Help on PozyxSerial in module pypozyx.pozyx\_serial object:

class PozyxSerial(pypozyx.lib.PozyxLib)

| PozyxSerial(port, baudrate=115200, timeout=0.1, write\_timeout=0.1, print\_output=False, debug\_trace=False, show\_trace=False, suppress\_warnings=False)

|

| 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.

|

| Args:

| 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])

|

| Method resolution order:

| PozyxSerial

| pypozyx.lib.PozyxLib

| pypozyx.core.PozyxCore

| builtins.object

|

| Methods defined here:

|

| \_\_init\_\_(self, port, baudrate=115200, timeout=0.1, write\_timeout=0.1, print\_output=False, debug\_trace=False, show\_trace=False, suppress\_warnings=False)

| Initializes the PozyxSerial object. See above for details.

|

| connectToPozyx(self, port, baudrate, timeout, write\_timeout)

| Attempts to connect to the Pozyx via a serial connection

|

| regFunction(self, address, params, data)

| Performs a register function on the Pozyx, if the address is a register

| function.

|

| Args:

| 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(self, address, data)

| Reads data from the Pozyx registers, starting at a register address,

| if registers are readable.

|

| Args:

| 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(self, address, data)

| Writes data to the Pozyx registers, starting at a register address,

| if registers are writable.

|

| Args:

| 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(self, s)

| Auxiliary. Performs a serial write to and read from the Pozyx.

|

| Args:

| 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(self)

| Validates whether the connected device is indeed a Pozyx device

|

| waitForFlag(self, interrupt\_flag, timeout\_s, interrupt=None)

| Waits for a certain interrupt flag to be triggered, indicating that

| that type of interrupt occured.

|

| Args:

| 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

|

| ----------------------------------------------------------------------

| Methods inherited from pypozyx.lib.PozyxLib:

|

| addDevice(self, device\_coordinates, remote\_id=None)

| Adds a device to the Pozyx's device list. Can be either a tag or anchor.

|

| Args:

| 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(self, id\_=None)

|

| changeDeviceCoordinates(self, 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

|

| Args:

| 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(self, interrupt\_flag, timeout\_s, interrupt=None)

| A fast variant of checkForFlag, using waitForFLagFast, useful for ranging on very fast UWB settings.

|

| Args:

| 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(self, suspected\_uwb\_settings, remote\_id=None, equal\_gain=True)

|

| clearConfiguration(self, remote\_id=None)

| Clears the Pozyx's flash memory.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| clearDevices(self, remote\_id=None)

| Clears the Pozyx's device list.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| configInterruptPin(self, pin\_number=0, mode=0, active\_high=False, latch=False, remote\_id=None)

| Configures the interrupt pin via the PozyxRegisters.INTERRUPT\_PIN register.

|

| Args:

| 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(self, anchor\_list, anchor\_select=1, remote\_id=None)

| Configures a set of anchors as the relevant anchors on a device

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, function, uwb\_settings, \*args, \*\*kwargs)

|

| doOptimalDiscovery(self, discovery\_type=2, slots=3, timeout=None)

| Performs a discovery with slot\_duration optimised for the device's UWB settings.

|

| doPositioning(self, position, dimension=3, height=<pypozyx.structures.generic.Data object at 0x000001A6F0FC1D30>, 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.

|

| Args:

| 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(self, 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

|

| Args:

| position: Container for the positioning coordinates. Coordinates object.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| doPositioningWithData(self, positioning\_data, remote\_id=None, timeout=None)

| # TODO needs a lot of refactoring...

|

| doPositioningWithDataSlave(self, 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

|

| Args:

| positioning\_data: Container for the positioning coordinates. PositioningData object.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| doRanging(self, 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.

|

| Args:

| 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(self, 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

|

| Args:

| 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(self, acceleration, remote\_id=None)

| Obtain the Pozyx's acceleration sensor data in mg.

|

| Args:

| acceleration: Container for the read data. Acceleration().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getAllSensorData(self, sensor\_data, remote\_id=None)

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

|

| Args:

| sensor\_data: Container for the read data. SensorData() or RawSensorData().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getAnchorIds(self, 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.

|

| Args:

| 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(self, mode, remote\_id=None)

| Obtains the Pozyx's anchor selection mode.

|

| Args:

| 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(self, angular\_vel, remote\_id=None)

| Obtain the Pozyx's angular velocity sensor data in dps(degrees per second).

