# ORB-Python Documentation

**Nils Hoffmann**

**Nov 08, 2024**

# CONTENTS

Welcome to the documentation of the **ORB-Python** project.

CHAPTER

ONE

# PYTHON API

This part of the Documentation is the Python API. Here you can find how to program your Open Robotics Board using Python.

## 1.1 devices

The Devices Module allows you to program the ORBs configurable periferals as well as onboard devices.

### 1.1.1 sensor

The *Sensor* class represents a sensor device with configurable parameters and multiple data retrieval methods.

**class** devices.**sensor**(*port: int*, *type: int*, *mode: int*, *option: int*)

Initializes a sensor instance connected to a specified port with defined type, mode, and optional parameters.

**Parameters**

- **port** (*int*) – The port number the sensor is connected to.
- **type** (*int*) – Type of the sensor, such as *Analog*, *I2C*, *TOF*, *Touch*, or *UART*.
- **mode** (*int*) – Operating mode of the sensor.
- **option** (*int*) – Additional configuration option.

**config**(*type: int*, *mode: int*, *option: int*)

Configures the sensor.

**Parameters**

- **type** (*int*) – Type of the sensor (*Analog*, *I2C*, *TOF*, *Touch*, or *UART*).
- **mode** (*int*) – Operating mode of the sensor.
- **option** (*int*) – Additional configuration option.

**get**() → Dict["values": [int ,int], "type": int, "option": int, "lenExp": int]

Retrieves the sensor report as a dictionary representation.

**getDigital**() → int

Returns a digital reading from the sensor.

**Returns**

The digital value.

**Return type**
int

`getValueExt`(*channel: int*) → int

**Parameters**
**channel** (*int*) – The channel number to retrieve the value from.

**Returns**
The value from the specified channel.

**Return type**
int

**Ports**

`S1 = 0`
Sensor port 1.

`S2 = 1`
Sensor port 2.

`S3 = 2`
Sensor port 3.

`S4 = 3`
Sensor port 4.

**Types**

`Analog = 0`
Analog sensor type.

`I2C = 1`
I2C sensor type.

`TOF = 2`
Time-of-Flight sensor type.

`Touch = 3`
Touch sensor type.

`UART = 4`
UART sensor type.

### 1.1.2 motor

The *Motor* class represents a motor device with configurable parameters and various modes of operation.

`class devices.motor`(*port: int*, *direction: int*, *ticks: int*, *acc: int*, *kp: int*, *ki: int*)
Initializes a motor instance.

**Parameters**

- **port** (*int*) – The port that the motor is connected to.
- **direction** (*int*) – Direction of the motor (*FORWARD* or *REVERSE*.)
- **ticks** (*int*) – Number of encoder ticks for one revolution.
- **acc** (*int*) – Acceleration rate of the motor.

- **kp** (*int*) – Proportional gain for the motor control.
- **ki** (*int*) – Integral gain for the motor control.

**config**(*direction: int*, *ticks: int*, *acc: int*, *kp: int*, *ki: int*)

Configures the motor parameters.

**Parameters**

- **direction** (*int*) – Direction of the motor, either *FORWARD* or *REVERSE*.
- **ticks** (*int*) – Number of encoder ticks for one revolution.
- **acc** (*int*) – Acceleration rate.
- **kp** (*int*) – Proportional gain.
- **ki** (*int*) – Integral gain.

**set**(*mode: int*, *speed: int*, *position: int*)

Sets the motor's operating mode, speed, and position.

**Parameters**

- **mode** (*int*) – The mode in which to operate the motor (*POWER_MODE*, *BRAKE_MODE*, *SPEED_MODE*, or *MOVETO_MODE*).
- **speed** (*int*) – The speed of the motor.
- **position** (*int*) – The target position for the motor.

**get**() → Dict["speed": int, "power": int, "position": int]

**Returns**

A dictionary containing: - **speed**: The current speed. - **power**: The current power. - **position**: The current position.

**Ports**

**M1 = 0**

Motor port 1.

**M2 = 1**

Motor port 2.

**M3 = 2**

Motor port 3.

**M4 = 3**

Motor port 4.

