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()`.

myCobot / myPalletizer / mechArm / myArm

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

1.6 `read_next_error()`

- **Fuction**: Robot Error Detection.
- **Return Value**
  - `0`: No abnormality
  - `1`: Communication disconnected
  - `2`: Unstable communication
  - `3` : Servo abnormality

1.7 `set_fresh_mode(mode)`

- **Fuction**: Set command refresh mode
- Parameters
  - **mode** – int 1 - Always execute the latest command first.
    0 - Execute instructions sequentially in the form of a queue.
- **Return Value** None

1.8 `get_fresh_mode()`

- **Fuction**: get command refresh mode
- **Return Value**
  - 1 - Always execute the latest command first.
  - 0 - Execute instructions sequentially in the form of a queue.

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

2.15 `set_joint_min(id,angle)`

- **Function:** Sets the minimum angle for the specified joint.
- **Parameters**:
    - `id`

      : (

      `int`

)
        - for mycobot / mecharm: int 1-6.
        - for mypalletizer: int 1-4.
        - for myArm: int 1 - 7.
    - `angle`: 0 - 180.
- **Return Value:** None

2.16 `set_joint_max(id,angle)`

- **Function:** Sets the maximum angle of the specified joint.
- **Parameter:**
    - `id`

      : (

      `int`

) joint id
        - for mycobot / mecharm: int 1-6.
        - for mypalletizer: int 1-4.
        - for myArm: int 1 - 7.
    - `angle`: 0 - 180
- **Return Value:** None

2.17 `get_basic_version()`

- **Function:** Get basic firmware version.
- Return Value:
    - `version` (`float`)

2.18 `set_communicate_mode(mode)`

- **Function:** Set basic communication mode.
- Parameter:
  - `mode` (`int`) 0 - Turn off transparent transmission. 1 - Open transparent transmission
- **Return Value:** None

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

3.18 `set_encoders_drag(encoders,speeds)`

- **Function:** Send all encoders and speeds
- Parameters:
  - `encoders` (`list`) : encoders list.
  - `speeds`: Obtained by the get_servo_speeds() method
- **Return Value:** None

3.19 `get_solution_angles()`

- **Function:** Get zero space deflection angle value.(`This interface is only applicable to MyArm`)
- **Return Value:** angles

3.20 `set_solution_angles(angle,speed)`

- **Function:** Set zero space deflection angle value.(`This interface is only applicable to MyArm`)
- Parameters:
  - `angle`: Angle of joint 1.
  - `speed`: 1 - 100.
- **Return Value:** None

3.21 `set_transponder_mode(mode)`

- **Function:** Set serial port transmission mode.(`This interface is only applicable to MyArm`)
- Parameters:
  - `mode`

    :
    - `0`: Turn off transparent transmission
    - `1`: Turn on transparent transmission, verify all data
    - `2`: Turn on transparent transmission, only verify communication forwarding mode configuration information

- **Return Value:** None

3.22 `get_transponder_mode()`

- **Function:** Obtain the configuration information of serial transmission mode.(`This interface is only applicable to MyArm`)
- **Return Value:**
  - `0`: Turn off transparent transmission
  - `1`: Turn on transparent transmission, verify all data
  - `2`: Turn on transparent transmission, only verify communication forwarding mode configuration information (default is 0)

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`)

