class XArmAPI

XArmAPI(const std::string &port="",
bool is_radian=DEFAULT_IS_RADIAN,
bool do_not_open=false,
bool check_tcp_limit=true,
bool check_joint_limit=true,
bool check_cmdnum_limit=true,
bool check_robot_sn=false,
bool check_is_ready=false,
bool check_is_pause=true,
int max_callback_thread_count=10,
int max_cmdnum = 512,
int init_axis = 7,
bool debug = false,
std::string report_type = "rich",
bool baud_checkset = true)

> @param port: ip-address(such as "192.168.1.185")
> Note: this parameter is required if parameter do_not_open is false
> @param is_radian: set the default unit is radians or not, default is false
> @param do_not_open: do not open, default is false, if true, you need to manually call the connect interface.
> @param check_tcp_limit: reversed, whether checking tcp limit, default is true
> @param check_joint_limit: reversed, whether checking joint limit, default is true
> @param check_cmdnum_limit: whether checking command num limit, default is true
> @param check_robot_sn: whether checking robot sn, default is false
> @param check_is_ready: check robot is ready to move or not, default is true
> Note: only available if firmware_version < 1.5.20
> @param check_is_pause: check robot is pause or not, default is true
> @param max_callback_thread_count: max callback thread count, default is -1
> Note: greater than 0 means the maximum number of threads that can be used to process callbacks
> Note: equal to 0 means no thread is used to process the callback
> Note: less than 0 means no limit on the number of threads used for callback
> @param max_cmdnum: max cmdnum, default is 512
> Note: only available in the param `check_cmdnum_limit` is true
> @param init_axis: init axis variable
> @param debug: reversed
> @param report_type: report type
> @param baud_checkset: auto check set the baud when use the gripper/bio/robotiq/lineartrack api or not

Property

- **int state**

  > xArm state
  >
  > @return:
  > 1: in motion
  > 2: sleeping
  > 3: suspended
  > 4: stopping

- **int mode**

  > xArm mode，only available in socket way and enable_report is true
  >
  > @return:
  > 0: position control mode
  > 1: servo motion mode
  > 2: joint teaching mode
  > 3: cartesian teaching mode (invalid)
  > 4: joint velocity control mode
  > 5: cartesian velocity control mode

- **int cmd_num**

  > Number of command caches in the controller

- **fp32 joints_torque[7]**

  > Joints torque, only available in socket way
  >
  > @return: fp32[7]{servo-1, ..., servo-7}

- **bool motor_brake_states[8]**

  > Motor brake state list, only available in socket way
  > Note:
  > For a robot with a number of axes n, only the first n states are valid, and the latter are reserved.
  >
  > @return: bool[8]{servo-1, ..., servo-7, reversed}

- **bool motor_enable_states[8]**

  > Motor enable state list, only available in socket way
  > Note:
  > For a robot with a number of axes n, only the first n states are valid, and the latter are reserved.
  >
  > @return: bool[8]{servo-1, ..., servo-7, reversed}

- **int error_code**

  > Controller error code. See the Controller Error Code Documentation for details.

- **int warn_code**

  > Controller warn code. See the Controller Warn Code Documentation for details.

- **fp32 tcp_load[4]**

  > xArm tcp load, only available in socket way
  >
  > @return: fp32[4]{weight, x, y, z}

- **int collision_sensitivity**

  > The sensitivity value of collision, only available in socket way

- **int teach_sensitivity**

  > The sensitivity value of drag and teach, only available in socket way

- **int device_type**

  > Device type, only available in socket way

- **int axis**

  > Axis number, only available in socket way

- **unsigned char version[30]**

  > xArm firmware version

- **unsigned char sn[40]**

  > xArm sn

- **int version_number[3]**

  > Frimware version number

- **fp32 tcp_jerk**

  > tcp jerk

- **fp32 joint_jerk**

  > joint jerk

- **fp32 rot_jerk**

  > rot jerk

- **fp32 max_rot_acc**

  > max rot acc

- **fp32 tcp_speed_limit[2]**

  > Joint acceleration limit, only available in socket way

> @return: fp32[2]{min, max}

- **fp32 tcp_acc_limit[2]**

  > Joint acceleration limit, only available in socket way
  >
  > @return: fp32[2]{min, max}

