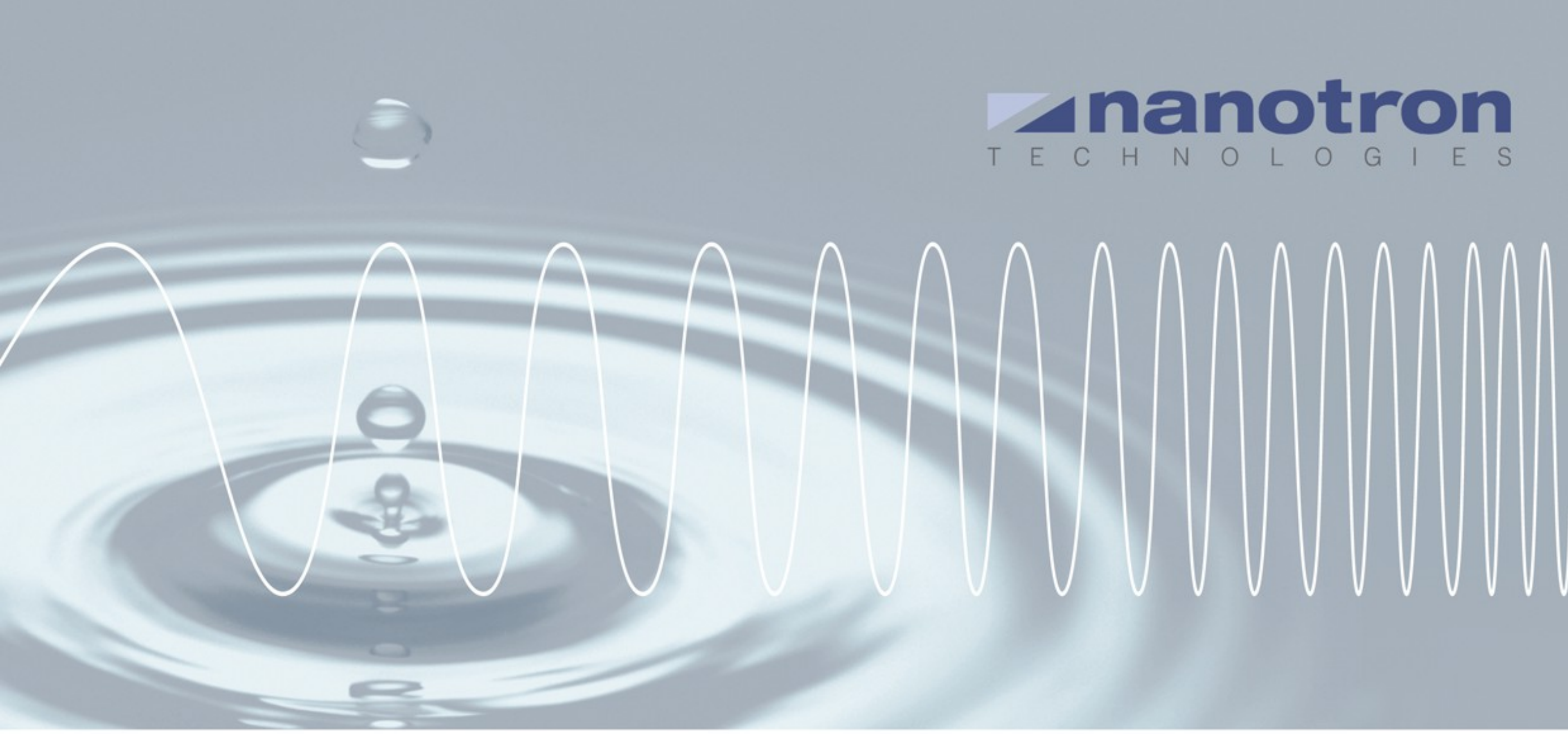


swarm bee - User Guide

1.0

NA-15-0356-0019-1.0

Document Information

Document Title:	swarm bee - User Guide
Document Version:	1.0
Current Date:	2015-08-25
Print Date:	2015-04-14
Document ID:	NA-15-0356-0019-1.0
Document Author:	

Contents

1. Scope

The Scope of this document is to define a hardware independent Application Programming Interface (API) to realize the ranging functionality for a *swarm* radio described in chapter 2. A swarm is defined as a congregation of independent radios or nodes which share a common interest in their relative positioning and can communicate together, and detect newcomers and departures from the local swarm area.

The main goal of the API is to support and simplify the development of ranging applications and allow swarm mobility. For these purposes, it provides a list of commands which act as interface with the hardware and can be used to build more complex applications in an easy way.

From API 2.0 onward power management for mobile devices is supported. The API 2.x.x is not backward compatible with previous releases (V1.6.x).

2. Functionality Overview

The intended application is a multi-node peer-to-peer ranging solution in which every node shall be independent from each other, capable of ranging and communicating with other nodes and battery operated to allow fully mobile applications. For this purpose the *swarm* radio shall be able to perform two tasks:

1. Send out ID broadcast information. Swarm radio nodes shall periodically broadcast their own node ID together with their node information data, so that other swarm radio nodes can be aware of the presence of the first one and can decide to interact with it. This feature can be deactivated.
2. Listening to other broadcast IDs and react on them. When a swarm radio receives a broadcast ID packet, it shall store the node's ID (and if relevant the information data received) and initiate a ranging operation with the swarm radio owning this broadcast ID.

A ranging operation consists of 3 packets. The first packet is a ranging request sent by the node initiating the operation; the other two packets are the automatic response of the node receiving the request. The automatic ranging response for packets can be also deactivated by “privacy mode”.

The received sensor data consist of sensors data (temperature and acceleration), battery level, node class and node power management mode. The node class is set by the user and it is used to classify the nodes and configure their automatic ranging feature. All this node information data is stored in the receiver swarm node and can be accessed via the API as well as the battery status of the node itself.

The node class is an application dependent parameter; it is used to define different groups of swarms. They can be classified according to their power mode, the kind of device they are placed on, how fast they move... the user is free to use its own criteria and what better fits with the application. Each node can then be given a list of classes so that whenever it receives a node-id blink, it checks whether that class is in the list and, if yes, it immediately starts ranging with that node.

To implement the described functionalities the *swarm* radio shall either be connected to a host platform or work as an autonomous node. The node is controlled through the USART interface connected to a host platform. This interface is described in chapters 4. Furthermore the swarm radio nodes that are host controlled should be capable of communicating data packets of variable length to another *swarm* radio B, which can be host-controlled or autonomous.

The higher level application layer is not part of the API specification and will be programmed by the user.

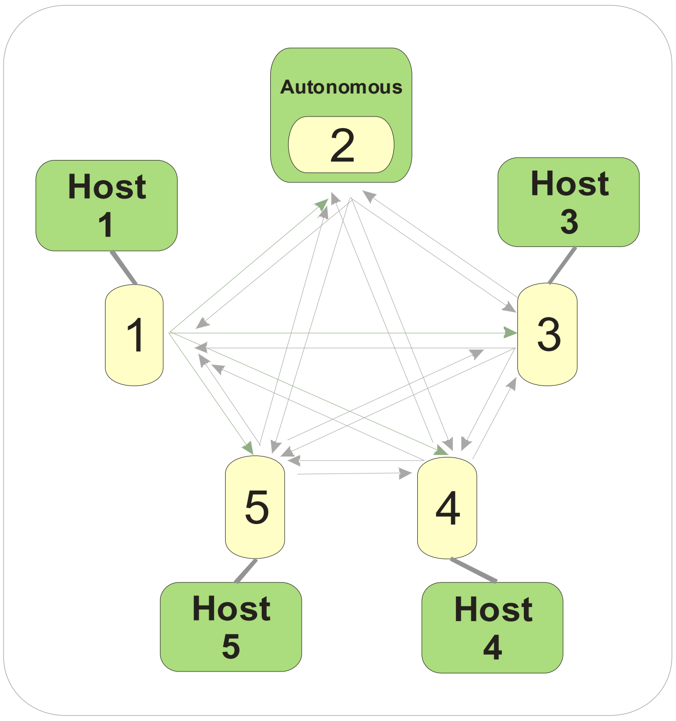


Figure 1. Example of *swarm* radio nodes interacting with each other. Nodes 1,3,4,5 are host-controlled and node 2 works autonomously.

3. Locating Methods

Swarm radios can perform location by using either collaborative or fixed location methods. The selection of one or the other will be influenced by the application requirements.

When working in a collaborative mode the swarms are aware of the presence of other swarms in their neighborhood and are capable of estimating their range or 1-D position relative to its own position. They can then inform other nodes in the area and send them the estimated positions (as well as the received ones) so that all swarms can use that information to make an estimation of the position of other swarms that are not in range.

Collaborative Location

Collaborative location uses relative positions to provide location-awareness. Radio nodes determine the distance to neighbors by exchanging packets and measuring their time of flight (TOF) at the speed of light. This method is called ranging. Radios are autonomous, location infrastructure is not required.

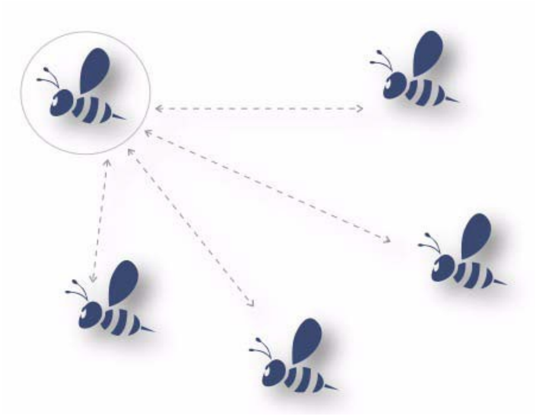


Figure 2 Collaborative location for mobile nodes using nanotron's swarm concept

Fixed Location

This uses fixed reference points or 'Anchors' to provide location awareness. Anchors are connected to a standard network, and a central computer or server tracks the positions of the tags. Because this system is based on time difference of arrival (TDOA), only one data packet sent from the tag is required to get a position fix in 1D, 2D or 3D. The need to only transmit a single packet reduces power consumption of the tag significantly. Less packets in the air per position fix allow for a larger number of objects to be tracked.

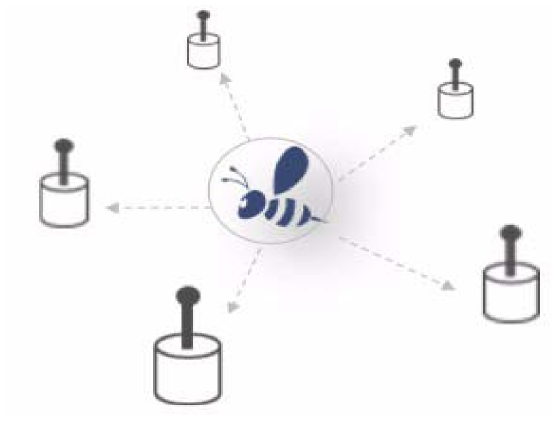


Figure 3 Fixed location utilizing location infrastructure with fixed radio nodes as anchors

4. Application Programming Interface

4.1. General Purpose

This application programming interface has been implemented so that it allows users to create their own applications, if desired. It consists of a low level API based on SDS-TWR ranging and other commands supporting data communication.

4.2. General Communication Protocols

Communication between the host controller and the swarm radio is supported by an ASCII and an BINARY protocol. This defines a set of commands with corresponding parameters and return values. The return values can be sent by the swarm radio immediately after the command is received or later once the action indicated by the command is performed, asynchronous return values.

In order to reconfigure remote swarm devices over the air, an AIR protocol is also implemented. With the help of the AIR protocol a connected host can send commands to remote devices over the air.

4.2.1. ASCII protocol

For the general ASCII communication protocol the following conventions apply:

1. All communication via the interface is done by ASCII characters. This implies that, every numeric parameter, e.g. a 6 byte node ID (hexadecimal), will be transmitted in the following format:

Node ID (hex)												
0000BF260468	0	0	0	0	B	F	2	6	0	4	6	8
ASCII (hex)	30	30	30	30	42	46	32	36	30	34	36	3

2. All command communication ends with carriage return / line feed:

Command termination			
… \r\n	…	\r	\n
ASCII (hex)	…	0D	0A

3. All command codes and their respective parameters are separated by one space character (ASCII 20)

   Example (String): `“RATO 0 0000BF260468\r\n"`

   Example (Bytes): `{0x52, 0x41, 0x54, 0x4F, 0x20, 0x30, 0x20, 0x30, 0x30, 0x30, 0x30, 0x42, 0x46, 0x32, 0x36, 0x30, 0x34, 0x36, 0x38, 0x0D, 0x0A}`

4. Return code for unknown or erroneous command is „=ERR\r\n“

   Example:
   ```
   WrongCommand xyz
   =“ERR\r\n”
   ```

5. To all commands which return one single line, the reply begins with ‘=’ (ASCII Hex 3D).

Example:
```
GNID

=001122334455
```

6. To all commands which return multiple lines, the reply begins with '#' (ASCII Hex 23), followed by the number of lines.

Example:
```
GRWL
#003
DDF451534C23
134683567ABC
33A441FFB311
```

7. All asynchronous lines start with '*' (ASCII Hex 2A).

Example: `*RRN:001122334455,0,0010.3,-37`

4.2.2. BINARY protocol

This protocol describes the bidirectional binary interface between swarm- and hostdevice. The information is provided in high density. Each transfer is done in a single frame which is described within general packet structure chapter.

4.2.2.1. General Packet Structure

The general packet structure is given by the following table.

SYN	LEN	DATA	CRC_LOW	CRC_HIGH
1 byte	1 byte	1 .. 256 byte	1 byte	1 byte

Table 4.1: General packet structure

The SYN field is used for the synchronization of the data stream and by default "0x7f". Next, the LEN field describes the length of the following DATA. The range of value is 0 .. 255, 0 is representing 256 byte of DATA. It is possible that the SYN value or "0x7F" occurs in the data, so in order to prevent false detection of packet starting with this value, it must be modified to an escape value. The DATA field contains the user data/payload. To escape SYN in data field use the Escape sign ESC = "0x1b". However, the ESC value or "0x1b" may also appear in the data. To prevent false detection of ESC it used another modified value "0x1b45". Before transmitting from host to swarm device, each message must be checked for SYN or ESC signs. Every discovered sign must be escaped beside the initial SYN. After receiving a message from a swarm device, it must be de-escaped.

To escape a SYN:	ESC + 'S'
	0x1b53
To escape a ESC:	ESC + 'E'
	0x1b45

Example1: Sending SMBW 2 command in binary mode:

1) 0x7f03555402ba9f => nothing to escape, no SYN (0x7f) or ESC (0x1b) spotted after initial SYN.

2) The answer to this request from swarm to host is: 0x7f03575402**1b45**5f. This is already escaped and must be de-escaped after reception: 0x7f03555402**1b**5f.