|

| Args:

| angular\_vel: Container for the read data. AngularVelocity().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getCalibrationStatus(self, calibration\_status, remote\_id=None)

| Obtains the Pozyx's calibration status.

|

| Args:

| 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(self, gpio\_num, mode, remote\_id=None)

| Obtain the Pozyx's configuration mode of the selected GPIO pin.

|

| Args:

| 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(self, gpio\_num, pull, remote\_id=None)

| Obtain the Pozyx's selected GPIO pin pull.

|

| Args:

| 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(self, coordinates, remote\_id=None)

| Obtains the Pozyx's coordinates. These are either set manually or by positioning.

|

| Args:

| coordinates: Container for the read data. Coordinates().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getDeviceCoordinates(self, device\_id, coordinates, remote\_id=None)

| Obtain the coordinates of the device with selected ID in the Pozyx's device list.

|

| Args:

| device\_id: ID of desired device whose coordinates are of interest. NetworkID()

| or Data([ID], 'H') or integer ID.

| coordinates: Container for the read data. Coordinates().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getDeviceDetails(self, system\_details, remote\_id=None)

| Args:

| system\_details: Container for the read data. DeviceDetails.

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getDeviceIds(self, 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.

|

| Args:

| devices: Container for the read data. DeviceList(list\_size=size)

| or Data([0] \* size, 'H' \* size).

| 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(self, device\_list\_size, remote\_id=None)

| Obtain the size of Pozyx's list of added devices.

|

| Args:

| 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(self, device\_id, device\_range, remote\_id=None)

| Obtain the range information of the device with selected ID in the Pozyx's device list.

|

| Args:

| device\_id: ID of desired device whose range measurement is of interest. NetworkID()

| or Data([ID], 'H') or integer ID.

| device\_range: Container for the read data. DeviceRange().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getErrorCode(self, error\_code, remote\_id=None)

| Obtains the Pozyx's error code.

|

| Args:

| 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(self, error\_code)

| Returns the system error string for the given error code

|

| Args:

| 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(self, euler\_angles, remote\_id=None)

| Obtain the Pozyx's euler angles sensor data in degrees(heading, roll, pitch).

|

| Args:

| euler\_angles: Container for the read data. EulerAngles().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getFirmwareVersion(self, firmware, remote\_id=None)

| Obtains the Pozyx's firmware version.

|

| Args:

| firmware: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getGPIO(self, 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).

|

| Args:

| 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(self, gravity\_vector, remote\_id=None)

| Obtain the Pozyx's gravity vector sensor data in mg.

|

| Args:

| gravity\_vector: Container for the read data. Acceleration().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getHardwareVersion(self, hardware, remote\_id=None)

| Obtains the Pozyx's hardware version.

|

| Args:

| hardware: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getHeight(self, height, remote\_id=None)

| Obtains the Pozyx's height coordinate.

|

| Args:

| height: Container for the read height data. Data([0], 'i').

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getInterruptMask(self, mask, remote\_id=None)

| Obtains the Pozyx's interrupt mask.

|

| Args:

| mask: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getLastDataLength(self, data\_length, remote\_id=None)

| Obtain the size of the most recent data packet received by the Pozyx.

|

| Args:

| 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(self, network\_id, remote\_id=None)

| Obtain the network ID of the last device Pozyx communicated with.

|

| Args:

| 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(self, linear\_acceleration, remote\_id=None)

| Obtain the Pozyx's linear acceleration sensor data in mg.

|

| Args:

| 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(self, magnetic, remote\_id=None)

| Obtain the Pozyx's magnetic sensor data in uT(microtesla).

|

| Args:

| magnetic: Container for the read data. Magnetic().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getMaxLinearAcceleration\_mg(self, max\_linear\_acceleration, remote\_id=None)

| Obtain the Pozyx's acceleration sensor data in mg.

|

| Args:

| max\_linear\_acceleration: Container for the read data. MaxLinearAcceleration.

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getNetworkId(self, network\_id)

| Obtains the Pozyx's network ID.

|

| Args:

| network\_id: Container for the read data. NetworkID() or SingleRegister(size=2) or Data([0], 'H').

