Pozyx Arduino Library Documentation

Release 1.2.2

Samuel Van de Velde, Vadim Vermeiren, Laurent Van Acker

Jun 24, 2020

CONTENTS:

CHAPTER

ONE

ARDUINO POZYX

1.1 About

Arduino Pozyx is an Arduino library providing a high-level interface to a Pozyx shield attached to the Arduino, making the Pozyx shield compatible with Arduino Uno, Arduino Mega, and Arduino Nano.

1.2 Features

- Easy to use, allowing both high-level and low-level interfacing with the Pozyx device.
- Straightforward API
- Specialized structs for all important Pozyx data.
- Compatible with Arduino Uno, Mega, and Nano.

1.3 Requirements

- Arduino Web IDE or Arduino IDE 1.6+

1.4 Installation

1.4.1 Web IDE

When you're using the Web IDE, Arduino Pozyx should install automagically when you include `<Pozyx.h>`.

1.4.2 Arduino Library Manager

Currently, the Arduino Pozyx library is easily installable from the Arduino Library manager.

1.4.3 From source

To install from source, you'll have to download the source files first and put them in your Arduino sketchbook libraries folder. You can do this either by:

- `git clone https://github.com/pozyxLabs/Pozyx-Arduino-library` and move the folder to your sketchbook libraries
- Download it from the repository and extracting it.
- Downloading the zip file directly, and extracting it.

CHAPTER

TWO

GETTING STARTED

Note: In this text, the word tag and shield will be used interchangeably.

2.1 Arduino

If you're not familiar with the Arduino platform as a whole, this is a good place to get started and set up with testing your Arduino's basic functionality.

2.2 Setup

2.2.1 Role of Arduino

The Arduino communicates to the Pozyx device using the I2C protocol.

Something that has been confusing to a lot of people is how to set up the Pozyx on Arduino, especially when multiple devices come into play.

And with the additional existence of the Python library that provides direct USB access to the Pozyx, we have even seen people trying to use the Python library with an Arduino which had a Pozyx shield attached, which of course did not work.

In this section, we intend to give you insight of the hardware you need!

2.2.2 Pozyx on Arduino

The only Pozyx device that needs an Arduino is the one you're running the positioning/ranging sketches on. This device is called the master device, as this will also direct the operation of other Pozyx devices through UWB communication.

2.2.3 Remote Pozyx devices on Arduino

Remote Pozyx tags don't need to have an Arduino attached! This is an important point. They just need to be powered. Phone powerbanks play well with the micro USB port on the tags!

Only perform positioning and ranging functions on your master device.

Note: You might want to have an Arduino on remote shields if you want to read their sensor data locally. However, do not use functions like positioning and ranging on multiple devices at the same time!

An example is the Cloud example where an Arduino is used to read the tag's position. In this example, the positioning is directed by the Pozyx attached to the server, and the Arduino checks the tag's status to see whether a new position has been calculated.

2.2.4 Pozyx via USB

Instead of using an Arduino locally, you can skip the Arduino and use an USB cable directly.

This has advantages like:

- Very cross platform serial protocol.
- Easy to use Python programming language for flexible functionality (and extendability).
- Computer can be as powerful as you want.
- You can implement the serial communication in any other programming language.

But disadvantages too:

- Arduino is very cheap and small standalone hardware, compared to a Raspberry Pi or regular PC.
- The new Arduino Web IDE is amazing.
- You're comfortable with Arduino programming but not with Python.

Ultimately, the decision which you want to use depends on your application and your available hardware.

The documentation for using USB directly can be found here.

2.3 Required headers

To use the Pozyx library, you have to include the following headers:

```
#include <Pozyx.h>
#include <Pozyx_definitions.h>
```

2.4 Connecting to the Pozyx

Connecting with the Pozyx is very straightforward. A safe way is presented here:

```
void setup(){
     Serial.begin(115200);
     if(Pozyx.begin() == POZYX_FAILURE){
          Serial.println(F("ERROR: Unable to connect to POZYX shield"));
          Serial.println(F("Reset required"));
          delay(100);
          abort();
     }
 }
```

With this, you initialize the Pozyx and can use the entire API in the rest of your script!

2.5 General philosophy

Essentially, you can do three things with Pozyx:

1. Reading register data, which includes sensors and the device's configuration
2. Writing data to registers, making it possible to change the device's configuration ranging from its positioning algorithm to its very ID.
3. Performing Pozyx functions like ranging, positioning, saving the device's configuration to its flash memory...

All these things are possible to do on the shield connected to your Arduino, and powered remote devices as well. In this section we'll go over all of these.

2.6 Reading data

To read data from the Pozyx, a simple pattern is followed. This pattern can be used with almost all methods starting with the words 'get':

1. Initialize the appropriate container for your data read.
2. Pass this container along with the get functions.
3. Check the status to see if the operation was successful and thus the data trustworthy.

You can see the same pattern in action above when reading the UWB data.

```
// initialize the data container
uint8_t who_am_i;
uint8_t status = Pozyx.getWhoAmI(&whoami);

// check the status to see if the read was successful. Handling failure is covered␣
↪later.
if (status == POZYX_SUCCESS) {
  // print the container. Note how a SingleRegister will print as a hex string by␣
↪default.
  Serial.println(who_am_i); // will print '0x43'
}
```

(continues on next page)

(continued from previous page)

```
# and repeat
# initialize the data container
acceleration_t acceleration;
# get the data, passing along the container
Pozyx.getAcceleration_mg(&acceleration);

# initialize the data container
euler_angles_t euler_angles;
# get the data, passing along the container
Pozyx.getEulerAngles_deg(&euler_angles)
```

2.7 Writing data

Writing data follows a similar pattern as reading, but making a container for the data is optional. This pattern can be used with all methods starting with the words 'set':

1. (Optional) Initialize the appropriate container with the right contents for your data write.
2. Pass this container or the right value along with the set functions.
3. Check the status to see if the operation was successful and thus the data written.

Some typical write operations

```
# initialize Pozyx as above

uint8_t status = Pozyx.setPositionAlgorithm(POZYX_POS_ALG_UWB_ONLY);
// Note: this shouldn't ever happen when writing locally.
if (status == POZYX_FAILURE) {
  Serial.println("Settings the positioning algorithm failed");
}

Pozyx.setPositioningFilter(FILTER_TYPE_MOVING_AVERAGE, 10);
```

2.8 Performing functions

Positioning, ranging, configuring the anchors for a tag to use... While the line is sometimes thin, these aren't per se writes or reads as they are implemented as functions on the Pozyx.

A Pozyx device function typically can take a container object for storing the function's return data, and a container object for the function parameters.

For example, when adding an anchor to a tag's device list, the anchor's ID and position are the function's parameters, but there is no return data. Thus, the function addDevice only needs a container object containing the anchor's properties.

In the library, function wrappers are written in such a way that when no parameters are required, they are hidden from the user, and the same goes for return data.

```
// assume an anchor 0x6038 that we want to add to the device list and
// immediately save the device list after.
device_coordinates_t anchor;
anchor.network_id = 0x6038;
```

(continues on next page)

(continued from previous page)

```
anchor.flag = 0x1;
anchor.pos.x = 5000;
anchor.pos.y = 5000;
anchor.pos.z = 0;
Pozyx.addDevice(anchor);
```

2.9 Remote

To interface with a remote device, every function has a remote_id optional parameter. Thus, every function you just saw can be performed on a remote device as well!

(The exceptions are doPositioning and doRanging, which have doRemotePositioning and doRemoteRanging)

```
// let's assume there is another tag present with ID 0x6039
uint16_t remote_device_id = 0x6039;

// this will read the WHO_AM_I register of the remote tag
uint8_t who_am_i;
Pozyx.getWhoAmI(&whoami, remote_device_id);
```

2.10 Saving writable register data

Basically, every register you can write data to as a user can be saved in the device's flash memory. This means that when the device is powered on, its configuration will remain. Otherwise, the device will use its default values again.

This is useful for multiple things:

- Saving the UWB settings so all your devices remain on the same UWB settings.
- Saving the anchors the tag uses for positioning. This means that after a reset, the tag can resume positioning immediately and doesn't need to be reconfigured!
- Saving positioning algorithm, dimension, filter... you'll never lose your favorite settings when the device shuts down.

There are various helpers in the library to help you save the settings you prefer, not requiring you to look up the relevant registers.

2.11 Finding out the error

Pozyx functions typically return a status to indicate the success of the function. This is useful to indicate failure especially. When things go wrong, it's advised to read the error as well.

A code snippet shows how this is typically done

CHAPTER

THREE

TROUBLESHOOTING

3.1 FAQ

3.2 Lost a device?

3.3 Contacting support

If you want to contact support, please include the following:

- Run the pozyx_basic_troubleshooting.ino (provide link to github location of script) script and attach its output.
- Mention what you want to achieve. Our support team has experience with many use cases and can set you on the right track.
- If you get an error or exception, please include this in your mail instead of just saying something is broken.

Ultimately, the more information you can provide our support team from the start, the less they'll have to ask of you and the quicker your problem resolution.

CHAPTER

FOUR

POZYX CLASS FULL

4.1 Core

group **core**

Functions

int **remoteRegFunctionWithoutCheck**(uint16_t *destination*, uint8_t *reg_address*, uint8_t *params = NULL, int *param_size = 0, uint8_t *pData = NULL, int *size = 0)

Wait until the Pozyx shields has raised a specfic event flag or until timeout. This functions halts the process flow until a specific event flag was raised by the Pozyx shield. The event flag is checked by reading the contents of the reg:POZYX_INT_STATUS register. This function can work in both polled and interupt mode

Parameters

- `interrupt_flag`: the exepected Pozyx interrupt. Possible values are POZYX_INT_STATUS_ERR, POZYX_INT_STATUS_POS, POZYX_INT_STATUS_IMU, POZYX_INT_STATUS_RX_DATA, POZYX_INT_STATUS_FUNC, or combinations.
- `timeout_ms`: maximum waiting time in milliseconds for flag to occur
- `interrupt`: a pointer that will contain the value of the interrupt status register

Return Value

- `#true`: event occured.
- `#false`: event did not occur, this function timed out.

boolean **waitForFlag**(uint8_t *interrupt_flag*, int *timeout_ms*, uint8_t **interrupt* = NULL)

Does the same as the remoteRegFunction, but doesn't wait for nearly as long and doesn't care about whether the function worked.

Parameters

- `interrupt_flag`: the exepected Pozyx interrupt. Possible values are POZYX_INT_STATUS_ERR, POZYX_INT_STATUS_POS, POZYX_INT_STATUS_IMU, POZYX_INT_STATUS_RX_DATA, POZYX_INT_STATUS_FUNC, or combinations.
- `timeout_ms`: maximum waiting time in milliseconds for flag to occur
- `interrupt`: a pointer that will contain the value of the interrupt status register

Return Value

- `#true`: event occured.
- `#false`: event did not occur, this function timed out.

int **begin**(boolean *print_result* = false, int *mode* = MODE_INTERRUPT, int *interrupts* = POZYX_INT_MASK_ALL, int *interrupt_pin* = POZYX_INT_PIN0)

Initiates the Pozyx shield. This function initializes the pozyx device. It will verify that the device is functioning correctly by means of the self-test, and it will configure the interrupts. See the register reg:POZYX_INT_MASK for more details about the interrupts.

Parameters

- `print_result`: outputs the result of the function to the Serial output
- `mode`: The modus of the system: MODE_POLLING or MODE_INTERRUPT
- `interrupts`: defines which events trigger interrupts. This field is only required for MODE_INTERRUPT. Possible values are bit-wise combinations of POZYX_INT_MASK_ERR, POZYX_INT_MASK_POS, POZYX_INT_MASK_IMU, POZYX_INT_MASK_RX_DATA and POZYX_INT_MASK_FUNC. Use POZYX_INT_MASK_ALL to trigger on all events.
- `interrupt_pin`: Pozyx interrupt pin: POZYX_INT_PIN0 or POZYX_INT_PIN1. This field is only required for MODE_INTERRUPT.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getRead**(uint8_t *reg_address*, uint8_t **pData*, int *size*, uint16_t *remote_id* = NULL)

Read from the registers on a local or remote Pozyx device (anchor or tag).

Parameters

- `reg_address`: the specific register address to start reading from
- `pData`: a pointer to the data thas will be read
- `size`: the number of bytes to read
- `remote_id`: this is the network id of the remote Pozyx tag, if used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setWrite**(uint8_t *reg_address*, uint8_t **pData*, int *size*, uint16_t *remote_id* = NULL)

Write to the registers of a local remote Pozyx device (anchor or tag).