Example2: Sending GRWL command in binary mode:

1) 0x7f02541547**1b** => last byte is 0x1b and must be escaped to: 0x7f02541547**1b45**

2) The answer to this request, if the list is empty, is: 0x7f03561500fb0e => nothing to de-escape.

The cyclic redundancy check (CRC) detects accidental changes to raw data. The swarm implementation uses CRC-16-ansi reversed, which is x^16 + x^14 + x^1 + 1 (0xA001, Koopman notation) and is initialized with 0The CRC check will be applied to SYN, LEN and DATA field. It only considers the original sign, not the escape sequences.

Example: ESC + 'S' reflects a 0x7f and a ESC + 'E' a 0x1b, therefore the 0x7f or 0x1b is used for calculating the CRC. Not the ESC + 'S' or ESC + 'E' itself.

Note: The CRC is transmitted LSB first. Ensure that order for CRC before transmitting.

CRC_LOW = second byte of CRC result

CRC_HIGH = first byte of CRC result

4.2.2.2. DATA Structure

The DATA consists of a TYPE, CMD opcode and a CMD_DATA field. The TYPE describes the kind of interaction with the swarm module and the CMD, the chosen command with the corresponding command data (CMD_DATA) eg. parameters.

TYPE	CMD	CMD_DATA
1 byte	1 byte	0 .. 254 byte

Table 4.2: Data Structure

4.2.2.3. TYPE

The command types are divided into two groups. Commands which are sent to the swarm devices (GET, SET) and command types which are included into a response or message of a swarm device (RESP, ERR, NOTI).

TYPE	Value	description
GET	0x54	Get command
SET	0x55	Set command
G_RESP	0x56	Get response
S_RESP	0x57	Set response
ERR	0x60	Error
NOTI	0x61	Asynchronous notification

Table 4.3: Type field definitions

TYPE GET must be used to read from swarm device. If something goes wrong, ERR is returned.

Example: GNID (get node id) request from host to swarm:

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x00	0x86	0xd4

Table 4.4: GET example request

Note:

CMD_DATA dosen't exist for GNID, because this command dosen't have parameters, other commands may have this field.

GNID response from swarm to host:

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x56	0x00	0x0000b6f31103	0x4a	0xe7

*Table 4.5: Get example response TYPE **G_RESP***

TYPE SET must be used to write to swarm device. If something goes wrong, ERR is returned.

Example: SNID (set node id) request from host to swarm:

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x55	0x00	0x0000b6f31103	0x0a	0xf2

Table 4.6: SET example request

SNID response from swarm to host with OK:

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x57	0x00	0x0000b6f31103	0x8b	0x2b

*Table 4.7: SET example response TYPE **S_RESP***

TYPE ERR indicates that something went wrong. Following errors may occur:

ERR_CRC	= 0x01	CRC is wrong
ERR_CMD_UNKNOWN	= 0x02	unknown command
ERR_PARAMETER	= 0x03	wrong parameter for command (too many / too few parameters, or range of values violated)
ERR_BUFFER_OVERFLOW	= 0x04	data did not fit into reception buffer
ERR_GARBAGE	= 0x06	unexpected sign during frame reception
ERR_TIMEOUT	= 0x07	incomplete packet (timeout 5ms)
ERR_LOCKED	= 0x08	parameter is locked

In any response with the TYPE *ERR* the CMD field does not contain the opcode instead it contains the corresponding error code, which is one of the values listed above.

Example: ERR because CMD unknown:

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x60	0x02	0x95	0xad

Table 4.8: ERR example request

TYPE NOTI occurs withhin asynchronous communication messages. Several notification messages exist. To separate the several notifications,types are defined:

Data Notification Message	DNO (0x60)
Node ID Notification Message	NIN (0x61)
Ranging Result Notification Message	RRN (0x62)
SDAT Notification Messages	SDAT (0x63)
AIR Notification Message	AIR (0x64)

The CMD field holds the information which type of notifications was transmitted. Depending on the type of the notification the CMD_DATA field is set up.

Example: NOTI because of Node ID Notification Message (Parameter is a node ID)

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x61	0x61	0x0000b6f31103	0x78	0xd3

4.2.3. AIR protocol

This protocol describes the bidirectional AIR interface. This allows reconfiguration of remote swarm devices through another swarm device using the SDAT command, where the parameter (or message sent) is the AIR packet. The information is provided the same way as it is done by the binary interface. Therefore an AIR protocol packet can be easily created using the binary description. SDAT always returns whether the packet reached the destination or not. In addition to that, a response packet from remote swarm device is generated. The response is generated right before the change takes effect. If the command is not known to the device, an error packet is generated.

Note: In autonomous mode, changed settings must be saved immediately. Otherwise they are lost, with the next blink, because all settings will be restored from EEPROM after each wake-up. Therefore send a streaming start (SSTART) to enable the receiver for a certain time on the remote device. Now change settings with the corresponding commands. Finish it with save settings (SSET) and stop streaming (SSTOP).

4.2.3.1. General Packet Structure

The general packet structure is given by the following table.

P_TYPE	P_VERSION	C_TYPE	RESERVED	C_LEN	C_OPCODE	DATA
4 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 ... 119 bytes

Table 4.9: General packet structure

P_TYPE = Protocol identifier = **0x08125554**

P_VERSION = Protocol version = **0x02**

C_TYPE = Command type =

C_TYPE	Value	description
GET	0x54	Get command
SET	0x55	Set command
G_RESP	0x56	Get response
S_RESP	0x57	Set response
ERR	0x60	Error

Table 4.10: Type field definitions

C_LEN = Payload length (all bytes after C_LEN until end of packet without CRC)

C_OPCODE = Command opcode (same as binary command opcode)

DATA = Contains OPCODE specific data and is described in the command description. (same as binary parameter description)

ERR

P_TYPE	P_VERSION	C_TYPE	RESERVED	C_LEN	C_OPCODE	DATA
4 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

Table 4.11: Error packet structure

TYPE ERR (0x60 in C_TYPE) indicates that something went wrong. Following errors may occur:

ERR_CMD_UNKNOWN	= 0x02	unknown command
ERR_PARAMETER	= 0x03	wrong parameter for command (too many / too few parameters, or range of values violated or GET/SET is not available for this C_OPCODE)

DATA = contains the corresponding error code (ERR_CMD_UNKNOWN or ERR_PARAMETER).

How to assemble an air packet for reconfiguration of swarm nodes

To reconfigure a certain setting an over the air packet must be assembled. The protocol type P_TYPE (0x08125554) and version P_VERSION (0x02) are always the same.

=> **0x0812555402**

Next part of protocol is the command type C_TYPE, which is either GET (0x54) or SET (0x55). - Table 4.10: Type field definitions.

=> 0x0812555402**55**

Followed by a reserved byte.

=> 0x081255540255**00**

Followed by the length C_LEN, which depends on the command itself. For changing the node ID blink interval the command SBIV is used. This command has one parameter with length of 2 bytes. The one byte command opcode counts also into length. This information can be found in the binary description of each command.

=> 0x08125554025500**03**

Next to C_LEN field is the opcode field (C_OPCODE). SBIV opcode is 0x31. Opcode information can be found at 4.5 Command Opcode Overview.

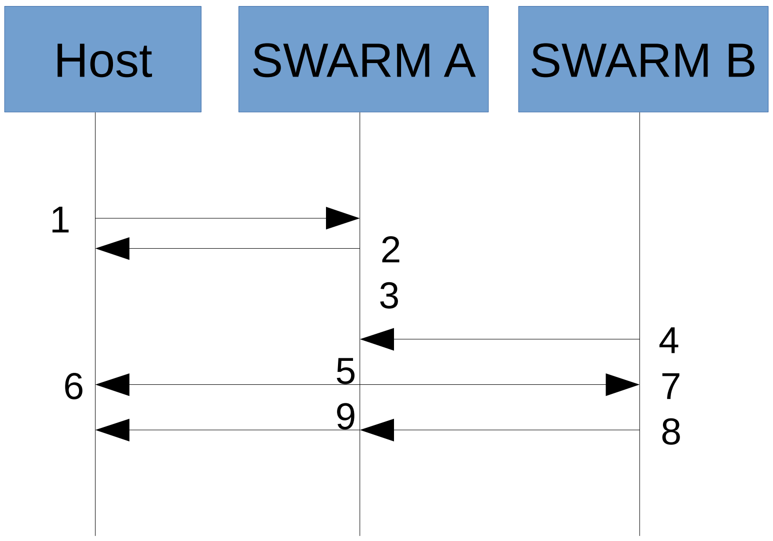
=> 0x0812555402550003**31**

After C_OPCODE the command specific parameter must be placed. In this case it has only 2 bytes, which configures the new blink interval. It is transmitted MSB first. See command parameter description for detailed information about range and order of each parameter. One second blink interval is 0x03e8.

=> 0x081255540255000331**03e8**

This can now be transmitted to any *swarm* device to reconfigure its blink interval. This can be done with SDAT command, or any other device which is able to send nanoLOC packets.

Example GTXP (get tx power) on ASCII interface using SDAT



1) Host transmits a SDAT command with GTXP (get tx power) as content:

SDAT <option> <ID> <len> **<data>** <timeout>

SDAT 1 000000000011 9 **081255540254000105** 60000

Data part which is transmitted over air:

P_TYPE	P_VERSION	C_TYPE	RESERVED	C_LEN	C_OPCODE	DATA
0x08125554	0x02	0x54	0x00	0x01	0x05	-

2) *swarm* A sends back payload ID to host.

=575090200

3) Wait for next node ID blink with RX slot open of *swarm* B device with ID (000000000011)

4) Node ID blink broadcast from *swarm* B.

5) *swarm* A transmits the prepared payload to *swarm* B (081255540254000105) and sends error code after transmission to Host.

6) Host receives asynchronous *SDAT response. No error occurred.

*SDAT:000000000011,0,575090200

7) *swarm* B receives the packet, and identifies the AIR protocol through P_TYPE and P_VERSION. Reads the C_TYPE and C_OPCODE which is set to GET and GTXP.

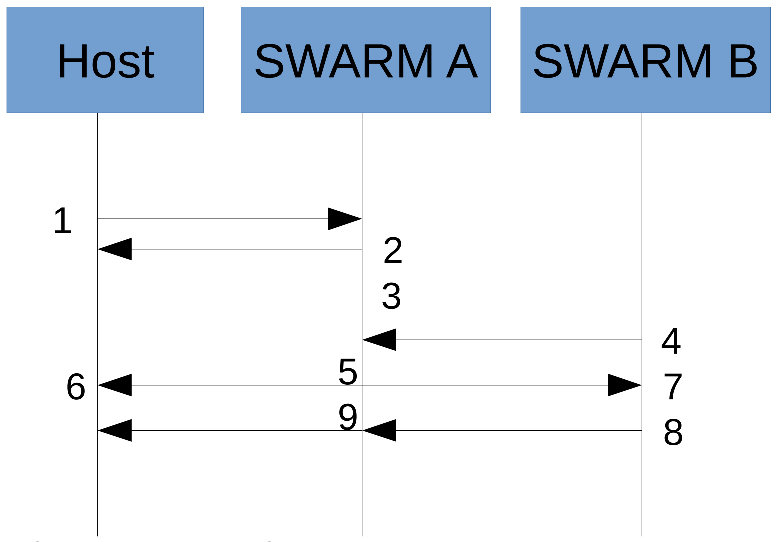
8) *swarm* B sends a GTXP response packet back to *swarm* A over air.

P_TYPE	P_VERSION	C_TYPE	RESERVED	C_LEN	C_OPCODE	DATA
0x08125554	0x02	0x56	0x00	0x02	0x05	0x3f

9) *swarm* A receives the response packet, and generates an *AIR notification at Host.

*AIR:000000000011,05,56,3f

Example GTXP (get tx power) on BINARY interface using SDAT



1) Host transmits a SDAT command with GTXP (get tx power) as content:

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x14	0x55	0x21	0x0100000000001108125554025400 0105ea60	0x79	0x86

SDAT command contains <Option>, <ID>, <Len>, <Data>, <TimeOut>.

In this case, <Option> is set to 1, which means it must wait for next rx slot of *swarm* B. Destination <ID> is 0x000000000011. The <Timeout> is 60000ms (0xea60).

<Data> is the part, which is transmitted over air:

P_TYPE	P_VERSION	C_TYPE	RESERVED	C_LEN	C_OPCODE
0x08125554	0x02	0x54	0x00	0x01	0x05

2) *swarm* A sends back the payload ID (0x0e4a0a96)

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x06	0x57	0x21	0x0e4a0a96	0x35	0x74

3) Wait for next node ID blink with RX slot open of *swarm* B device with ID (000000000011)

4) Node ID blink broadcast from *swarm* B.

5) SWARM A transmit the prepared payload to *swarm* B (081255540254000105) and sends error code after transmission to Host.

6) Host receives asynchronous *SDAT response. No error occurred.

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0d	0x61	0x63	0x000000000011020e4a 0a96	0xeb	0x39

7) *swarm* B receives the packet, and identifies the AIR protocol through P_TYPE and P_VERSION. Reads the C_TYPE and C_OPCODE which is set to GET and GTXP.

8) *swarm* B sends a GTXP response packet back to *swarm* A over air.

P_TYPE	P_VERSION	C_TYPE	RESERVED	C_LEN	C_OPCODE	DATA
0x08125554	0x02	0x56	0x00	0x02	0x05	0x3f

9) *swarm* A receives the response packet, and generates an *AIR notification at Host.

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0b	0x61	0x64	0x00000000001105563f	0x9c	0xff

4.3. Power management

Swarm radios support three different power modes, which can be selected with the API or on the hardware. The three modes are:

- Receiver always active (no power management). Can be set with the API command `SPSA 0`
- Node sleeping, receiver only active after node ID blink with RxSlot open and only for the specified RxWindow. UART is still active. Can be set with the API command `SPSA 1.` The RxWindow can be changed with the command `SRXW`. The RxSlot occurance can be changed with command `SRXO`.
- Deep sleep, receiver only active after node ID blink for the specified RxWindow. UART is not active. This mode is set in the hardware and it overwrites the two previous modes.

Note: When power saving is active, the *swarm* radio is **only** able to receive ranging requests, data packets and NodeIDs during the time span specified by the SRXW command following immediately after its own NodeID broadcast. Otherwise any communication over the air will be ignored. One exception is streaming (SSTART). If streaming is enabled, the receiver is on for the specified time regardless what is set for power saving mode (SPSA). It falls back to configured power saving operation if streaming is stopped (SSTOP) or the timeout expires. Except the power saving mode, everything else is working normal during streaming (e.g. node ID blinks).

The default configuration receiver always active, which is compatible to previous API command descriptions:

When a swarm radio A node wants to communicate to another swarm radio B in sleep mode it first stores the information until it receives the swarm radio B's NodeID.