4.12 `jog_absolute(joint_id,angle,spedd)`

- **Function:** Jog absolute angle.
- Parameters

  :

  - ```
    joint_id
    ```

    : (

    ```
    int
    ```

)
    - for mycobot / mecharm: int 1-6.
    - for mypalletizer: int 1-4.
    - for myArm: int 1 - 7.
    ```

- `direction`:
- `speed`: 0 ~ 100

- **Returns**: None

4.13 `jog_increment(joint_id,angle,speed)`

- **Function:** Setp mode.
- Parameters

  :

  - `coord_id`

    : (

    `int`

)

    - for mycobot / mecharm: int 1-6.
    - for mypalletizer: int 1-4.
    - for myArm: int 1 - 7.
  - `direction`:
  - `speed`: 0 ~ 100
- **Returns**: None

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

5.5 `joint_brake(joint_id)`

- **Function:** Make it stop when the joint is in motion, and the buffer distance is positively related to the existing speed
- **Parameters**: `joint_id`: 1~7
- **Return Value:** None

5.6 `get_servo_speeds()`

- **Function:** Get joint velocity.
- **Return Value:** `list` Speed of each joint.

5.7 `get_servo_currents()`

- **Function:** Get joint current.
- **Return Value:** `list` Current of each joint.

5.8 `get_servo_voltages()`

- **Function:** Get joint voltage.
- **Return Value:** `list` Voltage of each joint.

5.9 `get_servo_temps()`

- **Function:** Get the temperature of each joint.
- **Return Value:** `list` the temperature of each joint.

5.10 `get_servo_status()`

- **Function:** power on designated servo
- **Return Value:** `list` the state of each joint.

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

6.4 `set_pwm_output(channel, frequency, pin_val)`

- **Function:** PWM control.
- Parameters
  :
  - `channel` (`int`): IO number.
  - `frequency` (`int`): clock frequency
  - `pin_val` (`int`): Duty cycle 0 ~ 256; 128 means 50%
- **Return Value**: None

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°)

7.7 `set_gripper_calibration()`

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

7.8 `set_eletric_gripper(status)`

- **Function:** Set Electric Gripper Mode (only for 350).
- Parameters
  - `status` (`int`): 0 - open, 1 - close.
- **Return Value:** None

7.9 `set_gripper_mode(status)`

- **Function:** Set gripper mode.
- Parameters
  - `status` (`int`): 0 - transparent transmission. 1 - Port Mode.
- **Return Value:** None

7.10 `get_gripper_mode()`

- **Function:** Get gripper mode.
- Return Value:
  - `status` (`int`): 0 - transparent transmission. 1 - Port Mode.

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.

```
# 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

12.5 `gpio_output(pin,v)`

- **Function:** Set GPIO port output value.
- Parameters:
  - `pin` (`int`) Pin number.
  - `v` (`int`): 0 / 1
- **Return Value:** None

12.6 `set_gpio_out(pin_no, mode)`

- **Function:** Set the pin as input or output.
- **Parameters:**
  - `pin_no` (`int`) pin id.
  - `mode` (`str`) "in" or "out"
- **Return Value:** None

13 Coordinate Transformation

13.1 `set_tool_reference(coords)`

- **Function**: Set tool coordinate system.
- **Parameters**:
  - `coords`: (`list`) [x, y, z, rx, ry, rz].
- **Return**:None

13.2 `set_world_reference(coords)`

- **Function**: Set world coordinate system.
- **Parameters**:
  - `coords`: (`list`) [x, y, z, rx, ry, rz].
- **Return**:None

13.3 `get_world_reference()`

- **Function**: Get world coordinate system.
- **Return**: `list` [x, y, z, rx, ry, rz].

13.4 `set_reference_frame(rftype)`

- **Function**: Set base coordinate system.
- **Parameters**:
  - `rftype`: 0 - base 1 - tool.
- **Return**:None

13.5 `get_reference_frame()`

- **Function**: Get base coordinate system.
- **Return**: 0 - base 1 - tool.

13.6 `set_movement_type(move_type)`

- **Function**: Set movement type.
- **Parameters**:
  - `move_type`: 1 - movel, 0 - moveJ.
- **Return**:None

13.7 `get_movement_type()`

- **Function**: Get movement type.
- **Return**: 1 - movel, 0 - moveJ.

13.8 `set_end_type(end)`

- **Function**: Set end coordinate system.
- Parameters

  :

  - `end`: 0 - flange, 1 - tool.
- **Return**:None

13.9 `get_end_type()`

- **Function**: Get end coordinate system.
- **Return**: 0 - flange, 1 - tool.

14 Speed Planning

14.1 `get_plan_speed()`

- **Function**: Get planning speed.
- **Return**: [`movel planning speed`, `movej planning speed`].