- **fp32 last_used_tcp_speed**

  > The last used cartesion speed, default value of parameter speed of interface set_position/move_circle

- **fp32 last_used_tcp_acc**

  > The last used cartesion acceleration, default value of parameter mvacc of interface set_position/move_circle

- **fp32 angles[7]**

  > Servo angles
  >
  > @return: fp32[7]{servo-1, ..., servo-7}

- **fp32 last_used_angles[7]**

  > The last used servo angles, default value of parameter angle of interface set_servo_angle
  >
  > @return: fp32[7]{servo-1, ..., servo-7}

- **fp32 joint_speed_limit[2]**

  > Joint speed limit, only available in socket way
  >
  > @return: fp32[2]{min, max}

- **fp32 joint_acc_limit[2]**

  > Joint acceleration limit, only available in socket way
  >
  > @return: fp32[2]{min, max}

- **fp32 last_used_joint_speed**

  > The last used joint speed, default value of parameter speed of interface set_servo_angle

- **fp32 last_used_joint_acc**

  > The last used joint acceleration, default value of parameter mvacc of interface set_servo_angle

- **fp32 position[6]**

  > Cartesion position
  >
  > @return: fp32[6]{x, y, z, roll, pitch, yaw}

- **fp32 last_used_position[6]**

> The last used cartesion position, default value of parameter x/y/z/roll/pitch/yaw of interface set_position
>
> @return: fp32[6]{x, y, z, roll, pitch, yaw}

- **fp32 tcp_offset[6]**

  > Cartesion position offset, only available in socket way and enable_report is true
  >
  > @return: fp32[6]{x, y, z, roll, pitch, yaw}

- **fp32 gravity_direction[3]**

  > gravity direction, only available in socket way
  >
  > @return: fp32[3]{x_direction, y_direction, z_direction}

- **fp32 realtime_tcp_speed**

  > The real time speed of tcp motion, only available if version > 1.2.11

- **fp32 realtime_joint_speeds[7]**

  > The real time speed of joint motion, only available if version > 1.2.11

- **fp32 world_offset[6]**

  > Base coordinate offset, only available if version > 1.2.11
  >
  > Note:
  > 1. If self.default_is_radian is true, the returned value(roll_offset/pitch_offset/yaw_offset) is in radians
  >
  > @return: fp32[6]{x_offset(mm), y_offset(mm), z_offset(mm), roll_offset(° or rad), pitch_offset(° or rad), yaw_offset(° or rad)}

- **int count**

  > Counter value

- **fp32 temperatures[7]**

  > Motor temperature, only available if version > 1.2.11
  >
  > @return: fp32[7]{motor-1-temperature, ..., motor-7-temperature}

- **unsigned char gpio_reset_config[2]**

  > The gpio reset enable config, only available if version > 1.5.0
  >
  > @return: unsigned char[2]{cgpio_reset_enable, tgpio_reset_enable}

- **bool default_is_radian**

  > The default unit is radians or not

- **fp32 voltages[7]**

  > Servos voltage

> @return: fp32[7]{servo-1-voltage, ..., servo-7-voltage}

- **fp32 currents[7]**

  > Servos electric current
  >
  > @return: fp32[7]{servo-1-current, ..., servo-7-current}

- **int is_collision_detection**

  > self collision detection or not

- **int collision_tool_type**

  > self collision tool type

- **fp32 collision_model_params[6]**

  > self collision model params

- **int iden_progress**

  > the progress of identifition

- **fp32 ft_ext_force[6]**

  > the ext force data of ft sensor

- **fp32 ft_raw_force[6]**

  > the raw force data of ft sensor

- **unsigned char only_check_result**

  > the result of only check motion

Method

- **bool has_err_warn(void)**

  > xArm has error/warn or not, only available in socket way

- **bool has_error(void)**

  > xArm has error or not, only available in socket way

- **bool has_warn(void)**

  > xArm has warn or not, only available in socket way

- **bool is_connected(void)**

  > xArm is connected or not

- **bool is_reported(void)**

> xArm is reported or not, only available in socket way

- **int connect(const std::string &port="")**

> Connect to xArm
>
> @param port: port name or the ip address
> @return:
> 0: success
> -1: port is empty
> -2: tcp control connect failed
> -3: tcp report connect failed

- **void disconnect(void)**

> Disconnect to xArm

- **int get_version(unsigned char version[40])**

> Get the xArm version
>
> @param version:
> @return: see the API Code Documentation for details.

