**Introduction to API**

API or Application Programming Interface refers to a number of preset programs. Before utilization, it is required to import API library:

# for mycobot,mecharm

from pymycobot.mycobot import MyCobot

# for mypalletizer

from pymycobot.mypalletizer import MyPalletizer

# for myBuddy

from pymycobot.mybuddy import MyBuddy

**Notice:** Functions with return value are required to use print() to print value. For example, if you want to get the speed value, type print(get\_speed()), instead of get\_speed().

**1 Overall Status**

**1.1** power\_on()

* **Function:** Atom open communication (default open)
* **Return Value:** None

**1.2** power\_off()

* **Function**: Atom turn off communication
* **Return Value:** None

**1.3** is\_power\_on()

* **Function:** judge whether robot arms is powered on or not
* **Return Value:**
  + 1: power on
  + 0: power off
  + -1: error

**1.4** release\_all\_servos

* **Function:** release all robot arms
* **Return Value:** None

**1.5** is\_controller\_connected

* **Function:** check if connected with Atom.
* **Return Value:**
  + 1: connected
  + 0: not connected
  + -1: error

**2 Operating Mode**

**2.1**pause()

* **Function:** pause motion
* **Return Value:** None

**2.2** stop()

* **Function:** stop motion
* **Return Value:** None

**2.3** resume()

* **Function:** resume motion
* **Return Value:** None

**2.4** is\_paused()

* **Function:** judge whether motion pauses or not
* **Return Value:**
  + 1: pause
  + 0: not pause
  + -1: error

**2.5** get\_speed()

* **Function:** get motion speed
* **Return Value:** range from 0-100

**2.6** set\_speed()

* **Function:** set motion speed
* **Parameter:** range from 0-100
* **Return Value:** None

**2.7** get\_joint\_min\_angle(joint\_id)

* **Function:** get minimum speed of a joint
* **Parameter:** range from 1-6 or 1-4
* **Return Value:** angle value

**2.8** get\_joint\_max\_angle(joint\_id)

* **Function:** get maximum speed of a joint
* **Parameter:** range from 1-6 or 1-4
* **Return Value:** angle value

**2.9** is\_servo\_enable(servo id)

* **Function:** judge whether a servo is enabled
* **Parameter:** range from 1-6 or 1-4
* **Return Value:**
  + 1: enabled
  + 0: not enabled
  + -1: error

**2.10** is\_all\_servo\_enable()

* **Function:** judge whether all servos are enabled
* **Return Value:**
  + 1: enabled
  + 0: not enabled
  + -1: error

**2.11** release\_servo(servo\_id)

* **Function:** release a servo
* **Parameter:** range from 1-6 or 1-4
* **Return Value:**
  + 1: enabled
  + 0: not enabled
  + -1: error

**2.12** get\_tof\_distance()

* **Function:** get tested distance
* **Return Value:** distance value

**2.13** get\_error\_information()

* **Function:** get error message.
* **Return Value:**
  + 0: no error message.
  + 1 ~ 6: The corresponding joint exceeds the limit.
  + 16 ~ 19: collision protection.
  + 32: Kinematics inverse solution has no solution.
  + 33 ~ 34: Linear motion has no adjacent solution.

**2.14** clear\_error\_information()

* **Function:** clear error message

**3 MDI Mode**

**Notice:** Different types of manipulators have different limits, and the angle and coordinate limits that can be set are also different. Refer to the parameter introduction section.

**3.1** get\_angles()

* **Function:** get the degree of all joints.
* **Returns**: A float list of all degree.