14.1 `get_plan_acceleration()`

- **Function**: Get planning acceleration.
- **Return**: [`movel planning acceleration`, `movej planning acceleration`].

14.1 `set_plan_speed(speed, is_linear)`

- **Function**: Set planning speed.
- **Parameters**
  - `speed` (`int`) 0 - 100.
  - `is_linear` (`int`): 0 / 1 (0 ->joint, 1 -> line)
- **Return**: None

14.1 `set_plan_acceleration(acceleration, is_linear)`

- **Function**: Set planning acceleration.
- **Parameters**
  - `acceleration` (`int`) 0 - 100.
  - `is_linear` (`int`): 0 / 1 (0 ->joint, 1 -> line)
- **Return**:None

myBuddy

1 Overall Status

1.1 `power_off(id=0)`

- **Fuction:** Close communication with Atom.
- **Parameters:**

  id – 0/1/2/3 (ALL/L/R/W)

1.2 `power_on(id=0)`

- **Function:** Open comminication with Atom.
- **Parameter**

id – 0/1/2/3 (ALL/L/R/W)

1.3 `read_next_error(id=0)`

- **Function:** Robot Error Detection
- **Parameter**

  id – 0/1/2/3 (ALL/L/R/W)

1.4 `release_all_servos(id=0)`

- **Function:** Robot turns off torque output
- **Parameter**

  id – 0/1/2/3 (ALL/L/R/W) **1.5** `is_power_on(id=0)`

- **Function:** Adjust robot arm status
- **Parameter**

  id – 0/1/2/3 (ALL/L/R/W)

- **Return Value:**

  1 - power on 0 - power off -1 - error data

1.6 `is_controller_connected(id=0)`

- **Function:** Wether connected with Atom.
- **Parameter**

  id – 0/1/2/3 (ALL/L/R/W)

- **Return Value:**

  0 - Not connected 1 - Connected

1.7 `set_free_mode(id, value)`

- **Function:** set free mode
- **Parameter**
  - **id** – 0/1/2/3 (ALL/L/R/W)
  - **value** - 0 - close 1 - open

1.8 `set_fresh_mode(id, mode)`

- **Function:** set command refresh mode
- **Parameter**
  - **id** – 1/2 (L/R) .
  - **mode** - int 0 - Always execute the latest command first. 1 - Execute instructions sequentially in the form of a queue.

1.9 `release_servo (id, servo_id)`

- **Function:** Power off designated servo
- **Parameter**
  - **id** – 1/2/3 (L/R/W)
  - **servo_id** - 1 - 6.

1.10 `is_free_mode(id)`

- **Function：** check if it is free mode
- **Parameter**

  id – 0/1/2/3 (ALL/L/R/W)
- **Return Value：**

  0 - No 1 - Yes

2 Operating Mode

2.1 `stop (id)`

- **Function：** Stop moving
- **Parameters：**

  id – 0/1/2/3 (ALL/L/R/W).

2.2 `resume (id)`

- **Function：** Recovery movement
- **Parameters：**

  id – 0/1/2/3 (ALL/L/R/W).

2.3 `is_paused(id)`

- **Function：** Judge whether the manipulator pauses or not.
- **Parameters：**

  id – 0/1/2/3 (ALL/L/R/W).
- **Return Value：**

  1 - paused 0 - not paused -1 - error

2.4 `get_speed(id)`

- **Function：** get speed
- **Parameters：**

  id – 1/2/3（L/R/W）.
- **Return Value：**

  speed
- **Return Value：类型**

  int

2.5 `set_speed (id, speed)`

- **Function：** set speed value
- **Parameters：**
  - **id** – 1/2/3 (L/R/W)
  - **speed** (*int*) - 0 - 100

2.6 `get_joint_min_angle (id, joint_id)`

- **Function：** Gets the minimum movement angle of the specified joint
- **Parameters：**

