to the complete backchannel message, which will be:

```
tagbc msg 122.122.122.122 4646 00:00:09:2f:dd:70 0x08 0x12 0x55 0x54 0x01 0x55
0x00 0x03 0x32 0x03 0xe8
```

Once the message is sent to the server we will receive two acknowledgements back, the first from the anchor and the second from the tag.

b) For the second step we want to verify the blink interval. For this we need a command type GET:

Protocol version: 0x01
For this the required command type is 'get': 0x54
Command code: 0x32
Command length: 0x01 (according to Table 3-1)
No parameter required

The instruction is added to the complete message:

```
tagbc msg 122.122.122.122 4646 00:00:09:2f:dd:70 0x08 0x12 0x55 0x54 0x01 0x54
0x00 0x01 0x32
```

After the backchannel message is sent we will receive the two acknowledgments and the answer from the tag, this will be type 0x56:

```
tag response 00:00:09:2f:dd:70 0x08 0x12 0x55 0x54 0x01 0x56 0x00 0x03 0x32 0x03
0xe8
```

Example 2: Using nanoLES 3, read out the status of the optocoupler of a nanoTAG RX.

Again we start by building the instruction for the backchannel message:

```
0x08 0x12 0x55 0x54 <protocol version> <command type> <reserved> <command length>
<command code> <command parameter>
```

Command type is 'get': 0x54
Command length: 0x01 (this length indicates that there is no parameter)
Command code: 0x27

The instruction is: `0x08 0x12 0x55 0x54 0x01 0x54 0x00 0x01 0x27`

This instruction should be added to the complete backchannel message. There are two options: we can either a) leave up to nanoLES the selection of the anchor that it will use, or b) include this information in the backchannel message.

a) `tagbc msg 00:00:09:2f:dd:70 "0x08 0x12 0x55 0x54 0x01 0x54 0x00 0x01 0x27"`

b) `tagbc msg 00:00:09:2f:dd:70 "0x08 0x12 0x55 0x54 0x01 0x54 0x00 0x01 0x27" 122.122.122.122 4646`

The response from nanoLES will be:

```
ok
```

`INFO 1045: message successfully transmitted to the tag` (1st ack to confirm that the instruction was sent)
`INFO 1046: tag response 00:00:09:2f:dd:70 0x0a "0x08 0x12 0x55 0x54 0x01 0x56 0x00 0x02 0x27 0x01"` (2nd ack, answer from the tag)

`0x0a` → indicates the length of the message that comes after. (Only present with nanoLES 3)

The message sent to the tag is: `0x08 0x12 0x55 0x54 0x01 0x57 0x00 0x02 0x27 0x01`
`0x08 0x12 0x55 0x54`
`0x01` → protocol version 1
`0x57` → message type 'response to set'
`0x00` → reserved byte
`0x02` → length 2
`0x27` → command: optocoupler state
`0x01` → optocoupler ON

3.2. swarm bee

The protocol version used by the swarm family devices is 2. This new version was created to adapt the command codes to the API commands of the swarm bee.

Table 3-3 shows the list of commands available indicating the length of the command code plus the required parameter depending on the selected message type. An empty box indicates that the command cannot be used as that type of message. The commands with a * are better explained below the table.

Table 3-3: Backchannel commands for the swarm bee family

Code	Command	Length				Parameter
		Get	Resp. Get	Set	Resp. Set	
0x00	Node ID	0x01	0x07	0x07	0x07*	ID
0x01	Save settings	-	-	0x01	0x01	-
0x02	Reset EEPROM	-	-	0x01	0x02	0 = successful 1 = failed
0x03	Set factory settings	-	-	0x01	0x01	0 = successful 1 = failed
0x04	Power mode	0x01	0x02	0x02	0x02	0 = always active 1 = active / sleep 2 = active / snooze 3 = active / nap
0x05	Transmission power	0x01	0x02	0x02	0x02	0... 63, register value
0x06	Sync word	0x01	0x02	0x02	0x02	0... 8, and 12
0x08	Firmware version	0x01	0x05	-	-	0 …0x00000000 i.e. 0x02003062 for ver2.0.3-rc0-98
0x09	Unique ID	0x01	0x0d	-	-	0 …FFFFFFFFFFFFFFFFFFFFFFF
0x0A	UART baud rate	0x01	0x05	-	-	500 … 2000000 Bps
0x10	Privacy mode	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x11	Privacy blacklist	-	-	0x02	0x02	Parameter 1:

						0 = clear white list
		-	-	0x08	0x08	Parameter 1: 1 = add to list 2 = remove from list Parameter 2: ID (6 Bytes)
		0x01	0x02 to 0x72	-	-	Parameter 1: 0… 19 Parameter 2: IDs, as many as indicated by parameter 1
0x13	Broadcast range result	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x14	Class to range to	0x01	0x02	0x02	0x02	8 bits mask (one bit per class) 0 = class not allowed 1 = class allowed
0x15	Ranging white list	-	-	0x02	0x02	Parameter 1: 0 = clear white list
				0x08	0x08	Parameter 1: 1 = add to list 2 = remove from list Parameter 2: ID (6 Bytes)
		0x01	0x02 to 0x72	-	-	Parameter 1: 0… 19 Parameter 2: IDs, as many as indicated by parameter 1
0x16	Range result notification	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x17	Ranging offset	0x01	0x03	0x03	0x03	0 .. 65000 ms.
0x20	Data notification	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x23	Start streaming	-	-	0x03	0x03	1… 6500 ms
0x24	Extend streaming	-	-	0x01	0x01	-
0x25	Stop streaming	-	-	0x01	0x01	-
0x26	Node ID notifications	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x28	Data in ID notification, delete	0x01	0x02	0x02	0x02	Parameter 1: 0 = no data/ delete data
	Data in ID notification, fill	0x01	0x03 to 0x5D	0x03 to 0x5D	0x03 to 0x05	Parameter 1: 1…91 Bytes Parameter 2: data
0x2A	Data in ranging packets,	0x01	0x02	0x02	0x02	Parameter 1:

	delete					0 = no data/ delete data
	Data in ranging packets, delete	0x01	0x03 to 0x76	0x03 to 0x76	0x03 to 0x76	Parameter 1: 1…116 Bytes Parameter 2: data
0x30	Blink ID notification broadcast	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x31	Blink interval	0x01	0x03	0x03	0x03	0…65000 ms
0x32	Notifications configuration	0x01	0x03	0x03	0x03	10-bit mask, one per sensor field 0 = field not added 1 = field added
0x40	Reception window length	0x01	0x03	0x03	0x03	0…65000 ms
0x41	Reception window occurrence	0x01	0x02	0x02	0x02	0 … 255 0 = RX window disable x = after every x blinks
0x42	Device class	0x01	0x02	0x02	0x02	1… 8
0x43	FEC	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x44	Transmission mode	0x01	0x02	0x02	0x02	1 = 80/1 2 = 80/4
0x45	CSMA off	0x01	0x02	0x02	0x02	0 = csma off
	CSMA symbol detection	0x01	0x03	0x03	0x03	Parameter 1: 1 = symbol detection, random seed 2= symbol detection, fix back-off Parameter 2: 0 …255 24us-slots
	CSMA energy detection	0x01	0x04	0x04	0x04	Parameter 1: 1 = energy detection, random seed 2= energy detection, fix back-off Parameter 2: 0 …255 24us-slots Parameter3: 0…63 register value
0x50	Enable/disable MEMS	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x51	Broadcast MEMS data	0x01	0x02	0x02	0x02	0 = disable 1 = enable
0x52	MEMS acceleration range	0x01	0x02	0x02	0x02	1 = +/- 2g 2 = +/- 2g 3 = +/- 2g 4 = +/- 16g
0x53	MEMS threshold	0x01	0x02	0x02	0x02	0… 255

0x54	MEMS bandwidth	0x01	0x02	0x02	0x02	1… 8
0x55	MEMS' sleep time	0x01	0x02	0x02	0x02	1… 11
0x56	MEMS' death time	0x01	0x03	0x03	0x03	0… 65000 ms
0x57	Get own acceleration	0x01	0x07	-	-	x mg, y mg, z mg
0x58	Get own temperature	0x01	0x02	-	-	-99 °C … 99°C
0x59	Get battery level	0x01	0x02	-	-	0… 255 dV
0x5A	GPIO, input/output mode	0x02	0x06	0x06	0x06	Parameter 1: pin 0… 3 Parameter 2: 0 = input 1 = output Parameter 3: 0 = 400 KHz 1 = 2MHz 2 = 10MHz 3 = 40MHz Parameter 4: 0 = push pull 1= open drain Parameter 5: 0 = No pull 1 = Pull up 2 = Pull down
	GPIO, wake-up mode	0x02	0x03	0x03	0x03	Parameter 1: pin 0… 3 selected Parameter 2: 2= wake-up pin
	GPIO, alternative blink	0x02	0x06	0x06	0x06	Parameter 1: pin 0… 3 selected Parameter 2: 3=alternative blink Parameter 3: 0 ... 65000 ms Parameter 4: 0 = Low active 1 = High active Parameter 5: 0 … 255
0x5B	Pin status	0x01	0x02	0x03	0x03	4-bit mask, 1 per pin Status: 0 = reset 1 = set
0x5C	Interrupts configuration	0x01	0x03	0x03	0x03	9-bit mask, one per possible interrupt
0x3C	Alternative blink interval (MEMS)	0x01	0x06	0x06	0x06	Interval: 0… 6500 ms Priority level. 0…255 Timeout: 0…6500 ms

All the commands in the table and the parameters that they required are explained in detail in the swarm API document [1]

When a swarm family device receives a command, either over the air or through the UART, it first checks whether the parameter received is inside the range of valid parameters. If it is not, it does not modify the setting and, instead of the acknowledgement it sends and error message. To indicate this kind of message over the backchannel it uses the message type 0x60.