**3.2** send\_angle(id, degree, speed)

* **Function:** Send one degree of joint to robot arm.
* **Parameters**
  + id: Joint id(genre.Angle) / int 1-6
  + degree: degree value(float)
  + speed: (int) 0 ~ 100

##Example:

from pymycobot.mycobot import MyCobot

from pymycobot.genre import Angle

mycobot = MyCobot('/dev/ttyUSB0')

mycobot.send\_angle(Angle.J2.value, 10, 50)

**3.3** send\_angles(degrees, speed)

* **Function:** Send the degrees of all joints to robot arm.
* **Parameters:**
  + degrees: a list of degree value(List[float]), length 6 or 4.
  + speed: (int) 0 ~ 100

##Example:

from pymycobot.mycobot import MyCobot

from pymycobot.genre import Angle

mycobot = MyCobot('/dev/ttyUSB0')

mycobot.send\_angles([0,0,0,0,0,0], 80)

**3.4** get\_coords()

* **Function:** get the Coords from robot arm, coordinate system based on base.
* **Returns**: A float list of coord:[x, y, z, rx, ry, rz] or [x, y, z, rx]

**3.5** send\_coord(id, coord, speed)

* **Function:** send one coord to robot arm.
* **Parameters:**
  + id: coord id(genre.Coord) / int 1-6
  + coord: coord value(float)
  + speed: (int) 0 ~ 100

##Example:

from pymycobot.mycobot import MyCobot

from pymycobot.genre import Coord

mycobot = MyCobot('/dev/ttyUSB0')

mycobot.send\_coord(Coord.X.value, -40, 70)

**3.6** send\_coords(coords, speed, mode)

* **Function:** send all coords to robot arm.
* **Parameters:**
  + coords: a list of coords value(List[float]), length 6.
  + speed: (int) 0 ~ 100
  + mode: (int): 0 - angular, 1 - linear

##Example:

from pymycobot.mycobot import MyCobot

from pymycobot.genre import Coord

mycobot = MyCobot('/dev/ttyUSB0')

mycobot.send\_coords([160, 160, 160, 0, 0, 0], 70, 0)

**3.7** get\_encoders()

* **Function:** get encoders of all joint
* **Parameter**: a list of encoder values, at the length of 4 or 6

**3.8** get\_encoder(joint\_id)

* **Function:** get encoders of a joint
* **Parameter**: joint ID, ranging from 1-4 or 1-6

**3.9** get\_radians()

* **Function:** get the radians of all joints
* **Returns**: A float list of radian

**3.10** send\_radians(radians, speed)

* **Function:** send the radians of all joint to robot arm.
* **Parameter**:
  + radians: a list of radian value(List[float]), length 6 or 4.
  + speed: (int) 0 ~ 100

##Example:

from pymycobot.mycobot import MyCobot

from pymycobot.genre import Angle

mycobot = MyCobot('/dev/ttyUSB0')

mycobot.send\_radian([1,1,1,1,1,1], 70)

**3.11** sync\_send\_angles(degrees, speed, timeout=7)

* **Description**: send the angle in synchronous state and return when the target point is reached
* **Parameters:**
  + degrees: a list of degree value(List[float]), length 6.
  + speed: (int) 0 ~ 100
  + timeout: default 7s.

**3.12** sync\_send\_coords(coords, speed, mode, timeout=7)

* **Function:** send the coord in synchronous state and return when the target point is reached
* **Parameters:**
  + coords: a list of coords value(List[float])
  + speed: (int) 0 ~ 100
  + mode: (int): 0 - angular, 1 - linear
  + timeout: default 7s.

**3.13** is\_in\_position(data, flag)

* **Function:** judge whether in the position.
* **Parameters:**
  + data: A data list, angles or coords, length 6 or 4.
  + flag: Tag the data type, 0 - angles, 1 - coords.
* **Return Value:**
  + 1 - true
  + 0 - false
  + -1 - error

**3.14** is\_moving()

* **Function:** judge whether the robot is moving
* **Return Value:**
  + 1 moving
  + 0 not moving
  + -1 error

**3.15** set\_color(r, g, b)

* **Function:** set the color of RGB light panel
* **Parameters:**
  + R: 0-255
  + G: 0-255
  + B: 0-255
* **Return Value:** None

**3.16** get\_radians()

* **Function:** get radians of all arms
* **Return Value:** a list of radian values