    - **id** – 1/2/3 (L/R/W)
    - **joint_id** - (int) 1 - 6
- **Return Value:**

  angle value(float)

2.7 `is_servo_enable (id, servo_id)`

- **Function:** Determine whether all steering gears are connected
- **Parameters:**
    - **id** – 1/2/3 (L/R/W)
    - **servo_id** - (int) 1 ~ 6
- **Return Value:**

  0 - disable 1 - enable -1 - error

2.8 `is_all_servo_enable(id)`

- **Function:** Determine whether the specified steering gear is connected
- **Parameters:**

  id – 1/2/3 (L/R/W)
- **Return Value:**

  0 - disable 1 - enable -1 - error

2.9 `set_joint_min (id, joint_id, angle)`

- **Function:** Set the joint minimum angle
- **Parameters:**
    - **id** – 1/2/3 (L/R/W)
    - **joint_id** - int 1-6.
    - **angle** - 0 ~ 180

2.10 `get_robot_version(id)`

- **Function:** get robot version
- **Parameters:**

  id – 0/1/2/3 (ALL/L/R/W)

2.11 `get_system_version(id)`

- **Function:** get system version
- **Parameters:**

  id – 0/1/2/3 (ALL/L/R/W)

2.12 `get_joint_max_angle (id, joint_id`

- **Function:** Gets the maximum movement angle of the specified joint
- **Parameters**
    - **id** – 1/2/3 (L/R/W)
    - **joint_id** - (int) 1 - 6
- **Return Value:**

  angle(float)

2.13 `set_robot_id(id, new_id)`

- **Function:** set the robot's id
- **Parameters**
  - **id** – 0/1/2/3 (ALL/L/R/W)
  - **new_id** - 1 - 253

2.14 `joint_brake(id, joint_id)`

- **Function:** Make it stop when the joint is in motion, and the buffer distance is positively related to the existing speed
- **Parameters**
  - **id** – 1/2/3 (L/R/W)
  - **joint_id** - 1 - 6

2.15 `get_robot_id(id)`

- **Function:** Detect this robot id
- **Parameters:**

  id – 0/1/2/3 (ALL/L/R/W)

3 MDI Mode

3.1 `get_angles(id)`

- **Function:** Get the degree of all joints.
- **Parameters**

  id – 1/2 (L/R)
- **Return Value:**

  A float list of all degree.
- **Return Value:**

  list

3.2 `send_angle(id, joint, angle, speed)`

- **Function:** Send one degree of joint to robot arm.
- **Parameters**
  - **id** – 1/2/3 (L/R/W)
  - **joint** – 1 ~ 6
  - **angle** - int
  - **speed** – 1 ~ 100
- **Return Value:**
  - None

3.3 `send_angles (id, degrees, speed)`

- **Function:** Send all angles to the robotic arm
- **Parameters**
  - **id** – 1/2 (L/R) .
  - **degrees** - [angle_list] len 6

- **speed** - 1 - 100

3.4 `set_joint_max (id, joint_id, angle)`

- **Function：** set the joint maximum angle
- **Parameters**
  - **id** – 1/2/3 (L/R/W)
  - **joint_id** - int 1-6.
  - **角度** - 0 ~ 180

3.5 `send_coord(id, coord, data, speed)`

- **Function：** Send a single coordinate to the robotic arm
- **Parameters**
  - **id** – 1/2/3 (L/R/W).
  - **coord** – 1 ~ 6 (x/y/z/rx/ry/rz)
  - **data** - int
  - **speed** - 0 ~ 100

3.6 `send_coords(id, coords, speed, mode)`

- **Function：** Send all coordinates to robotic arm
- **Parameters**
  - **id** – 1/2（L/R）.
  - **coords** – a list of coords（List[float]），length 6，[x(mm), y, z, rx(angle), ry, rz]
  - **speed** - (int) 1 ~ 100
  - **mode** - (int) 0 - moveJ, 1 - moveL, 2 - moveC