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| getNormalizedQuaternion(self, quaternion, remote\_id=None)

| Obtain the Pozyx's normalized quaternion sensor data that is required for ROS.

|

| Args:

| quaternion: Container for the read data. Quaternion().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getNumRegistersSaved(self, remote\_id=None)

| Obtains the number of registers saved to the Pozyx's flash memory.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| The number of saved registers.

|

| getNumberOfAnchors(self, nr\_anchors, remote\_id=None)

| Obtains the Pozyx's number of selected anchors.

|

| Args:

| 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(self, mode, remote\_id=None)

| Obtains the Pozyx's mode of operation.

|

| Args:

| mode: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getPositionAlgorithm(self, algorithm, remote\_id=None)

| Obtains the Pozyx's positioning algorithm.

|

| Args:

| algorithm: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getPositionDimension(self, dimension, remote\_id=None)

| Obtains the Pozyx's positioning dimension.

|

| Args:

| dimension: Container the for read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getPositionError(self, positioning\_error, remote\_id=None)

| Obtains the Pozyx's positioning error.

|

| Args:

| positioning\_error: Container for the read data. PositionError().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getPositionFilterData(self, 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.

|

| Args:

| 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(self, remote\_id=None)

| \*\*NEW\*\*! Get the positioning filter strength.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getPositioningAnchorIds(self, 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.

|

| Args:

| anchors: Container for the read data. DeviceList(list\_size=size)

| or Data([0] \* size, 'H' \* size).

| 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(self, positioning\_data)

|

| getPressure\_Pa(self, pressure, remote\_id=None)

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

|

| Args:

| 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(self, quaternion, remote\_id=None)

| Obtain the Pozyx's quaternion sensor data.

|

| Args:

| quaternion: Container for the read data. Quaternion().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getRangingProtocol(self, protocol, remote\_id=None)

| Obtains the Pozyx's ranging protocol

|

| Args:

| 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(self, remote\_id=None)

|

| getSelftest(self, selftest, remote\_id=None)

| Obtains the Pozyx's selftest.

|

| Args:

| selftest: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getSensorMode(self, sensor\_mode, remote\_id=None)

| Obtains the Pozyx's sensor mode.

|

| Args:

| 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(self, remote\_id=None)

| Returns the Pozyx's system error string.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| string with error description

|

| See Also:

| getErrorCode, getErrorMessage

|

| getTagIds(self, 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.

|

| Args:

| 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(self, temperature, remote\_id=None)

| Obtain the Pozyx's temperature sensor data in C(celsius).

|

| Args:

| 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(self, txgain\_db, remote\_id=None)

| DEPRECATED: use getUWBGain instead. Obtains the Pozyx's transmitter UWB gain in dB, as a float.

|

| Args:

| txgain\_db: Container for the read data. Data([0], 'f').

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getUWBChannel(self, channel\_num, remote\_id=None)

| Obtains the Pozyx's UWB channel.

|

| Args:

| 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(self, uwb\_gain\_db, remote\_id=None)

| Obtains the Pozyx's transmitter UWB gain in dB, as a float.

|

| Args:

| 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(self, UWB\_settings, remote\_id=None)

| Obtains the Pozyx's UWB settings.

|

| Args:

| UWB\_settings: Container for the read data. UWBSettings().

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getUpdateInterval(self, ms, remote\_id=None)

| Obtains the Pozyx's update interval.

|

| Args:

| ms: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getWhoAmI(self, who\_am\_i, remote\_id=None)

| Obtains the Pozyx's WHO\_AM\_I.

|

| Args:

| 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(self, register\_address, remote\_id=None)

| Returns whether the given register is saved to the Pozyx's flash memory.

|

| Args:

| 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(self, remote\_id=None)

| Prints a Pozyx's basic info, such as firmware.

|

| Mostly for debugging

|

| printDeviceList(self, remote\_id=None, include\_coordinates=True, prefix='\t- ')

| Prints a Pozyx's device list.

|

| Args:

| 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(self, destination\_id, device\_range, remote\_id=None)

|

| remoteRegFunctionOnlyData(self, 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(self, destination, address, params)

| # TODO find new group for these four functions?

|

| removeDevice(self, device\_id, remote\_id=None)

| Removes a device from the Pozyx's device list, keeping the rest of the list intact

|

| Args:

| device\_id: ID that needs to be removed. NetworkID or integer.

| remote\_id (optional): Remote Pozyx ID

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| resetSystem(self, remote\_id=None)

| Resets the Pozyx device.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| saveAnchorIds(self, 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.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| saveConfiguration(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| savePositioningSettings(self, remote\_id=None)

|

| saveRegisters(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| setConfigGPIO(self, gpio\_num, mode, pull, remote\_id=None)

| Set the Pozyx's selected GPIO pin configuration(mode and pull).

|

| Args:

| 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(self, coordinates, remote\_id=None)

| Set the Pozyx's coordinates.

|

| Args:

| coordinates: Desired Pozyx coordinates. Coordinates() or [x, y, z].