Parameters

- `reg_address`: the specific register address to start writing to
- `pData`: a pointer to the data thas will be written
- `size`: the number of bytes to write
- `remote_id`: this is the network id of the remote Pozyx tag, if used.

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **useFunction** (uint8_t *reg_address*, uint8_t **params* = NULL, int *param_size* = 0, uint8_t **pData* = NULL, int *size* = 0, uint16_t *remote_id* = NULL)

Call a register funcion on a local or remote Pozyx device (anchor or tag).

Parameters

- reg_address: the specific register address of the function
- params: this is the pointer to a parameter array required for the specific function that is called
- param_size: the number of bytes in the params array
- pData: a pointer to the data thas will be read
- size: the number of bytes that will be stored in pData
- remote_id: this is the network id of the remote Pozyx tag, if used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

4.2 System functions

group **System_functions**

Functions

int **getWhoAmI** (uint8_t **whoami*, uint16_t *remote_id* = NULL)

Obtain the who_am_i value. This function reads the reg:POZYX_WHO_AM_I register.

Parameters

- whoami: reference to the pointer where the read data should be stored e.g. whoami
- remote_id: optional parameter that determines the remote device to be read

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getFirmwareVersion** (uint8_t **firmware*, uint16_t *remote_id* = NULL)

Obtain the firmware version. This function reads the reg:POZYX_FIRMWARE_VER register.

Parameters

- `firmware`: reference to the pointer where the read data should be stored e.g. the firmware version
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getHardwareVersion`**(uint8_t **hardware*, uint16_t *remote_id* = NULL)
Obtain hte hardware version. This function reads the reg:POZYX_HARDWARE_VER register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. hardware version
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getSelftest`**(uint8_t **selftest*, uint16_t *remote_id* = NULL)
Obtain the selftest result. This function reads the reg:POZYX_ST_RESULT register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. the selftest result
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getErrorCode`**(uint8_t **error_code*, uint16_t *remote_id* = NULL)
Obtain the error code. This function reads the reg:POZYX_ERRORCODE register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. the error code
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getInterruptStatus**(uint8_t *_interrupts_, uint16_t _remote_id_ = NULL)
Obtain the interrupt status. This function reads the reg:POZYX_INT_STATUS register.

See *waitForFlag*

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. the interrupt status
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getCalibrationStatus**(uint8_t *_calibration_status_, uint16_t _remote_id_ = NULL)
Obtain the calibration status. This function reads the reg:POZYX_CALIB_STATUS register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. calibration status
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getGPIO**(int _gpio_num_, uint8_t *_value_, uint16_t _remote_id_ = NULL)
Obtain the digital value on one of the GPIO pins. Function to retieve the value of the given General Purpose Input/output pin (GPIO) on the target device

Note In firmware version v0.9. The GPIO state cannot be read remotely.

Parameters

- `gpio_num`: the gpio pin to be set or retrieved. Possible values are 1, 2, 3 or 4.
- `value`: the pointer that stores the value for the GPIO.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setGPIO**(int _gpio_num_, uint8_t _value_, uint16_t _remote_id_ = NULL)
Set the digital value on one of the GPIO pins. Function to set or set the value of the given GPIO on the target device

Parameters

- `gpio_num`: the gpio pin to be set or retrieved. Possible values are 1, 2, 3 or 4.

- `value`: the value for the GPIO. Can be O (LOW) or 1 (HIGH).
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

void **`resetSystem`**(uint16_t *remote_id* = NULL)

Trigger a software reset of the Pozyx device. Function that will trigger the reset of the system. This will reload all configurations from flash memory, or to their default values.

See *clearConfiguration*, *saveConfiguration*

Parameters

- `remote_id`: optional parameter that determines the remote device to be used.

int **`setLed`**(int *led_num*, boolean *state*, uint16_t *remote_id* = NULL)

Function for turning the leds on and off. This function allows you to turn one of the 4 LEDs on the Pozyx board on or off. By default, the LEDs are controlled by the Pozyx system to give status information. This can be changed using the function setLedConfig.

See *setLedConfig*

Parameters

- `led_num`: the led number to be controlled, value between 1, 2, 3 or 4.
- `state`: the state to set the selected led to. Can be 0 (led is off) or 1 (led is on)
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getInterruptMask`**(uint8_t **mask*, uint16_t *remote_id* = NULL)

Function to obtain the interrupt configuration. This functions obtains the interrupt mask from the register reg:POZYX_INT_MASK. The interrupt mask is used to determine which event trigger an interrupt.

Parameters

- `mask`: the configured interrupt mask
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setInterruptMask**(uint8_t *mask*, uint16_t *remote_id* = NULL)

Function to configure the interrupts. Function to configure the interrupts by means of the interrupt mask from the register reg:POZYX_INT_MASK. The interrupt mask is used to determine which event trigger an interrupt.

Parameters

- `mask`: reference to the interrupt mask to be written. Bit-wise combinations of the following flags are allowed: POZYX_INT_MASK_ERR, POZYX_INT_MASK_POS, POZYX_INT_MASK_IMU, POZYX_INT_MASK_RX_DATA, POZYX_INT_MASK_FUNC.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getConfigModeGPIO**(int *gpio_num*, uint8_t **mode*, uint16_t *remote_id* = NULL)

Obtain the pull configuration of a GPIO pin. Function to obtain the configured pin mode of the GPIO for the given pin number. This is performed by reading from the reg:POZYX_CONFIG_GPIO1 up to reg:POZYX_CONFIG_GPIO4 register.

Parameters

- `gpio_num`: the pin to update
- `mode`: pointer to the mode of GPIO. Possible values POZYX_GPIO_DIGITAL_INPUT, POZYX_GPIO_PUSHPULL, POZYX_GPIO_OPENDRAIN
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getConfigPullGPIO**(int *gpio_num*, uint8_t **pull*, uint16_t *remote_id* = NULL)

Obtain the pull configuration of a GPIO pin. Function to obtain the configured pull resistor of the GPIO for the given pin number. This is performed by reading from the reg:POZYX_CONFIG_GPIO1 up to reg:POZYX_CONFIG_GPIO4 register.

Parameters

- `gpio_num`: the pin to update
- `pull`: pull configuration of GPIO. Possible values are POZYX_GPIO_NOPULL, POZYX_GPIO_PULLUP, POZYX_GPIO_PULLDOWN.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setConfigGPIO**(int *gpio_num*, int *mode*, int *pull*, uint16_t *remote_id* = NULL)

Configure the GPIOs. Function to set the configuration mode of the GPIO for the given pin number. This is performed by writing to the reg:POZYX_CONFIG_GPIO1 up to reg:POZYX_CONFIG_GPIO4 register.

Parameters

- `gpio_num`: the GPIO pin to update. Possible values are 1, 2, 3 or 4.
- `mode`: the mode of GPIO. Possible values POZYX_GPIO_DIGITAL_INPUT, POZYX_GPIO_PUSHPULL, POZYX_GPIO_OPENDRAIN
- `pull`: pull configuration of GPIO. Possible values are POZYX_GPIO_NOPULL, POZYX_GPIO_PULLUP, POZYX_GPIO_PULLDOWN.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setLedConfig**(uint8_t *config* = 0x0, uint16_t *remote_id* = NULL)

Configure the LEDs. This function configures the 6 LEDs on the pozyx device by writing to the register reg:POZYX_LED_CTRL. More specifically, the function configures which LEDs give system information. By default all the LEDs will give system information.

See *setLed*

Parameters

- `config`: default: the configuration to be set. See POZYX_LED_CTRL for details.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **configInterruptPin**(int *pin*, int *mode*, int *bActiveHigh*, int *bLatch*, uint16_t *remote_id* = NULL)

Configure the interrupt pin.

Parameters

- `pin`: pin id on the pozyx device, can be 1,2,3,4 (or 5 or 6 on the pozyx tag)
- `mode`: push-pull or pull-mode
- `bActiveHigh`: is the interrupt active level HIGH (i.e. 3.3V)
- `bLatch`: should the interrupt be a short pulse or should it latch until the interrupt status register is read

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.

int **saveConfiguration**(int *type*, uint8_t *registers*[] = NULL, int *num_registers* = 0, uint16_t *remote_id* = NULL)

Save (part of) the configuration to Flash memory. This functions stores (parts of) the configurable Pozyx settings in the non-volatile flash memory of the Pozyx device. This function will save the current settings and on the next startup of the Pozyx device these saved settings will be loaded automatically. settings from the flash memory. All registers that are writable, the anchor ids for positioning and the device list (which contains the anchor coordinates) can be saved.

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- `type`: this specifies what should be saved. Possible options are POZYX_FLASH_REGS, POZYX_FLASH_ANCHOR_IDS or POZYX_FLASH_NETWORK.
- `registers`: an option array that holds all the register addresses for which the value must be saved. All registers that are writable are allowed.
- `num_registers`: optional parameter that determines the length of the registers array.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.
- `POZYX_TIMEOUT`: function timed out.

int **saveRegisters**(uint8_t *registers*[] = NULL, int *num_registers* = 0, uint16_t *remote_id* = NULL)

Save registers to the flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- `registers`: an option array that holds all the register addresses for which the value must be saved. All registers that are writable are allowed.
- `num_registers`: optional parameter that determines the length of the registers array.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.
- `POZYX_TIMEOUT`: function timed out.

int **saveNetwork**(uint16_t *remote_id* = NULL)

Save the Pozyx's stored network to its flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.
- POZYX_TIMEOUT: function timed out.

int **saveAnchorIds**(uint16_t *remote_id* = NULL)

Save the Pozyx's stored anchor list to its flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- remote_id: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.
- POZYX_TIMEOUT: function timed out.

int **saveUWBSettings**(uint16_t *remote_id* = NULL)

Save the Pozyx's stored UWB settings to its flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- remote_id: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.
- POZYX_TIMEOUT: function timed out.

int **clearConfiguration**(uint16_t *remote_id* = NULL)

Clears the configuration. This function clears (part of) the configuration that was previously stored in the non-volatile Flash memory. The next startup of the Pozyx device will load the default configuration values for the registers, anchor_ids and network list.

Version Requires firmware version v1.0

See *saveConfiguration*

Parameters

- remote_id: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.

bool **isRegisterSaved**(uint8_t *regAddress*, uint16_t *remote_id* = NULL)

Verify if a register content is saved in the flash memory. This function verifies if a given register variable, specified by its address, is saved in flash memory.

Version Requires firmware version v1.0

Parameters

- `regAddress`: the register address to check
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `true(1)`: if the register variable is saved
- `false(0)`: if the register variable is not saved

int **getNumRegistersSaved**(uint16_t *remote_id* = NULL)
Return the number of register variables saved in flash memory.

Return the number of register variables saved in flash memory.

Parameters

- `remote_id`: optional parameter that determines the remote device to be used.

4.3 Communication

group **Communication_functions**

Functions

int **sendData**(uint16_t *destination*, uint8_t **pData*, int *size*)
Wirelessly transmit data to a remote pozyx device. This function combines writeTXBufferData and sendTXBufferData to write data to the transmit buffer and immediately transmit it.

Parameters

- `destination`: the network id of the device that should receive the data. A value of 0 will result in a broadcast
- `pData`: pointer to the data that should be transmitted
- `size`: number of bytes to transmit

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **writeTXBufferData**(uint8_t *data*[], int *size*, int *offset* = 0)
Write data bytes in the transmit buffer. This function writes bytes in the transmit buffer without sending it yet.

See *sendTXBufferData*

Parameters

- `data[]`: the array with data bytes
- `size`: size of the data array
- `offset`: The offset in the memory

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **sendTXBufferData**(uint16_t *destination* = 0x0)

Wirelessly transmit data. Wirelessly transmit the contents of the transmit buffer over UWB.

See *writeTXBufferData*

Parameters

- `destination`: the network id of the device that should receive the data. A value of 0 will result in a broadcast.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **readRXBufferData**(uint8_t **pData*, int *size*)

Read data bytes from receive buffer. This function reads the bytes from the receive buffer from the last received message.

See *getLastDataLength getLastNetworkId*

Parameters

- `pData`: pointer to where the data will be stored
- `size`: the number of bytes to read from the receive buffer.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getLastNetworkId**(uint16_t **network_id*, uint16_t *remote_id* = NULL)

Obtain the network id of the last message. This function identifies the source of the last message that was wirelessly received.

