pypozyx Documentation

Release 1.2.6

Laurent Van Acker

Jan 22, 2019

Contents:

This module provides wrappers for interfacing with a Pozyx device over a serial connection.

CHAPTER 1

pypozyx

1.1 About

pypozyx is a Python library made for providing an interface with a Pozyx device over a serial connection. This serial interface then also allows communication with remote Pozyx devices through the connected device.

The library was largely inspired by the already existing Arduino library, yielding a lot of advantages that carried over from the Arduino library. However, as a result pypozyx is not that Pythonic. It is our intention to improve this with our next big release.

1.2 Features

- Easy to use, allowing both high-level and low-level interfacing with the Pozyx device.
- Uses the excellent pySerial library for cross-platform functionality.
- Works with Python 2 and Python 3
- Pozyx device serial port detection.
- Specialized data structures for all important Pozyx data.

1.3 Requirements

- Python 2.7 or newer, or Python 3.4 and newer
- pySerial > 3.0

1.4 Pozyx serial connection requirements

1.4.1 Linux

- Make sure you have permissions set to use the serial device.
- To get permission once (given your device is on port ACM0) you can run `sudo chmod 666 /dev/ttyACM0`
- To get permission forever you can run `sudo adduser $USER dialout` or the relevant group for your distro.

1.4.2 MacOS

- Normally everything should work out of the box.

1.4.3 Windows

- Please use Windows 7 or newer.
- Install STM32 Virtual COM Port Driver.

Note: After running this installer, you have to install the correct driver package for your system.

The driver installers are located in *C:\Program Files (x86)\STMicroelectronics\Software\Virtual comport driver*.

Choose Win7 if you run Windows 7 or older. Choose Win8 for Windows 8 or newer. Run "dpinst_amd64.exe" on a 64-bit system, "dpinst_x86.exe"on a 32-bit system.

1.5 Installation

Currently, pypozyx is easily installable from Python Package index (PyPi), but can also be installed from source.

1.5.1 From PyPi

To install the library from PyPi, you can run either of the following:

- `pip install pypozyx` or `python -m pip install pypozyx`.
- `pip3 install pypozyx` or `python3 -m pip install pypozyx`.

Note: If installation fails due to permission issues, or the package doesn't seem to install, please try to add –user as a flag to pip, `pip install --user pypozyx`, or use sudo `sudo pip install pypozyx`.

1.5.2 From source

To install from source, you'll have to download the source files first, of course. You can do this either by:

- `git clone https://github.com/pozyxLabs/Pozyx-Python-library`
- Download it from the repository and extracting it.
- Downloading the zip file directly, and extracting it.

Then, in your extracted/downloaded folder, run `python setup.py install` or `python3 setup.py install`.

CHAPTER 2

Getting started

2.1 Finding your serial port

There's a helper in the library for identifying the first serial port that is a Pozyx. This is easily done with a Python snippet

```
import pypozyx

print(pypozyx.get_first_pozyx_serial_port())
```

Or from the command line

```
python -c "from pypozyx import *;print(get_first_pozyx_serial_port())"
```

If there is no Pozyx device recognized, the function will return `None` and thus nothing will be printed.

2.2 Connecting to the Pozyx

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

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port

serial_port = get_first_pozyx_serial_port()

if serial_port is not None:
    pozyx = PozyxSerial(serial_port)
    print("Connection success!")
else:
    print("No Pozyx port was found")
```

With this, you have a pozyx object with the full API at your fingertips. For example, you can read the UWB settings with the following snippet.

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port, UWBSettings

serial_port = get_first_pozyx_serial_port()

if serial_port is not None:
    pozyx = PozyxSerial(serial_port)
    uwb_settings = UWBSettings()
    pozyx.getUWBSettings(uwb_settings)
    print(uwb_settings)
else:
    print("No Pozyx port was found")
```

2.3 General philosophy

As said in the introduction, the pypozyx library was heavily inspired by the Arduino library, making it less pythonic.

- The functions are camelCased
- Almost all functions return a status and take the relevant data container as an argument.

This had as an advantage that users coming from Arduino could easily adapt their code, and that the documentation was very similar. However, I'd love to change these things when I make a 2.0 release.

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 device connected to your computer, and powered remote devices as well. In this section we'll go over all of these.

2.4 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.

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port, POZYX_SUCCESS,␣
↪SingleRegister, EulerAngles, Acceleration
# initalize the Pozyx as above

# initialize the data container
who_am_i = SingleRegister()
# get the data, passing along the container
status = pozyx.getWhoAmI(who_am_i)
```

(continues on next page)

(continued from previous page)

```
# 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.
    print(who_am_i) # will print '0x43'

# and repeat
# initialize the data container
acceleration = Acceleration()
# get the data, passing along the container
pozyx.getAcceleration_mg(acceleration)

# initialize the data container
euler_angles = EulerAngles()
# get the data, passing along the container
pozyx.getEulerAngles_deg(euler_angles)
```

2.5 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.

Note: All set functions are tolerant for values that aren't per se a data object. An integer value or respectively fitting array with the relevant data as contained in the register will pass as well.

```
# method 1: making a data object
uwb_channel = SingleRegister(5)
pozyx.setUWBChannel(uwb_channel)
# method 2: or just using the channel number directly
pozyx.setUWBChannel(5)

# both have the same effect!
```

The advantage of using the data object approach lies especially with more complex data, where your Python editor will give you more information on what content you're putting in, or where the object will convert data to the right form for you.