Swarm radios store one ranging request or data packet per node ID. The number of node IDs that can be stored is 19. When a second ranging or data operation is signaled to the same node before the previous communication has been successfully established the first communication will be canceled and replaced by the new request. If the maximum number of ranging requests or data packets is reached an error will occur.

4.4. UART API Command Set Overview

This chapter summarizes and categorizes the API Command Set:

4.4.1. *swarm* radio Setup Commands

SNID	(0x00)	**S**ets the **N**ode **ID** of *swarm* node
GNID	(0x00)	**G**et **N**ode **ID** of the node connected to host
SSET	(0x01)	**S**ave **SET**tings; saves all settings including Node ID permanently to EEPROM
RSET	(0x02)	**R**estore **SET**tings from EEPROM (node configuration)
GSET	(n.a.)	**G**et current **SET**tings (node configuration)
SFAC	(0x03)	**S**et node configuration to **FAC**tory default settings
SPSA	(0x04)	**S**et **P**ower **S**aving **A**ctive; sets power management mode on/off
STXP	(0x05)	**S**ets transmission (**TX**) **P**ower of the node
SSYC	(0x06)	**S**et the PHY **SY**n**C**word of *swarm* node
BLDR	(0x07)	**B**oot**L**oa**D**e**R**, switch to bootloader.
SBIN	(n.a.)	**S**et/**S**elect **BIN**ARY mode/interface
GFWV	(0x08)	**G**et the **F**irm**W**are **V**ersion of the swarm Node
GUID	(0x09)	**G**et **U**nique **ID** of swarm Node

4.4.2. Ranging Commands

EPRI	(0x10)	**E**nable **PRI**vacy mode**;** Enables and disables automatic response to received ranging equests.
SPBL	(0x11)	**S**ets **P**rivacy **B**lack**L**ist; handles the privacy black list used to block ranging operation with specified node IDs; maximum number of entries is 19.
GPBL	(0x11)	**G**et **P**rivacy **B**lack**L**ist.
RATO	(0x12)	**RA**nge **TO**; initiates an elementary ranging cycle to another *swarm* node
BRAR	(0x13)	**B**roadcast **RA**nging **R**esults; enabled (or disabled) the broadcasting of ranging **r**esults after each successful ranging operation.
SROB	(0x14)	**S**elects **R**anging **O**peration **B**links; sets which classes of devices the node will initiate a ranging operation with upon reception of a node blink ID packet.
SRWL	(0x15)	**S**ets **R**ange **W**hite **L**ist; Handles the list of node IDs the node should range to. Maximum number of entries is 19.
GRWL	(0x15)	**G**et **R**anging **W**hite **L**ist.
ERRN	(0x16)	**E**nables (or disables) **R**anging **R**esult **N**otification
SROF	(0x17)	**S**ets **R**ange **OF**fset (in ms), this is fixed delay before ranging to autonomous devices

4.4.3. Data Communication Commands

EDAN	(0x20)	**E**nables and disables **DA**ta **N**otification
SDAT	(0x21)	**S**ends **DAT**a to node ID
GDAT	(0x27)	**G**ets received **DAT**a
BDAT	(0x22)	**B**roadcasts **DAT**a
FNIN	(0x28)	**F**ill data into **N**ode **I**D **N**otification packets.
FRAD	(0x2A)	**F**ills the **RA**nging data buffer. This data will be transmitted with the next RATO operation.
EIDN	(0x26)	**E**nables and disables Node **ID** Broadcast **N**otification

4.4.4. *swarm* radio Node Identification

EBID	(0x30)	**E**nable **B**roadcast **ID.** Enables and disables broadcast of Node ID blink packets
SBIV	(0x31)	**S**ets the **B**roadcast **I**nterval **V**alue (or blinking rate)
NCFG	(0x32)	**N**otification **C**on**F**i**G**uration is used to define which information is visible after receiving *RRN or *NIN type of notifications.

4.4.5. Medium Access Commands

SRXW	(0x40)	**S**ets reception, **RX, W**indow during which the receiver listens after its ID Broadcast.
SRXO	(0x41)	**S**ets **RX** Window **O**ccurrence; sets whether the receives listens after every blink, after every 2 blink,.. It will listen after every SRXO
SDCL	(0x42)	**S**ets the **D**evice **CL**ass of the node (1…8).
SFEC	(0x43)	**S**witches **F**orward **E**rror **C**orrection (FEC) on and off.
SDAM	(0x44)	**S**ets **DA**ta **M**ode and air communication speed (80/1 and 80/4 mode).
CSMA	(0x45)	Switches **CSMA** mode on and off and determines back-off factor for CSMA.

4.4.6. Sensors Commands

EMSS	(0x50)	**E**nables the **M**EMS **S**ensor
EBMS	(0x51)	**E**nable **B**roadcast **M**EM**S** within Node ID Blink packets.
SMRA	(0x52)	**S**ets **M**EMS' **RA**nge, i.e. +/-2,4,8 or 16g.
SMTH	(0x53)	**S**ets the **M**EMS' **TH**reshold for the slope interrupt.
SMBW	(0x54)	**S**ets **M**EMS' filter **B**and**W**idth of the MEMS.
SMSL	(0x55)	**S**ets **M**EMS' **SL**eep time of the sensor (0.5 … 1000ms)
SMDT	(0x56)	**S**ets **M**EMS' **D**ead**T**ime. Its the minimum time between two possible interrupts.
GMYA	(0x57)	**G**ets **MY A**cceleration; acceleration value of the node connected to the host
GMYT	(0x58)	**G**ets **MY T**emperature; temperature of the node connected to the host
GBAT	(0x59)	**G**ets **BAT**tery status of the node connected to the host
GPIO	(0x5A)	Configure **GPIO** pins.
SPIN	(0x5B)	**S**ets GPIO **PIN**s.
GPIN	(0x5B)	**G**ets GPIO **PIN** status.
ICFG	(0x5C)	**I**nterrupt **C**on**F**i**G**uration**.**

4.4.7. AIR Interface Commands

SSTART	(0x23)	**S**treaming **START** until timeout or stop command.
SEXTEND	(0x24)	**S**treaming **EXTEND** refresh timeout from start command..
SSTOP	(0x25)	**S**treaming **STOP**.
MRATO	(0x18)	**M**ultiple **RA**nge **TO**; request allows a remote swarm to perform several ranging requests to the same or different nodes.

4.5. Command OP_CODE Overview

CMD		ASCII		BINARY		AIR	
NAME	OP_CODE	GET	SET	GET	SET	GET	SET
snid	0x00		x	x	x	x	locked
gnid	0x00	x		x	x	x	locked
sset	0x01		x		x		x
rset	0x02		x		x		locked
gset	NA	x					
sfac	0x03		x		x		locked
spsa	0x04		x	x	x	x	x
stxp	0x05		x	x	x	x	locked
ssyc	0x06		x	x	x	x	locked
bldr	0x07		x		x		
sbin	NA		x				
gfwv	0x08	x		x		x	
guid	0x09	x		x		x	
epri	0x10		x	x	x	x	x
spbl	0x11		x	x	x	x	x
gpbl	0x11	x		x	x	x	x
rato	0x12		x		x		
brar	0x13		x	x	x	x	x
srob	0x14		x	x	x	x	x
srwl	0x15		x	x	x	x	x
grwl	0x15	x		x	x	x	x
errn	0x16		x	x	x	x	x
srof	0x17		x	x	x	x	x
mrato	0x18						x
edan	0x20		x	x	x	x	x
sdat	0x21		x		x		
bdat	0x22		x		x		
sstart	0x23						x
sextend	0x24						x
sstop	0x25						x
gdat	0x27	x		x			
fnin	0x28		x	x	x	x	x
frad	0x2a		x	x	x	x	x
eidn	0x26		x	x	x	x	x
ebid	0x30		x	x	x	x	locked
sbiv	0x31		x	x	x	x	x
ncfg	0x32		x	x	x	x	x
srxw	0x40		x	x	x	x	locked
srxo	0x41		x	x	x	x	locked
sdcl	0x42		x	x	x	x	x
sfec	0x43		x	x	x	x	locked
sdam	0x44		x	x	x	x	locked
csma	0x45		x	x	x	x	locked
emss	0x50		x	x	x	x	x
ebms	0x51		x	x	x	x	x
smra	0x52		x	x	x	x	x
smth	0x53		x	x	x	x	x
smbw	0x54		x	x	x	x	x

smsl	0x55		x	x	x	x	x
smdt	0x56		x	x	x	x	x
gmya	0x57		x	x	x	x	
gmyt	0x58		x	x	x	x	
gbat	0x59		x	x	x	x	
gpio	0x5a		x	x	x	x	x
spin	0x5b		x	x	x	x	x
gpin	0x5b	x		x	x	x	x
icfg	0x5c		x	x	x	x	x

In the table above an overview about the get or set capability of each command is given. To avoid changing some command settings by mistake and losing access to the remote node, a locking mechanism is introduced. “locked” stands for not changeable without unlocking first. Locked commands may not be changed because it would disrupt air communication. If you really need to change locked settings via the AIR interface, please contact nanotron support and they will provide you with the necessary information.

5. UART API Command Set

In order to interact with the embedded ranging hardware platform the following API command set is implemented:

swarm radio Setup Commands

SNID <ID>:

Sets the Node ID of *swarm* node to **<ID>**

Parameter Description:

<ID>	000000000000 is not a valid address but resets the original unique Node ID which was set during device production if supported by µC otherwise: 000000000001
	Range: 000000000000 … FFFFFFFFFFFE

Return Value Description: **<ID>**

<ID>	configured 6 byte Node ID of *swarm* node if set ID = 000000000000 then default ID is returned
	Range: 000000000000 … FFFFFFFFFFFE

ASCII

Parameter (Format):

<ID> 12 bytes(hex)

```
Example:      SNID 0000BF260468
```

Return value (Format):

<ID> 12 bytes(hex)

```
Example:      =0000BF260468
```

BINARY

Type: SET

Parameter (Format):

<ID> 6 bytes (uint8_t [6])

```
Example:      SET SNID 0000b6f31103
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x55	0x00	0x0000b6f31103	0x0a	0xf2

Return value (Format):

<ID> 6 bytes (uint8_t [6])

```
Example:      RESP SNID 0000b6f31103
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x57	0x00	0x0000b6f31103	0x8b	0x2b

GNID <void>:

Readback of configured Node ID of node connected to host

Parameter description:	void
Return value description:	**=<ID>**
<ID>	configured 6 byte Node ID of *swarm* node
Range:	000000000001 … FFFFFFFFFFFE

ASCII

Parameter (Format):

<ID> 12 bytes(hex)

`Example:      GNID`

Return value (Format):

<ID> 12 bytes (hex)

`Example:      =0000BF260468`

BINARY

Type: GET

Parameter (Format):

none

`Example:      GET GNID`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x00	0x86	0xd4

Return value (Format):

<ID> 6 bytes (uint8_t [6])

`Example:      RESP GNID 0000b6f31103`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x56	0x00	0x0000b6f31103	0x4a	0xe7

SSET<void>:

Saves all settings including Node ID permanently to EEPROM. This must be done after changing settings if they should persist through a power cycle or switch off state.

Parameter description:

<void>

Return value description: **=<ErrorCode>**

<ErrorCode>		Result of saving operation
	Range:	0 … 1
		0 = Saving of all parameters successfully verified
		1 = Saving of parameters not successful; verification failed

ASCII

```
Example:      SSET
```

Return value (Format):

<ErrorCode> 1 byte (dec)

```
Example:      =0
```

BINARY

Type: SET

```
Example:      SET SSET
```

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x55	0x01	0x46	0x84

Return value (Format):

<ErrorCode> 1 byte (uint8_t)

```
Example:      RESP SSET 0
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x01	0x00	0xa5	0xce

RSET<void>:

Restores all parameter settings from EEPROM

Parameter description:

<void>

Return value description: **=<ErrorCode>**

<ErrorCode>	Result of restoring operation
Range:	0 … 1
	0 = Restoring of all parameters successful
	1 = Restoring parameters from EEPROM failed

ASCII

```
Example:      RSET
```

Return value (Format):

<ErrorCode> 1 byte (dec)

```
Example:      =0
```

BINARY

Type: SET

```
Example:      SET RSET
```

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x55	0x02	0x06	0x85

Return value:

<ErrorCode> 1 byte (uint8_t)

```
Example:      RESP  RSET
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x02	0x00	0xa5	0x3e

GSET<void>:

Reads current device configuration. First line is the number of following lines. All others state the name of parameter separated with ':' and value. The value depends on parameter.

Parameter description:

<void>

Return value description: **#<NumLines>**\r\n**<ParameterName>**:**<Value>**\r\n ..

<NumLines>	Number of lines after this line
Range:	000 … 255
<ParameterName>	Name of following parameter value.
<Value>	Depends on parameter.

ASCII

```
Example:      GSET
```

Return value (Format):

<NumLines>	4 bytes (dec), first byte fixed „#“
<ParameterName>	ASCII
<Value>	depends on parameter

```
Example:      #029
              BRAR:1
              CSMA:0,0,0
              EBID:1
              EBMS:1
              EDAN:1
              EIDN:0
              EMSS:1
              EPRI:0
              ERRN:1
              GPIO:0F,0,3,0,1
              ICFG:0
              NCFG:0004
              SBIV:30000
              SDAM:1
              SDCL:1
              SFEC:0
              SMBW:6
              SMDT:01000
              SMRA:2
              SMSL:01
              SMTH:30
              SNID:XXXXXXXXXXXX
              SPSA:0
              SROB:01
              SROF:000
              SRXO:001
              SRXW:00010
              SSYC:1
              STXP:63
```

BINARY

Type: Not supported, use the get command for each setting.

SFAC<void>:

Resets device configuration to factory settings. Default configuration is described in chapter 8 Default Settings.

Note: Settings not saved to EEPROM until SSET command is issued.

Parameter description:

<void>

Return value description: **=<ErrorCode>**

<ErrorCode> Result of configuration.

Range: 0…1

0 = Successful.

1 = Not successful.