Parameters

- `network_id`: the pointer that stores the network_id
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getLastDataLength**(uint8_t **data_length*, uint16_t *remote_id* = NULL)

Obtain the number of bytes received. This function gives the number of bytes in the last message that was wirelessly received.

Parameters

- `data_length`: the pointer that stores the number of received bytes
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getNetworkId`**(uint16_t **network_id*)

Obtain the network id of the connected Pozyx device. The network id is a unique 16bit identifier determined based on the hardware components. When the system is reset the orignal value is restored

Parameters

- `network_id`: reference to the network_id pointer to store the data

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`setNetworkId`**(uint16_t *network_id*, uint16_t *remote_id* = NULL)

Overwrite the network id. This function overwrites the network id of the pozyx device either locally or remotely. The network id must be unique within a network. When the system is reset the orignal network id is restored.

Parameters

- `network_id`: the new network id
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getUWBSettings`**(UWB_settings_t **UWB_settings*, uint16_t *remote_id* = NULL)

Obtain the current UWB settings. Functions to retrieve the current UWB settings. In order to communicate, two pozyx devices must have exactly the same UWB settings.

Parameters

- `UWB_settings`: reference to the UWB settings object to store the data
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setUWBSettings**(UWB_settings_t **UWB_settings*, uint16_t *remote_id* = NULL)
Overwrite the UWB settings. This function overwrites the UWB settings such as UWB channel, gain, PRF, etc. In order to communicate, two pozyx devices must have exactly the same UWB settings. Upon reset, the default UWB settings will be loaded.

Parameters

- `UWB_settings`: reference to the new UWB settings. If the gain_db is set to 0, it will automatically load the default gain value for the given uwb paramters.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setUWBSettingsExceptGain**(UWB_settings_t **UWB_settings*, uint16_t *remote_id* = NULL)
Overwrite the UWB settings except the gain. This function overwrites the UWB settings such as UWB channel, PRF, etc. In order to communicate, two pozyx devices must have exactly the same UWB settings. Upon reset, the default UWB settings will be loaded.

Parameters

- `UWB_settings`: reference to the new UWB settings. If the gain_db is set to 0, it will automatically load the default gain value for the given uwb paramters.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setUWBChannel**(int *channel_num*, uint16_t *remote_id* = NULL)
Set the Ultra-wideband frequency channel. This function sets the ultra-wideband (UWB) frequency channel used for transmission and reception. For wireless communication, both the receiver and transmitter must be on the same UWB channel. More information can be found in the register reg:POZYX_UWB_CHANNEL.

Parameters

- `channel_num`: the channel number, possible values are 1, 2, 3, 4, 5 and 7.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getUWBChannel**(int **channel_num*, uint16_t *remote_id* = NULL)
Get the Ultra-wideband frequency channel. This function reads the ultra-wideband (UWB) frequency channel used for transmission and reception. For wireless communication, both the receiver and

transmitter must be on the same UWB channel. More information can be found in the register reg:POZYX_UWB_CHANNEL.

Parameters

- `channel_num`: the channel number currently set, possible values are 1, 2, 3, 4, 5 and 7.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setTxPower`**(float *txgain_dB*, uint16_t *remote_id* = NULL)
configure the UWB transmission power.

This function configures the UWB transmission power gain. The default value is different for each UWB channel. Setting a larger transmit power will result in a larger range. For increased communication range (which is two-way), both the transmitter and the receiver must set the appropriate transmit power. Changing the UWB channel will reset the power to the default value.

Note setting a large TX gain may cause the system to fall out of regulation. Applications that require regulations must set an appropriate TX gain to meet the spectral mask of your region. This can be verified with a spectrum analyzer.

Parameters

- `txgain_dB`: the transmission gain in dB. Possible values are between 0dB and 33.5dB, with a resolution of 0.5dB.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getTxPower`**(float **txgain_dB*, uint16_t *remote_id* = NULL)
Obtain the UWB transmission power.

This function obtains the configured UWB transmission power gain. The default value is different for each UWB channel. Changing the UWB channel will reset the TX power to the default value.

Parameters

- `txgain_dB`: the configured transmission gain in dB. Possible values are between 0dB and 33.5dB, with a resolution of 0.5dB.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

4.4 Device list

group **device_list**

Functions

int **getDeviceListSize**(uint8_t **device_list_size*, uint16_t *remote_id* = NULL)
Obtain the number of devices stored internally. The following function retrieves the number of devices stored in the device list.

See *doDiscovery*, *doAnchorCalibration*

Parameters

- `device_list_size`: the pointer that stores the device list size
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getDeviceIds**(uint16_t *devices*[], int *size*, uint16_t *remote_id* = NULL)
Obtain the network IDs from all the devices in the device list. Function to get all the network ids of the devices in the device list

Parameters

- `devices[]`: array that will be filled with the network ids
- `size`: the number of network IDs to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getAnchorIds**(uint16_t *anchors*[], int *size*, uint16_t *remote_id* = NULL)
Obtain the network IDs from all the anchors in the device list. Function to get all the network ids of the anchors in the device list

Parameters

- `devices[]`: array that will be filled with the network ids
- `size`: the number of network IDs to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getTagIds**(uint16_t *tags*[], int *size*, uint16_t *remote_id* = NULL)

Obtain the network IDs from all the tags in the device list. Function to get all the network ids of the tags in the device list

Parameters

- `tags[]`: array that will be filled with the network ids
- `size`: the number of network IDs to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doDiscovery**(int *type* = 0x0, int *slots* = 3, int *slot_duration* = 10)

Discover Pozyx devices in range. Function to wirelessly discover anchors/tags/all Pozyx devices in range. The discovered devices are added to the internal device list.

See *getDeviceListSize*, *getDeviceIds*

Parameters

- `type`: which type of device to discover. Possible values are POZYX_DISCOVERY_ANCHORS_ONLY: anchors only, POZYX_DISCOVERY_TAGS_ONLY: tags only or POZYX_DISCOVERY_ALL_DEVICES: anchors and tags
- `slots`: The number of slots to wait for a response of an undiscovered device
- `slot_duration`: Time duration of an idle slot

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doAnchorCalibration**(int *dimension* = POZYX_2D, int *num_measurements* = 10, int *num_anchors* = 0, uint16_t *anchors*[] = NULL, int32_t *heights*[] = NULL)

Automatically obtain the relative anchor positions. WARNING: This is currently experimental and will be improved in the next firmware version! This function triggers the automatic anchor calibration to obtain the relative coordinates of up to 6 pozyx devices in range. This function can be used for quickly setting up the positioning system. The procedure may take several hundres of milliseconds depending on the number of devices in range and the number of range measurements requested. During the calibration proces LED 2 will be turned on. At the end of calibration the corresponding bit in the reg:POZYX_CALIB_STATUS register will be set. The resulting coordinates are stored in the internal device list. Please read the tutorial Ready to Localize to learn how to use this function.

Parameters

- `type`: dimension of the calibration, can be POZYX_2D or POZYX_2_5D
- `measurements`: The number of measurements per link. Recommended 10. Theoretically, a larger number should result in better calibration accuracy.
- `anchor_num`: The number of anchors in the anchors[] array
- `anchors[]`: The anchors that determine the axis (see datasheet)

- `heights`: The heights in mm of the anchors in the anchors[] array (only used for POZYX_2_5D)

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`clearDevices`**(uint16_t *remote_id* = NULL)
Empty the internal list of devices. This function empties the internal list of devices.

Parameters

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`addDevice`**(device_coordinates_t *device_coordinates*, uint16_t *remote_id* = NULL)
Manualy adds a device to the device list. This function can be used to manually add a device and its coordinates to the device list. Once the device is added, it can be used for positioning.

Parameters

- `device_coordinates`: the device information to be added
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getDeviceCoordinates`**(uint16_t *device_id*, coordinates_t **coordinates*, uint16_t *remote_id* = NULL)
Retrieve the stored coordinates of a device. This function retrieves the device coordinates stored in the internal device list.

Parameters

- `device_id`: device from which the information needs to be retrieved
- `device_coordinates`: data object to store the information
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

4.5 Positioning

group **Positioning_functions**

Functions

int **getCoordinates**(coordinates_t **coordinates*, uint16_t *remote_id* = NULL)
Obtain the last coordinates of the device. Retrieve the last coordinates of the device or the remote device. Note that this function does not trigger positioning.

See *doPositioning*, *doRemotePositioning*

Parameters

- `coordinates`: reference to the coordinates pointer object.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setCoordinates**(coordinates_t *coordinates*, uint16_t *remote_id* = NULL)
Set the coordinates of the device. Manually set the coordinates of the device or the remote device.

See *getCoordinates*

Parameters

- `coordinates`: coordinates to be set
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getPositionError**(pos_error_t **pos_error*, uint16_t *remote_id* = NULL)
Obtain the last estimated position error covariance information. Retrieve the last error covariance of the position for the device or the remote device. Note that this function does not trigger positioning.

Parameters

- `pos_error`: reference to the pos error object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setPositioningAnchorIds**(uint16_t *anchors*[], int *anchor_num*, uint16_t *remote_id* = NULL)
Manually set which anchors to use for positioning. Function to manually set which anchors to use for positioning by calling the register function reg:POZYX_POS_SET_ANCHOR_IDS. Notice that the anchors specified are only used if this is configured with setSelectionOfAnchors. Furthermore, the anchors specified in this functions must be available in the device list with their coordinates.

See *setSelectionOfAnchors*, *getPositioningAnchorIds*

Parameters

- `anchors[]`: an array with network id's of anchors to be used
- `anchor_num`: the number of anchors write
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getPositioningAnchorIds**(uint16_t *anchors*[], int *anchor_num*, uint16_t *remote_id* = NULL)
Obtain which anchors used for positioning. Function to retrieve the anchors that used for positioning by calling the register function reg:POZYX_POS_GET_ANCHOR_IDS. When in automatic anchor selection mode, the chosen anchors are listed here.

See *setSelectionOfAnchors*, *setPositioningAnchorIds*

Parameters

- `anchors[]`: an array with network id's of anchors manually set
- `anchor_num`: the number of anchors to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getUpdateInterval**(uint16_t **ms*, uint16_t *remote_id* = NULL)
Read the update interval continuous positioning. This function reads the configured update interval for continuous positioning from the register reg:POZYX_POS_INTERVAL.

See *setUpdateInterval*

Parameters

- `ms`: pointer to the update interval in milliseconds. A value of 0 means that continuous positioning is disabled.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

- POZYX_TIMEOUT: function timed out, no response received.

int **setUpdateInterval** (uint16_t *ms*, uint16_t *remote_id* = NULL)

Configure the udpate interval for continuous positioning. This function configures the update interval by writing to the register reg:POZYX_POS_INTERVAL. By writing a valid value, the system will start continuous positioning which will generate a position estimate on regular intervals. This will generate a POZYX_INT_STATUS_POS interrupt when interrupts are enabled.

See *getUpdateInterval*

Parameters

- ms: update interval in milliseconds. The update interval must be larger or equal to 100ms.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **getRangingProtocol** (uint8_t **protocol*, uint16_t *remote_id* = NULL)

Obtain the configured ranging protocol. This function obtains the configured ranging protocol by reading from the reg:POZYX_RANGE_PROTOCOL register.

See *doRanging*, *setRangingProtocol*

Parameters

- protocol: pointer to the variable holding the ranging protocol used when ranging. Possible values for the ranging protocol are POZYX_RANGE_PROTOCOL_FAST and POZYX_RANGE_PROTOCOL_PRECISION.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **setRangingProtocol** (uint8_t *protocol*, uint16_t *remote_id* = NULL)

Configure the ranging protocol. This function configures the ranging protocol by writing to the reg:POZYX_RANGE_PROTOCOL register.

See *doRanging*, *getRangingProtocol*

Parameters

- protocol: Ranging protocol used when ranging. Possible values for the ranging protocol are POZYX_RANGE_PROTOCOL_FAST and POZYX_RANGE_PROTOCOL_PRECISION.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getPositionFilterStrength**(uint8_t **filter_strength*, uint16_t *remote_id* = NULL)

Obtain the configured positioning filter strength. This function obtains the configured positioning filter strength by reading from the reg:POZYX_POS_FILTER register.