```
# method 1: making a data object
# this is much more readable
uwb_settings = UWBSettings(channel=5, bitrate=1, prf=2, plen=0x08, gain_db=25.0)
pozyx.setUWBChannel(uwb_channel)
# method 2: using the register values directly
# this isn't readable and also not writable (need to search in depth register␣
↪documentation)
pozyx.setUWBSettings([5, 0b10000001, 0x08, 50])
```

(continues on next page)

(continued from previous page)

```
# both still have the same effect, but note how bitrate and prf combine in a register␣
↪value,
# and gain is doubled when converted to its register contents.
```

Some typical write operations

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port, POZYX_SUCCESS,␣
↪SingleRegister, PozyxConstants

# initialize Pozyx as above

pozyx.setPositionAlgorithm(PozyxConstants.POSITIONING_ALGORITHM_UWB_ONLY)

new_id = NetworkId(0x1)
pozyx.setNetworkId(new_id)

pozyx.setPositioningFilter(PozyxConstant.FILTER_TYPE_MOVING_AVERAGE, 10)
```

Note that you seemingly need to know that the positioning filter has `PozyxConstant.FILTER_TYPE_MOVING_AVERAGE` as a possible type of filter. This is pretty low-level knowledge and may remain hidden when not knowing about, and so in a recent version we added a lot of helpers that do away with having to know the appropriate constants for certain operations.

```
# instead of pozyx.setPositionAlgorithm(PozyxConstants.POSITIONING_ALGORITHM_UWB_ONLY)
pozyx.setPositionAlgorithmNormal()

# instead of pozyx.setPositioningFilter(PozyxConstant.FILTER_TYPE_MOVING_AVERAGE, 10)
pozyx.setPositioningFilterMovingAverage(10)
```

2.6 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 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.

```
from pypozyx import ..., Coordinates, DeviceCoordinates

# assume an anchor 0x6038 that we want to add to the device list and immediately save␣
↪the device list after.
anchor = DeviceCoordinates(0x6038), 0, Coordinates(5000, 5000, 0))
pozyx.addDevice(anchor)
pozyx.saveNetwork()

# after, we can start positioning. Positioning takes its parameters from the␣
↪configuration in the tag's
```

(continues on next page)

(continued from previous page)

```
# registers, and so we only need the coordinates.
position = Coordinates()
pozyx.doPositioning(position)
```

2.7 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!

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

# this will read the WHO_AM_I register of the remote tag
who_am_i = SingleRegister()
pozyx.getWhoAmI(who_am_i)
print(who_am_i) # will print 0x43
```

2.8 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.

```
# Saves the positioning settings
pozyx.savePositioningSettings()
# Saves the device list used for positioning
pozyx.saveNetwork()
# Saves the device's UWB settings
pozyx.saveUWBSettings()
```

2.9 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

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port, POZYX_SUCCESS,␣
↪SingleRegister

# initialize Pozyx as above

if pozyx.saveUWBSettings() != POZYX_SUCCESS:
    # this is one way which retrieves the error code
    error_code = SingleRegister()
    pozyx.getErrorCode(error_code)
    print('Pozyx error code: %s' % error_code)
    # the other method returns a descriptive string
    print(pozyx.getSystemError())
```

CHAPTER 3

pypozyx API

3.1 PozyxSerial

This also includes the serial helpers you can use to connect to the Pozyx device.

pypozyx.pozyx_serial - contains the serial interface with Pozyx through PozyxSerial.

class pypozyx.pozyx_serial.**PozyxSerial**(*port*, *baudrate=115200*, *timeout=0.1*, *write_timeout=0.1*, *print_output=False*, *debug_trace=False*, *show_trace=False*, *suppress_warnings=False*)

Bases: *pypozyx.lib.PozyxLib*

This class provides the Pozyx Serial interface, and opens and locks the serial port to use with Pozyx. All functionality from PozyxLib and PozyxCore is included.

Parameters

- **port** (*str*) – Name of the serial port. On UNIX this will be '/dev/ttyACMX', on Windows this will be 'COMX', with X a random number.
- **baudrate** (*optional*) – the baudrate of the serial port. Default value is 115200.
- **timeout** (*optional*) – timeout for the serial port communication in seconds. Default is 0.1s or 100ms.
- **print_output** (*optional*) – boolean for printing the serial exchanges, mainly for debugging purposes
- **suppress_warnings** (*optional*) – boolean for suppressing warnings in the Pozyx use, usage not recommended
- **debug_trace** (*optional*) – boolean for printing the trace on bad serial init (DEPRECATED)
- **show_trace** (*optional*) – boolean for printing the trace on bad serial init (DEPRECATED)

Example usage:

```
>>> pozyx = PozyxSerial('COMX') # Windows
>>> pozyx = PozyxSerial('/dev/ttyACMX', print_output=True) # Linux and OSX.␣
↪Also puts debug output on.
```

Finding the serial port can be easily done with the following code:

```
>>> import serial.tools.list_ports
>>> print serial.tools.list_ports.comports()[0]
```

Putting one and two together, automating the correct port selection with one Pozyx attached:

```
>>> import serial.tools.list_ports
>>> pozyx = PozyxSerial(serial.tools.list_ports.comports()[0])
```

connectToPozyx (*port*, *baudrate*, *timeout*, *write_timeout*)
Attempts to connect to the Pozyx via a serial connection

regFunction (*address*, *params*, *data*)
Performs a register function on the Pozyx, if the address is a register function.

Parameters

- **address** – Register function address of function to perform.
- **params** – Parameters for the register function. Has to be ByteStructure-derived object.
- **data** – Container for the data the register function returns. Has to be ByteStructure-derived object.

Returns POZYX_SUCCESS, POZYX_FAILURE

regRead (*address*, *data*)
Reads data from the Pozyx registers, starting at a register address, if registers are readable.

Parameters

- **address** – Register address to start writing at.
- **data** – Data to write to the Pozyx registers. Has to be ByteStructure-derived object.

Returns POZYX_SUCCESS, POZYX_FAILURE

regWrite (*address*, *data*)
Writes data to the Pozyx registers, starting at a register address, if registers are writable.

Parameters

- **address** – Register address to start writing at.
- **data** – Data to write to the Pozyx registers. Has to be ByteStructure-derived object.

Returns POZYX_SUCCESS, POZYX_FAILURE

serialExchange (*s*)
Auxiliary. Performs a serial write to and read from the Pozyx.