**3.17** send\_radians(radians, speed)

* **Function:** send radians and speed to all arms
* **Parameters:**
  + radians: radians values of arms
  + speed: speed of arms

**4 JOG Mode**

**4.1** jog\_angle(joint\_id, direction, speed)

* **Function:** jog control angle
* **Parameters:**
  + joint\_id: (int) 1 ~ 6
  + direction: 0 - decrease, 1 - increase
  + speed: 0 ~ 100

**4.2** jog\_coord(coord\_id, direction, speed)

* **Function:** jog control coord.
* **Parameters:**
  + coord\_id: (int) 1 ~ 6
  + direction: 0 - decrease, 1 - increase
  + speed: 0 ~ 100

**4.3**jog\_stop()

* **Function:** stop jog moving
* **Return Value:** None

**4.4** pause()

* **Function:** Pause motion
* **Return Value:** None

**4.5** resume()

* **Function:** recovery motion
* **Return Value:** None

**4.6** stop()

* **Function:** stop motion
* **Return Value:** None

**4.7** is\_paused()

* **Function**: judge whether the manipulator pauses or not
* **Returns** :
  + 1 - paused
  + 0 - not paused
  + -1 - error

**4.8** set\_encoder(joint\_id, encoder)

* **Function:** set a single joint rotation to the specified potential value.
* **Parameters:** 
  + joint\_id: (int) 1 ~ 6 or 1~4
  + encoder: 0 ~ 4096

**4.9** get\_encoder(joint\_id)

* **Function:** obtain the specified joint potential value.
* **Parameters**: joint\_id: (int) 1 ~ 6 or 1~4
* **Returns**:
  + encoder: 0 ~ 4096

**4.10** set\_encoders(encoders, sp)

* **Function:** Set the six joints of the manipulator to execute synchronously to the specified position.
* **Parameters:**
  + encoders: A encoder list, length 6.
  + speed: speed 0 - 100

**4.11** get\_encoders()

* **Function:** get the six joints of the manipulator.
* **Returns**: a list of encoder (list)

**5 Servo Control**

**5.1** set\_servo\_data(servo\_no, data\_id, value)

* **Function:** set the data parameters of the specified address of the steering gear
* **Parameters**:
  + servo\_no: Serial number of articulated steering gear, 1 - 6.
  + data\_id: Data address.
  + value: 0 - 4096
* **Return Value:** None

**5.2** get\_servo\_data(servo\_no, data\_id)

* **Function:** read the data parameter of the specified address of the steering gear.
* **Parameters**:
  + servo\_no: Serial number of articulated steering gear, 1 - 6.
  + data\_id: Data address.
* **Returns**: value: 0 - 4096
  + 0: disable
  + 1: enable
  + -1: error

**5.3** set\_servo\_calibration(servo\_no)

* **Function:** the current position of the calibration joint actuator is the angle zero point, and the corresponding potential value is 2048.
* **Parameters**:
  + servo\_no: Serial number of articulated steering gear, 1 - 6.
* **Return Value:** None

**5.4** focus\_servo(servo\_id)

* **Function:** power on designated servo
* **Parameters**: servo\_id: 1 ~ 6 or 1~4
* **Return Value:** None

**6 Atom IO Control**

**6.1** set\_pin\_mode(pin\_no, pin\_mode)

* **Function:** set the state mode of the specified pin in atom
* **Parameters:**
  + pin\_no (int): Pin number.
  + pin\_mode (int): 0 - input, 1 - output, 2 - input\_pullup
* **Return Value:** None

**6.2** set\_digital\_output(pin\_no, pin\_signal)

* **Function:** set digital state of a pin
* **Parameters**
  + pin\_no (int):
  + pin\_signal (int): 0 / 1
* **Return Value:** None

**6.3** get\_digital\_input(self, pin\_no)

* **Function:** get digital state of a pin
* **Parameters**: pin\_no (int)
* **Return Value**: signal value