See *getPositionFilterType*, *setPositionFilter*

Parameters

- `filter_strength`: pointer to the variable holding the filter strength used in the built-in filter. This strength is the amount of previous samples used in positioning. Possible values for the position filter strength is between 0 and 15 samples.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getPositionFilterType**(uint8_t **filter_type*, uint16_t *remote_id* = NULL)

Obtain the configured positioning filter type. This function obtains the configured positioning filter type by reading from the reg:POZYX_POS_FILTER register.

See *getPositionFilterStrength*, *setPositionFilter*

Parameters

- `filter_type`: pointer to the variable holding the filter type data. Possible values for the positioning algorithm are FILTER_TYPE_NONE, FILTER_TYPE_FIR, FILTER_TYPE_MOVINGMEDIAN, and FILTER_TYPE_MOVINGAVERAGE.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setPositionFilter**(uint8_t *filter_type*, uint8_t *filter_strength*, uint16_t *remote_id* = NULL)

Configure the positioning filter. This function configures the positioning filter by writing to the reg:POZYX_POS_FILTER register.

See *getPositionFilterStrength*, *getPositionFilterType*

Parameters

- `filter_type`: Filter type used when positioning. Possible values for the positioning algorithm are FILTER_TYPE_NONE, FILTER_TYPE_FIR, FILTER_TYPE_MOVINGMEDIAN, and FILTER_TYPE_MOVINGAVERAGE.
- `filter_strength`: Filter strength used when positioning. Possible values for the position filter strength is between 0 and 15 samples.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.

- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getPositionAlgorithm**(uint8_t *_algorithm_, uint16_t _remote_id_ = NULL)

Obtain the configured positioning algorithm. This function obtains the configured positioning algorithm by reading from the reg:POZYX_POS_ALG register.

See _getPositionDimension_, _setPositionAlgorithm_

Parameters

- algorithm: pointer to the variable holding the algorithm used to determine position. Possible values for the positioning algorithm are POZYX_POS_ALG_UWB_ONLY and POZYX_POS_ALG_LS.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getPositionDimension**(uint8_t *_dimension_, uint16_t _remote_id_ = NULL)

Obtain the configured positioning dimension. This function obtains the configuration of the physical dimension by reading from the reg:POZYX_POS_ALG register.

See _getPositionAlgorithm_, _setPositionAlgorithm_

Parameters

- dimension: pointer to physical dimension used for the algorithm. Possible values for the dimension are POZYX_3D, POZYX_2D or POZYX_2_5D.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **setPositionAlgorithm**(int _algorithm_ = POZYX_POS_ALG_UWB_ONLY, int _dimension_ = 0x0, uint16_t _remote_id_ = NULL)

Configure the positioning algorithm. This function configures the positioning algorithm and the desired physical dimension by writing to the register reg:POZYX_POS_ALG.

See _getPositionAlgorithm_, _getPositionDimension_

Parameters

- algorithm: algorithm used to determine the position. Possible values are POZYX_POS_ALG_UWB_ONLY and POZYX_POS_ALG_LS.
- dimension: physical dimension used for the algorithm. Possible values are POZYX_3D, POZYX_2D or POZYX_2_5D.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **getAnchorSelectionMode**(uint8_t **mode*, uint16_t *remote_id* = NULL)

Obtain the anchor selection mode. This function reads the anchor selection mode bit from the register reg:POZYX_POS_NUM_ANCHORS. This bit describes how the anchors are selected for positioning, either manually or automatically.

Parameters

- mode: reference to the anchor selection mode. Possible results are POZYX_ANCHOR_SEL_MANUAL for manual anchor selection or POZYX_ANCHOR_SEL_AUTO for automatic anchor selection.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getNumberOfAnchors**(uint8_t **nr_anchors*, uint16_t *remote_id* = NULL)

Obtain the configured number of anchors used for positioning. This functions reads out the reg:POZYX_POS_NUM_ANCHORS register to obtain the number of anchors that are being used for positioning.

Parameters

- nr_anchors: reference to the number of anchors
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **setSelectionOfAnchors**(int *mode*, int *nr_anchors*, uint16_t *remote_id* = NULL)

Configure how many anchors are used for positioning and how they are selected. This function configures the number of anchors used for positioning. Theoretically, a larger number of anchors leads to better positioning performance. However, in practice this is not always the case. The more anchors used for positioning, the longer the positioning process will take. Furthermore, it can be chosen which anchors are to be used: either a fixed set given by the user, or an automatic selection between all the anchors in the internal device list.

See *setPositioningAnchorIds* to set the anchor IDs in manual anchor selection mode.

Parameters

- mode: describes how the anchors are selected. Possible values are POZYX_ANCHOR_SEL_MANUAL for manual anchor selection or POZYX_ANCHOR_SEL_AUTO for automatic anchor selection.

- `nr_anchors`: the number of anchors to use for positioning. Must be larger than 2 and smaller than 16.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getOperationMode`**(uint8_t **mode*, uint16_t *remote_id* = NULL)

Obtain the operation mode of the device. This function obtains the operation mode (anchor or tag) by reading from the register reg:POZYX_OPERATION_MODE. This operation mode is independent of the hardware it is on and will have it's effect when performing discovery or auto calibration.

See *setOperationMode*

Parameters

- `mode`: The mode of operations POZYX_ANCHOR_MODE or POZYX_TAG_MODE
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setOperationMode`**(uint8_t *mode*, uint16_t *remote_id* = NULL)

Define if the device operates as a tag or an anchor. This function defines how the device should operate (as an anchor or tag) by writing to the register reg:POZYX_OPERATION_MODE. This operation mode is independent of the hardware it is on and will have it's effect when performing discovery or auto calibration. This function overrules the jumper that is present on the board.

See *getOperationMode*

Parameters

- `mode`: The mode of operations POZYX_ANCHOR_MODE or POZYX_TAG_MODE
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

String **`getSystemError`**(uint16_t *remote_id* = NULL)

Get the textual system error. This function reads out the reg:POZYX_ERRORCODE register and converts the error code to a textual message.

Parameters

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `String`: the textual error

int **doPositioning**(coordinates_t **coordinates*, uint8_t *dimension*, int32_t *height*, uint8_t *algorithm*)

int **doPositioning**(coordinates_t **position*, uint8_t *dimension*, int32_t *height* = 0)

Obtain the coordinates. This function triggers the positioning algorithm to perform positioning with the given parameters. By default it will automatically select 4 anchors from the internal device list. It will then perform ranging with these anchors and use the results to compute the coordinates. This function requires that the coordinates for the anchors are stored in the internal device list.

Please read the tutorial ready to localize to get started with positioning.

See *doRemotePositioning*, *doAnchorCalibration*, *addDevice*, *setSelectionOfAnchors*, *setPositionAlgorithm*

Parameters

- `position`: data object to store the result
- `height`: optional parameter that is used for POZYX_2_5D to give the height in mm of the tag

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doRemotePositioning**(uint16_t *remote_id*, coordinates_t **coordinates*, uint8_t *dimension*, int32_t *height*, uint8_t *algorithm*)

int **doRemotePositioning**(uint16_t *remote_id*, coordinates_t **coordinates*, uint8_t *dimension*, int32_t *height* = 0)

Obtain the coordinates of a remote device. Don't use with 2.5D!

This function triggers the positioning algorithm on a remote pozyx device to perform positioning with the given parameters. By default it will automatically select 4 anchors from the internal device list on the remote device. The device will perform ranging with the anchors and use the results to compute the coordinates. This function requires that the coordinates for the anchors are stored in the internal device list on the remote device. After positioning is completed, the remote device will automatically transmit the result back.

See *doPositioning*, *addDevice*, *setSelectionOfAnchors*, *setPositionAlgorithm*

Parameters

- `remote_id`: the remote device that will do the positioning
- `position`: data object to store the result
- `height`: optional parameter that is used for POZYX_2_5D to give the height in mm of the tag

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doRanging**(uint16_t *destination*, device_range_t **range*)

Trigger ranging with a remote device. This function performs ranging with a remote device using the UWB signals.

See *doRemoteRanging*, *getDeviceRangeInfo*

Parameters

- `destination`: the target device to do ranging with

- `range`: the pointer to where the resulting data will be stored

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doRemoteRanging**(uint16_t *device_from*, uint16_t *device_to*, device_range_t **range*)
Trigger ranging between two remote devivces. Function allowing to trigger ranging between two remote devices A and B. The ranging data is collected by device A and transmitted back to the local device.

See *doRanging*, *getDeviceRangeInfo*

Parameters

- `device_from`: device A that will initiate the range request.
- `device_to`: device B that will respond to the range request.
- `range`: the pointer to where the resulting data will be stored

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getDeviceRangeInfo**(uint16_t *device_id*, device_range_t **device_range*, uint16_t *remote_id* = NULL)
Retrieve stored ranging information. Functions to retrieve the latest ranging information (i.e., the distance, signal strength and timestamp) with respect to a remote device. This function does not trigger ranging.

See *doRanging*, *doRemoteRanging*

Parameters

- `device_id`: network id of the device for which range information is requested
- `device_range`: data object to store the information
- `remote_id`: optional parameter that determines the remote device where this function is called.

4.6 Sensor Data

group **sensor_data**

Functions

int **getSensorMode**(uint8_t **sensor_mode*, uint16_t *remote_id* = NULL)
Retrieve the configured sensor mode. This function reads out the register reg:POZYX_SENSORS_MODE which describes the configured sensor mode.

Parameters

- `sensor_mode`: reference to the sensor mode
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **setSensorMode**(uint8_t *sensor_mode*, uint16_t *remote_id* = NULL)

Configure the sensor mode. This function reads out the register reg:POZYX_SENSORS_MODE which describes the configured sensor mode.

Parameters

- sensor_mode: the desired sensor mode.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **getRawSensorData**(sensor_raw_t **sensor_raw*, uint16_t *remote_id* = NULL)

Obtain all raw sensor data at once as it's stored in the registers. This functions reads out the pressure, acceleration, magnetic field strength, angular velocity, orientation in Euler angles, the orientation as a quaternion, the linear acceleration, the gravity vector and temperature.

Parameters

- sensor_raw: reference to the sensor_raw object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getAllSensorData**(sensor_data_t **sensor_data*, uint16_t *remote_id* = NULL)

Obtain all sensor data at once. This functions reads out the pressure, acceleration, magnetic field strength, angular velocity, orientation in Euler angles, the orientation as a quaternion, the linear acceleration, the gravity vector and temperature.

Parameters

- sensor_data: reference to the sensor_data object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getPressure_Pa**(float32_t **pressure*, uint16_t *remote_id* = NULL)

Obtain the atmospheric pressure in Pascal. This function reads out the pressure starting from the register POZYX_PRESSURE. The maximal update rate is 10Hz. The units are Pa.

Parameters

- `pressure`: reference to the pressure variable
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getMaxLinearAcceleration`**(uint16_t **max_lin_acc*, uint16_t *remote_id* = NULL)

Obtain the max linear acceleration This registers functions similarly to the interrupt and error registers in that it clears the register's value upon reading. This is the max linear acceleration since the last read.

Parameters

- `max_lin_acc`: pointer to a variable that will hold the max linear acceleration
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getAcceleration_mg`**(acceleration_t **acceleration*, uint16_t *remote_id* = NULL)

Obtain the 3D acceleration vector in mg. This function reads out the acceleration data starting from the register reg:POZYX_ACCEL_X. The maximal update rate is 100Hz. The units are mg. The vector is expressed in body coordinates (i.e., along axes of the device).

Parameters

- `acceleration`: reference to the acceleration object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getMagnetic_uT`**(magnetic_t **magnetic*, uint16_t *remote_id* = NULL)

Obtain the 3D magnetic field strength vector in µTesla. This function reads out the magnetic field strength data starting from the register reg:POZYX_MAGN_X. The maximal update rate is 100Hz. The vector is expressed in body coordinates (i.e., along axes of the device).

Parameters

- `magnetic`: reference to the magnetic object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.

- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getAngularVelocity_dps**(angular_vel_t **angular_vel*, uint16_t *remote_id* = NULL)

Obtain the 3D angular velocity vector degrees per second. This function reads out the angular velocity from the gyroscope using the register reg:POZYX_GYRO_X. The maximal update rate is 100Hz. The rotations are expressed in body coordinates (i.e., the rotations around the axes of the device).