ASCII

`Example:      SFAC`

Return value (Format):

<ErrorCode> 1 byte (dec)

`Example:      =0`

BINARY

Type: SET

`Example:      SET SFAC`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x55	0x03	0xc7	0x45

Return value (Format):

<ErrorCode> 1 byte (uint8_t)

`Example:       RESP SFAC 0`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x03	0x00	0xa4	0xae

SPSA <Type>

Sets power management mode for *swarm* radio

Parameter description:

<Type>

Range:	0 … 1
	0 = Power saving off, device is continuously receiving
	1 = Power saving on, wake up on any interrupt

Return value description: **=<Type>**

ASCII

Parameter (Format):

<Type> 1 byte (dec)

```
Example:      SPSA 1
```

Return value (Format):

<Type> 1 byte (dec)

```
Example:      =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Type> 1 byte (uint8_t)

```
Example:      SET SPSA 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x04	0x01	0xc6	0x9e

Return value (Format):

<Type> 1 byte (uint8_t)

```
Example:      RESP SPSA 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x04	0x01	0x67	0x5e

STXP <Power>:

Sets the node’s transmission power (0=minimum … 63=maximum).

Parameter description:

<Power>	transmission power Range: 00 … 63

Return value description: **=<Power>**

ASCII

Parameter (Format):

<**Power**>	2 bytes (dec)

```
Example:      STXP 63
```

Return value (Format):

<**Power**>	2 byte (dec)

```
Example:      =63
```

BINARY

Type: GET/SET

Parameter (Format):

<**Power**>	1 byte (uint8_t)

```
Example:      SET STXP 63
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x05	0x3F	0x46	0xde

Return value (Format):

<**Power**>	1 byte (uint8_t)

```
Example:      RESP STXP 63
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x05	0x3F	0xe7	0x1e

SSYC <SyncWord>:

Sets the node's PHY syncword (0 … 8). The node will only listen to messages with its same syncword.

Parameter description:

<SyncWord>

Range: 0 … 8

Return value description: **=<SyncWord>**

ASCII

Parameter (Format):

<SyncWord> 1 byte (dec)

```
Example:      SSYC 7
```

Return value (Format):

<SyncWord> 1 byte (dec)

```
Example:      =7
```

BINARY

Type: GET/SET

Parameter (Format):

<SyncWord> 1 byte (uint8_t)

```
Example:      SET SSYC 7
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x06	0x07	0x47	0xfc

Return value (Format):

<SyncWord> 1 byte (uint8_t)

```
Example:      RESP SSYC 7
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x06	0x07	0xe6	0x3c

BLDR <void>:

Restarts device and start bootloader.

Parameter description:

<void>

Return value description: =0

ASCII

`Example:      BLDR`

Return value (Format):

<> 1 byte (dec)

`Example:      =0`

BINARY

Type: SET

`Example:      SET BLDR`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x55	0x07	0xc6	0x86

Return value (Format):

NO PARAMETER

`Example:      RESP BLDR`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x57	0x07	0xc7	0xe6

SBIN<void>:

The command switchs the interface from ASCII to BINARY. The selected interface isn't stored. When the module resets, it is again in ASCII mode. If you switch the mode from ASCII to binary, still the "OK" response is transmitted by commanded node. To indicate for the host that the node has understood the command yet. A subsequent confirm is not necessary. To return back to ASCII you have to trigger a RESET or a BREAK command, which returns the module into the bootloader. Then you get back to ASCII Mode. The Command is not supported for BINARY Interface

ASCII

`Example:` `SBIN`

Return value (Format):

<> 1 byte (dec)

`Example:` **`=0`**

BINARY

Type: NOT SUPPORTED

GFWV<void>:

Returns the firmware version of the swarm Node.

Parameter description:

<void>

Return values: **=<FirmwareVersion>**

<FirmwareVersion>	Firmware version of the swarm node. The version consists of 5 parts. Major number, minor number, sub-minor, release candidate number, number of changes after last tag. Example: ver2.0.3-rc0-98 is translated to 0x02003062.

Range: major[0 … 0xFF]

minor[0 ... 0xFF]

sub-minor[0 … 0xF]

release candidate[0 … 0xF] (F stands for final release version)

changes after last tag [0 .. 0xFF]

ASCII

`Example:` `GFWV`

Return value (Format):

<FirmwareVersion> 8 byte (hex)

`Example:` `=02003062`

BINARY

Type: GET

`Example:` `GET GFWV`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x08	0x87	0x12

Return value (Format):

<FirmwareVersion> 4 byte (uint8_t)

`Example:` `RESP GFWV`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x06	0x56	0x08	0x02003500	0xda	0x7b

GUID <void>:

Description: Read the unique ID of µC.

Return value description: **=<UID>**

<UID>

Range: 0 … FFFFFFFFFFFFFFFFFFFFFFFF

ASCII

`Example:` `GUID`

Return value (Format):

<UID> 24 byte (hex)

`Example:` `=353347083138373128003200`

BINARY

Type: GET

Parameter (Format):

<void>

`Example:` `GET UID`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x0e	0x54	0x09	0x46	0xd2

Return value (Format):

<UID> 12 byte (uint8_t)

`Example:` `RESP GUID 353347083138373128003200`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0e	0x56	0x09	0x353347083138373128003200	0x5a	0xb8

Ranging Commands

EPRI <Enable>:

Enable Privacy mode: Enables and disables response to a received ranging request.

Parameter description:

<Enable>

Range:	0 … 1
	0 = Node will respond to ranging requests
	1 = Node will not respond to ranging requests

Return value description **=<Enable>**

ASCII

Parameter (Format):

<Enable> 1 byte (dec)

```
Example:      EPRI 1
```

Return value (Format):

<Enable> 1 byte (dec)

```
Example:      =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Enable> 1 byte (uint8_t)

```
Example:      SET EPRI 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x10	0x01	0xc9	0x9e

Return value (Format):

<Enable> 1 byte (uint8_t)

```
Example:      RESP EPRI 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x10	0x01	0x68	0x5e

SPBL <Option=0> :

SPBL <Option=1/2> <ID>:

Sets privacy blacklist: Handles the privacy black list used to block ranging operation with specified node IDs; maximum number of entries is 19.

Parameter description:

<Option>

Range: 0 … 2

0 = Clear black list

1 = Add ID to the black list

2 = Remove ID from the black list

<ID> 6 byte Node ID of ranging partner node

Range: 000000000001 … FFFFFFFFFFFE

Return value description **=<ErrorCode>**

<ErrorCode> **indicating status of operation**

Format: 1 byte (dec)

Range: 0…2

0 = success value stored/removed or list cleared

1 = list full, not possible to add more ID

2 = ID to be removed is not in the list

ASCII

Parameter (Format):

<Option> 1 byte (dec)

<ID> 12 bytes (hex)

```
Example:      SPBL 1 0000BF260468
```

Return value (Format):

<ErrorCode> 1 byte (dec)

```
Example:      =1
```

BINARY

Type: SET

Parameter (Format):

<Option>	1 byte (uint8_t)
<ID>	6 bytes (uint8_t[6])

`Example:      SET SPBL 1 0000bf260468`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x09	0x55	0x11	0x010000BF260468	0x9e	0x98

Return value (Format):

<ErrorCode>	1 byte (uint8_t)

`Example:      RESP SPBL 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x11	0x00	0xa8	0x0e

GPBL <void>:

Gets privacy blacklist.

Parameter description:

<void>

Return value description: **<NumEntries>**\r\n[**<NodeID>**]*

<NumEntries> **Number of lines after this line**

Range: 0 … 19

<NodeID>

Range: 000000000001 … FFFFFFFFFFFE

ASCII

`Example:      GPBL`

Return value (Format):

<NumEntries>	4 bytes (dec), first byte fixed „#“
<NodeID>	12 bytes (hex)

```
Example:      #003
        DDF451534C23
        134683567ABC
        33A441FFB311
```

BINARY

Type: GET

`Example:      GET GPBL`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x11	0x46	0xd8

Return value (Format):

<NumEntries>	1 byte (uint8_t)
<NodeID>	6 bytes (uint8_t[6])

```
Example:     RESP GPBL 3 DDF451534C23 134683567ABC 33A441FFB311
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x15	0x56	0x11	0x03DDF451534C23134683567ABC 33A441FFB311	0x3a	0xc7

RATO <Option=0> <ID>:

RATO <Option=1> <ID> <TimeOut>:

Range to: Initiates an elementary ranging cycle to node with node ID **<ID>**

Parameter description:

<Option> defines if ranging starts synchronous or asynchronous

Range: 0 … 1

0 = Immediate start ranging operation

1 = Wait for node <ID> blink before starting ranging operation

<ID> 6 byte Node ID of ranging partner node

Range: 000000000001 … FFFFFFFFFFFE

<TimeOut> maximum time in ms to wait for the node <ID> blink.

Range: 0 … 65000

Return Value description:

Return value with	**<Option=0>:**	**=<ErrorCode>,<RangingResult>,<RSSI>**
Return value with	**<Option=1>:**	**=<Error>**

<ErrorCode> indicating status of ranging operation

Range: 0 … 6

0 = success ranging result valid

1 = ranging to own ID

2 = ID out of range, no ACK

3 = ranging unsuccessful, ACK OK, then timeout

5 = <TimeOut> has been reached

6 = medium blocked (CSMA give up)

<RangingResult> returning the measured ranging distance in centimeter. Range depends on mode.

Range(ASCII): 000000 … 999999 ranging distance in centimeter [cm]

Range(BINARY): 0...999999 ranging distance in centimeter [cm]

<RSSI> RSSI value of remote node

Range: -128 .. -35

<Error> Indicate status of operation

Range: 0 … 1

0 = success, request accepted

1 = error, ranging to own ID is not allowed

ASCII

Parameter (Format):

<Option>	1 bytes (dec)
<ID>	12 bytes (hex)
<TimeOut>	5 bytes (dec)

```
Example:     RATO 1 0000BF260468 1000
```

Return Values (Format):

<ErrorCode>	1 byte (dec)
<RangingResult>	6 bytes (dec) in [cm]
<RSSI>	4 byte (dec)
<Error>:	1 byte (dec)

```
Example:     0, 001843,-56
```

BINARY

Types: SET

Parameter (Format):

<Option>	1 byte (uint8_t)
<ID>	6 bytes (uint8_t[6])
<TimeOut>	2 bytes (uint16_t)

Return Values (Format with **Option=0**):

<ErrorCode>	1 byte (uint8_t)
<RangingResult>	4 bytes (uint32_t) in [cm]
<RSSI>	1 byte (int8_t)

Return Values (Format with **Option=1**):

<Error>	1 byte (uint8_t)

```
Example:     SET RATO 1 0000BF260468 1000 (asynchronous)
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0B	0x55	0x12	0x010000BF26046803E8	0x60	0xbc

```
1. Response: Return Value <Error=0> (accept request)
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x12	0x00	0xa8	0xfe

```
2. Response: *RRN (ranging result notification)
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x16	0x61	0x62	0x0000000000020000bf26046800000000940004cd	0x8d	0xd8

```
Example:     SET RATO 0 0000BF260468
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x09	0x55	0x12	0x000000bf260468	0xce	0x4d

```
Response:    RESP RATO 0
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x57	0x12	0x0000000045cb	0x50	0x07

BRAR <Enable>:

Broadcast ranging results: Enables/Disables the broadcast transmission of ranging results aftereach successful ranging.

Parameter description:

<Enable>

Range: 0 … 1

0 = broadcast disabled

1 = broadcast enabled

Return value description: **=<Enable>**

ASCII

Parameter (Format):

<Enable> 1 byte (dec)

```
Example:     BRAR 0
```

Return value (Format):

<Enable> 1 byte (dec)

```
Example:     =0
```

BINARY:

Type: GET/SET

Parameter (Format):

<Enable> 1 byte (uint8_t)

```
Example:      SET BRAR 0
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x13	0x00	0x08	0xae

Return value (Format):

<Enable> 1 byte (uint8_t)

```
Example:      RESP BRAR 0
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x13	0x00	0xa9	0x6e

SROB <ClassMask>:

Sets range on broadcast: Selects if and to which class of devices <ClassMask> the Node will initiate a ranging operation upon reception of a node blink ID.

Parameter description:

<ClassMask> Device classes can be combined by performing a logical OR of ClassMask, e.g. ClassMask=0xA2 (=bin 1010 0010). The Node will initiate a ranging operation on broadcast to classes 2,6,8.

Range: 00 … FF

00	= (=bin 0000 0000) Range on Broadcast disabled
01	= (=bin 0000 0001) Range on Broadcast to device class 1
02	= (=bin 0000 0010) Range on Broadcast to device class 2
04	= (=bin 0000 0100) Range on Broadcast to device class 3
08	= (=bin 0000 1000) Range on Broadcast to device class 4
10	= (=bin 0001 0000) Range on Broadcast to device class 5
20	= (=bin 0010 0000) Range on Broadcast to device class 6
40	= (=bin 0100 0000) Range on Broadcast to device class 7
80	= (=bin 1000 0000) Range on Broadcast to device class 8

Return value: **=<ClassMask>**

ASCII

Parameter (Format):

<ClassMask> 2 bytes (hex)

`Example:      SROB A2`

Return value (Format):

<ClassMask> 2 bytes (hex)

`Example:      =A2`

BINARY:

Type: GET/SET

Parameter (Format):

<ClassMask> 1 byte (uint8_t)

`Example:      SET SROB A2`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x14	0xa2	0x8b	0x27

Return value (Format):

<ClassMask> 1 byte (uint8_t)

`Example:      RESP SROB A2`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x14	0xa2	0x2a	0xe7

SRWL <Option> <ID>:

Sets range white list: Handles the list of node IDs the node should range to. Maximum number of entries is 19.

Parameter description:

<Option>

Range: 0 … 2

0 = Clear list

1 = Add ID to the list

2 = Remove ID from the list

<ID> 6 byte Node ID of ranging partner node

Range: 000000000001 … FFFFFFFFFFFE

Return value description: **=<Error>**

<Error> Indicating status of operation

Range: 0…2

0 = success, value stored/removed or list cleared

1 = list full, not possible to add more ID

2 = ID to be removed is not in the list

ASCII

Parameter (Format):

<Option>	1 byte (dec)
<ID>	12 bytes (hex)

`Example:      SRWL 1 0000BF260468`

Return value (Format):

<Error> 1 byte (dec)

`Example:      =0`

BINARY

Type: SET

Parameter (Format):

<Option>	1 byte (uint8_t)
<ID>	6 bytes (uint8_t[6])

`Example:      SET SRWL 1 0000BF260468`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x09	0x55	0x15	0x010000BF260468	0x9f	0x6b

Return value (Format):

<Error> 1 byte (uint8_t)

`Example:      RESP SRWL 0`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x15	0x00	0xaa	0xce

GRWL <void>:

Gets ranging whitelist.

Parameter description:

<void>

Return value description: **<NumEntries>[<NodeID>]***

<NumEntries> Number of lines after this line

Range: 0 … 19

<NodeID>

Range: 000000000001 … FFFFFFFFFFFE

ASCII

`Example:      GRWL`

Return value (Format):

<NumEntries>	4 bytes (dec), first byte fixed „#“
<NodeID>	12 bytes (hex)

```
Example:      #003
              DDF451534C23
              134683567ABC
              33A441FFB311
```

BINARY

Type: GET

```
Example:      GRWL
```

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x15	0x47	0x1b

Note: CRC_HIGH is 0x1b and must be escaped before transmitting (0x1b43).