Parameters **s** – Serial message to send to the Pozyx

Returns

Serial message the Pozyx returns, stripped from 'D,' at its start and NL+CR at the end.

validatePozyx()
Validates whether the connected device is indeed a Pozyx device

waitForFlag(*interrupt_flag*, *timeout_s*, *interrupt=None*)
Waits for a certain interrupt flag to be triggered, indicating that that type of interrupt occured.

Parameters

- **interrupt_flag** – Flag indicating interrupt type.
- **timeout_s** – time in seconds that POZYX_INT_STATUS will be checked for the flag before returning POZYX_TIMEOUT.

Kwargs: interrupt: Container for the POZYX_INT_STATUS data

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

pypozyx.pozyx_serial.**get_first_pozyx_serial_port**()
Returns the first encountered Pozyx serial port's identifier

pypozyx.pozyx_serial.**get_port_object**(*device*)
Returns the PySerial port object from a given port path

pypozyx.pozyx_serial.**get_pozyx_ports**()
Returns the Pozyx serial ports. Windows only. Needs driver installed

pypozyx.pozyx_serial.**get_pozyx_ports_windows**()
Returns the Pozyx serial ports. Windows only. Needs driver installed

pypozyx.pozyx_serial.**get_serial_ports**()
Returns the open serial ports

pypozyx.pozyx_serial.**is_correct_pyserial_version**()
Returns whether the pyserial version is supported

pypozyx.pozyx_serial.**is_pozyx**(*device*)
Returns whether the device is a recognized Pozyx device

pypozyx.pozyx_serial.**is_pozyx_port**(*port*)
Returns whether the port is a Pozyx device

pypozyx.pozyx_serial.**list_serial_ports**()
Prints the open serial ports line per line

pypozyx.pozyx_serial.**print_all_serial_ports**()
Prints the open serial ports line per line

3.2 Functions

Do a split like in the current library documentation here?

pypozyx.lib - Contains core and extended Pozyx user functionality through the PozyxLib class.

class pypozyx.lib.**Device**(*id_*)
Bases: object

firmware_version

has_cloud_firmware()

has_firmware_version()

class pypozyx.lib.**PozyxLib**
Bases: pypozyx.core.PozyxCore

Implements the functionality users expect from Pozyx, using the methods from PozyxCore to communicate and interface with Pozyx both locally and remotely. This does not limit itself to positioning, ranging, and reading the sensor data of the various Pozyx sensors, but also features an assortment of troubleshooting functions, abstractions of frequently used registers, UWB settings, etc.

Unlike the Arduino library, this isn't divided into parts such as 'device functions', 'system functions', etc, but will be in the future. For now, the Arduino library should work as a great reference.

addDevice(*device_coordinates*, *remote_id=None*)
Adds a device to the Pozyx's device list. Can be either a tag or anchor.

Parameters

- **device_coordinates** – Device's ID, flag, and coordinates structure. DeviceCoordinates(ID, flag, Coordinates(x, y, z)) or [ID, flag, x, y, z]
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

addIdToDeviceMesh(*id_=None*)

changeDeviceCoordinates(*device_id*, *new_coordinates*, *remote_id=None*)
Changes a device's coordinates in the Pozyx's device list, keeping the rest of the list intact

Parameters

- **device_id** – ID that needs to be removed. NetworkID or integer.
- **new_coordinates** – new coordinates for the device
- **remote_id** (*optional*) – Remote Pozyx ID

Returns POZYX_SUCCESS, POZYX_FAILURE

checkForFlagFast(*interrupt_flag*, *timeout_s*, *interrupt=None*)
A fast variant of checkForFlag, using waitForFLagFast, useful for ranging on very fast UWB settings.

Parameters

- **interrupt_flag** – Flag of interrupt type to check the interrupt register against.
- **timeout_s** – duration to wait for the interrupt in seconds
- **interrupt** (*optional*) – Container for the interrupt status register data.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

checkUWBSettings(*suspected_uwb_settings*, *remote_id=None*, *equal_gain=True*)

clearConfiguration(*remote_id=None*)
Clears the Pozyx's flash memory.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

clearDevices(*remote_id=None*)
Clears the Pozyx's device list.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

configInterruptPin(*pin_number=0*, *mode=0*, *active_high=False*, *latch=False*, *remote_id=None*)
Configures the interrupt pin via the PozyxRegisters.INTERRUPT_PIN register.

Parameters

- **pin_number** (*optional*) – The Pozyx's pin ID. 1 to 4 on anchor, 1 to 6 on tag. 0 means no pin. SingleRegister or integer.
- **mode** (*optional*) – Push-pull (0) or pull (1). SingleRegister or integer. SingleRegister or integer.
- **active_high** (*optional*) – Is the interrupt voltage active high or low. Boolean.
- **latch** (*optional*) – Is the interrupt a short pulse or latch till read? Boolean.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

configureAnchors(*anchor_list*, *anchor_select=1*, *remote_id=None*)
Configures a set of anchors as the relevant anchors on a device

Parameters

- **anchor_list** (*list*) – Python list of either DeviceCoordinates or [ID, flag, x, y, z]
- **anchor_select** (*optional*) – How to select the anchors in positioning
- **remote_id** (*optional*) – Remote Pozyx ID

Returns POZYX_SUCCESS, POZYX_FAILURE

doAnchorCalibration(*dimension*, *num_measurements*, *anchors*, *heights=None*, *remote_id=None*)
Performs automatic anchor calibration on the Pozyx.

Using manual calibration over automatic calibration is highly recommended, as this will not only be less robust to use, the results will also be worse than a carefully accurately manually measured setup. Using a laser measurer for this purpose is also adviced.

When insisting on using automatic calibration, make sure that all devices are in range and able to communicate with the device. Try ranging with all devices first, and make sure they're on the same UWB settings.