Parameters

- angular_vel: reference to the angular velocity object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getEulerAngles_deg**(euler_angles_t **euler_angles*, uint16_t *remote_id* = NULL)

Obtain the orientation in Euler angles in degrees. This function reads out the Euleur angles: Yaw, Pitch and Roll that represents the 3D orientation of the device

Parameters

- euler_angles: reference to the euler_angles object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getQuaternion**(quaternion_t **quaternion*, uint16_t *remote_id* = NULL)

Obtain the orientation in quaternions. This function reads out the 4 coordinates of the quaternion that represents the 3D orientation of the device. The quaternions are unitless and normalized.

Parameters

- quaternion: reference to the quaternion object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getLinearAcceleration_mg**(linear_acceleration_t **linear_acceleration*, uint16_t *remote_id* = NULL)

Obtain the 3D linear acceleration in mg. This function reads out the linear acceleration data starting from the register reg:POZYX_LIA_X. The Linear acceleration is the acceleration compensated for gravity. The

maximal update rate is 100Hz. The units are mg. The vector is expressed in body coordinates (i.e., along axes of the device).

Parameters

- `linear_acceleration`: reference to the acceleration object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getGravityVector_mg`**(gravity_vector_t **gravity_vector*, uint16_t *remote_id* = NULL)

Obtain the 3D gravity vector in mg. This function reads out the gravity vector coordinates starting from the register reg:POZYX_GRAV_X. The maximal update rate is 100Hz. The units are mg. The vector is expressed in body coordinates (i.e., along axes of the device). This vector always points to the ground, regardless of the orientation.

Parameters

- `gravity_vector`: reference to the gravity_vector object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getTemperature_c`**(float32_t **temperature*, uint16_t *remote_id* = NULL)

Obtain the temperature in degrees Celcius. This function reads out the temperature from the register reg:POZYX_TEMPERATURE. This function is unsupported in firmware version v0.9.

Parameters

- `temperature`: reference to the temperature variable
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

CHAPTER

FIVE

POZYX CLASS FULL

5.1 Members

class PozyxClass
Provides an interface to an attached Pozyx shield.

Public Static Functions

boolean **waitForFlag_safe**(uint8_t *interrupt_flag*, int *timeout_ms*, uint8_t **interrupt* = NULL)
This function calls the waitForFlag function in polling mode. After this, the previous mode is reset.

Parameters

- `interrupt_flag`: the exepected Pozyx interrupt. Possible values are POZYX_INT_STATUS_ERR, POZYX_INT_STATUS_POS, POZYX_INT_STATUS_IMU, POZYX_INT_STATUS_RX_DATA, POZYX_INT_STATUS_FUNC, or combinations.
- `timeout_ms`: maximum waiting time in milliseconds for flag to occur
- `interrupt`: a pointer that will contain the value of the interrupt status register

Return Value

- `#true`: event occured.
- `#false`: event did not occur, this function timed out.

int **regRead**(uint8_t *reg_address*, uint8_t **pData*, int *size*)
Read from the registers of the connected Pozyx shield.

Reads a number of bytes from the specified pozyx register address using I2C

Parameters

- `reg_address`: the specific register address to start reading from
- `pData`: a pointer to the data thas will be read
- `size`: the number of bytes to read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **regWrite**(uint8_t *reg_address*, uint8_t **pData*, int *size*)
Write to the registers of the connected Pozyx shield.

Writes a number of bytes to the specified pozyx register address using I2C

Parameters

- `reg_address`: the specific register address to start writing to
- `pData`: a pointer to the data thas will be written
- `size`: the number of bytes to write

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **regFunction**(uint8_t *reg_address*, uint8_t **params* = NULL, int *param_size* = 0, uint8_t **pData* = NULL, int *size* = 0)
Call a register funcion on the connected Pozyx shield.

Call a register function using i2c with given parameters, the data from the function is stored in pData

Parameters

- `reg_address`: the specific register address of the function
- `params`: this is the pointer to a parameter array required for the specific function that is called
- `param_size`: the number of bytes in the params array
- `pData`: a pointer to the data thas will be read
- `size`: the number of bytes that will be stored in pData

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **remoteRegWrite**(uint16_t *destination*, uint8_t *reg_address*, uint8_t **pData*, int *size*)
Write to the registers on a remote Pozyx device (anchor or tag).

Wirelessly write a number of bytes to a specified register address on a remote Pozyx device using UWB.

Parameters

- `destination`: this is the network id of the receiving Pozyx tag
- `reg_address`: the specific register address to start writing to
- `pData`: a pointer to the data thas will be written
- `size`: the number of bytes to write

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **remoteRegRead**(uint16_t *destination*, uint8_t *reg_address*, uint8_t **pData*, int *size*)
Read from the registers on a remote Pozyx device (anchor or tag).

Wirelessly read a number of bytes from a specified register address on a remote Pozyx device using UWB.

Parameters

- `destination`: this is the network id of the receiving Pozyx tag
- `reg_address`: the specific register address to start reading from
- `pData`: a pointer to the data thas will be read
- `size`: the number of bytes to read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **remoteRegFunction**(uint16_t *destination*, uint8_t *reg_address*, uint8_t **params* = NULL, int *param_size* = 0, uint8_t **pData* = NULL, int *size* = 0)

Call a register funcion on a remote Pozyx device (anchor or tag).

Parameters

- `destination`: this is the network id of the receiving Pozyx tag
- `reg_address`: the specific register address of the function
- `params`: this is the pointer to a parameter array required for the specific function that is called
- `param_size`: the number of bytes in the params array
- `pData`: a pointer to the data thas will be read
- `size`: the number of bytes that will be stored in pData

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **remoteRegFunctionWithoutCheck**(uint16_t *destination*, uint8_t *reg_address*, uint8_t **params* = NULL, int *param_size* = 0, uint8_t **pData* = NULL, int *size* = 0)

Wait until the Pozyx shields has raised a specfic event flag or until timeout. This functions halts the process flow until a specific event flag was raised by the Pozyx shield. The event flag is checked by reading the contents of the reg:POZYX_INT_STATUS register. This function can work in both polled and interupt mode

Parameters

- `interrupt_flag`: the exepected Pozyx interrupt. Possible values are POZYX_INT_STATUS_ERR, POZYX_INT_STATUS_POS, POZYX_INT_STATUS_IMU, POZYX_INT_STATUS_RX_DATA, POZYX_INT_STATUS_FUNC, or combinations.
- `timeout_ms`: maximum waiting time in milliseconds for flag to occur
- `interrupt`: a pointer that will contain the value of the interrupt status register

Return Value

- `#true`: event occured.
- `#false`: event did not occur, this function timed out.

boolean **waitForFlag**(uint8_t *interrupt_flag*, int *timeout_ms*, uint8_t **interrupt* = NULL)

Does the same as the remoteRegFunction, but doesn't wait for nearly as long and doesn't care about whether the function worked.

Parameters

- `interrupt_flag`: the exepected Pozyx interrupt. Possible values are POZYX_INT_STATUS_ERR, POZYX_INT_STATUS_POS, POZYX_INT_STATUS_IMU, POZYX_INT_STATUS_RX_DATA, POZYX_INT_STATUS_FUNC, or combinations.
- `timeout_ms`: maximum waiting time in milliseconds for flag to occur
- `interrupt`: a pointer that will contain the value of the interrupt status register

Return Value

- `#true`: event occured.
- `#false`: event did not occur, this function timed out.

int **begin**(boolean *print_result* = false, int *mode* = MODE_INTERRUPT, int *interrupts* = POZYX_INT_MASK_ALL, int *interrupt_pin* = POZYX_INT_PIN0)

Initiates the Pozyx shield. This function initializes the pozyx device. It will verify that the device is functioning correctly by means of the self-test, and it will configure the interrupts. See the register reg:POZYX_INT_MASK for more details about the interrupts.

Parameters

- `print_result`: outputs the result of the function to the Serial output
- `mode`: The modus of the system: MODE_POLLING or MODE_INTERRUPT
- `interrupts`: defines which events trigger interrupts. This field is only required for MODE_INTERRUPT. Possible values are bit-wise combinations of POZYX_INT_MASK_ERR, POZYX_INT_MASK_POS, POZYX_INT_MASK_IMU, POZYX_INT_MASK_RX_DATA and POZYX_INT_MASK_FUNC. Use POZYX_INT_MASK_ALL to trigger on all events.
- `interrupt_pin`: Pozyx interrupt pin: POZYX_INT_PIN0 or POZYX_INT_PIN1. This field is only required for MODE_INTERRUPT.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getRead**(uint8_t *reg_address*, uint8_t **pData*, int *size*, uint16_t *remote_id* = NULL)

Read from the registers on a local or remote Pozyx device (anchor or tag).

Parameters

- `reg_address`: the specific register address to start reading from
- `pData`: a pointer to the data thas will be read
- `size`: the number of bytes to read
- `remote_id`: this is the network id of the remote Pozyx tag, if used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

- POZYX_TIMEOUT: function timed out, no response received.

int **setWrite**(uint8_t *reg_address*, uint8_t **pData*, int *size*, uint16_t *remote_id* = NULL)
Write to the registers of a local remote Pozyx device (anchor or tag).

Parameters

- reg_address: the specific register address to start writing to
- pData: a pointer to the data thas will be written
- size: the number of bytes to write
- remote_id: this is the network id of the remote Pozyx tag, if used.

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **useFunction**(uint8_t *reg_address*, uint8_t **params* = NULL, int *param_size* = 0, uint8_t **pData* = NULL, int *size* = 0, uint16_t *remote_id* = NULL)
Call a register funcion on a local or remote Pozyx device (anchor or tag).

Parameters

- reg_address: the specific register address of the function
- params: this is the pointer to a parameter array required for the specific function that is called
- param_size: the number of bytes in the params array
- pData: a pointer to the data thas will be read
- size: the number of bytes that will be stored in pData
- remote_id: this is the network id of the remote Pozyx tag, if used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **sendData**(uint16_t *destination*, uint8_t **pData*, int *size*)
Wirelessly transmit data to a remote pozyx device. This function combines writeTXBufferData and sendTXBufferData to write data to the transmit buffer and immediately transmit it.

Parameters

- destination: the network id of the device that should receive the data. A value of 0 will result in a broadcast
- pData: pointer to the data that should be transmitted
- size: number of bytes to transmit

Return Value

- POZYX_SUCCESS: success.

- POZYX_FAILURE: function failed.

int **writeTXBufferData**(uint8_t *data*[], int *size*, int *offset* = 0)

Write data bytes in the transmit buffer. This function writes bytes in the transmit buffer without sending it yet.

See *sendTXBufferData*

Parameters

- data[]: the array with data bytes
- size: size of the data array
- offset: The offset in the memory

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **sendTXBufferData**(uint16_t *destination* = 0x0)

Wirelessly transmit data. Wirelessly transmit the contents of the transmit buffer over UWB.

See *writeTXBufferData*

Parameters

- destination: the network id of the device that should receive the data. A value of 0 will result in a broadcast.

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **readRXBufferData**(uint8_t **pData*, int *size*)

Read data bytes from receive buffer. This function reads the bytes from the receive buffer from the last received message.

See *getLastDataLength getLastNetworkId*

Parameters

- pData: pointer to where the data will be stored
- size: the number of bytes to read from the receive buffer.

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **getLastNetworkId**(uint16_t **network_id*, uint16_t *remote_id* = NULL)

Obtain the network id of the last message. This function identifies the source of the last message that was wirelessly received.

Parameters

- network_id: the pointer that stores the network_id

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getLastDataLength`**(uint8_t **data_length*, uint16_t *remote_id* = NULL)
Obtain the number of bytes received. This function gives the number of bytes in the last message that was wirelessly received.

Parameters

- `data_length`: the pointer that stores the number of received bytes
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getNetworkId`**(uint16_t **network_id*)
Obtain the network id of the connected Pozyx device. The network id is a unique 16bit identifier determined based on the hardware components. When the system is reset the orignal value is restored

Parameters

- `network_id`: reference to the network_id pointer to store the data

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`setNetworkId`**(uint16_t *network_id*, uint16_t *remote_id* = NULL)
Overwrite the network id. This function overwrites the network id of the pozyx device either locally or remotely. The network id must be unique within a network. When the system is reset the orignal network id is restored.

Parameters

- `network_id`: the new network id
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getUWBSettings`**(UWB_settings_t **UWB_settings*, uint16_t *remote_id* = NULL)
Obtain the current UWB settings. Functions to retrieve the current UWB settings. In order to communicate, two pozyx devices must have exactly the same UWB settings.