Return value (Format):

<NumEntries>	1 byte (uint8_t)
<NodeID>	6 bytes (uint8_t[6])

```
Example:      RESP GRWL 3 DDF451534C23 134683567ABC 33A441FFB311
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x15	0x56	0x15	0x03DDF451534C23134683567ABC 33A441FFB311	0x43	0x52

ERRN <Notify>:

Enables and disables ranging result notification

Parameter description:

<Notify>

Range: 0 … 1

0 = Node will not trigger host when data packet has been received

1 = Node will trigger host when data packet has been received

Return value description: **=<Notify>**

ASCII

Parameter (Format):

<Notify> 1 byte (dec)

```
Example:      ERRN 1
```

Return value (Format):

<Notify> 1 byte (dec)

```
Example:      =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Notify> 1 byte (uint8_t)

```
Example:      SET ERRN 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x16	0x01	0xca	0x3e

Return value (Format):

<Notify> 1 byte (uint8_t)

```
Example:      RESP  ERRN 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x16	0x01	0x6b	0xfe

SROF <Duration>

Sets ranging offset: fixed delay in ms before starting automatic ranging.

Parameter description:

<Duration>

Range: 0 … 65000

Return value description **=<Duration>**

ASCII

Parameter (Format):

<Duration> 5 bytes (dec)

```
Example:      SROF 00100
```

Return value (Format):

<Duration> 5 bytes (dec)

```
Example:      =00100
```

BINARY

Type: GET/SET

Parameter (Format):

<Duration> 2 bytes (uint16_t)

`Example:      SET SROF 100`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x17	0x0064	0x5a	0x2c

Return value (Format):

<Duration> 2 byte (uint16_t)

`Example:      RESP SROF 100`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x17	0x0064	0x5b	0x94

Data Communication Commands

EDAN <Notify>:

Enables and disables data notification

Parameter description:

<Notify>

Range: 0 … 1

0 = Node will not trigger host when data packet has been received

1 = Node will trigger host when data packet has been received

Return value description **=<Notify>**

ASCII

Parameter (Format):

<Notify> 1 byte (dec)

`Example:      EDAN 1`

Return value (Format):

<Notify> 1 byte (dec)

`Example: =1`

BINARY:

Type: GET/SET

Parameter (Format):

<Notify> 1 byte (uint8_t)

`Example: SET EDAN 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x20	0x01	0xdd	0x9e

Return value (Format):

<Notify> 1 byte (uint8_t)

`Example: RESP EDAN 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x20	0x01	0x7c	0x5e

SDAT <Option=0> <ID> <Len> <Data>:

SDAT <Option=1> <ID> <Len> <Data> <TimeOut>:

Sends <Data> of length <Len> to node <ID>.Depending on <OPTION> it transmits immediately or after the node <ID> blink.

Parameter description:

<Option>

Range: 0 … 1

0 = Immediate transmission

1 = Wait for node **<ID>** blink before transmitting

Note: The receiver must be enabled to receive node ID blinks. Disable power saving (SPSA 0) to receive asynchronous node ID blinks.

<ID> 6 byte Node ID of ranging partner node.

Note: If ID is all FF than it send always after receiving a node ID notification until **<TimeOut>**

Range: 000000000001 … FFFFFFFFFFFE

<Len> length of payload in bytes (hex)

Range: 01 … 80

<Data> payload to be transmitted

Range: 00 … FF

<TimeOut> maximum time in ms to wait for the node **<ID>** blink.

Range: 0 … 65000

Return value description: **<Option=0>**: **=<ErrorCode>**

<Option=1>: **=<PayloadID>**

<ErrorCode> Status of transmission.

Range: 0 ... 1

0 = Successfully transmitted

1 = Transmission failed

2 = Overload, try again later

<PayloadID> ID of <Data>, used to identify the asynchronous return value (*SDAT)

Range: 00000000 … FFFFFFFF

Note: After issuing a SDAT command and asynchronous response will be generated when the transmission is processed. The response is described in chapter 7.

ASCII

Parameter (Format):

<Option>	1 byte (dec)
<ID>	12 bytes (hex)
<Len>	2 bytes (hex) = times 2 bytes of payload
<Data>	2 bytes (hex)
<TimeOut>	5 bytes (dec)

```
Example:      SDAT 1 1F318052001A 02 FA13 1000
```

Return value (Format):

<ErrorCode>	1 byte (hex)
<PayloadID>	8 bytes (hex)

```
Example:      =A4A865F8
```

BINARY

Type: GET/SET

Parameter (Format):

<Option>	1 byte (uint8_t)
<ID>	6 bytes (uint8_t[6])
<Len>	1 byte (uint8_t)
<Data>	<Len> bytes (uint8_t[Len])
<TimeOut>	2 bytes (uint16_t)

```
Example:      SET SDAT 0 1F318052001A 02 FA13
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0C	0x55	0x21	0x001F318052001A02FA13	0x6b	0xa4

Return value (Format):

<ErrorCode> 1 byte (uint8_t)

`Example: RESP SDAT <ErrorCode=1>`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x21	0x01	0x7d	0xce

`Example: SET SDAT 1 1F318052001A 02 FA13 1000`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0c	0x55	0x21	0x011F318052001A02FA1303e8	0x51	0xaf3

Return value (Format):

<PayloadID> 4 bytes (uint8_t[4])

`Example: RESP SDAT <PayloadID=0xa4a865f8>`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x06	0x57	0x21	0xa4a865f8	0x18	0x86

GDAT <void>:

Gets data: Reads out transmitted data

Parameter description:

<void>

Return values: **=<Number of bytes>,<ID>,<Payload>**

<Number of bytes>	returns the number of bytes in pending message
	Number of bytes = 00:no pending message available
	Number of bytes = 01…80 number of bytes in message

Range: 00…80

<ID> returns ID of node which sent message

Range: 000000000001 … FFFFFFFFFFFF

<Payload> payload received

Range: 00 … FF

ASCII

`Example:      GDAT`

Return value (Format):

<Number of bytes>	2 bytes (hex)
<ID>	12 Bytes (hex)
<Payload>	2 bytes (hex)

`Example:      =02,1F318052001A,FA13`

BINARY:

Type: GET

`Example:      GET GDAT`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x27	0xc6	0xce

Return value (Format):

<Number of bytes>	1 byte (uint8_t)
<ID>	6 bytes (uint8_t[6])
<Payload>	**<Number of bytes>** bytes (uint8_t[**<Number of bytes>**])

Note: If <Number of bytes> is 0, because no data is available, than <ID> and <Payload> do not exist in return message.

`Example:      RESP  GDAT  02 1F318052001A FA13`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0C	0x56	0x27	0x021F318052001AFA13	0xe8	0x5f

`Example:      RESP  GDAT  no data present`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x56	0x27	0x00	0xee	0xe6

BDAT <Option=0> <Len> <Data>:

BDAT <Option=1> <Len> <Data> <TimeOut>:

BDAT <Option=2>:

Broadcasts **<Data>** of length **<Len>** to all nodes

Parameter description:

<Option>

Range: 0 … 2

0 = Immediate broadcast transmission

1 = Send the packet after any node ID blink.

2 = Cancel broadcasting

<Len> length of payload in bytes (hex)

Range: 01 … 70

<Data> payload to be transmitted

Range: 00 … FF

<TimeOut>

Range: 0 … 65000

0 = timeout disabled, broadcast will always occur.

> 0 = time in ms during which the node transmits after a node ID blink.

Return value with **<Option=0>**: **=<ErrorCode>**

<Option=1>: **=<PayloadID>**

<Option=2>: **=0**

<ErrorCode> Status of transmission.

Format: 1 byte (dec)

Range: 0 ... 1

0 = Successfully transmitted

1 = Transmission failed

<PayloadID> ID of <Data>, used to identify the asynchronous return value (*BDAT)

Format: 8 byte (hex)

Range: 00000000 … FFFFFFFF

Note: After issuing a BDAT command and asynchronous response will be generated when the transmission is processed. The response is described in chapter 7.

ASCII

Parameter (Format):

<Option>	1 bytes (dec)
<Len>	2 bytes (hex)
<Data>	2 bytes (hex)
<TimeOut>	5 bytes (dec)

```
Example:    BDAT 0 2 FA13
```

Return value (Format):

<ErrorCode>	1 byte (dec)
<PayloadID>	8 bytes (hex)

```
Example:    =0
```

BINARY:

Type: SET

Parameter (Format):

<Option>	1 byte (uint8_t)
<Len>	1 byte (uint8_t)

<Data> <Len> bytes (uint8_t[Len])

<TimeOut> 2 bytes (uint16_t)

`Example:      SET BDAT 0 2 FA13`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x06	0x55	0x22	0x0002FA13	0x77	0xcb

Return value (Format):

<ErrorCode> 1 byte (uint8_t)

<PayloadID> 4 bytes (uint8_t[4])

`Example:      RESP BDAT 0`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x22	0x00	0xbc	0xfe

FRAD <Len> <Data>:

Fills the ranging data buffer with <Data> of length <Len>. The ranging data <Data> is contained within every following ranging packet itself. Each receiving swarm will generate a *DNO. This cause a lot of traffic for each swarm device.

Parameter description:

<Len>		length of ranging data payload in bytes (hex)
	Range:	00 … 74
		0 = delete data
<Data>		payload to be transmitted
	Range:	00 … FF

Return value description **=<ErrorCode>**

<ErrorCode> Status on ranging data buffer fill operation

Range: 0 … 1

0 = successful

1 = not successful

ASCII

Parameter (Format):

<Len> 2 bytes (hex)

<Data> <Len> * 2 bytes (hex)

`Example: FRAD 0A FA13192F680426AE2345`

Return value (Format):

<ErrorCode> 1 byte (dec)

`Example: =1`

BINARY

Type: GET/SET

Parameter (Format):

<Len> 1 byte (uint8_t)

<Data> <Len> byte (uint8_t[Len])

`Example: SET FRAD 0A FA13192F680426AE2345`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0d	0x55	0x2a	0x0AFA13192F680426AE2345	0xef	0xf2

Return value (Format for SET response):

<ErrorCode> 1 byte (uint8_t)

Return value (Format for GET response):

<Len> 1 byte (uint8_t)

<Data> <Len> byte (uint8_t[Len])

`Example: RESP FRAD <ErrorCode=0>`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x2a	0x00	0xbb	0x3e

EIDN <Notify>:

Enables and disables Node ID broadcast notification

Parameter description:

<Notify>

Range: 0 … 1

0 = Node will not trigger host when Node ID Broadcast has been received

1 = Node will trigger host when Node ID Broadcast has been received

Return value description **=<Notify>**

<Notify> returning parameter which has been set

Range: 0 …1

ASCII

Parameter (Format):

<Notify> 1 byte (dec)

```
Example:      EIDN 1
```

Return value (Format):

<Notify> 1 byte (dec)

```
Example:      =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Notify> 1 byte (uint8_t)

```
Example:      SET EIDN 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x26	0x01	0xde	0x3e

Return value (Format):

<Notify> 1 byte (uint8_t)

```
Example:      RESP EDIN 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x26	0x01	0x7f	0xfe

Note: CRC_LOW is 0x7f and must be escaped before transmitting (0x1b53).

FNIN <Len=0>:

FNIN <Len>0> <Data>:

Fills the data buffer of each node ID notification with <Data> of length <Len>. This data will be transmitted with every node ID notification until deleted by host application. The data is not visible for swarm devices, because no *DNO is generated. This command is used to provide data to other products such as TDOA RTLS.

Parameter description:

<Len> length of ranging data payload in bytes (hex)

Range: 00 … 5B

0 = delete data

<Data> payload to be transmitted

Range: 00 … FF

Return value description **=<ErrorCode>**

<ErrorCode>	Status on ranging data buffer fill operation
Range:	0 … 1
	0 = successful
	1 = not successful

ASCII

Parameter (Format):

<Len>	2 bytes (hex)
<Data>	<Len> * 2 bytes (hex)

`Example:      FNIN 0A FA13192F680426AE2345`

Return value (Format):

<ErrorCode>	1 byte (dec)

`Example:      =0`

BINARY

Type: GET/SET

Parameter (Format):

<Len>	1 byte (uint8_t)
<Data>	<Len> bytes (uint8_t[Len])

`Example:      SET FNIN 0A FA13192F680426AE2345`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0C	0x55	0x28	0x0AFA13192F680426AE2345	0xba	0x5e

Return value (Format for SET response):

<ErrorCode>	1 byte (uint8_t)

Return value (Format for GET response):

<Len>	1 byte (uint8_t)
<Data>	<Len> byte (uint8_t[Len])

`Example:      RESP  FNIN 0`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x28	0x00	0xba	0x5e

swarm radio node Identification

EBID <Enable>:

Enable/Disable broadcast node ID: Enables/disables broadcast of Node ID blink packets.

Parameter description:

<Enable>

Range: 0 … 1

0 = Broadcast of Node ID blink packets disabled

1 = Broadcast of Node ID blink packets enabled

Return value description **=<Enable>**

ASCII

Parameter (Format):

<Enable>	1 byte (dec)

```
Example:      EBID 1
```

Return value (Format):

<Enable>	1 byte (dec)

```
Example:      =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Enable>	1 byte (uint8_t)

```
Example:      SET EBID 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x30	0x01	0xd0	0x5e

Return value (Format):