3.7 `get_coord (id, joint_id)`

- **Function：** Get the coordinates of the robotic arm
- **Parameters**
  - **id** (*int*) – 1/2/3 (L/R/W).
  - **joint_id** (*int*) – 1 - 7（7 is gripper）

3.8 `get_encoder(id,joint_id)`

- **Function：** Obtain the specified joint potential value.
- **Parameters**
  - **id** - 1/2/3（L/R/W）.
  - **joint_id** - (int) 1 ~ 6
- **Return Value：**

  0 ~ 4096

3.9 `get_encoders(id)`

- **Function：** Get the six joints of the manipulator
- **Parameters**

  id – 1/2（L/R）.
- **Return Value：**

  list

3.10 `get_radians(id)`

- **Function:** Get the radians of all joints
- **Parameters**

  id – 1/2（L/R）
- **Return Value:**

  A list of float radians [radian1, ...]
- **Return Value:** list

3.11 `send_radians(id, radians, speed)`

- **Function:** Send the radians of all joints to robot arm
- **Parameters**
  - **id** – 1/2（L/R）.
  - **radians** – a list of radian values（List[float]），length 6
  - **speed** - (int)0 ~ 100

3.12 `is_in_position(id, data, mode)`

- **Function:** Detect whether in the position.
- **Parameters**
  - **id** – 0/1/2/3 (ALL/L/R/W).
  - **data** – A data list, angles or coords. If id is 1/2. data length is 6. If id is 0. data len 13. if id is 3. data len 1
- **mode** - 1 - coords, 0 - angles
- **Return Value:**

  1 - True 0 - False -1 - error

3.13 `is_moving(id)`

- **Function:** Detect if the robot is moving
- **Parameters**

  id – 0/1/2/3 (ALL/L/R/W).
- **Return Value:**

  0 - not moving 1 - is moving -1 - error data

3.14 `set_color(id, r=0, g=0, b=0)`

- **Function:** Set the light color on the top of the robot arm.
- **Parameters**
  - **id** - 1/2（L/R）
  - **r** (*int*) – 0 ~ 255
  - **g** (*int*) – 0 ~ 255
  - **b** (*int*) – 0 ~ 255

3.15 `set_encoder (id, joint_id, encoder, speed)`

- **Function:** Set a single joint rotation to the specified potential value.
- **Parameters**

- **id** – 1/2/3 (L/R/W).
- **joint_id** - 1 - 6.
- **encoder** – The value of the set encoder.

3.16 `set_encoders (id, encoder, speed)`

- **Function：** Set the six joints of the manipulator to execute synchronously to the specified position.
- **Parameters**
  - **id** – 1/2（L/R）.
  - **encoders** - A encoder list, length 6.
  - **speed** – speed 1 ~ 100

3.17 `get_angle (id, joint_id)`

- **Function：** Get the angle of a single joint
- **Parameters**
  - **id** (*int*) – 1/2/3 (L/R/W).
  - **joint_id** (*int*) – 1 - 7（7 is gripper）

3.18 `set_servo_calibration (id, servo_no`

- **Function：** The current position of the calibration joint actuator is the angle zero point, and the corresponding potential value is 2048.
- **Parameters：**
  - **id** – 1/2/3 (L/R/W)
  - **servo_no** – Serial number of articulated steering gear, 1 - 6.

3.19 `set_joint_current (id, joint_id, current)`

- **Function：** Set Collision Current
- **Parameters:**
  - **id** - 0/1/2 (ALL/L/R)
  - **joint_id** - 1 - 6
  - **current** – current value

3.19 `get_coords(id)`

- **Function：** Read a single coordinate parameter
- **Parameters**

  id – 1/2（L/R）

4 JOG Mode

4.1 `jog_absolute (id, joint_id, angle, speed)`

- **Function：** Absolute joint control
- **Parameters:**
  - **id** – 1/2/3 (L/R/W).
  - **joint_id** - int 1-6.
  - **angle** - int

- **speed** - int (0 - 100)