Parameters

- `UWB_settings`: reference to the UWB settings object to store the data
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setUWBSettings`**(UWB_settings_t **UWB_settings*, uint16_t *remote_id* = NULL)

Overwrite the UWB settings. This function overwrites the UWB settings such as UWB channel, gain, PRF, etc. In order to communicate, two pozyx devices must have exactly the same UWB settings. Upon reset, the default UWB settings will be loaded.

Parameters

- `UWB_settings`: reference to the new UWB settings. If the gain_db is set to 0, it will automatically load the default gain value for the given uwb paramters.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setUWBSettingsExceptGain`**(UWB_settings_t **UWB_settings*, uint16_t *remote_id* = NULL)

Overwrite the UWB settings except the gain. This function overwrites the UWB settings such as UWB channel, PRF, etc. In order to communicate, two pozyx devices must have exactly the same UWB settings. Upon reset, the default UWB settings will be loaded.

Parameters

- `UWB_settings`: reference to the new UWB settings. If the gain_db is set to 0, it will automatically load the default gain value for the given uwb paramters.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setUWBChannel`**(int *channel_num*, uint16_t *remote_id* = NULL)

Set the Ultra-wideband frequency channel. This function sets the ultra-wideband (UWB) frequency channel used for transmission and reception. For wireless communication, both the receiver and transmitter must be on the same UWB channel. More information can be found in the register reg:POZYX_UWB_CHANNEL.

Parameters

- `channel_num`: the channel number, possible values are 1, 2, 3, 4, 5 and 7.

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getUWBChannel`**(int **channel_num*, uint16_t *remote_id* = NULL)

Get the Ultra-wideband frequency channel. This function reads the ultra-wideband (UWB) frequency channel used for transmission and reception. For wireless communication, both the receiver and transmitter must be on the same UWB channel. More information can be found in the register reg:POZYX_UWB_CHANNEL.

Parameters

- `channel_num`: the channel number currently set, possible values are 1, 2, 3, 4, 5 and 7.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setTxPower`**(float *txgain_dB*, uint16_t *remote_id* = NULL)

configure the UWB transmission power.

This function configures the UWB transmission power gain. The default value is different for each UWB channel. Setting a larger transmit power will result in a larger range. For increased communication range (which is two-way), both the transmitter and the receiver must set the appropriate transmit power. Changing the UWB channel will reset the power to the default value.

Note setting a large TX gain may cause the system to fall out of regulation. Applications that require regulations must set an appropriate TX gain to meet the spectral mask of your region. This can be verified with a spectrum analyzer.

Parameters

- `txgain_dB`: the transmission gain in dB. Possible values are between 0dB and 33.5dB, with a resolution of 0.5dB.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getTxPower`**(float **txgain_dB*, uint16_t *remote_id* = NULL)

Obtain the UWB transmission power.

This function obtains the configured UWB transmission power gain. The default value is different for each UWB channel. Changing the UWB channel will reset the TX power to the default value.

Parameters

- `txgain_dB`: the configured transmission gain in dB. Possible values are between 0dB and 33.5dB, with a resolution of 0.5dB.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getWhoAmI`**(uint8_t **whoami*, uint16_t *remote_id* = NULL)
Obtain the who_am_i value. This function reads the reg:POZYX_WHO_AM_I register.

Parameters

- `whoami`: reference to the pointer where the read data should be stored e.g. whoami
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getFirmwareVersion`**(uint8_t **firmware*, uint16_t *remote_id* = NULL)
Obtain the firmware version. This function reads the reg:POZYX_FIRMWARE_VER register.

Parameters

- `firmware`: reference to the pointer where the read data should be stored e.g. the firmware version
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getHardwareVersion`**(uint8_t **hardware*, uint16_t *remote_id* = NULL)
Obtain hte hardware version. This function reads the reg:POZYX_HARDWARE_VER register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. hardware version
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getSelftest**(uint8_t *_selftest_, uint16_t _remote_id_ = NULL)
Obtain the selftest result. This function reads the reg:POZYX_ST_RESULT register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. the selftest result
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getErrorCode**(uint8_t *_error_code_, uint16_t _remote_id_ = NULL)
Obtain the error code. This function reads the reg:POZYX_ERRORCODE register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. the error code
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getInterruptStatus**(uint8_t *_interrupts_, uint16_t _remote_id_ = NULL)
Obtain the interrupt status. This function reads the reg:POZYX_INT_STATUS register.

See *waitForFlag*

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. the interrupt status
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getCalibrationStatus**(uint8_t *_calibration_status_, uint16_t _remote_id_ = NULL)
Obtain the calibration status. This function reads the reg:POZYX_CALIB_STATUS register.

Parameters

- `data`: reference to the pointer where the read data should be stored e.g. calibration status
- `remote_id`: optional parameter that determines the remote device to be read

Return Value

- `POZYX_SUCCESS`: success.

- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getGPIO**(int *gpio_num*, uint8_t **value*, uint16_t *remote_id* = NULL)

Obtain the digital value on one of the GPIO pins. Function to retieve the value of the given General Purpose Input/output pin (GPIO) on the target device

Note In firmware version v0.9. The GPIO state cannot be read remotely.

Parameters

- gpio_num: the gpio pin to be set or retrieved. Possible values are 1, 2, 3 or 4.
- value: the pointer that stores the value for the GPIO.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **setGPIO**(int *gpio_num*, uint8_t *value*, uint16_t *remote_id* = NULL)

Set the digital value on one of the GPIO pins. Function to set or set the value of the given GPIO on the target device

Parameters

- gpio_num: the gpio pin to be set or retrieved. Possible values are 1, 2, 3 or 4.
- value: the value for the GPIO. Can be O (LOW) or 1 (HIGH).
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

void **resetSystem**(uint16_t *remote_id* = NULL)

Trigger a software reset of the Pozyx device. Function that will trigger the reset of the system. This will reload all configurations from flash memory, or to their default values.

See *clearConfiguration*, *saveConfiguration*

Parameters

- remote_id: optional parameter that determines the remote device to be used.

int **setLed**(int *led_num*, boolean *state*, uint16_t *remote_id* = NULL)

Function for turning the leds on and off. This function allows you to turn one of the 4 LEDs on the Pozyx board on or off. By default, the LEDs are controlled by the Pozyx system to give status information. This can be changed using the function setLedConfig.

See *setLedConfig*

Parameters

- `led_num`: the led number to be controlled, value between 1, 2, 3 or 4.
- `state`: the state to set the selected led to. Can be 0 (led is off) or 1 (led is on)
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getInterruptMask`**(uint8_t **mask*, uint16_t *remote_id* = NULL)

Function to obtain the interrupt configuration. This functions obtains the interrupt mask from the register reg:POZYX_INT_MASK. The interrupt mask is used to determine which event trigger an interrupt.

Parameters

- `mask`: the configured interrupt mask
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setInterruptMask`**(uint8_t *mask*, uint16_t *remote_id* = NULL)

Function to configure the interrupts. Function to configure the interrupts by means of the interrupt mask from the register reg:POZYX_INT_MASK. The interrupt mask is used to determine which event trigger an interrupt.

Parameters

- `mask`: reference to the interrupt mask to be written. Bit-wise combinations of the following flags are allowed: POZYX_INT_MASK_ERR, POZYX_INT_MASK_POS, POZYX_INT_MASK_IMU, POZYX_INT_MASK_RX_DATA, POZYX_INT_MASK_FUNC.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getConfigModeGPIO`**(int *gpio_num*, uint8_t **mode*, uint16_t *remote_id* = NULL)

Obtain the pull configuration of a GPIO pin. Function to obtain the configured pin mode of the GPIO for the given pin number. This is performed by reading from the reg:POZYX_CONFIG_GPIO1 up to reg:POZYX_CONFIG_GPIO4 register.

Parameters

- `gpio_num`: the pin to update

- `mode`: pointer to the mode of GPIO. Possible values POZYX_GPIO_DIGITAL_INPUT, POZYX_GPIO_PUSHPULL, POZYX_GPIO_OPENDRAIN
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getConfigPullGPIO`**(int *gpio_num*, uint8_t **pull*, uint16_t *remote_id* = NULL)

Obtain the pull configuration of a GPIO pin. Function to obtain the configured pull resistor of the GPIO for the given pin number. This is performed by reading from the reg:POZYX_CONFIG_GPIO1 up to reg:POZYX_CONFIG_GPIO4 register.

Parameters

- `gpio_num`: the pin to update
- `pull`: pull configuration of GPIO. Possible values are POZYX_GPIO_NOPULL, POZYX_GPIO_PULLUP, POZYX_GPIO_PULLDOWN.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setConfigGPIO`**(int *gpio_num*, int *mode*, int *pull*, uint16_t *remote_id* = NULL)

Configure the GPIOs. Function to set the configuration mode of the GPIO for the given pin number. This is performed by writing to the reg:POZYX_CONFIG_GPIO1 up to reg:POZYX_CONFIG_GPIO4 register.

Parameters

- `gpio_num`: the GPIO pin to update. Possible values are 1, 2, 3 or 4.
- `mode`: the mode of GPIO. Possible values POZYX_GPIO_DIGITAL_INPUT, POZYX_GPIO_PUSHPULL, POZYX_GPIO_OPENDRAIN
- `pull`: pull configuration of GPIO. Possible values are POZYX_GPIO_NOPULL, POZYX_GPIO_PULLUP, POZYX_GPIO_PULLDOWN.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setLedConfig`**(uint8_t *config* = 0x0, uint16_t *remote_id* = NULL)

Configure the LEDs. This function configures the 6 LEDs on the pozyx device by writing to the register reg:POZYX_LED_CTRL. More specifically, the function configures which LEDs give system information. By default all the LEDs will give system information.

See *setLed*

Parameters

- `config`: default: the configuration to be set. See POZYX_LED_CTRL for details.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **configInterruptPin**(int *pin*, int *mode*, int *bActiveHigh*, int *bLatch*, uint16_t *remote_id* = NULL)

Configure the interrupt pin.

Parameters

- `pin`: pin id on the pozyx device, can be 1,2,3,4 (or 5 or 6 on the pozyx tag)
- `mode`: push-pull or pull-mode
- `bActiveHigh`: is the interrupt active level HIGH (i.e. 3.3V)
- `bLatch`: should the interrupt be a short pulse or should it latch until the interrupt status register is read

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.

int **saveConfiguration**(int *type*, uint8_t *registers*[] = NULL, int *num_registers* = 0, uint16_t *remote_id* = NULL)

Save (part of) the configuration to Flash memory. This functions stores (parts of) the configurable Pozyx settings in the non-volatile flash memory of the Pozyx device. This function will save the current settings and on the next startup of the Pozyx device these saved settings will be loaded automatically. settings from the flash memory. All registers that are writable, the anchor ids for positioning and the device list (which contains the anchor coordinates) can be saved.

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- `type`: this specifies what should be saved. Possible options are POZYX_FLASH_REGS, POZYX_FLASH_ANCHOR_IDS or POZYX_FLASH_NETWORK.
- `registers`: an option array that holds all the register addresses for which the value must be saved. All registers that are writable are allowed.
- `num_registers`: optional parameter that determines the length of the registers array.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.

- POZYX_TIMEOUT: function timed out.

int **saveRegisters**(uint8_t *registers*[] = NULL, int *num_registers* = 0, uint16_t *remote_id* = NULL)
Save registers to the flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- registers: an option array that holds all the register addresses for which the value must be saved. All registers that are writable are allowed.
- num_registers: optional parameter that determines the length of the registers array.
- remote_id: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.
- POZYX_TIMEOUT: function timed out.

int **saveNetwork**(uint16_t *remote_id* = NULL)
Save the Pozyx's stored network to its flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- remote_id: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.
- POZYX_TIMEOUT: function timed out.

int **saveAnchorIds**(uint16_t *remote_id* = NULL)
Save the Pozyx's stored anchor list to its flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- remote_id: optional parameter that determines the remote device to be used.

Return Value

- POZYX_SUCCESS: success.
- #POZYX_FAIL: function failed.
- POZYX_TIMEOUT: function timed out.

int **saveUWBSettings**(uint16_t *remote_id* = NULL)
Save the Pozyx's stored UWB settings to its flash memory. See saveConfiguration for more information

Version Requires firmware version v1.0

See *clearConfiguration*

Parameters

- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.
- `POZYX_TIMEOUT`: function timed out.

int **clearConfiguration**(uint16_t *remote_id* = NULL)

Clears the configuration. This function clears (part of) the configuration that was previously stored in the non-volatile Flash memory. The next startup of the Pozyx device will load the default configuration values for the registers, anchor_ids and network list.