Examples

Example 1: With nanoLES 2, **s**etting the CSMA in symbol detection (and random seed) in the remote swarm bee with ID 00:00:09:2f:dd:70

First we need to set the blink interval. We will build the instruction that should be include in the backchannel message:

```
0x08 0x12 0x55 0x54 <protocol version> <command type> <reserved> <command length>
<command code> <command parameter>
```

Protocol version: 0x02
For this the required command type is 'set': 0x55
Command length: 0x03 (we need to check in the row for symbol detection)
Command code: 0x45
Parameter 1: 0x01 (random seed)
Parameter 2: 0x05
The instruction is then: `0x08 0x12 0x55 0x54 0x02 0x55 0x00 0x03 0x45 0x01 0x05`

Finally this instruction should be added to the complete backchannel message, which will be:

```
tagbc msg 122.122.122.122 4646 00:00:09:2f:dd:70 0x08 0x12 0x55 0x54 0x02 0x55
0x00 0x03 0x45 0x01 0x05
```

The answer that we will receive will be:

`ok` (is tag from the anchor)

`ok` (2nd ack from swarm)

Example 2: With nanoLES 3, we would like to know the configuration of GPIO1 of the remote swarm device

Again we start by the instruction itself:

```
0x08 0x12 0x55 0x54 <protocol version> <command type> <reserved> <command length>
<command code> <command parameter>
```

Protocol version: 0x02
For this the required command type is 'get': 0x54
Command length: 0x02
Command code: 0x5a
Parameter: 0x01 (As it is a get command the only required parameters is the GPIO that should be checked)

The instruction will be: `0x08 0x12 0x55 0x54 0x02 0x54 0x00 0x02 0x5a 0x01`

The instruction should be added to the backchannel message. As we are using nanoLES 3, we have 2 options: a) we can indicate what anchor should retransmit the message for the tag, or b) we can let nanoLES to pick the most appropriate anchor for the task.

a) `tagbc msg 00:00:09:2f:dd:70 "0x08 0x12 0x55 0x54 0x02 0x54 0x00 0x02 0x5a 0x01" 122.122.122.122 4646`

b) `tagbc msg 00:00:09:2f:dd:70 "0x08 0x12 0x55 0x54 0x02 0x54 0x00 0x02 0x5a 0x01"`

In both cases the answer that we will receive will be:
`ok`

`INFO 1045: message successfully transmitted to the tag` (ack from the anchor)

```
INFO 1046: tag response 00:00:09:2f:dd:70 0x0e "0x08 0x12 0x55 0x54 0x02 0x56
0x00 0x06 0x5a 0x01 0x01 0x03 0x00 0x02"
```

`0x0e` → indicates that the message length is 14 bytes. (Only present with nanoLES 3)

We finally need to interpret the answer from the tag, which in this case is the swarm device. We will analyze it byte by byte: `0x08 0x12 0x55 0x54 0x02 0x56 0x01 0x02 0x5a 0x01`

`0x08 0x12 0x55 0x54` → initial bytes
`0x02` protocol version, 2 → swarm bee
`0x56` command type, 56 →answer to get
`0x00` reserved byte
`0x06` length, the command code is 1 byte, thus the parameters are 5 bytes
`0x5a` command code, 5a → GPIO
`0x01 0x01 0x03 0x00 0x02` → parameters, they appear in the message in the same order as they are shown in Table 3-3:

- `0x01` → GPIO 1
- `0x01` → output mode
- `0x03` → speed: 40 MHz
- `0x00` → push pull type
- `0x02` → set when pulled down

4. References

[1] swarm API, v 3.0, NA-13-0267-0003, April 2016, nanotron Technologies

Document History

Date	Author	Version	Description
15-7-2016	MLA	1.0	Initial document
28-9-2016	MLA	1.1	Small correction

Life Support Policy

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nanotron Technologies GmbH customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify nanotron Technologies GmbH for any damages resulting from such improper use or sale.

About Nanotron Technologies GmbH

Today nanotron's *embedded location platform* delivers location-awareness for safety and productivity solutions across industrial and consumer markets. The platform consists of chips, modules and software that enable precise real-time positioning and concurrent wireless communication. The ubiquitous proliferation of interoperable location platforms is creating the location-aware Internet of Things.

Further Information

For more information about products from nanotron Technologies GmbH, contact a sales representative at the following address:

nanotron Technologies GmbH
Alt-Moabit 60
10555 Berlin, Germany
Phone: +49 30 399 954 – 0
Fax: +49 30 399 954 – 188
Email: sales@nanotron.com
Internet: www.nanotron.com