Parameters

- **dimension** – Dimension for the automatic calibration. When 2.5D, make sure to pass along heights as well.
- **num_measurements** – Number of measurements to use in calibration. The
- **anchors** – List of anchor IDs that will be used in the calibration. DeviceList() or [anchor_id1, anchor_id2, . . .]
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doDiscovery(*discovery_type=0*, *slots=3*, *slot_duration=0.01*, *remote_id=None*)
Performs discovery on the Pozyx, which will let it discover other Pozyx devices with the same UWB settings in range.

Parameters

- **discovery_type** (*optional*) – Type of devices to discover, defaults to discovering the anchors. PozyxConstants.DISCOVERY_ALL_DEVICES, PozyxConstants.DISCOVERY_TAGS_ONLY are alternatives.

- **slots** (*optional*) – Number of timeslots used in attempt to discover devices. Default is 3 slots.
- **slot_duration** (*optional*) – Duration in seconds of each timeslot used in the discovery process. Default is 10 ms.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doDiscoveryAll(*slots=3*, *slot_duration=0.01*, *remote_id=None*)
Performs general discovery on the Pozyx, which will let it discover both Pozyx tags and anchors with the same UWB settings in range.

Parameters

- **slots** (*optional*) – Number of timeslots used in attempt to discover devices. Default is 3 slots.
- **slot_duration** (*optional*) – Duration in seconds of each timeslot used in the discovery process. Default is 10 ms.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doDiscoveryAnchors(*slots=3*, *slot_duration=0.01*, *remote_id=None*)
Performs anchor discovery on the Pozyx, which will let it discover Pozyx anchors with the same UWB settings in range.

Parameters

- **slots** (*optional*) – Number of timeslots used in attempt to discover devices. Default is 3 slots.
- **slot_duration** (*optional*) – Duration in seconds of each timeslot used in the discovery process. Default is 10 ms.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doDiscoveryTags(*slots=3*, *slot_duration=0.01*, *remote_id=None*)
Performs tag discovery on the Pozyx, which will let it discover Pozyx tags with the same UWB settings in range.

Parameters

- **slots** (*optional*) – Number of timeslots used in attempt to discover devices. Default is 3 slots.
- **slot_duration** (*optional*) – Duration in seconds of each timeslot used in the discovery process. Default is 10 ms.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doFunctionOnDifferentUWB(*function*, *uwb_settings*, **args*, ***kwargs*)

doOptimalDiscovery(*discovery_type=2*, *slots=3*, *timeout=None*)
Performs a discovery with slot_duration optimised for the device's UWB settings.

doPositioning(*position*, *dimension=3*, *height=<pypozyx.structures.generic.Data object>*, *algorithm=None*, *remote_id=None*, *timeout=None*)
Performs positioning with the Pozyx. This is probably why you're using Pozyx.

This function only performs the positioning and doesn't take care of the previous steps required to get this operational, so be sure to adhere to this checklist: - while you can perform automatic calibration, manual calibration is much more stable and reliable. - when using manual calibration, add all anchors using addDevice. - all anchors are on the same UWB settings as the device performing positioning. - if you're using more than four anchors, be sure to set this with setSelectionOfAnchors.

Basic troubleshooting: - try to perform ranging with all devices - are you using a Coordinates object for your position? - if you perform getDeviceListSize and subsequently getDeviceIds, are these your anchors?

While in the Arduino library doRemotePositioning is used for remote ranging, this function follows the library's convention to add remote_id as a keyword argument.

For an in-action example, check the "Ready to localize" tutorial on the Pozyx homepage(www.pozyx.io), and the ready_to_localize.py example found in this library's tutorial folder.

Parameters

- **position** – Container for the positioning coordinates. Coordinates object.
- **dimension** (*optional*) – Dimension to perform positioning in. Default 3D. When 2.5D, make sure height is also passed along.
- **height** (*optional*) – Height of Pozyx in 2.5D positioning. Default 0. Either integer height or Data([height], 'i').
- **algorithm** (*optional*) – Algorithm set before positioning. No new algorithm is set by default.
- **remote_id** (*optional*) – Remote Pozyx ID. Local Pozyx is used when None or omitted.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doPositioningSlave(*position*, *timeout=None*)

Checks whether the device has positioned and if so, reads the position.

This is useful for slave devices with a controller that needs to know the device's positions too

Parameters **position** – Container for the positioning coordinates. Coordinates object.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doPositioningWithData(*positioning_data*, *remote_id=None*, *timeout=None*)

doPositioningWithDataSlave(*positioning_data*, *timeout=None*)

Checks whether the device has positioned and if so, reads the position with data.

This is useful for slave devices with a controller that needs to know the device's positions (with data) too

Parameters **positioning_data** – Container for the positioning coordinates. PositioningData object.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doRanging(*destination_id*, *device_range*, *remote_id=None*)

Performs ranging with another destination device, resulting in range information.

This is pretty straightforward, the range information consists of the following:

- the timestamp of the range measurement.
- the distance between the local / remote tag and the destination
- the RSS, which indicates the signal strength between origin and destination.

While in the Arduino library doRemoteRanging is used for remote ranging, this function follows the library's convention to add remote_id as a keyword argument. Make sure that the destination is on the same UWB settings as this, and to pass a DeviceRange object for the device_range parameter.

For an in-action example, check the "Ready to range" tutorial on the Pozyx homepage(www.pozyx.io), and the ready_to_range.py example found in this library's tutorial folder.

Parameters

- **destination_id** – Network ID of the destination, to perform ranging with. integer ID or NetworkID(ID)
- **device_range** – Container for device range measurement data. DeviceRange object.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

doRangingSlave(*destination_id*, *device_range*)

Checks whether the device has ranged and if so, reads the range.

This is useful for slave devices with a controller that needs to know the range measurements too

Parameters

- **destination_id** – Network ID of the destination, to perform ranging with. integer ID or NetworkID(ID)
- **device_range** – Container for device range measurement data. DeviceRange object.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getAcceleration_mg(*acceleration*, *remote_id=None*)

Obtain the Pozyx's acceleration sensor data in mg.