Version Requires firmware version v1.0

See *saveConfiguration*

Parameters

- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `#POZYX_FAIL`: function failed.

bool **isRegisterSaved**(uint8_t *regAddress*, uint16_t *remote_id* = NULL)

Verify if a register content is saved in the flash memory. This function verifies if a given register variable, specified by its address, is saved in flash memory.

Version Requires firmware version v1.0

Parameters

- `regAddress`: the register address to check
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `true(1)`: if the register variable is saved
- `false(0)`: if the register variable is not saved

int **getNumRegistersSaved**(uint16_t *remote_id* = NULL)

Return the number of register variables saved in flash memory.

Return the number of register variables saved in flash memory.

Parameters

- `remote_id`: optional parameter that determines the remote device to be used.

int **getCoordinates**(coordinates_t **coordinates*, uint16_t *remote_id* = NULL)

Obtain the last coordinates of the device. Retrieve the last coordinates of the device or the remote device. Note that this function does not trigger positioning.

See *doPositioning*, *doRemotePositioning*

Parameters

- `coordinates`: reference to the coordinates pointer object.
- `remote_id`: optional parameter that determines the remote device to be used.

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setCoordinates**(coordinates_t *coordinates*, uint16_t *remote_id* = NULL)
Set the coordinates of the device. Manually set the coordinates of the device or the remote device.

See *getCoordinates*

Parameters

- `coordinates`: coordinates to be set
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getPositionError**(pos_error_t **pos_error*, uint16_t *remote_id* = NULL)
Obtain the last estimated position error covariance information. Retrieve the last error covariance of the position for the device or the remote device. Note that this function does not trigger positioning.

Parameters

- `pos_error`: reference to the pos error object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setPositioningAnchorIds**(uint16_t *anchors*[], int *anchor_num*, uint16_t *remote_id* = NULL)
Manually set which anchors to use for positioning. Function to manually set which anchors to use for positioning by calling the register function reg:POZYX_POS_SET_ANCHOR_IDS. Notice that the anchors specified are only used if this is configured with setSelectionOfAnchors. Furthermore, the anchors specified in this functions must be available in the device list with their coordinates.

See *setSelectionOfAnchors*, *getPositioningAnchorIds*

Parameters

- `anchors[]`: an array with network id's of anchors to be used
- `anchor_num`: the number of anchors write

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getPositioningAnchorIds`**(uint16_t *anchors*[], int *anchor_num*, uint16_t *remote_id* = NULL)

Obtain which anchors used for positioning. Function to retrieve the anchors that used for positioning by calling the register function reg:POZYX_POS_GET_ANCHOR_IDS. When in automatic anchor selection mode, the chosen anchors are listed here.

See *setSelectionOfAnchors*, *setPositioningAnchorIds*

Parameters

- `anchors[]`: an array with network id's of anchors manually set
- `anchor_num`: the number of anchors to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getUpdateInterval`**(uint16_t **ms*, uint16_t *remote_id* = NULL)

Read the update interval continuous positioning. This function reads the configured update interval for continuous positioning from the register reg:POZYX_POS_INTERVAL.

See *setUpdateInterval*

Parameters

- `ms`: pointer to the update interval in milliseconds. A value of 0 means that continuous positioning is disabled.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setUpdateInterval`**(uint16_t *ms*, uint16_t *remote_id* = NULL)

Configure the udpate interval for continuous positioning. This function configures the update interval by writing to the register reg:POZYX_POS_INTERVAL. By writing a valid value, the system will start continuous positioning which will generate a position estimate on regular intervals. This will generate a POZYX_INT_STATUS_POS interrupt when interrupts are enabled.

See *getUpdateInterval*

Parameters

- `ms`: update interval in milliseconds. The update interval must be larger or equal to 100ms.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getRangingProtocol`**(uint8_t **protocol*, uint16_t *remote_id* = NULL)
Obtain the configured ranging protocol. This function obtains the configured ranging protocol by reading from the reg:POZYX_RANGE_PROTOCOL register.

See *doRanging*, *setRangingProtocol*

Parameters

- `protocol`: pointer to the variable holding the ranging protocol used when ranging. Possible values for the ranging protocol are POZYX_RANGE_PROTOCOL_FAST and POZYX_RANGE_PROTOCOL_PRECISION.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setRangingProtocol`**(uint8_t *protocol*, uint16_t *remote_id* = NULL)
Configure the ranging protocol. This function configures the ranging protocol by writing to the reg:POZYX_RANGE_PROTOCOL register.

See *doRanging*, *getRangingProtocol*

Parameters

- `protocol`: Ranging protocol used when ranging. Possible values for the ranging protocol are POZYX_RANGE_PROTOCOL_FAST and POZYX_RANGE_PROTOCOL_PRECISION.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getPositionFilterStrength`**(uint8_t **filter_strength*, uint16_t *remote_id* = NULL)
Obtain the configured positioning filter strength. This function obtains the configured positioning filter strength by reading from the reg:POZYX_POS_FILTER register.

See *getPositionFilterType*, *setPositionFilter*

Parameters

- `filter_strength`: pointer to the variable holding the filter strength used in the built-in filter. This strength is the amount of previous samples used in positioning. Possible values for the position filter strength is between 0 and 15 samples.

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getPositionFilterType`**(uint8_t **filter_type*, uint16_t *remote_id* = NULL)
Obtain the configured positioning filter type. This function obtains the configured positioning filter type by reading from the reg:POZYX_POS_FILTER register.

See *getPositionFilterStrength*, *setPositionFilter*

Parameters

- `filter_type`: pointer to the variable holding the filter type data. Possible values for the positioning algorithm are FILTER_TYPE_NONE, FILTER_TYPE_FIR, FILTER_TYPE_MOVINGMEDIAN, and FILTER_TYPE_MOVINGAVERAGE.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setPositionFilter`**(uint8_t *filter_type*, uint8_t *filter_strength*, uint16_t *remote_id* = NULL)
Configure the positioning filter. This function configures the positioning filter by writing to the reg:POZYX_POS_FILTER register.

See *getPositionFilterStrength*, *getPositionFilterType*

Parameters

- `filter_type`: Filter type used when positioning. Possible values for the positioning algorithm are FILTER_TYPE_NONE, FILTER_TYPE_FIR, FILTER_TYPE_MOVINGMEDIAN, and FILTER_TYPE_MOVINGAVERAGE.
- `filter_strength`: Filter strength used when positioning. Possible values for the position filter strength is between 0 and 15 samples.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getPositionAlgorithm`**(uint8_t **algorithm*, uint16_t *remote_id* = NULL)
Obtain the configured positioning algorithm. This function obtains the configured positioning algorithm by reading from the reg:POZYX_POS_ALG register.

See *getPositionDimension*, *setPositionAlgorithm*

Parameters

- `algorithm`: pointer to the variable holding the algorithm used to determine position. Possible values for the positioning algorithm are POZYX_POS_ALG_UWB_ONLY and POZYX_POS_ALG_LS.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **getPositionDimension**(uint8_t **dimension*, uint16_t *remote_id* = NULL)

Obtain the configured positioning dimension. This function obtains the configuration of the physical dimension by reading from the reg:POZYX_POS_ALG register.

See *getPositionAlgorithm*, *setPositionAlgorithm*

Parameters

- `dimension`: pointer to physical dimension used for the algorithm. Possible values for the dimension are POZYX_3D, POZYX_2D or POZYX_2_5D.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setPositionAlgorithm**(int *algorithm* = POZYX_POS_ALG_UWB_ONLY, int *dimension* = 0x0, uint16_t *remote_id* = NULL)

Configure the positioning algorithm. This function configures the positioning algorithm and the desired physical dimension by writing to the register reg:POZYX_POS_ALG.

See *getPositionAlgorithm*, *getPositionDimension*

Parameters

- `algorithm`: algorithm used to determine the position. Possible values are POZYX_POS_ALG_UWB_ONLY and POZYX_POS_ALG_LS.
- `dimension`: physical dimension used for the algorithm. Possible values are POZYX_3D, POZYX_2D or POZYX_2_5D.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getAnchorSelectionMode**(uint8_t **mode*, uint16_t *remote_id* = NULL)

Obtain the anchor selection mode. This function reads the anchor selection mode bit from the register reg:POZYX_POS_NUM_ANCHORS. This bit describes how the anchors are selected for positioning, either manually or automatically.

Parameters

- `mode`: reference to the anchor selection mode. Possible results are POZYX_ANCHOR_SEL_MANUAL for manual anchor selection or POZYX_ANCHOR_SEL_AUTO for automatic anchor selection.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getNumberOfAnchors`** (uint8_t **nr_anchors*, uint16_t *remote_id* = NULL)

Obtain the configured number of anchors used for positioning. This functions reads out the reg:POZYX_POS_NUM_ANCHORS register to obtain the number of anchors that are being used for positioning.

Parameters

- `nr_anchors`: reference to the number of anchors
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`setSelectionOfAnchors`** (int *mode*, int *nr_anchors*, uint16_t *remote_id* = NULL)

Configure how many anchors are used for positioning and how they are selected. This function configures the number of anchors used for positioning. Theoretically, a larger number of anchors leads to better positioning performance. However, in practice this is not always the case. The more anchors used for positioning, the longer the positioning process will take. Furthermore, it can be chosen which anchors are to be used: either a fixed set given by the user, or an automatic selection between all the anchors in the internal device list.

See *setPositioningAnchorIds* to set the anchor IDs in manual anchor selection mode.

Parameters

- `mode`: describes how the anchors are selected. Possible values are POZYX_ANCHOR_SEL_MANUAL for manual anchor selection or POZYX_ANCHOR_SEL_AUTO for automatic anchor selection.
- `nr_anchors`: the number of anchors to use for positioning. Must be larger than 2 and smaller than 16.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getOperationMode`** (uint8_t **mode*, uint16_t *remote_id* = NULL)

Obtain the operation mode of the device. This function obtains the operation mode (anchor or tag) by

reading from the register reg:POZYX_OPERATION_MODE. This operation mode is independent of the hardware it is on and will have it's effect when performing discovery or auto calibration.

See *setOperationMode*

Parameters

- `mode`: The mode of operations POZYX_ANCHOR_MODE or POZYX_TAG_MODE
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **setOperationMode**(uint8_t *mode*, uint16_t *remote_id* = NULL)

Define if the device operates as a tag or an anchor. This function defines how the device should operate (as an anchor or tag) by writing to the register reg:POZYX_OPERATION_MODE. This operation mode is independent of the hardware it is on and will have it's effect when performing discovery or auto calibration. This function overrules the jumper that is present on the board.

See *getOperationMode*

Parameters

- `mode`: The mode of operations POZYX_ANCHOR_MODE or POZYX_TAG_MODE
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

String **getSystemError**(uint16_t *remote_id* = NULL)

Get the textual system error. This function reads out the reg:POZYX_ERRORCODE register and converts the error code to a textual message.

Parameters

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `String`: the textual error

int **getSensorMode**(uint8_t **sensor_mode*, uint16_t *remote_id* = NULL)

Retrieve the configured sensor mode. This function reads out the register reg:POZYX_SENSORS_MODE which describes the configured sensor mode.

Parameters

- `sensor_mode`: reference to the sensor mode
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.

- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **setSensorMode**(uint8_t *sensor_mode*, uint16_t *remote_id* = NULL)
Configure the sensor mode. This function reads out the register reg:POZYX_SENSORS_MODE which describes the configured sensor mode.

Parameters

- sensor_mode: the desired sensor mode.
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

int **getRawSensorData**(sensor_raw_t **sensor_raw*, uint16_t *remote_id* = NULL)
Obtain all raw sensor data at once as it's stored in the registers. This functions reads out the pressure, acceleration, magnetic field strength, angular velocity, orientation in Euler angles, the orientation as a quaternion, the linear acceleration, the gravity vector and temperature.

Parameters

- sensor_raw: reference to the sensor_raw object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getAllSensorData**(sensor_data_t **sensor_data*, uint16_t *remote_id* = NULL)
Obtain all sensor data at once. This functions reads out the pressure, acceleration, magnetic field strength, angular velocity, orientation in Euler angles, the orientation as a quaternion, the linear acceleration, the gravity vector and temperature.

Parameters

- sensor_data: reference to the sensor_data object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getPressure_Pa**(float32_t **pressure*, uint16_t *remote_id* = NULL)
Obtain the atmospheric pressure in Pascal. This function reads out the pressure starting from the register POZYX_PRESSURE. The maximal update rate is 10Hz. The units are Pa.