<Enable>	1 byte (uint8_t)

```
Example:      RESP EBID 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x30	0x01	0x71	0x9e

SBIV <Time>:

Sets the broadcast interval in which the Node ID will be sent

Parameter description:

<Time> Blink interval in milliseconds [ms]

Range: 00000 … 65000

Note: AIR interface SBIV is blocked, if someone tries to set an interval less than 10 ms.

Return value description **=<Time>**

ASCII

Parameter (Format):

<Time> 5 bytes (dec)

`Example:      SBIV 10000`

Return value (Format):

<Time> 5 bytes (dec)

`Example:      =10000`

BINARY

Type: GET/SET

Parameter (Format):

<Time> 2 bytes (uint16_t)

`Example:      SET SBIV 10000`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x31	0x2710	0xa0	0x30

Return value (Format):

<Time> 2 bytes (uint16_t)

`Example:      RESP SBIV 10000`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x31	0x2710	0xa1	0x88

NCFG <Mask>

<Mask> sets visible information in ranging result notifications and node ID notifications. This allows control over information displayed each time a “*RRN:” or “*NIN:” is generated. If information for a desired field is missing, it is filled up with “?”.

Parameter description:

<Mask>

Range: 0...FFFF

Bit 0 = Device Class

Format ASCII:	1 byte (dec)
Format BINARY:	1 byte (uint8_t)
Range:	1 .. 8
Description:	Device class of remote node

Bit 1 = MEMS values consists **of** x,y,z acceleration.

<XACC>

Format ASCII:	6 bytes (dec), signed INT16, first byte “+” or “-”
Format BINARY:	2 bytes (int16_t)
Range:	-32768 … +32767
Description:	Acceleration value.

<YACC> See <XACC>

<ZACC> See <XACC>

Bit 2 = RSSI values

Format ASCII:	4 byte (dec)
Format BINARY:	1 byte (int8_t)
Range:	-128 .. -35
Description:	RSSI value of remote node.

Bit 3 = TEMP values

Format ASCII:	3 byte (dec), frist byte “+” or “-”
Format BINARY:	1 byte (int8_t)
Range:	-99 .. +99
Description:	Temperature value of remote node.

Bit 4 = Power saving mode

Format ASCII:	1 byte (dec)
Format BINARY:	1 byte (uint8_t)
Range:	0 .. 2
	0 = receiver of remote node is always active
	1 = remote node is in sleep mode
	2 = remote node is in deep sleep
Description:	Power saving mode of remote node.

Bit 5 = Battery level

Format ASCII:	3 bytes (dec)
Format BINARY:	1 byte (uint8_t)
Range:	0 … 255
Description:	Node’s own battery level is in dV.
	(Example: 32 = 3.2V)

Bit 6 = GPIO Status

Format ASCII:	2 bytes (hex)
Format BINARY:	1 byte (uint8_t)
Range:	00 … FF
	Bit 0 = PA0 = DIO_0
	Bit 1 = PA2 = DIO_1

	Bit 2 = PA3 = DIO_2
	Bit 3 = PA8 = DIO_3
Description:	GPIO Pin status.

Bit 7 = Wakeup reason

Format ASCII	2 bytes (hex)
Format BINARY:	1 byte (uint8_t)
Range:	00 … FF
	Bit 0 = DIO_0
	Bit 1 = DIO_1
	Bit 2 = DIO_2
	Bit 3 = DIO_3
	Bit 4 = MEMS
Description:	Wake up reason of remote device. If no bit is set, its the normal blink wake up for node ID notification.

Bit 8 = BlinkID

Format ASCII:	3 bytes (dec)
Format BINARY:	1 byte (uint8_t)
Range:	0 … 255
Description:	Blink ID.

Bit 9 = RX slot counter.

Format ASCII:	3 bytes (dec)
Format BINARY:	1 byte (uint8_t)
Range:	0 … 255
Description:	RX slot counter.

Bit 10 = Timestamp of broadcast message.

Format ASCII:	8 bytes (dec)
Format BINARY:	4 bytes (uint32_t)
Range:	0 … 4294967295
Description:	Timestamp in milliseconds of broadcast message.

Return value description: **=<Mask>**

ASCII

Parameter (Format):

<Mask>	4 bytes (hex)

```
Example:      NCFG 1ff
```

Return value (Format):

<Mask>	4 bytes (hex)

```
Example:     =1
```

BINARY:

Type: GET/SET

Parameter (Format):

<Mask> 2 bytes (uint16_t)

`Example: SET NCFG 1ff`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x32	0x01ff	0x0b	0xdc

Return value (Format):

<Mask> 2 bytes (uint16_t)

`Example: RESP NCFG 1ff`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x32	0x01ff	0x0a	0x64

Medium Access Commands

SRXW <Time>:

Sets reception window, time span the receiver listens after sending its broadcast ID in power saving mode.

Parameter description:

<Time> Blink interval in milliseconds [ms]

Range: 0 … 65000

0 = Receiver is continuously disabled

1 – 65000 = Time in ms that the receiver stays on after its Node ID broadcast. If TIME> NodeID broadcast intervall the receiver is always active

Return value description **=<Time>**

ASCII

Parameter (Format):

<Time> 5 bytes (dec)

`Example:      SRXW 00010`

Return value (Format):

<Time> 5 bytes (dec)

`Example:      =00010`

BINARY:

Type: GET/SET

Parameter (Format):

<Time> 2 bytes (uint16_t)

`Example:      SET SRXW 10`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x40	0x000A	0x6a	0x10

Return value (Format):

<Time> 2 bytes (uint16_t)

`Example:      RESP SRXW 10`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x40	0x000A	0x6b	0xa8

SRXO <Value>:

Sets reception window occurrence.

Note: If streaming is enabled, or power saving mode is disabled, the receiver is always open, but asynchronous transmissions still use this window for sending packets. If SRXO is set to 2, even if the receiver is always open, asynchronous packets are only send every other node ID notification.

Parameter description:

<Value>

Range: 0 … 255

0 = Reception window disabled.

1-255 = Number of Node ID Blinks with a RxWindow. If 1, every Node ID Blink a RxWindow, if 2, every second Node ID Blink a RxWindow.

Return value description **=<Value>**

ASCII

Parameter (Format):

<Value> 3 bytes (dec)

```
Example:     SRXO 1
```

Return value (Format):

<Value> 3 bytes (dec)

```
Example:     =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Value> 1 byte (uint8_t)

```
Example:     SET SRXO 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x41	0x01	0xf4	0x0e

Return value (Format):

<Value> 1 byte (uint8_t)

```
Example:     RESP SRXO 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x41	0x01	0x55	0xce

SDCL <Class>:

Sets the node's device class which will be broadcasted in its blink ID packets.

Parameter description:

<Class>

Range: 1 … 8

Return value description **=<Class>**

ASCII

Parameter (Format):

<Class> 1 byte (dec)

`Example: SDCL 1`

Return value (Format):

<Class> 1 byte (dec)

`Example: =1`

BINARY:

Type: GET/SET

Parameter (Format):

<Class> 1 byte (uint8_t)

`Example: SET SDCL 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x42	0x01	0xf4	0xfe

Return value (Format):

<Class> 1 byte (uint8_t)

`Example: RESP SDCL 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x42	0x01	0x55	0x3e

SFEC <Enable>:

Enables forward error correction (FEC)

Parameter description:

<Enable>

Range:	0 … 1
	0 = FEC off
	1 = FEC on

Return value description **=<Enable>**

ASCII

Parameter (Format):

<Enable> 1 byte (dec)

```
Example:      SFEC 1
```

Return value (Format):

<Enable> 1 byte (dec)

```
Example:      =1
```

BINARY

Type: GET/SET

Parameter (Format):

<Enable> 1 byte (uint8_t)

```
Example:      SET SFEC 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x43	0x01	0xf5	0x6e

Return value (Format):

<Enable> 1 byte (uint8_t)

```
Example:      RESP SFEC 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x43	0x01	0x54	0xae

SDAM <Mode>:

Sets the node's data mode

Parameter description:

<Mode>

Range: 1 … 2

1 = 80MHz, 1µs per bit mode

2 = 80MHz, 4µs per bit mode

Return value description **=<Mode>**

ASCII

Parameter (Format):

<Mode> 1 byte (dec)

```
Example:      SDAM 2
```

Return value (Format):

<Mode> 1 byte (dec)

```
Example:      =2
```

BINARY

Type: GET/SET

Parameter (Format):

<Mode> 1 byte (uint8_t)

```
Example:      SET SDAM 2
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x44	0x02	0xb7	0x5f

Return value (Format):

<Mode> 1 byte (uint8_t)

```
Example:      RESP SDAM 2
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x44	0x02	0x16	0x9f

CSMA <M> <Duration> <Threshold>:

Sets CSMA mode on and off and determines back-off factor for CSMA or uses fixed back-off factor of length <DURATION>*24µs. <Threshold> is for energy detection CSMA. When gain is below this <Threshold> the medium is interpreted as busy.

Parameter needed depending on <M>.

CSMA <M=0>

CSMA <M=1/2> <Duration>

CSMA <M=3/4> <Duration> <Threshold>

Parameter description:

<M>

Range: 0 ... 4

<M> = 0 CSMA off → ALOHA

<M> = 1 CSMA symbol detection on, random seed, <Duration> parameter needed.

<M> = 2 CSMA symbol detection on, fixed back-off time, <Duration> parameter needed.

<M> = 3 CSMA energy detection on, random seed, <Duration> parameter and <Threshold> parameter needed.

<M> = 4 CSMA energy detection on, fixed back-off time, <Duration> parameter and <Threshold> parameter needed.

<Duration>

Range: 0 … 255

For <M>=1 or <M>=3 → random seed

For <M>=2 or <M>=4 → time in 24µs ticks

<Threshold>

Range: 0 … 63

Return value: **=<M>,<Duration>,<Threshold>**

ASCII

Parameter (Format):

<M>	1 byte (dec)
<Duration>	3 bytes (dec)
<Threshold>	2 bytes (dec)

`Example:      CSMA 2 10`

Return value (Format):

=<M>,<Duration>,<Threshold>

`Example:      =2,10`

BINARY

TYPE: GET/SET

Parameter (Format):

<M>	1 byte (uint8_t)
<Duration>	1 byte (uint8_t)
<Threshold>	1 byte (uint8_t)

`Example:      SET CSMA 4 255 18`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x05	0x55	0x45	0x04ff12	0x17	0xae

Return value (Format):

<M>	1 byte (uint8_t)
<Duration>	1 byte (uint8_t)
<Threshold>	1 byte (uint8_t)

`Example:      RESP CSMA 4 255 18`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x05	0x57	0x45	0x04ff12	0x6e	0x6e

MEMS & Temperature Sensor Commands

EMSS <Enable>:

Enables and disables the MEMS sensor on the *swarm* radio.

Parameter description:

<Enable>

Range: 0 … 1

0 = MEMS sensor will be disabled on node

1 = MEMS sensor will be enabled on Node

Return value description **=<Enable>**

ASCII

Parameter (Format):

<Enable> 1 byte (dec)

`Example:      EMSS 1`

Return value (Format):

<Enable> 1 byte (dec)

`Example:      =1`

BINARY

Type: GET/SET

Parameter (Format):

<Enable> 1 byte (uint8_t)

`Example:      SET EMSS 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x50	0x01	0xf8	0x5e

Return value (Format):

<Enable> 1 byte (uint8_t)

`Example:      RESP EMSS 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x50	0x01	0x59	0x9e

EBMS <Notify>:

Description: Enables and disables broadcasting of MEMS and TEMP values in blink packet

Parameter description:

<Notify>

Range: 0 … 1

0 =Node will not broadcast MEMS & TEMP values in blink packet

1 = Node will broadcast MEMS &TEMP values in blink packet

Return value description **=<Notify>**

<Notify> returning parameter which has been set

Range: 0 …1

ASCII

Parameter (Format):

<Notify>	1 byte (dec)

`Example:       EBMS 1`

Return value (Format):

<Notify>	1 byte (dec)

`Example:       =1`

BINARY

Type: GET/SET

Parameter (Format):

<Notify>	1 byte (uint8_t)

`Example:       SET EBMS 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x51	0x01	0xf9	0xce

Return value (Format):

<Notify>	1 byte (uint8_t)

`Example:       RESP EBMS 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x51	0x01	0x58	0x0e

SMRA <Grange>:

Sets the MEMS range, that is the sensor's acceleration scale.

Parameter description:

<Grange>

Range:	1 … 4
	1 = MEMS g range set to +/- 2g
	2 = MEMS g range set to +/- 4g
	3 = MEMS g range set to +/- 8g
	4 = MEMS g range set to +/- 16g

Return value description: **=<Grange>**

Note:

ERR returned if MEMS is switched off

ASCII

Parameter (Format):

<Grange> 1 byte (dec)

```
Example:      SMRA 2
```

Return value (Format):

<Grange> 1 byte (dec)

```
Example:      =2
```

BINARY

Type: GET/SET

Parameter (Format):

<Grange> 1 byte (uint8_t)

```
Example:      SET SMRA 2
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x52	0x02	0xb9	0x3f

Return value (Format):

<Grange>	1 byte (uint8_t)

`Example:      RESP SMRA 2`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x52	0x02	0x18	0xff

SMTH <Thres>:

Sets the MEMS sensor's threshold definition for the slope interrupt. An LSB corresponds to an LSB of acceleration data and therefore depends on the selected g range (see above).

Parameter description:

<Thres>

Range: 0 … 255

For <Grange> =	1	Threshold = 3.91 mg/LSB * <Thres>
For <Grange> =	2	Threshold = 7.81 mg/LSB * <Thres>
For <Grange> =	3	Threshold = 15.62 mg/LSB * <Thres>
For <Grange> =	4	Threshold = 31.25 mg/LSB * <Thress>

Return value description **=<Thres>**

Note:

ERR returned if MEMS is switched off

ASCII

Parameter (Format):

<Thres>	3 bytes (dec)

`Example:      SMTH 100`

Return value (Format):

<Thres>	3 bytes (dec)

`Example:      =100`

BINARY

Type: GET/SET

Parameter (Format):

<Thres>	1 byte (uint8_t)

`Example:      SET SMTH 100`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x53	0x64	0x38	0x85

Return value (Format):

<Thres> 1 byte (uint8_t)

`Example:` `RESP SMTH 100`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x53	0x64	0x99	0x45

SMBW <BandWidth>:

Sets the MEMS sensor’s filter bandwidth.

Parameter description:

<BandWidth>

Range: 1 … 8

1 = 7.81 Hz

2 = 5.63 Hz

3 = 31.25 Hz

4 = 62.50 Hz

5 = 125.00 Hz

6 = 250.00 Hz

7 = 500.00 Hz

8 = 1000.00 Hz

Return value description **=<BandWidth>**

Note:

ERR returned if MEMS is switched off

ASCII

Parameter (Format):

<BandWidth> 1 byte (dec)

`Example:` `SMBW 3`

Return value (Format):

<BandWidth> 1 byte (dec)

`Example:` `=3`

BINARY

Type: GET/SET

Parameter (Format):

<BandWidth> 1 byte (uint8_t)

`Example: SET SMBW 3`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x54	0x03	0x7b	0x5f

Return value (Format):

<BandWidth> 1 byte (uint8_t)

`Example: RESP SMBW 3`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x54	0x03	0xda	0x9f

SMSL <SleepTime>:

Sets the MEMS sensor’s sleep time

Parameter description:

<SleepTime>

Range: 1 … 11

1 = 0.5 ms

2 = 1 ms

3 = 2 ms

4 = 4 ms

5 = 6 ms

6 = 10 ms

7 = 25 ms

8 = 50 ms

9 = 100 ms

10 = 500 ms

11 = 1000 ms

Return value description **=<SleepTime>**

Note:

ERR returned if MEMS is switched off

ASCII

Parameter (Format):

<SleepTime> 2 bytes (dec)

`Example: SMSL 11`

Return value (Format):

<SleepTime> 2 bytes (dec)

`Example:      =11`

BINARY

Type: GET/SET

Parameter (Format):

<SleepTime> 1 byte (uint8_t)

`Example:      SET SMSL 11`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x55	0x55	0x0B	0x7b	0x09

Return value (Format):

<SleepTime> 1 byte (uint8_t)

`Example:      RESP SMSL 11`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x57	0x55	0x0B	0xda	0xc9

SMDT <DeadTime>:

Sets the MEMS sensor's dead time.

Parameter description:

<DeadTime> Time in milliseconds with MEMS interrupt disabled, after MEMS interrupt occurred.

Range: 0 … 65000

Return value: =**<DeadTime>**

ASCII

Parameter (Format):

<DeadTime> 5 bytes (dec)

`Example:      SMDT 1000`

Return value (Format):

<DeadTime> 5 bytes (dec)

`Example:      =1000`

BINARY

Type: GET/SET

Parameter (Format):

<DeadTime> 2 bytes (uint16_t)

`Example:      SET SMDT 1000`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x56	0x03E8	0x0b	0x6d

Return value (Format):

<DeadTime>	2 bytes (uint16_t)