4.2 `jog_angle (id, joint_id, direction, speed)`

- **Function：** Jog control joint
- **Parameters:**
  - **id** – 1/2/3 (L/R/W).
  - **joint_id** - int 1-6.
  - **direction** - 0 - decrease，1 - increase
  - **speed** - int (0 - 100)

4.3 `jog_coord(id, coord_id, direction, speed)`

- **Function：** Jog control coordinate
- **Parameters:**
  - **id** – 1/2/3 (L/R/W).
  - **coord_id** – int 1-6 (x/y/z/rx/ry/rz).
  - **direction** - 0 - decrease，1 - increase
  - **speed** - int (0 - 100)

4.4 `jog_inc_coord (axis, increment, speed)`

- **Function：** Double-arm coordinated coordinate stepping
- **Parameters:**
  - **axis** – 1 - 6 (x/y/z/rx/ry/rz)
  - **increment** -
  - **speed** - 1 - 100

4.5 `jog_increment (id, joint_id, increment, speed)`

- **Function：** step mode
- **Parameters:**
  - **id** – 1/2/3 (L/R/W).
  - **joint_id** - int 1-6.
  - **increment** -
  - **speed** - int (1 - 100)

4.6 `jog_stop(id)`

- **Function：** JOG stop
- **Parameters:**

  id – 1/2/3（L/R/W)

5 Servo COntrol

5.1 `focus_servo (id, servo_id)`

- **Function：** Power on designated servo
- **Parameters：**
  - **id** – 1/2/3 (L/R/W)
  - **servo_id** - 1 - 6

5.2 `get_servo_currents(id)`

- **Function:** Get joint current
- **Parameters:**

  id – 1/2/3 (L/R/W)
- **Return Value:**

  value mA

5.3 `get_servo_data (id, servo_no, data_id)`

- **Function:** Read the data parameter of the specified address of the steering gear.
- **Parameters:**
  - **id** – 1/2/3 (L/R/W)
  - **servo_no** – Serial number of articulated steering gear, 1 - 6.
  - **data_id** – Data address.
- **Return Value:**

  values (0 - 4096) 0 - disable 1 - enable -1 - error

5.4 `get_servo_status(id)`

- **Function:** Get joint status
- **Parameters:**

  id – 1/2/3 (L/R/W)
- **Return Value:**

  [voltage, sensor, temperature, current, angle, overload], a value of 0 means no error

5.5 `get_servo_temps(id)`

- **Function:** Get joint temperature
- **Parameters:**

  id – 1/2/3 (L/R/W)

5.6 `get_servo_voltages(id)`

- **Function:** Get joint voltages
- **Parameters:**

  id – 1/2/3 (L/R/W)
- **Return Value:**

  volts < 24 V

5.7 `set_servo_data (id, servo_no, data_id, value)`

- **Function:** Set the data parameters of the specified address of the steering gear
- **Parameters**
  - **id** – 1/2/3 (L/R/W)
  - **servo_no** – Serial number of articulated steering gear, 1 - 6.
  - **data_id** – Data address.
  - **value** – 0 - 4096

6 Atom IO Control

6.1 `set_pin_mode(id, pin_no, pin_mode)`

- **Function:** Set the state mode of the specified pin in atom.
- **Parameters:**
  - **id** - 1/2 (L/R)
  - **pin_no** (*int*) – pin number (1 - 5).
  - **pin_mode** (*int*) – 0 - input, 1 - output

6.2 `set_digital_output(id, pin_no, pin_signal)`

- **Function:** Set atom IO output level
- **Parameters:**
  - **id** - 1/2 (L/R)
  - **pin_no** (*int*) - 1 - 5
  - **pin_signal** (*int*) – 0 / 1

6.3 `get_digital_input(id, pin_no)`

- **Function:** signal
- **Parameters:**
  - **id** - 1/2 (L/R)
  - **pin_no** (*int*) - 1 - 5

6.4 `set_pwm_output(id, channel, frequency, **6.1pin_val)`

- **Function:** PWM control
- **Parameters:**
  - **id** - 1/2 (L/R)
  - **channel** (*int*) – IO number (1 - 5).
  - **frequency** (*int*) – clock frequency (0/1: 0 - 1Mhz 1 - 10Mhz)
  - **pin_val** (*int*) – Duty cycle 0 ~ 100: 0 ~ 100%