Parameters

- `pressure`: reference to the pressure variable
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getMaxLinearAcceleration`**(uint16_t **max_lin_acc*, uint16_t *remote_id* = NULL)

Obtain the max linear acceleration This registers functions similarly to the interrupt and error registers in that it clears the register's value upon reading. This is the max linear acceleration since the last read.

Parameters

- `max_lin_acc`: pointer to a variable that will hold the max linear acceleration
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getAcceleration_mg`**(acceleration_t **acceleration*, uint16_t *remote_id* = NULL)

Obtain the 3D acceleration vector in mg. This function reads out the acceleration data starting from the register reg:POZYX_ACCEL_X. The maximal update rate is 100Hz. The units are mg. The vector is expressed in body coordinates (i.e., along axes of the device).

Parameters

- `acceleration`: reference to the acceleration object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getMagnetic_uT`**(magnetic_t **magnetic*, uint16_t *remote_id* = NULL)

Obtain the 3D magnetic field strength vector in µTesla. This function reads out the magnetic field strength data starting from the register reg:POZYX_MAGN_X. The maximal update rate is 100Hz. The vector is expressed in body coordinates (i.e., along axes of the device).

Parameters

- `magnetic`: reference to the magnetic object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

- POZYX_TIMEOUT: function timed out, no response received.

int **getAngularVelocity_dps**(angular_vel_t **angular_vel*, uint16_t *remote_id* = NULL)

Obtain the 3D angular velocity vector degrees per second. This function reads out the angular velocity from the gyroscope using the register reg:POZYX_GYRO_X. The maximal update rate is 100Hz. The rotations are expressed in body coordinates (i.e., the rotations around the axes of the device).

Parameters

- angular_vel: reference to the angular velocity object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getEulerAngles_deg**(euler_angles_t **euler_angles*, uint16_t *remote_id* = NULL)

Obtain the orientation in Euler angles in degrees. This function reads out the Euleur angles: Yaw, Pitch and Roll that represents the 3D orientation of the device

Parameters

- euler_angles: reference to the euler_angles object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getQuaternion**(quaternion_t **quaternion*, uint16_t *remote_id* = NULL)

Obtain the orientation in quaternions. This function reads out the 4 coordinates of the quaternion that represents the 3D orientation of the device. The quaternions are unitless and normalized.

Parameters

- quaternion: reference to the quaternion object
- remote_id: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.
- POZYX_TIMEOUT: function timed out, no response received.

int **getLinearAcceleration_mg**(linear_acceleration_t **linear_acceleration*, uint16_t *remote_id* = NULL)

Obtain the 3D linear acceleration in mg. This function reads out the linear acceleration data starting from the register reg:POZYX_LIA_X. The Linear acceleration is the acceleration compensated for gravity. The maximal update rate is 100Hz. The units are mg. The vector is expressed in body coordinates (i.e., along axes of the device).

Parameters

- `linear_acceleration`: reference to the acceleration object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getGravityVector_mg`**(gravity_vector_t **gravity_vector*, uint16_t *remote_id* = NULL)

Obtain the 3D gravity vector in mg. This function reads out the gravity vector coordinates starting from the register reg:POZYX_GRAV_X. The maximal update rate is 100Hz. The units are mg. The vector is expressed in body coordinates (i.e., along axes of the device). This vector always points to the ground, regardless of the orientation.

Parameters

- `gravity_vector`: reference to the gravity_vector object
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`getTemperature_c`**(float32_t **temperature*, uint16_t *remote_id* = NULL)

Obtain the temperature in degrees Celcius. This function reads out the temperature from the register reg:POZYX_TEMPERATURE. This function is unsupported in firmware version v0.9.

Parameters

- `temperature`: reference to the temperature variable
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.
- `POZYX_TIMEOUT`: function timed out, no response received.

int **`doPositioning`**(coordinates_t **position*, uint8_t *dimension*, int32_t *height* = 0)

Obtain the coordinates. This function triggers the positioning algorithm to perform positioning with the given parameters. By default it will automatically select 4 anchors from the internal device list. It will then perform ranging with these anchors and use the results to compute the coordinates. This function requires that the coordinates for the anchors are stored in the internal device list.

Please read the tutorial ready to localize to get started with positioning.

See *doRemotePositioning*, *doAnchorCalibration*, *addDevice*, *setSelectionOfAnchors*, *setPositionAlgorithm*

Parameters

- `position`: data object to store the result
- `height`: optional parameter that is used for POZYX_2_5D to give the height in mm of the tag

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doRemotePositioning**(uint16_t *remote_id*, coordinates_t **coordinates*, uint8_t *dimension*, int32_t *height* = 0)

Obtain the coordinates of a remote device. Don't use with 2.5D!

This function triggers the positioning algorithm on a remote pozyx device to perform positioning with the given parameters. By default it will automatically select 4 anchors from the internal device list on the remote device. The device will perform ranging with the anchors and use the results to compute the coordinates. This function requires that the coordinates for the anchors are stored in the internal device list on the remote device. After positioning is completed, the remote device will automatically transmit the result back.

See *doPositioning*, *addDevice*, *setSelectionOfAnchors*, *setPositionAlgorithm*

Parameters

- `remote_id`: the remote device that will do the positioning
- `position`: data object to store the result
- `height`: optional parameter that is used for POZYX_2_5D to give the height in mm of the tag

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doRanging**(uint16_t *destination*, device_range_t **range*)

Trigger ranging with a remote device. This function performs ranging with a remote device using the UWB signals.

See *doRemoteRanging*, *getDeviceRangeInfo*

Parameters

- `destination`: the target device to do ranging with
- `range`: the pointer to where the resulting data will be stored

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doRemoteRanging**(uint16_t *device_from*, uint16_t *device_to*, device_range_t **range*)

Trigger ranging between two remote devivces. Function allowing to trigger ranging between two remote devices A and B. The ranging data is collected by device A and transmitted back to the local device.

See *doRanging*, *getDeviceRangeInfo*

Parameters

- `device_from`: device A that will initiate the range request.

- `device_to`: device B that will respond to the range request.
- `range`: the pointer to where the resulting data will be stored

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getDeviceRangeInfo`**(uint16_t *device_id*, device_range_t **device_range*, uint16_t *remote_id* = NULL)

Retrieve stored ranging information. Functions to retrieve the latest ranging information (i.e., the distance, signal strength and timestamp) with respect to a remote device. This function does not trigger ranging.

See *doRanging*, *doRemoteRanging*

Parameters

- `device_id`: network id of the device for which range information is requested
- `device_range`: data object to store the information
- `remote_id`: optional parameter that determines the remote device where this function is called.

int **`getDeviceListSize`**(uint8_t **device_list_size*, uint16_t *remote_id* = NULL)

Obtain the number of devices stored internally. The following function retrieves the number of devices stored in the device list.

See *doDiscovery*, *doAnchorCalibration*

Parameters

- `device_list_size`: the pointer that stores the device list size
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getDeviceIds`**(uint16_t *devices*[], int *size*, uint16_t *remote_id* = NULL)

Obtain the network IDs from all the devices in the device list. Function to get all the network ids of the devices in the device list

Parameters

- `devices[]`: array that will be filled with the network ids
- `size`: the number of network IDs to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **`getAnchorIds`**(uint16_t *anchors*[], int *size*, uint16_t *remote_id* = NULL)

Obtain the network IDs from all the anchors in the device list. Function to get all the network ids of the anchors in the device list

Parameters

- `devices[]`: array that will be filled with the network ids
- `size`: the number of network IDs to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getTagIds**(uint16_t *tags*[], int *size*, uint16_t *remote_id* = NULL)

Obtain the network IDs from all the tags in the device list. Function to get all the network ids of the tags in the device list

Parameters

- `tags[]`: array that will be filled with the network ids
- `size`: the number of network IDs to read.
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doDiscovery**(int *type* = 0x0, int *slots* = 3, int *slot_duration* = 10)

Discover Pozyx devices in range. Function to wirelessly discover anchors/tags/all Pozyx devices in range. The discovered devices are added to the internal device list.

See *getDeviceListSize*, *getDeviceIds*

Parameters

- `type`: which type of device to discover. Possible values are POZYX_DISCOVERY_ANCHORS_ONLY: anchors only, POZYX_DISCOVERY_TAGS_ONLY: tags only or POZYX_DISCOVERY_ALL_DEVICES: anchors and tags
- `slots`: The number of slots to wait for a response of an undiscovered device
- `slot_duration`: Time duration of an idle slot

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **doAnchorCalibration**(int *dimension* = POZYX_2D, int *num_measurements* = 10, int *num_anchors* = 0, uint16_t *anchors*[] = NULL, int32_t *heights*[] = NULL)

Automatically obtain the relative anchor positions. WARNING: This is currently experimental and will be improved in the next firmware version! This function triggers the automatic anchor calibration to obtain the relative coordinates of up to 6 pozyx devices in range. This function can be used for quickly setting up the positioning system. The procedure may take several hundres of milliseconds depending on the number of devices in range and the number of range measurements requested. During the calibration proces LED 2 will be turned on. At the end of calibration the corresponding bit in the reg:POZYX_CALIB_STATUS

register will be set. The resulting coordinates are stored in the internal device list. Please read the tutorial Ready to Localize to learn how to use this function.

Parameters

- `type`: dimension of the calibration, can be POZYX_2D or POZYX_2_5D
- `measurements`: The number of measurements per link. Recommended 10. Theoretically, a larger number should result in better calibration accuracy.
- `anchor_num`: The number of anchors in the anchors[] array
- `anchors[]`: The anchors that determine the axis (see datasheet)
- `heights`: The heights in mm of the anchors in the anchors[] array (only used for POZYX_2_5D)

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **clearDevices**(uint16_t *remote_id* = NULL)
Empty the internal list of devices. This function empties the internal list of devices.

Parameters

- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **addDevice**(device_coordinates_t *device_coordinates*, uint16_t *remote_id* = NULL)
Manualy adds a device to the device list. This function can be used to manually add a device and its coordinates to the device list. Once the device is added, it can be used for positioning.

Parameters

- `device_coordinates`: the device information to be added
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- `POZYX_SUCCESS`: success.
- `POZYX_FAILURE`: function failed.

int **getDeviceCoordinates**(uint16_t *device_id*, coordinates_t **coordinates*, uint16_t *remote_id* = NULL)
Retrieve the stored coordinates of a device. This function retrieves the device coordinates stored in the internal device list.

Parameters

- `device_id`: device from which the information needs to be retrieved
- `device_coordinates`: data object to store the information
- `remote_id`: optional parameter that determines the remote device to be used

Return Value

- POZYX_SUCCESS: success.
- POZYX_FAILURE: function failed.

5.2 Protected

class PozyxClass
Provides an interface to an attached Pozyx shield.

Protected Static Functions

int **i2cWriteWrite**(**const** uint8_t *reg_address*, **const** uint8_t **pData*, int *size*)
Function: i2cWriteWrite

Internal function that writes a number of bytes to a specfified Pozyx register This function implements the I2C interface as described in the datasheet on our website Input values: Writes a number of bytes to the specified pozyx register address using I2C

Return POZYX_FAILURE: error occured during the process POZYX_SUCCESS: successful execution of the function

Parameters

- reg_address: Pozyx address to write to
- pData: reference to the data that needs to be writen
- size: size in byte of data to be written

int **i2cWriteRead**(uint8_t **write_data*, int *write_len*, uint8_t **read_data*, int *read_len*)
Function: i2cWriteRead

Internal function that writes and reads a number of bytes to call a specfic Pozyx register function This function implements the I2C interface as described in the datasheet on our website Input values: Call a register function using I2C with given parameters

Return POZYX_FAILURE: error occured during the process POZYX_SUCCESS: successful execution of the function

Parameters

- write_data: reference to the data that needs to be writen
- write_len: size in byte of data to be written
- read_data: reference to the pointer where the read data should be stored
- read_len: size in byte of data to be read

void **IRQ**()
Function: void IRQ

Internal function that sets the _interrupt variable on an Arduino interrupt

The interrupt handler for the pozyx device: keeping it uber short!

Protected Static Attributes

int **_interrupt**

This file contains the defintion of the core POZYX functions and variables

5.3 Private

class PozyxClass
Provides an interface to an attached Pozyx shield.

5.4 Undocumented...

class PozyxClass
Provides an interface to an attached Pozyx shield.

INDEX

R

S

U

W