```
Example:     RESP SMDT 1000
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x56	0x03E8	0x0a	0xd5

GMYA <void>:

Reports Node's own MEMS acceleration values

Parameter description:

<void>

Return value description:

<XACC> Acceleration value for X coordinate (positive and negative)

Range: -32768mg … +32767mg

<YACC> Acceleration value for Y coordinate (positive and negative)

Range: -32768 … +32767mg

<ZACC> Acceleration value for Z coordinate (positive and negative)

Range: -32768 … +32767mg

Note: Max. range depends on SMRA (2g,4g,8g,16g)

Note:

ERR returned if MEMS is switched off

ASCII

```
Example:     GMYA
```

Return value (Format):

<XACC>	6 bytes (dec), Signed INT16, first byte "+" or "-"

<YACC>	6 bytes (dec), Signed INT16, first byte "+" or "-"
<ZACC>	6 bytes (dec), Signed INT16, first byte "+" or "-"

`Example:      =+31599,+18501,-26468`

BINARY

Type: GET

`Example:      GET GMYA`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x57	0xc7	0x2a

Return value (Format):

<XACC>	2 bytes (int16_t)
<YACC>	2 bytes (int16_t)
<ZACC>	2 bytes (int16_t)

`Example:      RESP GMYA +31599 +18501 -26468`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x56	0x57	0x7B6F4845989C	0x45	0x96

GMYT <void>:

Reports Node's own MEMS temperature values

Parameter description:

<void>

Return values: **=<Temperature>**

<Temperature> Node's own current temperature (positive and negative) in °C

Range: -99 … +99

Note:

ERR returned if MEMS is switched off

ASCII

`Example:      GMYT`

Return value (Format):

<Temperature>	3 bytes (dec), first byte "+" or "-"

`Example:      =+26`

BINARY:

Type: GET

`Example: GET GMYT`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x58	0x87	0x2e

Return value (Format):

<Temperature>	1 byte (int8_t)

`Example: RESP GMYT +26`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x56	0x58	0x1A	0x4f	0x95

GBAT <void>:

Reports battery level of the node connected to the host

Parameter description:

<void>

Return values: **=<BatteryStatus>**

<BatteryStatus> Node's own battery level in dV (example: =025 is 2.5V)

Range: 000 … 255

ASCII

`Example: GBAT`

Return value (Format):

<BatteryStatus> 3 bytes (dec)

`Example: =025`

BINARY

Type: GET

`Example: GET GBAT`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x59	0x46	0xee

Return value (Format):

<BatteryStatus> 1 byte (uint8_t)

`Example:      RESP GBAT 25`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x56	0x59	0x19	0x0e	0x04

GPIO <Mask> <Mode> <Speed> <OType> <PUPD>:

Description: Configure GPIO's of µC.

Parameter description:

<Mask> defines which pins affected by changes.

Range: 00 .. FF

Bit 0 = PA0 = DIO_0

Bit 1 = PA2 = DIO_1

Bit 2 = PA3 = DIO_2

Bit 3 = PA8 = DIO_3

<Mode>

Range: 0 ... 3

0 = GPIO Input Mode

1 = GPIO Output Mode

2 = GPIO Alternate function Mode

3 = GPIO Analog Mode

<Speed>

Range: 0 ... 3

0 = Very Low Speed (400KHz)

1 = Low Speed (2MHz)

2 = Medium Speed (10MHz)

3 = High Speed (40MHz)

<OType>

Range: 0 … 1

0 = Push Pull

1 = Open Drain

<PUPD>

Range: 0 … 2

0 = No pull

1 = Pull up

2 = Pull down

Return values: **=<Pins>,<Mode>,<Speed>,<OType>,<PUPD>**

ASCII

Parameter (Format):

<Pins>	2 bytes (hex)
<Mode>	1 byte (dec)
<Speed>	1 byte (dec)
<OType>	1 byte (dec)
<PUPD>	1 byte (dec)

```
Example:      GPIO 03 1 3 0 1
```

Return value (Format):

<Pins>	2 bytes (hex)
<Mode>	1 byte (dec)
<Speed>	1 byte (dec)
<OType>	1 byte (dec)
<PUPD>	1 byte (dec)

```
Example:      =03,1,3,0,1
```

BINARY

Type: GET/SET

Parameter (Format):

<Pins>	1 bytes (uint8_t)
<Mode>	1 byte (uint8_t)
<Speed>	1 byte (uint8_t)
<OType>	1 byte (uint8_t)
<PUPD>	1 byte (uint8_t)

```
Example:      SET GPIO 3 1 3 0 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x07	0x55	0x5A	0x0301030001	0x18	0x17

Return value (Format):

<Pins>	1 bytes (uint8_t)
<Mode>	1 byte (uint8_t)
<Speed>	1 byte (uint8_t)
<OType>	1 byte (uint8_t)
<PUPD>	1 byte (uint8_t)

```
Example:      RESP GPIO 3 1 3 0 1
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x07	0x57	0x5A	0x0301030001	0x3b	0xd7

SPIN <Mask> <Status>:

Set new pin status.

Parameter description:

<Mask> defines which pins affected by changes.

Range: 00 .. FF

Bit 0 = PA0 = DIO_0

Bit 1 = PA2 = DIO_1

Bit 2 = PA3 = DIO_2

Bit 3 = PA8 = DIO_3

<Status>

Range: 00 ... FF

0 = Bit reset

1 = Bit set

Return values: **=<Mask>,<Status>**

ASCII

Parameter (Format):

<Mask>	2 bytes (hex)
<Status>	2 bytes (hex)

```
Example:      SPIN 03 01
```

Return value (Format):

<Mask>	2 bytes (hex)
<Status>	2 bytes (hex)

```
Example:      =03,01
```

BINARY

Type: SET

Parameter (Format):

<Mask>	1 byte (uint8_t)
<Status>	1 byte (uint8_t)

```
Example:      SET SPIN 03 01
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x5b	0x0301	0x5b	0x20

Return value (Format):

<Mask>	1 byte (uint8_t)
<Status>	1 byte (uint8_t)

```
Example:      RESP SPIN 03 01
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x5b	0x0301	0x5a	0x98

GPIN <void>:

Read current pin status.

Parameter description:

<void>

Return values: **=<Status>**

<Status> Each bit indicates a pin status 0 = reset, 1 = set.

Range: 00 … FF

ASCII

Parameter (Format):

```
Example:      GPIN
```

Return value (Format):

<Status>	2 byte (hex)

`Example:      =01`

BINARY

Type: GET

`Example:      GET GPIN`

SYN	LEN	TYPE	CMD	CRC_LOW	CRC_HIGH
0x7f	0x02	0x54	0x5B	0xc7	0x2f

Return value (Format):

<Status> 1 byte (uint8_t)

`Example:      RESP GPIN 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x03	0x56	0x5B	0x00	0xce	0xae

ICFG <Setting>:

Configure and enable interrupt sources.

Parameter description:

<Setting>

Range: 0 … FFFF

Bit 0 = PA0 = DIO_0 = Rising edge enabled/disabled

Bit 1 = PA0 = DIO_0 = falling edge enabled/disabled

Bit 2 = PA2 = DIO_1 = Rising edge enabled/disabled

Bit 3 = PA2 = DIO_1 = falling edge enabled/disabled

Bit 4 = PA3 = DIO_2 = Rising edge enabled/disabled

Bit 5 = PA3 = DIO_2 = falling edge enabled/disabled

Bit 6 = PA8 = DIO_3 = Rising edge enabled/disabled

Bit 7 = PA8 = DIO_3 = falling edge enabled/disabled

Bit 8 = MEMS interrupt enabled/disabled

Return values: **=<Setting>**

ASCII

Parameter (Format):

<Setting> 4 byte (hex)

`Example: ICFG 1`

Return value (Format):

<Setting> 4 byte (hex)

`Example: =1`

BINARY:

Type: GET/SET

Parameter (Format):

<Setting> **2** bytes (uint16_t)

`Example: SET ICFG 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x55	0x5C	0x0001	0xea	0x11

Return value (Format):

<Setting> 2 bytes (uint16_t)

`Example: RESP ICFG 1`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x04	0x57	0x5C	0x0001	0xeb	0xa9

6. AIR Interface (additional commands)

SSTART <Time>:

OPCODE: 0x23

Force receiver active for a certain time at remote node. This is useful to change multiple commands one after another without waiting for a RxSlot.

Parameter description:

<Time> force active receiver for n milliseconds

Range: 1 … 65000

Parameter (Format):

<Time> 2 bytes (uint16_t)

Return:

<Time>

Example via ASCII SDAT: Send air command SSTART to force 1000ms receive active.

1) SDAT 1 000000000011 0B 08125554025500032303e8 60000
2) =4130055640
3) *SDAT:000000000011,0,4130055640
4) *AIR:000000000011,23,57,03e8

SEXTEND:

OPCODE: 0x24

Refresh timeout set by SSTART to extend receiver active window.

Parameter description:

NO PARAMETER

Return:

<ErrorCode>

0 = successful extended
1 = failed to extend

Example via ASCII SDAT: Send air command SEXTEND immediately.

1) SDAT 0 000000000011 09 081255540255000124
2) =0
3) *AIR:000000000011,24,57

SSTOP:

OPCODE: 0x25

Stop streaming to this device. The remote device immediately turn back to normal operation regarding the receiver active state.

Parameter description:

NO PARAMETER

Return:

NO PARAMETER

Example via ASCII SDAT: Send air command SSTOP immediately.

1) SDAT 0 000000000011 09 081255540255000125
2) =0
3) *AIR:000000000011,25,57

MRATO:

Not supported.

7. Notification messages

This chapter describes the format of all asynchronous messages.

Format for Data Notification Messages

This chapter describes the communication structure for Data Notifications when data notification has been enabled.

Notification format: **<Data Notification Flag>** (fixed):**<ID>**

Parameter description:

<ID> returns ID of node which sent message

Range: 000000000001 … FFFFFFFFFFFE

ASCII:

Parameter(Format)

<Data Notification Flag>	4 bytes starting with "*" to signal identification flag with content *DNO
<ID>	12 bytes (hex)

`Example:      *DNO:1F3CFF322133`

BINARY:

Type: NOTI

CMD: Message type DNO (0x60)

Parameter (Format):

<ID> 6 bytes (uint8_t[6])

`Example:      NOTI DNO 1F3CFF322133`

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x08	0x61	0x60	0x1F3CFF322133	0xcd	0x69

Format for Node ID Notification Messages

This chapter describes the communication structure for Node ID Notifications when node ID notification has been enabled.

Notification format: **<NodeID Notification Flag>** (fixed):**<ID>**,**<NCFG>**,**<DataNCFG>**

Parameter description:

<ID> returns ID of node which sent message

Range: 000000000001 … FFFFFFFFFFFE

<NCFG> Defines which data follows within this value. For details see at NCFG command for more information.

<DataNCFG> Depends on command NCFG and defines the sensor data transmitted to the host. The data is ordered according to the bit numbering: Example: **<NCFG>** = 5 (1+4) Bit 1 and Bit 2 is transmitted from *swarm* node to host. After **<NCFG>** in the example follow MEMS-Values and then RSSI value. If a value isn't present because MEMS is disabled "?" is transimited in ASCII mode and MAX value from corresponding data type in BINARY mode.

ASCII:

Parameter(Format)

<Node ID Notification Flag>

4 bytes starting with "*" to signal identification flag with content *NIN

<ID>	12 bytes (hex)
<NCFG>	4 bytes (hex)
<DataNCFG>	depends on **<NCFG>**-Mask

`Example:      *NIN:1F3CFF322133,04,-56`

BINARY:

Type: NOTI

CMD: Message type NIN (0x61)

Parameter (Format):

<ID> 6 bytes (uint8_t[6])

<NCFG> **2** bytes (uint16_t)

<DataNCFG> depends on **<NCFG>**-Mask

```
Example:      NOTI NIN 1F3CFF322133 04 -56
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x09	0x61	0x61	0x1F3CFF32213304C8	0xe5	0xff

Format for Ranging Result Notification Messages

This chapter describes the communication structure for Ranging Results Notification when it has been enabled. This is printed whenever a ranging result broadcast is received or a automatic ranging is finished as long as Ranging Result Notification has been enabled.

Notification format: **Ranging Notification Flag**:**<SrcID>**, **<DestID>**, **<ErrorCode>**, **<Distance>**, **<NCFG>, <DataNCFG>**

Parameter description:

<SrcID> Returns ID of node which requested ranging

Range: 000000000001 … FFFFFFFFFFFE

<**DestID>** Returns ID of node which received ranging request

Range: 000000000001 … FFFFFFFFFFFE

<ErrorCode> indicating status of ranging operation

Range: 0 … 6

0 = success, ranging result valid

1 = ranging to own ID

2 = ID out of range, no ACK

3 = ranging unsuccessful, ACK OK, then timeout.

5 = timeout duration has been reached.

6 = medium blocked (CSMA give up)

<Distance> returning the measured ranging distance in centimeter. Range depends on mode.

Range(ASCII): 000000 … 999909 ranging distance in centimeter [cm]

Range(BINARY): 0...99999 ranging distance in centimeter [cm]

<NCFG> Defines which data follows within this value. For details see at NCFG command for more information.

<DataNCFG> Depends on command NCFG and defines the sensor data transmitted to the host. The data is ordered according to the bit numbering: Example: **<NCFG>** = 5 (1+4) Bit 1 and Bit 2 is transmitted from *swarm* node to host. After **<NCFG>** in the example follow MEMS-Values and then RSSI value. If a value isn't present because MEMS is disabled "?" is transimited in ASCII mode and MAX value from corresponding data type in BINARY mode.

ASCII:

Parameter(Format)

<Ranging Notification Flag> 4 bytes starting with "*" to signal identification flag with content *RRN

<SrcID>	12 bytes (hex)
<DestID>	12 bytes (hex)
<ErrorCode>	1 byte (dec)
<Distance>	6 bytes (dec) in [cm]
<NCFG>	4 bytes (hex)
<DataNCFG>	depends on **<NCFG>**-Mask

```
Example:     *RRN:1F3123123133,1F3CFF322133,0,001843,04,-56
```

BINARY:

Type: NOTI

CMD: Message type RRN (0x62)

Parameter (Format):

<SrcID>	6 bytes (uint8_t[6])
<DestID>	6 bytes (uint8_t[6])
<ErrorCode>	1 byte (uint8_t)
<Distance>	4 bytes (uint32_t) in [cm]
<NCFG>	2 bytes (uint16_t)
<DataNCFG>	depends on **<NCFG>**-Mask

```
Example:     NOTI RRN 1F3123123133 1F3CFF322133 0 1840 04 -56
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x13	0x61	0x62	0x1F31231231331F3CFF3221330000073004C8	0xd8	0x70

Format for SDAT Notification Messages

This chapter describes the return message format for asynchronous SDAT Notifications after issuing a SDAT command.