7 Gripper Control

7.1 `get_gripper_value(id)`

- **Function:** Get the value of gripper.
- **Parameters**

  id – 1/2 (L/R)
- **Return Value:**

  gripper value (int)

7.2 `is_gripper_moving(id)`

- **Function:** Judge whether the gripper is moving or not
- **Parameters**

  id – 1/2 (L/R)
- **Return Value:**

0 - not moving 1 - is moving -1 - error data

7.3 `set_gripper_calibration(id)`

- **Function:** Set the current position to zero, set current position value is 2048.
- **Parameters**

  id – 1/2 (L/R)

7.4 `set_gripper_state(id, flag)`

- **Function:** Set gripper switch state
- **Parameters**
  - **id** - 1/2 (L/R)
  - **flag** (*int*) - 0 - close, 1 - open

7.5 `set_gripper_value(id, value, speed)`

- **Function:** Set gripper value
- **Parameters**
  - **id** – 1/2 (L/R)
  - **value** (*int*) – 0 ~ 100
  - **speed** (*int*) – 0 ~ 100

8 Socket Control

```
from pymycobot import MyBuddySocket

mst = MyBuddySocket("192.168.0.1", 9000)
mst.connect("/dev/ttyACM0", "115200")

print(mst.get_angles(1))
```

9 Raspberry PI-GPIO

9.1 `set_gpio_input(pin)`

- **Function:** Set GPIO input value.
- **Parameters**

  pin - (int)pin number.

9.2 `set_gpio_mode(pin_no, mode)`

- **Function:** Init GPIO module, and set BCM mode.
- **Parameters**
  - **pin_no** – (int)pin number.
  - **mode** – 0 - input 1 - output

9.3 `set_gpio_output(pin, v)`

- **Function:** Set GPIO output value.
- **Parameters**
  - **pin** - (int)pin number.

- **v** - (int) 0 / 1

9.4 `set_gpio_pwm (pin, baud, dc)`

- **Function:** Set GPIO PWM value.
- **Parameters**
  - **pin** – (int)pin number.
  - **baud** – (int) 10 - 1000000
  - **dc** – (int) 0 - 100

10 Coordinate Transformation

10.1 `set_tool_reference (id, coords)`

- **Function:** Set tool coordinate system
- **Parameters**
  - **id** - 0/1/2 (ALL/L/R)
  - **coords** – a list of coords value(List[float]), length 6. [x(mm), y, z, rx(angle), ry, rz]

10.2 `set_world_reference (id, coords)`

- **Function:** Set the world coordinate system
- **Parameters**
  - **id** - 0/1/2 (ALL/L/R)
  - **coords** – a list of coords value(List[float]), length 6 [x(mm), y, z, rx(angle), ry, rz]

10.3 `get_reference_frame(id)`

- **Function:** Get the base coordinate system
- **Parameters**

  id – 0/1/2 (ALL/L/R)
- **Return Value:**

  0 - base 1 - tool

10.4 `get_tool_reference(id)`

- **Function:** Get tool coordinate system
- **Parameters**

  id – 0/1/2 (ALL/L/R)

10.5 `get_world_reference(id)`

- **Function:** Get the world coordinate system
- **Parameters**

  id – 0/1/2 (ALL/L/R)

10.6 `set_reference_frame(id, rftype)`

- **Function:** Set the base coordinate system
- **Parameters**
  - **id** - 0/1/2 (ALL/L/R)

- **rftype** - 0 - base 1 - tool.