Parameters

- **acceleration** – Container for the read data. Acceleration().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getAllSensorData(*sensor_data*, *remote_id=None*)

Obtains all the Pozyx's sensor data in their default units.

Parameters

- **sensor_data** – Container for the read data. SensorData() or RawSensorData().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getAnchorIds(*anchors*, *remote_id=None*)

Obtain the IDs of the anchors in the Pozyx's device list.

You need to make sure to know how many anchors are in the list, as an incorrect size of anchors will cause the function to fail.

Parameters

- **anchors** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getDeviceIds, getPositioningAnchorIds, getTagIds

getAnchorSelectionMode(*mode*, *remote_id=None*)
Obtains the Pozyx's anchor selection mode.

Parameters

- **mode** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getAngularVelocity_dps(*angular_vel*, *remote_id=None*)
Obtain the Pozyx's angular velocity sensor data in dps(degrees per second).

Parameters

- **angular_vel** – Container for the read data. AngularVelocity().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getCalibrationStatus(*calibration_status*, *remote_id=None*)
Obtains the Pozyx's calibration status.

Parameters

- **calibration_status** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getConfigModeGPIO(*gpio_num*, *mode*, *remote_id=None*)
Obtain the Pozyx's configuration mode of the selected GPIO pin.

Parameters

- **gpio_num** – GPIO pin number, 1 to 4.
- **mode** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getGPIO, getConfigPullGPIO

getConfigPullGPIO(*gpio_num*, *pull*, *remote_id=None*)
Obtain the Pozyx's selected GPIO pin pull.

Parameters

- **gpio_num** – GPIO pin number, 1 to 4.
- **pull** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getGPIO, getConfigModeGPIO

getCoordinates(*coordinates*, *remote_id=None*)
Obtains the Pozyx's coordinates. These are either set manually or by positioning.

Parameters

- **coordinates** – Container for the read data. Coordinates().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getDeviceCoordinates(*device_id*, *coordinates*, *remote_id=None*)
Obtain the coordinates of the device with selected ID in the Pozyx's device list.

Parameters

- **device_id** – ID of desired device whose coordinates are of interest. NetworkID()
- **Data** (*or*) –
- **coordinates** – Container for the read data. Coordinates().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getDeviceDetails(*system_details*, *remote_id=None*)

Parameters

- **system_details** – Container for the read data. DeviceDetails.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getDeviceIds(*devices*, *remote_id=None*)
Obtain the IDs of all devices in the Pozyx's device list.

You need to make sure to know how many devices are in the list, as an incorrect size of anchors will cause the function to fail. Use getDeviceListSize to know this number.

Parameters

- **devices** – Container for the read data. DeviceList(list_size=size)
- **Data** (*or*) –
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getAnchorIds, getTagIds, getPositioningAnchorIds

Example

>> > list_size = SingleRegister() >> > self.getDeviceListSize(list_size) >> > device_list = DeviceList(list_size=list_size[0]) >> > self.getDeviceIds(device_list) >> > print(device_list) '0x60A0, 0x6070, 0x6891'

getDeviceListSize(*device_list_size*, *remote_id=None*)
Obtain the size of Pozyx's list of added devices.

Parameters

- **device_list_size** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getDeviceRangeInfo(*device_id*, *device_range*, *remote_id=None*)
Obtain the range information of the device with selected ID in the Pozyx's device list.

Parameters

- **device_id** – ID of desired device whose range measurement is of interest. NetworkID()
- **Data** (*or*) –
- **device_range** – Container for the read data. DeviceRange().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getErrorCode(*error_code*, *remote_id=None*)
Obtains the Pozyx's error code.

Parameters

- **error_code** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getErrorMessage(*error_code*)
Returns the system error string for the given error code

Parameters **error_code** – Error code for which to return the error message. int or SingleRegister

Returns string with error description

See also:

getErrorCode, getSystemError

getEulerAngles_deg(*euler_angles*, *remote_id=None*)
Obtain the Pozyx's euler angles sensor data in degrees(heading, roll, pitch).

Parameters

- **euler_angles** – Container for the read data. EulerAngles().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getFirmwareVersion(*firmware*, *remote_id=None*)
Obtains the Pozyx's firmware version.

Parameters

- **firmware** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getGPIO(*gpio_num*, *value*, *remote_id=None*)
Obtain the Pozyx's value of the selected GPIO pin, being either HIGH or LOW(physically 3.3V or 0V).

Parameters

- **gpio_num** – GPIO pin number, 1 to 4.
- **value** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getConfigPullGPIO, getConfigModeGPIO

getGravityVector_mg(*gravity_vector*, *remote_id=None*)
Obtain the Pozyx's gravity vector sensor data in mg.

Parameters

- **gravity_vector** – Container for the read data. Acceleration().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getHardwareVersion(*hardware*, *remote_id=None*)
Obtains the Pozyx's hardware version.

Parameters

- **hardware** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getHeight(*height*, *remote_id=None*)
Obtains the Pozyx's height coordinate.

Parameters

- **height** – Container for the read height data. Data([0], 'i').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getInterruptMask(*mask*, *remote_id=None*)
Obtains the Pozyx's interrupt mask.

Parameters

- **mask** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getLastDataLength(*data_length*, *remote_id=None*)
Obtain the size of the most recent data packet received by the Pozyx.

Parameters

- **data_length** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getLastNetworkId(*network_id*, *remote_id=None*)
Obtain the network ID of the last device Pozyx communicated with.

Parameters

- **network_id** – Container for the read data. NetworkID() or SingleRegister(size=2) or Data([0], 'H').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getLinearAcceleration_mg(*linear_acceleration*, *remote_id=None*)
Obtain the Pozyx's linear acceleration sensor data in mg.