**Modes**

**FORWARD = 1**

**REVERSE = -1**

**POWER_MODE = 0**

**BRAKE_MODE = 1**

**SPEED_MODE = 2**

**MOVETO_MODE = 3**

### 1.1.3 servo

The *Servo* class represents a servo device with specific parameters.

**class** devices.**servo**(*port: int*)

A servo object.

**Parameters**
**port** (*int*) – The port number the servo is connected to.

**set**(*speed: int*, *angle: int*)

Sets the speed and angle for the servo.

**Parameters**

- **speed** (*int*) – Speed of the servo movement.
- **angle** (*int*) – Target angle for the servo.

**Ports**

**S1**

Servo port 1.

**S2**

Servo port 2.

## 1.2 memory

The *memory* module provides functions to manage memory operations.

memory.**setMemory**(*addr: int*, *data: list | bytes*)

Sets memory at the specified address with the given data.

**Parameters**

- **addr** (*int*) – The memory address to set.
- **data** (*list | bytes*) – The data to be stored, which can be a list or bytes.

memory.**getMemory**(*addr: int*, *length: int*) → bytes

Returns memory starting from the specified address in the user-memory-region.

**Parameters**

- **addr** (*int*) – The starting memory address.
- **length** (*int*) – The number of bytes to retrieve.

**Returns**
The retrieved memory content as bytes.

**Return type**
bytes

memory.**clearMemory**()

Clears all stored memory data.

## 1.3 time

The *time* module provides utilities for working with time, including retrieving the current time and introducing delays.

`time.getTime()` → int

Returns the time, since microcontroller start.

**Returns**
The current time in milliseconds.

**Return type**
int

`time.wait(ms: int)`

Pauses execution for a given number of milliseconds.

**Parameters**
**ms** – Number of milliseconds to wait.

**Type**
int

## 1.4 monitor

The *monitor* module provides functionality for ORB-Monitor communication.

`monitor.getKey()` → int

**Returns**
Current key pressed on the monitor.

**Return type**
int

`monitor.setText(text: str)`

Sets the text to be displayed on the monitor. At the Moment only 32 characters are supported for one Line.

**Parameters**
**text** (*str*) – The text to display.

`class monitor.keys`

A class representing key constants for the *monitor*.

Each key constant has a unique integer value.

**Available keys**:

- **NO_KEY** = 0
- **A1** = 1
- **A2** = 2
- **A3** = 3
- **A4** = 4
- **A5** = 5
- **A6** = 6

- **A7** = 7
- **A8** = 8
- **B1** = 9
- **B2** = 10
- **B3** = 11
- **B4** = 12
- **B5** = 13
- **B6** = 14
- **B7** = 15
- **B8** = 16
- **B9** = 17
- **B10** = 18
- **B11** = 19
- **B12** = 20
- **C1** = 21

CHAPTER

TWO

# VM API

class **PythonVM**

A class to manage the Micropython-VirtualMachine. Providing methods for running, stopping, and retrieving the VM's status.

void PythonVM::**run**(LoadLengthFunction loadLength, LoadProgramFunction loadProgram, uint8_t arg)

Starts the VM with the specified load functions and argument.

**Parameters**

- **loadLength** – Function pointer that returns the program length
- **loadProgram** – Function pointer that loads the program data
- **arg** – Additional argument passed to the program

bool PythonVM::**isRunning**()

Checks if the VM is currently running.

**Returns**
True if the VM is running, otherwise False

**Rtype**
bool

void PythonVM::**stopProgram**()

Stops the currently running program in the VM.

int PythonVM::**getExitStatus**()

Retrieves the exit status of the VM.

**Returns**
Exit status code

**Rtype**
int

const char *PythonVM::**getExitInfo**()

Returns additional information about the VM's exit status.

**Returns**
Exit information string

**Rtype**
const char*

enum **Status**

Enumeration for VM exit statuses, indicating various exit conditions.

**Status codes**:

- **NORMAL**: Program exited normally.
- **EXCEPTION**: Program exited with an exception.
- **INTERRUPT**: Program was interrupted by User.

# INDEX