**7 Gripper Control**

**7.1** is\_gripper\_moving()

* **Function:** judge whether the gripper is moving or not
* **Return Value:**
  + 0 : not moving
  + 1 : is moving
  + -1: error data

**7.2** set\_gripper\_state(flag, speed)

* **Function:** set gripper switch state
* **Parameter:**
  + flag (int): 0 - open, 1 - close
  + speed (int): 0 ~ 100
* **Return Value:** None

**7.3** get\_gripper\_value()

* **Function:** get gripper value
* **Return Value:** gripper value

**7.4** set\_gripper\_ini()

* **Function:** set the current position to zero, set current position value is 2048
* **Return Value:** None

**7.5** set\_gripper\_value(value, speed)

* **Function:** set gripper value
* **Parameters**
  + value (int): 0 ~ 100
  + speed (int): 0 ~ 100
* **Return Value:** None

**7.6** set\_gservo\_round(angle)

* **Function：** Drive the 9g steering gear clockwise for one revolution.
* **Parameters**
  + angle (int) 0 - 255. 0 : stop 255 : Keep turning 1 ~ 254: Based on 30° (1->30°, 2->60°)

**8 Basic IO Control**

**8.1** get\_basic\_input(pin\_no)

* **Function:** get bottom pin
* **Parameters:**
* pin\_no (int) Pin number.
* **Return Value:**
  + 0: in working state
  + 1: not in working state

**8.2** set\_basic\_output(pin\_no, pin\_signal)

* **Function:** set bottom pin
* **Parameters:**
  + pin\_no (int) Pin number
  + pin\_signal (int): 0 / 1

**9 Socket Control**

The robotic arm needs to open the server, the server file is [here](https://github.com/elephantrobotics/pymycobot/blob/main/demo/Server.py).

# for mycobot,mecharm

from pymycobot import MyCobotSocket

mc = MyCobotSocket("192.168.1.10", 9000)

print(mc.get\_angles())

**10 TCPIP**

**10.1** set\_ssid\_pwd(account, password)

* **Function:** change connected wifi (apply to m5 or seeed)
* **Parameters**
  + account (str) : new wifi account.
  + password (str): new wifi password.
* **Return Value:** None

**10.2** get\_ssid\_pwd()

* **Function:** get connected wifi account and password (apply to M5 or seed)
* **Return Value:** present WIFI account and password

**10.3** set\_server\_port(port)

* **Function:** change the connection port of the server
* **Parameters**
  + port (int): the new connection port of the server
* **Return Value:** None

**11 utils (Module)**

Import utils before using it:

from pymycobot import utils

**11.1** utils.get\_port\_list()

* **Function:** get the all serial port list
* **Return Value:** serial port list (list)

**11.2** utils.detect\_port\_of\_basic()

* **Description**: Returns the serial port string of the first detected M5 Basic. If it is not found, it returns None.
* **Return**: detected port (str) or `None

##Example:

from pymycobot import MyCobot, utils

port = utils.detect\_port\_of\_basic()

if port is None:

raise Exception('Detection failed.')

mycobot = MyCobot(port, 115200)

**12 Raspberry PI—GPIO**

Import pymycobot first:

from pymycobot import MyCobot

**12.1** gpio\_init()

* **Function:** init GPIO module, and set BCM mode
* **Return Value:** None

**12.2** set\_gpio\_mode()

* **Function:** set pin coding method.
* **Parameters**
  + mode (str) "BCM" or "BOARD"
* **Return Value:** None

**12.3** set\_gpio\_output(pin\_no, state)

* **Function:** set GPIO port output value.
* **Parameters:**
  + pin (int): pin number
  + v (int): 0 / 1
* **Return Value:** None

**12.4** get\_gpio\_in(pin\_no)

* **Function:** get pin level status.
* **Parameters:**
  + pin\_no (int) pin id
* **Return Value:**
  + 0:low
  + 1: high