Parameters

- **linear_acceleration** – Container for the read data. LinearAcceleration() or Acceleration().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getMagnetic_uT(*magnetic*, *remote_id=None*)
Obtain the Pozyx's magnetic sensor data in uT(microtesla).

Parameters

- **magnetic** – Container for the read data. Magnetic().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getMaxLinearAcceleration_mg(*max_linear_acceleration*, *remote_id=None*)
Obtain the Pozyx's acceleration sensor data in mg.

Parameters

- **max_linear_acceleration** – Container for the read data. MaxLinearAcceleration.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getNetworkId(*network_id*)
Obtains the Pozyx's network ID.

Parameters **network_id** – Container for the read data. NetworkID() or SingleRegister(size=2) or Data([0], 'H').

Returns POZYX_SUCCESS, POZYX_FAILURE

getNormalizedQuaternion(*quaternion*, *remote_id=None*)
Obtain the Pozyx's normalized quaternion sensor data that is required for ROS.

Parameters

- **quaternion** – Container for the read data. Quaternion().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getNumRegistersSaved(*remote_id=None*)
Obtains the number of registers saved to the Pozyx's flash memory.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns The number of saved registers.

getNumberOfAnchors(*nr_anchors*, *remote_id=None*)
Obtains the Pozyx's number of selected anchors.

Parameters

- **nr_anchors** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getOperationMode(*mode*, *remote_id=None*)
Obtains the Pozyx's mode of operation.

Parameters

- **mode** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getPositionAlgorithm(*algorithm*, *remote_id=None*)
Obtains the Pozyx's positioning algorithm.

Parameters

- **algorithm** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getPositionDimension(*dimension*, *remote_id=None*)
Obtains the Pozyx's positioning dimension.

Parameters

- **dimension** – Container the for read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getPositionError(*positioning_error*, *remote_id=None*)
Obtains the Pozyx's positioning error.

Parameters

- **positioning_error** – Container for the read data. PositionError().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getPositionFilterData(*filter_data*, *remote_id=None*)
NEW! Get the positioning filter data.

Use FilterData if you want to have a ready to go container for this data.

Parameters

- **filter_data** – Container for filter data. SingleRegister or FilterData
- **remote_id** (*optional*) – Remote Pozyx ID.

Example

```
>>> pozyx = PozyxLib()  # PozyxSerial has PozyxLib's functions, just for␣
↪generality
>>> filter_data = FilterData()
>>> pozyx.getPositionFilter(filter_data)
>>> print(filter_data)  # "Moving average filter with strength 10"
>>> print(filter_data.get_filter_name())  # "Moving average filter"
>>> print(filter_data.filter_type)  # "3"
>>> print(filter_data.filter_strength())  # "10"
```

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getPositionFilterStrength(*remote_id=None*)

NEW! Get the positioning filter strength.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getPositioningAnchorIds(*anchors*, *remote_id=None*)

Obtain the IDs of the anchors in the Pozyx's device list used for positioning.

You need to make sure to know how many anchors are used, as an incorrect size of anchors will cause the function to fail. Use getNumberOfAnchors to know this number.

Parameters

- **anchors** – Container for the read data. DeviceList(list_size=size)
- **Data** (*or*) –
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getAnchorIds, getTagIds, getDeviceIds

Example

>> > list_size = SingleRegister() >> > self.getNumberOfAnchors(list_size) >> > anchor_list = DeviceList(list_size=list_size[0]) >> > self.getPositioningAnchorIds(anchor_list) >> > print(anchor_list) '0x6720, 0x6811, 0x6891'

getPositioningData(*positioning_data*)

getPressure_Pa(*pressure*, *remote_id=None*)

Obtain the Pozyx's pressure sensor data in Pa(pascal).

Parameters

- **pressure** – Container for the read data. Pressure or Data([0], 'I') (Data is DEPRECATED).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getQuaternion(*quaternion*, *remote_id=None*)
Obtain the Pozyx's quaternion sensor data.

Parameters

- **quaternion** – Container for the read data. Quaternion().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getRangingProtocol(*protocol*, *remote_id=None*)
Obtains the Pozyx's ranging protocol

Parameters

- **protocol** – Container for the read protocol data. SingleRegister or Data([0])
- **remote_id** (*optional*) – Remote Pozyx ID

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getSavedRegisters(*remote_id=None*)

getSelftest(*selftest*, *remote_id=None*)
Obtains the Pozyx's selftest.

Parameters

- **selftest** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getSensorMode(*sensor_mode*, *remote_id=None*)
Obtains the Pozyx's sensor mode.

Parameters

- **sensor_mode** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getSystemError(*remote_id=None*)
Returns the Pozyx's system error string.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns string with error description

See also:

getErrorCode, getErrorMessage

getTagIds(*tags*, *remote_id=None*)
Obtain the IDs of the tags in the Pozyx's device list.

You need to make sure to know how many tags are in the list, as an incorrect size of tags will cause the function to fail.

Parameters

- **tags** – Container for the read data. SingleRegister() or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

See also:

getDeviceIds, getAnchorIds, getPositioningAnchorIds

getTemperature_c(*temperature*, *remote_id=None*)
Obtain the Pozyx's temperature sensor data in C(celsius).

Parameters

- **temperature** – Container for the read data. Temperature or Data([0], 'b') (DEPRECATED).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getTxPower(*txgain_db*, *remote_id=None*)
DEPRECATED: use getUWBGain instead. Obtains the Pozyx's transmitter UWB gain in dB, as a float.

Parameters

- **txgain_db** – Container for the read data. Data([0], 'f').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getUWBChannel(*channel_num*, *remote_id=None*)
Obtains the Pozyx's UWB channel.

Parameters

- **channel_num** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getUWBGain(*uwb_gain_db*, *remote_id=None*)
Obtains the Pozyx's transmitter UWB gain in dB, as a float.

Parameters

- **uwb_gain_db** – Container for the read data. Data([0], 'f').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getUWBSettings(*UWB_settings*, *remote_id=None*)
Obtains the Pozyx's UWB settings.