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| setGPIO(self, gpio\_num, value, remote\_id=None)

| Set the Pozyx's selected GPIO pin output.

|

| Args:

| 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(self, height, remote\_id=None)

| Sets the Pozyx device's height.

|

| Args:

| height: Desired Pozyx height. integer height or Data([height], 'i').

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| setInterruptMask(self, mask, remote\_id=None)

| Set the Pozyx's interrupt mask.

|

| Args:

| 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(self, led\_num, state, remote\_id=None)

| Set the Pozyx's selected LED state.

|

| Args:

| 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(self, config, remote\_id=None)

| Set the Pozyx's LED configuration.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, remote\_id=None)

|

| setPositionAlgorithmTracking(self, remote\_id=None)

|

| setPositionFilter(self, 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.

|

| Args:

| 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(self, anchors, remote\_id=None)

| Set the anchors the Pozyx will use for positioning.

|

| Args:

| 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(self, filter\_strength, remote\_id=None)

|

| setPositioningFilterMovingAverage(self, filter\_strength, remote\_id=None)

|

| setPositioningFilterMovingMedian(self, filter\_strength, remote\_id=None)

|

| setPositioningFilterNone(self, remote\_id=None)

|

| setRangingProtocol(self, protocol, remote\_id=None)

| Set the Pozyx's ranging protocol.

|

| Args:

| 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(self, remote\_id=None)

|

| setRangingProtocolPrecision(self, remote\_id=None)

|

| setSelectionOfAnchors(self, mode, number\_of\_anchors, remote\_id=None)

| Set the Pozyx's coordinates.

|

| Args:

| 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(self, number\_of\_anchors, remote\_id=None)

|

| setSelectionOfAnchorsManual(self, number\_of\_anchors, remote\_id=None)

|

| setSensorMode(self, sensor\_mode, remote\_id=None)

| Set the Pozyx's sensor mode.

|

| Args:

| 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(self, txgain\_db, remote\_id=None)

| DEPRECATED: use getUWBGain instead. Set the Pozyx's UWB transceiver gain.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, uwb\_gain\_db, remote\_id=None)

| Set the Pozyx's UWB transceiver gain.

|

| Args:

| 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(self, 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.

|

| Args:

| 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(self, ms, remote\_id=None)

| Set the Pozyx's update interval in ms(milliseconds).

|

| Args:

| 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(self, 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

|

| ----------------------------------------------------------------------

| Methods inherited from pypozyx.core.PozyxCore:

|

| checkForFlag(self, interrupt\_flag, timeout\_s, interrupt=None)

| Performs waitForFlag\_safe and checks against errors or timeouts.

|

| This abstracts the waitForFlag status check routine commonly encountered

| in more complex library functions and checks the given flag against

| the error flag.

|

| Args:

| 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

|

| clearInterruptStatus(self)

|

| getInterruptStatus(self, interrupts, remote\_id=None)

| Obtains the Pozyx's interrupt register.

|

| Args:

| interrupts: Container for the read data. SingleRegister or Data([0]).

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| getRead(self, address, data, remote\_id=None)

| Reads Pozyx register data either locally or remotely.

|

| Args:

| address: The register address

| data: A ByteStructure - derived object that is the container of the read data.

|

| Kwargs:

| remote\_id: Remote ID for remote read.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| Example:

| >>> who\_am\_i = SingleRegister()

| >>> self.getRead(PozyxRegisters.WHO\_AM\_I, who\_am\_i)

| >>> print(who\_am\_i)

| 67

|

| getRxInfo(self, rx\_info, remote\_id=None)

| Obtain's information on the data the Pozyx received over UWB

|

| Args:

| rx\_info: Container for the RX Info, Data or RXInfo

| remote\_id (optional): Remote Pozyx ID.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| readRXBufferData(self, data, offset=0)

| Reads the device's receive buffer's data completely.