10.7 `set_movement_type(id, move_type)`

- **Function：** Set movement type
- **Parameters**
  - **id** - 0/1/2 (ALL/L/R)
  - **move_type** - 1 - movel，0 - moveJ

10.8 `get_movement_type(id)`

- **Function：** Get movement type
- **Parameters**

  id – 0/1/2 (ALL/L/R)
- **Return Value：**

  1 - movel 0 - moveJ

10.9 `set_end_type(id, end)`

- **Function：** Set end coordinate system
- **Parameters**
  - **id** - 0/1/2 (ALL/L/R)
  - **end** – 0 - flange, 1 - tool

10.10 `get_end_type(id)`

- **Function：** Get end coordinate system
- **Parameters**

  id – 0/1/2 (ALL/L/R)
- **Return Value：**

  0 - flange 1 - tool

10.11 `write_base_coords(id, coords, speed)`

- **Function：** Base coordinate move
- **Parameters**
  - **id** - 1/2（L/R)
  - **coords** – coords: a list of coords value(List[float]), length 6, [x(mm), y, z, rx(angle), ry, rz]
  - **speed** - 1 - 100

10.12 `write_base_coord (id, axis, coord, speed)`

- **Function：** Base single coordinate movement
- **Parameters**
  - **id** - 1/2（L/R)
  - **axis** – 1 - 6 (x/y/z/rx/ry/rz)
  - **coord** - Coordinate value
  - **speed** - 1 - 100

10.13 `base_to_single_coords(base_coords, arm)`

- **Function：** Convert base coordinates to coordinates

- **Parameters:**
  - **coords** – a list of base coords value len 6
  - **arm** – 0 - left. 1 - right
- **Return Value: :**

  coords

10.14 `get_base_coord(id)`

- **Function:** Get the base coordinates of the single arm
- **Parameters:**

  id – 1/2 (L/R)

10.15 `get_base_coords(\*args: int)`

- **Function:** Convert coordinates to base coordinates. Pass in parameters or no parameters
- **Parameters:**
  - **coords** – a list of coords value(List[float]), length 6 [x(mm), y, z, rx(angle), ry, rz]
  - **arm** - 0 - L. 1 -
- **Return Value: :**

  Base coords

11 Speed Planning

11.1 `get_plan_acceleration(id=0)`

- **Function:** Get planning acceleration
- **Parameters:**

  id – 0/1/2/3 (ALL/L/R/W)
- **Return Value:**

  [movel planning acceleration, movej planning acceleration].

11.2 `get_plan_speed(id=0)`

- **Function:** Get planning speed
- **Parameters:**

  id – 0/1/2/3 (ALL/L/R/W)
- **Return Value:**

  [movel planning speed, movej planning speed].

11.3 `set_plan_acceleration(id, acceleration)`

- **Function:** Set planning acceleration
- **Parameters:**
  - **id** – 0/1/2/3 (ALL/L/R/W)
  - **Acceleration**(*int*) - (0 ~ 100).

11.4 `set_plan_speed (id, speed)`

- **Function:** Set planning speed
- **Parameters:**

- **id** – 0/1/2/3 (ALL/L/R/W)
- **speed** (*int*) - (0 ~ 100).

11.5 `set_acceleration(id, acc)`

- **Function：** Set acceleration during all moves
- **Parameters:**
  - **id** – 1/2/3 (L/R/W)
  - **acc** - 1 - 100

11.6 `get_acceleration(id)`

- **Function：** Read acceleration during all moves
- **Parameters：**

  id – 1/2/3 (L/R/W)

12 Collision Detection

12.1 `get_joint_current (id, joint_id)`

- **Function:** Get Collision Current
- **Parameters:**
  - **id** - 0/1/2 (ALL/L/R)
  - **joint_id** - 1 - 6

12.2 `collision_switch (state)`

- **Function:** Collision Detection Switch
- **Parameters:**

  state (*int*) – 0 - close 1 - open (Off by default)

12.3 `collision (left_angles, right_angles)`

- **Function:** Collision detection main program
- **Parameters:**
  - **left_angles** – left arm angle len 6.
  - **right_angles** – right arm angle len 6.
- **Returns**

  int

12.4 `is_collision_on()`

- **Parameters：** Get collision detection status
- **Return Value：：**

  0 - disbale 1 - enable

More interface description