Parameters

- **UWB_settings** – Container for the read data. UWBSettings().
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getUpdateInterval(*ms*, *remote_id=None*)
Obtains the Pozyx's update interval.

Parameters

- **ms** – Container for the read data. SingleRegister or Data([0]).

- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

getWhoAmI(*who_am_i*, *remote_id=None*)
Obtains the Pozyx's WHO_AM_I.

Parameters

- **who_am_i** – Container for the read data. SingleRegister or Data([0]).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

isRegisterSaved(*register_address*, *remote_id=None*)
Returns whether the given register is saved to the Pozyx's flash memory.

Parameters

- **register_address** – Register address to check if saved
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns 1 if the register is saved, 0 if it's not.

printDeviceInfo(*remote_id=None*)
Prints a Pozyx's basic info, such as firmware.

Mostly for debugging

printDeviceList(*remote_id=None*, *include_coordinates=True*, *prefix='\t- '*)
Prints a Pozyx's device list.

Parameters

- **remote_id** (*optional*) – Remote Pozyx ID
- **include_coordinates** (*bool, optional*) – Whether to include coordinates in the prints
- **prefix** (*str, optional*) – Prefix to prepend the device list

Returns None

rangingWithoutCheck(*destination_id*, *device_range*, *remote_id=None*)

remoteRegFunctionOnlyData(*destination*, *address*, *params*, *data*)
Performs a remote function without waiting for the acknowledgement.

Advanded custom internal use only, you're not expected to use this unless you know what you're doing.

remoteRegFunctionWithoutCheck(*destination*, *address*, *params*)

removeDevice(*device_id*, *remote_id=None*)
Removes a device from the Pozyx's device list, keeping the rest of the list intact

Parameters

- **device_id** – ID that needs to be removed. NetworkID or integer.
- **remote_id** (*optional*) – Remote Pozyx ID

Returns POZYX_SUCCESS, POZYX_FAILURE

resetSystem(*remote_id=None*)
Resets the Pozyx device.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

saveAnchorIds(*remote_id=None*)
Saves the anchor IDs used in positioning to the Pozyx's flash memory.

This means that upon reset, the Pozyx won't need to be recalibrated before performing positioning.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

saveConfiguration(*save_type*, *registers=None*, *remote_id=None*)
General function to save the Pozyx's configuration to its flash memory.

This constitutes three different Pozyx configurations to save, and each have their specialised derived function:
POZYX_FLASH_REGS: This saves the passed Pozyx registers if they're writable, see saveRegisters. PozyxConstants.FLASH_SAVE_ANCHOR_IDS: This saves the anchors used during positioning, see saveAnchorIds. POZYX_FLASH_NETWORK: This saves the device list to the Pozyx device, see saveNetwork.

It is recommended to use the derived functions, as these are not just easier to use, but also more descriptive than this general save function.

DISCLAIMER: Make sure to not abuse this function in your code, as the flash memory only has a finite number of writecycles available, adhere to the Arduino's mentality in using flash memory.

Parameters

- **save_type** – Type of configuration to save. See above.
- **registers** (*optional*) – Registers to save to the flash memory. Data([register1, register2, ...]) or [register1, register2, ...] These registers have to be writable. Saving the UWB gain is currently not working.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

saveNetwork(*remote_id=None*)
Saves the Pozyx's device list to its flash memory.

This means that upon a reset, the Pozyx will still have the same configured device list.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

savePositioningSettings(*remote_id=None*)

saveRegisters(*registers*, *remote_id=None*)
Saves the given registers to the Pozyx's flash memory, if these are writable registers.

This means that upon reset, the Pozyx will use these saved values instead of the default values. This is especially practical when changing UWB settings of an entire network, making it unnecessary to re - set these when resetting or repowering a device.

DISCLAIMER: Make sure to not abuse this function in your code, as the flash memory only has a finite number of writecycles available, adhere to the Arduino's mentality in using flash memory.

Parameters

- **registers** – Registers to save to the flash memory. Data([register1, register2, ...]) or [register1, register2, ...] These registers have to be writable. Saving the UWB gain is currently not working.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

saveUWBSettings(*remote_id=None*)
Saves the Pozyx's UWB settings to its flash memory.

This means that upon a reset, the Pozyx will still have the same configured UWB settings. As of writing, PozyxRegisters.UWB_GAIN is not savable yet.

Parameters **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setConfigGPIO(*gpio_num*, *mode*, *pull*, *remote_id=None*)
Set the Pozyx's selected GPIO pin configuration(mode and pull).

Parameters

- **gpio_num** – GPIO pin number, 1 to 4.
- **mode** – GPIO configuration mode. integer mode or SingleRegister(mode)
- **pull** – GPIO configuration pull. integer pull or SingleRegister(pull)
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setCoordinates(*coordinates*, *remote_id=None*)
Set the Pozyx's coordinates.

Parameters

- **coordinates** – Desired Pozyx coordinates. Coordinates() or [x, y, z].
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setGPIO(*gpio_num*, *value*, *remote_id=None*)
Set the Pozyx's selected GPIO pin output.

Parameters

- **gpio_num** – GPIO pin number, 1 to 4
- **value** – GPIO output value, either HIGH(1) or LOW(0). Physically, 3.3V or 0V. integer value or SingleRegister(value).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setHeight(*height*, *remote_id=None*)
Sets the Pozyx device's height.

Parameters

- **height** – Desired Pozyx height. integer height or Data([height], 'i').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setInterruptMask(*mask*, *remote_id=None*)
Set the Pozyx's interrupt mask.

Parameters

- **mask** – Interrupt mask. See PozyxRegisters.INTERRUPT_MASK register. integer mask or SingleRegister(mask)
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setLed(*led_num*, *state*, *remote_id=None*)
Set the Pozyx's selected LED state.