|

| Args:

| data: Container for the data to be read from the receiver buffer.

| offset (optional): Offset of where in the RX buffer to start read

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| remoteRegFunction(self, destination, address, params, data)

| Performs regFunction on a remote Pozyx device.

|

| Args:

| destination: Network ID of destination device. integer ID or NetworkID(ID).

| address: Register address to start the read operation from.

| params: Parameters for the register function. ByteStructure-derived object of uint8s.

| data: Container for the data returned by the register function. ByteStructure-derived object.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| remoteRegRead(self, destination, address, data)

| Performs regRead on a remote Pozyx device.

|

| Args:

| destination: Network ID of destination device. integer ID or NetworkID(ID).

| address: Register address to start the read operation from.

| data: Container for the read data. ByteStructure-derived object.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| remoteRegWrite(self, destination, address, data)

| Performs regWrite on a remote Pozyx device.

|

| Args:

| destination: Network ID of destination device. integer ID or NetworkID(ID).

| address: Register address to start the writing operation on.

| data: Contains the data to be written. ByteStructure-derived object.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| sendData(self, destination, data)

| Stores the data in the transmit buffer and then sends it to the device with ID destination.

|

| Args:

| destination: Network ID of destination. integer ID or NetworkID(ID)

| data: Data to send to the destination. Has to be a ByteStructure derived object.

|

| Performs the following code::

|

| >>>self.writeTXBufferData(data)

| >>>self.sendTXBufferData(destination)

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| sendTX(self, destination, operation)

| Sends the data in the transmit buffer to destination ID. Helper for sendData.

|

| Args:

| destination: Network ID of destination. integer ID or NetworkID(ID)

| operation: Type of TX operation. These vary depending on the desired operation.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| sendTXBufferData(self, destination)

| Sends the transmit buffer's data to the destination device.

|

| Args:

| destination: Network ID of destination. integer ID or NetworkID(ID)

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| sendTXFunction(self, destination)

| Sends the function parameters in the transmit buffer to the destination device.

|

| Args:

| destination: Network ID of destination. integer ID or NetworkID(ID)

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| sendTXRead(self, destination)

| Sends the read operation's data in the transmit buffer to the destination device.

|

| Args:

| destination: Network ID of destination. integer ID or NetworkID(ID)

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| sendTXWrite(self, destination)

| Sends the write operation's data in the transmit buffer to the destination device.

|

| Args:

| destination: Network ID of destination. integer ID or NetworkID(ID)

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| setWrite(self, address, data, remote\_id=None, local\_delay=0.001, remote\_delay=0.005)

| Writes data to Pozyx registers either locally or remotely.

|

| Args:

| address: The register address

| data: A ByteStructure - derived object that contains the data to be written.

| remote\_id (optional): Remote ID for remote read.

| local\_delay (optional): Delay after a local write

| remote\_delay (optional): Delay after a remote write

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| Examples:

| >>> leds = SingleRegister(0xFF)

| >>> self.setWrite(PozyxRegisters.LED\_CONTROL, leds)

|

| useFunction(self, function, params=None, data=None, remote\_id=None)

| Activates a Pozyx register function either locally or remotely.

|

| Args:

| address: The function address

|

| Kwargs:

| params: A ByteStructure - derived object that contains the parameters for the function.

| data: A ByteStructure - derived object that is the container of the read data.

| remote\_id: Remote ID for remote read.

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE, POZYX\_TIMEOUT

|

| Example:

| >>> self.useFunction(PozyxRegisters.CLEAR\_DEVICES)

|

| waitForFlagSafe(self, interrupt\_flag, timeout\_s, interrupt=None)

| Performs waitForFlag in polling mode.

|

| Args:

| 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

|

| waitForFlag\_safe(self, interrupt\_flag, timeout\_s, interrupt=None)

| Performs waitForFlag in polling mode.

|

| Args:

| 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

|

| writeTXBufferData(self, data, offset=0)

| Writes data to the device's transmit buffer at the offset address.

|

| Args:

| data: Data to write to the Pozyx buffer. Has to be a ByteStructure derived object.

| offset (optional): Offset in buffer to start writing data

|

| Returns:

| POZYX\_SUCCESS, POZYX\_FAILURE

|

| ----------------------------------------------------------------------

| Data descriptors inherited from pypozyx.core.PozyxCore:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)