Notification format: **<SDAT Notification Flag>**:**<ID>**,**<ErrorCode>**,**<PayloadID>**

Parameter description:

<ID> returns ID of node to which the message was sent

Range: 000000000001 … FFFFFFFFFFFE

<ErrorCode> indicating status of ranging operation

Range: 0 … 4

0 = success data communication valid

1 = error: no hardware ack received.

2 = timeout error, message could not be delivered

3 = error: medium blocked (CSMA give up)

4 = error: unknown.

<PayloadID> used to identify the payload

Range: 000000000001 … FFFFFFFFFFFF

ASCII:

Parameter(Format)

<SDAT Notification Flag>	5 bytes starting with "*" to signal identification flag with content *SDAT
<ID>	12 bytes (hex)
<ErrorCode>	1 byte (dec)
<PayloadID>	8 bytes (hex)

```
Example:     *SDAT:1F3CFF322133,0,45A6213F
```

BINARY:

Type: NOTI

CMD: Message type SDAT (0x63)

Parameter (Format):

<ID>	6 bytes (uint8_t[6])
<ErrorCode>	1 byte (uint8_t)
<PayloadID>	4 bytes (uint8_t[4])

```
Example:     NOTI SDAT 1F3CFF322133 0 45A6213F
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0D	0x61	0x63	0x1F3CFF3221330045A6213F	0xce	0xcb

Format for AIR Notification Messages

This chapter describes the return message format for asynchronous responses after issuing a SDAT command with the AIR interface protocol.

Notification format: **<AIR Notification Flag>**:**<ID>**,**<C_OPCODE>**,**<C_TYPE>**,**<LEN>**,**<DATA>**

Parameter description:

<ID> returns ID of remote node

Range: 000000000001 … FFFFFFFFFFFE

<C_OPCODE> indicating command opcode

Range: 00 .. FF

<C_TYPE> used to identify if its a SET, GET, or ERROR message

Range: 00 .. FF

0x54 = GET request

0x55 = SET request

0x56 = GET response

0x57 = SET response

0x60 = ERROR

<LEN> indicating length of <DATA> field.

Range: 00 .. 7F

<DATA> command data, or if its an error message, the error code. If no data appended, this field is missing.

Range: 00 .. FF

if C_TYPE is ERROR:

0x02 = unknown command opcode

0x03 = parameter error

ASCII:

Parameter(Format)

<AIR Notification Flag> 4 bytes starting with "*" to signal identification flag with content *AIR

<ID>	12 bytes (hex)
<C_OPCODE>	2 byte (hex)
<C_TYPE>	2 byte (hex)
<LEN>	2 byte (hex)
<DATA>	2 byte (hex)

```
Example:      *AIR:000000000011,05,56,01,3f
```

BINARY:

Type: NOTI

CMD: Message type AIR (0x64)

Parameter (Format):

<ID>	6 bytes (uint8_t[6])
<C_OPCODE>	1 byte (uint8_t)
<C_TYPE>	1 bytes (uint8_t)
<LEN>	1 bytes (uint8_t)
<DATA>	(n) bytes (uint8_t[n])

```
Example:      NOTI AIR 000000000011 05 56 01 3f
```

SYN	LEN	TYPE	CMD	CMD_DATA	CRC_LOW	CRC_HIGH
0x7f	0x0b	0x61	0x64	0x000000000110556013f	0xaf	0x19

8. API Default Settings

When starting the *swarm* Radio the following default settings, representing an active node, are valid:

Parameter/ Command	Default Value	Description
BRAR	1	Broadcast ranging results is enabled.
CSMA	0, 0, 0	Disabled (mode 0, random seed 0, threshold 0)
EBID	1	ID broadcast enabled
EBMS	1	Broadcast MEMS data is enabled.
EDAN	1	Data notification enabled
EIDN	0	Node ID notification is disabled
EMSS	1	MEMS sensor is enabled.
EPRI	0	Responds to ranging request is true.
ERRN	1	Ranging result notification is enabled
GPIO	0F, 0, 3, 0, 1	All inputs are with PullUp.
ICFG	0	All interrupts are disabled.
NCFG	0004	Notification configuration is set to RSSI only.
SBIV	30000	ID broadcast interval is 30000 ms
SDAM	1	80/1 mode enabled
SDCL	1	Device class is 1
SFEC	0	FEC is disabled.
SMBW	6	MEMS sensor's bandwidth is 250 Hz.
SMDT	01000	MEMS sensor'sdead time is 1000 ms.
SMRA	2	MEMS acceleration scale (g range) is 4 g.
SMSL	01	MEMS sensor's sleep time is 0.5 ms.
SMTH	30	MEMS sensor’s threshold is set to 30.
SNID	0000xxxxxxxx	Factory pre-configured MAC address
SPSA	0	Power saving is disabled.
SROB	001	Range on broadcast to device class 1
SROF	000	Range offset is 0.
SRXO	001	RX occurrence is 1 (RX window after every node ID blink)
SRXW	00010	RX window is 10 ms.
SSYC	1	Syncword is 1.
STXP	63	TX power is 63.

9. Settings for Different Node Behaviours

The *swarm* radio settings are predefined with some default values such that the user only needs to connect power to start working with it. This default behavior is as follows:

Every 30s the *swarm* device broadcasts a blink with its nodeID and sensor data in it. During the time between blinks the radio is in receiving mode, waiting for different kind of messages which are listed in the following:

- A blink from another *swarm* radio: Whenever it receives a blink from any other *swarm* radio, it initiates a ranging operation with that radio. The ranging operation consists of an exchange of packets in which both *swarm* radios participate. Once it has all the necessary data it estimates the distance to the other *swarm* radio and broadcast it.
- A ranging request: Any *swarm* in the neighborhood who receive its nodeID blink may start a ranging operation with the *swarm*. In this case, the *swarm* answers by sending all necessary packets.
- A range result broadcast: After a ranging operation the *swarms* broadcast the result so that all the neighbors have access to that information. The range result may indicate the distance between the receiving *swarm* itself and the sender or between the sender and any other *swarm* radio in the area. When a *swarm* which is connected to a host receives a range result broadcast packet, it passes it to the host as a range result notification (RRN).
- Any other data packet from any swarm in the neighborhood: In case the *swarm* is connected to a host it notifies the host that data was received. It is up to the host controller to read the data or not (API command: GDAT).

This ‘default’ behavior of the *swarm* can be changed using the API commands. Some of the possible changes are:

- The blinking interval can be changed (API command SBIV) or even deactivated (API command EBID) so that the *swarm* will not blink.
- The *swarm* radio can be set in low power mode: The swarm can be put in sleep mode so that it wakes up only when it needs to send a blink; after the blink it waits for a while in receive mode in case it receives ranging requests or other packet and it goes to sleep again. When a *swarm* is in this mode a notification is sent as part of each blink so that all the nodes around know that if they need to send something to this node they should do it immediately after receiving its blink. The time during which the *swarm* is listening after its blink and whether it listens after each blink or after every xth blink can be set by the user (API commands SRXW and SRXO).
- A privacy mode can be used: When a swarm receives a blink it reacts, by default, by initiating a ranging operation with the blink originator. This behavior can be modified (API command SPRI) so that the *swarm* only reacts to certain devices: Devices of the same class (API command SROB) or devices with certain IDs (API command RATO). This allows the user to reduce the number of packets over the air by not sending unnecessary messages.
- The notifications that are passed to the host can be selected and customized: The swarm can also notify the host controller every time it receives a blink (API command EIDN). Moreover, in every blink the *swarms* can add payload data; the user can decide what data is passed to the host in this notifications (RSSI, sensor data, battery status …). This can be done with the API command NCFG. By default the sensor data is added to the blink payload.
- Send to an individual *swarm* or broadcast data: The command SDAT can be used to send data to any other *swarm* in range. For the two operations the transmitting *swarm* needs to take into account that the receiving *swarms* may be in sleep mode. If they are, they should send the message every time one of the receivers sends a blink.

10. Certifications

10.1. FCC (United States) Certification

The PT-000320 RF Module complies with Part 15 of the FCC rules and regulations. Compliance with the labeling requirements [Figure A], FCC notices and antenna usage guidelines is required.

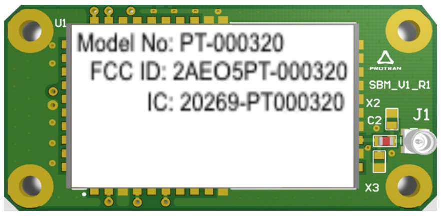


Figure 4 Required FCC label for Module

In order to operate under PT-000320 RF Module FCC Certification, integrators must comply with the following regulations:

The integrator must ensure that the text provided with this device [Figure B] is placed on the outside of the final product and within the final product operation manual.

The PT-000320 RF Module may only be used with antennas that have been tested and approved for use with this module [refer to 'FCC-approved Antennas' section].

15.21 Information to User
Changes or modifications not expressly approved by Harsco, could void the user's authority to operate the equipment.

10.1.1. Integrator Labeling Requirements

WARNING: The Original Equipment Manufacturer (OEM) must ensure that FCC labeling requirements are met. This includes a clearly visible label on the outside of the final product enclosure that displays the text shown in the figure below

Contains FCC ID: 2AEO5PT-000320

The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (i.) this device may not cause harmful interference and (ii.) this device must accept any interference received, including interference that may cause undesired operation.

Figure 5: Required FCC Label for OEM products containing the PT-000320 RF Module

10.1.2. FCC Notices

IMPORTANT: The PT-000320 RF Module has been certified by the FCC for use with other products without any further certification (as per FCC section 2.1091).

IMPORTANT: Integrators must test final product to comply with unintentional radiators (FCC sections 15.107 & 15.109) before declaring compliance of their final product to Part 15 of the FCC rules.

IMPORTANT: The PT-000320 RF module has been certified for remote and base radio applications. If the module will be used for portable applications, the device must undergo SAR testing.

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Re-orient or relocate the receiving antenna, Increase the separation between the equipment and receiver, Connect equipment and receiver to outlets on different circuits, or Consult the dealer or an experienced radio/TV technician for help.

10.1.3. Modular Approval

Single-modular transmitter is a self-contained, physically delineated, component for which compliance can be demonstrated independent of the host operating conditions, and which complies with all eight requirements of Section 15.212(a)(1) as summarized below. See Section 15.212 for more detailed information, and Section 2.901 (and sub-sections that follow) for general certification requirements.

i The radio elements must have the radio frequency circuitry shielded. Physical components and tuning capacitor(s) may be located external to the shield, but must be on the module assembly;

ii The module must have buffered modulation/data inputs to ensure that the device will comply with Part 15 requirements with any type of input signal;

iii The module must contain power supply regulation on the module;

iv The module must contain a permanently attached antenna, or contain a unique antenna connector, and be marketed and operated only with specific antenna(s), per Sections 15.203, 15.204(b), 15.204(c), 15.212 (a), 2.929(b);

v The module must demonstrate compliance in a stand-alone configuration;

vi The module must be labelled with its permanently affixed FCC ID label, or use an electronic display (See KDB Publication 784748 about labelling requirements);

vii The module must comply with all specific rules applicable to the transmitter. The grantee must provide comprehensive instructions to explain compliance requirements; viii The module must comply with RF exposure requirements. For any transmitters intended for use in portable devices, SAR compliance must be demonstrated to be independent of the host device. (See KDB Publication 447498. Attachment 447498)

10.1.4. FCC-approved Antennas

WARNING: This device has been tested with Reverse Polarity SMA connectors with the antennas listed in the table of this section. When integrated into products, fixed antennas require installation preventing end users from replacing them with non-approved antennas. Antennas not listed in the tables must be tested to comply with FCC Section 15.203 (unique antenna connectors) and Section 15.247 (emissions).

Antenna Type	Gain	Power output
Directional	≤12dBi	63mW
Omnidirectional	≤15dBi	63mW
Helical	≤18dBi	52mW

10.1.5. Fixed Base Station and Mobile Applications

The PT-000320 RF Module is pre-FCC approved for use in fixed base station and mobile applications. When the antenna is mounted at least 20cm (8") from nearby persons, the application is considered a mobile application.

10.1.6. Portable Applications and SAR Testing

When the antenna is mounted closer than 20cm to nearby persons, then the application is considered "portable" and requires an additional test be performed on the final product. This test is called Specific Absorption Rate (SAR) testing and measures the emissions from the module and how they affect the person.

10.1.7. RF Exposure

WARNING: This equipment is approved only for mobile and base station transmitting devices. Antenna(s) used for this transmitter must be installed to provide a separation distance of at least 30 cm from all persons and must not be co-located or operating in conjunction with any other antenna or transmitter.

NOTE: The separation distance indicated in the above is 30 cm, but any distance greater than or equal to 23 cm can be used (per MPE evaluation).

10.2. IC (Industry Canada) Certification

This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes : (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

10.2.1. Labeling Requirements

Labeling requirements for Industry Canada are similar to those of the FCC. A clearly visible label on the outside of the final product must display the following text:

Contains IC: 20269-PT000320

The integrator is responsible for its product to comply with IC ICES-003 and FCC Part 15, Sub. B - Unintentional Radiators. ICES-003 is the same as FCC Part 15 Sub. B and Industry Canada accepts FCC test report or CISPR 22 test report for compliance with ICES-003.

Figure 6: Required IC Label for OEM products containing the PT-000320 RF Module

10.2.2. Transmitters for Detachable Antennas

This radio transmitter has been approved by Industry Canada to operate with the antenna types listed in the table above with the maximum permissible gain and required antenna impedance for each antenna type indicated. Antenna types not included in this list, having a gain greater than the maximum gain indicated for that type, are strictly prohibited for use with this device. The required antenna impedance is 50 ohms.

egulations, this radio transmitter may only operate using an antenna of a type and maximum (or lesser) gain approved for the transmitter by Industry Canada. To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (e.i.r.p.) is not more than that necessary for successful communication.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peut fonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvé pour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillage radioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne et son gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépasse pas l'intensité nécessaire àl'établissement d'une communication satisfaisante.

11. Further Reading

- Apps Note – AN0509 swarm API Country Settings
- Apps Note – AN0508 How to Adjust and Measure the RF Output Power on swarm bee LE

12. Revision History

Date	Authors	Version	Description
2015-08-25	nanotron	1.0	

End of Document

Life Support Policy

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 such platforms is creating the location-aware Internet of Things.

Further Informatio

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