Parameters

- **led_num** – LED pin number, 1 to 4
- **state** – LED output state. Boolean. True = on and False = off, you can use POZYX_LED_ON and POZYX_LED_OFF instead.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setLedConfig(*config*, *remote_id=None*)
Set the Pozyx's LED configuration.

Parameters

- **config** – LED configuration. See PozyxRegisters.LED_CONFIGURATION register. integer configuration or SingleRegister(configuration)
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setNetworkId(*network_id*, *remote_id=None*)
Set the Pozyx's network ID.

If using this remotely, make sure to change the network ID to the new ID in subsequent code, as its ID will have changed and using the old ID will not work.

Parameters

- **network_id** – New Network ID. integer ID or NetworkID(ID) or SingleRegister(ID, size=2)
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setPositionAlgorithm(*algorithm*, *dimension*, *remote_id=None*)
Set the Pozyx's positioning algorithm.

Note that currently only PozyxConstants.POSITIONING_ALGORITHM_UWB_ONLY and PozyxConstants.POSITIONING_ALGORITHM_TRACKING are implemented.

Parameters

- **algorithm** – Positioning algorithm. integer algorithm or SingleRegister(algorithm).
- **dimension** – Positioning dimension. integer dimension or SingleRegister(dimension).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setPositionAlgorithmNormal(*remote_id=None*)

setPositionAlgorithmTracking(*remote_id=None*)

setPositionFilter(*filter_type*, *filter_strength*, *remote_id=None*)
Set the Pozyx's positioning filter.

Note that currently only PozyxConstants.FILTER_TYPE_MOVING_AVERAGE, PozyxConstants.FILTER_TYPE_MOVING_MEDIAN and PozyxConstants.FILTER_TYPE_FIR are implemented.

Parameters

- **filter_type** – Positioning filter type. Integer or SingleRegister.
- **filter_strength** – Positioning filter strength. Integer or SingleRegister.
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setPositioningAnchorIds(*anchors*, *remote_id=None*)
Set the anchors the Pozyx will use for positioning.

Parameters

- **anchors** – List of anchors that'll be used for positioning. DeviceList() or [anchor_id1, anchor_id2, ...]
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setPositioningFilterFIR(*filter_strength*, *remote_id=None*)

setPositioningFilterMovingAverage(*filter_strength*, *remote_id=None*)

setPositioningFilterMovingMedian(*filter_strength*, *remote_id=None*)

setPositioningFilterNone(*remote_id=None*)

setRangingProtocol(*protocol*, *remote_id=None*)
Set the Pozyx's ranging protocol.

Parameters

- **protocol** – the new ranging protocol. See PozyxRegisters.RANGING_PROTOCOL register. integer or SingleRegister(protocol)
- **remote_id** (*optional*) – Remote Pozyx ID

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setRangingProtocolFast(*remote_id=None*)

setRangingProtocolPrecision(*remote_id=None*)

setSelectionOfAnchors(*mode*, *number_of_anchors*, *remote_id=None*)
Set the Pozyx's coordinates.

Parameters

- **mode** – Anchor selection mode. integer mode or SingleRegister(mode).
- **number_of_anchors** (*int, SingleRegister*) – Number of anchors used in positioning. integer nr_anchors or SingleRegister(nr_anchors).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setSelectionOfAnchorsAutomatic(*number_of_anchors*, *remote_id=None*)

setSelectionOfAnchorsManual(*number_of_anchors*, *remote_id=None*)

setSensorMode(*sensor_mode*, *remote_id=None*)
Set the Pozyx's sensor mode.

Parameters

- **sensor_mode** – New sensor mode. See PozyxRegisters.SENSORS_MODE register. integer sensor_mode or SingleRegister(sensor_mode).
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setTxPower(*txgain_db*, *remote_id=None*)
DEPRECATED: use getUWBGain instead. Set the Pozyx's UWB transceiver gain.

Parameters

- **txgain_db** – The new transceiver gain in dB, a value between 0.0 and 33.0. float gain or Data([gain], 'f').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setUWBChannel(*channel_num*, *remote_id=None*)
Set the Pozyx's UWB channel.

If using this remotely, remember to change the local UWB channel as well to make sure you are still able to communicate with the remote device.

Parameters

- **channel_num** – The new UWB channel, being either 1, 2, 3, 4, 5 or 7. See PozyxRegisters.UWB_CHANNEL register. integer channel or SingleRegister(channel)
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setUWBGain(*uwb_gain_db*, *remote_id=None*)
Set the Pozyx's UWB transceiver gain.

Parameters

- **uwb_gain_db** – The new transceiver gain in dB, a value between 0.0 and 33.0. float gain or Data([gain], 'f').
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setUWBSettings(*uwb_settings*, *remote_id=None*, *save_to_flash=False*)
Set the Pozyx's UWB settings.

If using this remotely, remember to change the local UWB settings as well to make sure you are still able to communicate with the remote device.

Parameters **uwb_settings** – The new UWB settings. UWBSettings() or [channel, bitrate, prf, plen, gain_db]

Kwargs: remote_id: Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

setUpdateInterval(*ms*, *remote_id=None*)
Set the Pozyx's update interval in ms(milliseconds).

Parameters

- **ms** – Update interval in ms. integer ms or SingleRegister(ms, size=2)
- **remote_id** (*optional*) – Remote Pozyx ID.

Returns POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT

waitForFlagSafeFast(*interrupt_flag*, *timeout_s*, *interrupt=None*)
A fast variation of wait for flag, tripling the polling speed. Useful for ranging on very fast UWB settings.

Returns True, False

3.3 Data structures

3.3.1 Generic

3.3.2 Device data

3.3.3 Sensor data

3.4 Constants, bitmasks, registers

CHAPTER 4

Troubleshooting

4.1 FAQ

4.2 Lost a device?

4.3 Contacting support

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

- Run the troubleshooting.py (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.

Python Module Index

p

Index

H

I

L

P

R

S

V

W