- **int get_robot_sn(unsigned char robot_sn[40])**

> Get the xArm sn
>
> @param robot_sn:
> @return: see the API Code Documentation for details.

- **int get_state(int *state)**

> Get the xArm state
>
> @param: the state of xArm
> 1: in motion
> 2: sleeping
> 3: suspended
> 4: stopping
> @return: see the API Code Documentation for details.

- **int shutdown_system(int value=1)**

> Shutdown the xArm controller system
>
> @param value:
> 1: remote shutdown
> @return: see the API Code Documentation for details.

- **int get_cmdnum(int *cmdnum)**

> Get the cmd count in cache
>
> @return: see the API Code Documentation for details.

- **int get_err_warn_code(int err_warn[2])**

  > Get the controller error and warn code
  >
  > @return: see the API Code Documentation for details.

- **int get_position(fp32 pose[6])**

  > Get the cartesian position
  >
  > @param pose: the position of xArm, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
  > if default_is_radian is true, the value of roll/pitch/yaw should be in radians
  > if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
  > @return: see the API Code Documentation for details.

- **int get_servo_angle(fp32 angles[7])**

  > Get the servo angle
  >
  > @param angles: the angles of the servos, like [servo-1, ..., servo-7]
  > if default_is_radian is true, the value of servo-1/.../servo-7 should be in radians
  > if default_is_radian is false, The value of servo-1/.../servo-7 should be in degrees
  > @return: see the API Code Documentation for details.

- **int motion_enable(bool enable, int servo_id=8)**

  > Motion enable
  >
  > @param enable: enable or not
  > @param servo_id: servo id, 1-8, 8(enable/disable all servo)
  > @return: see the API Code Documentation for details.

- **int set_state(int state)**

  > Set the xArm state
  >
  > @param state: state
  > 0: sport state
  > 3: pause state
  > 4: stop state
  > @return: see the API Code Documentation for details.

- **int set_mode(int mode, int detection_param = 0)**

  > Set the xArm mode
  >
  > @param mode: mode
  > 0: position control mode
  > 1: servo motion mode
  > 2: joint teaching mode
  > 3: cartesian teaching mode (invalid)
  > 4: joint velocity control mode
  > 5: cartesian velocity control mode
  > 6: joint online trajectory planning mode

> 7: cartesian online trajectory planning mode
> @param detection_param: teaching detection parameters, default is 0
> 0: turn on motion detection
> 1: trun off motion detection
> Note:
> 1. only available if firmware_version >= 1.10.1
> 2. only available if set_mode(2)
> @return: see the API Code Documentation for details.

- **int set_servo_attach(int servo_id)**

> Attach the servo
>
> @param servo_id: servo id, 1-8, 8(attach all servo)
> @return: see the API Code Documentation for details.

- **int set_servo_detach(int servo_id)**

> Detach the servo, be sure to do protective work before unlocking to avoid injury or damage.
>
> @param servo_id: servo id, 1-8, 8(detach all servo)
> @return: see the API Code Documentation for details.

- **int clean_error(void)**

> Clean the controller error, need to be manually enabled motion and set state after clean error
>
> @return: see the API Code Documentation for details.

- **int clean_warn(void)**

> Clean the controller warn
>
> @return: see the API Code Documentation for details.

- **int set_pause_time(fp32 sltime)**

> Set the arm pause time, xArm will pause sltime second
>
> @param sltime: sleep second
> @return: see the API Code Documentation for details.

- **int set_collision_sensitivity(int sensitivity)**

> Set the sensitivity of collision
>
> @param sensitivity: sensitivity value, 0~5
> @return: see the API Code Documentation for details.

- **int set_teach_sensitivity(int sensitivity)**

> Set the sensitivity of drag and teach
>
> @param sensitivity: sensitivity value, 1~5
> @return: see the API Code Documentation for details.

- **int set_gravity_direction(fp32 gravity_dir[3])**

  > Set the direction of gravity
  >
  > @param gravity_dir: direction of gravity, such as [x(mm), y(mm), z(mm)]
  > @return: see the API Code Documentation for details.

- **int clean_conf(void)**

  > Clean current config and restore system default settings
  > Note:
  > 1. This interface will clear the current settings and restore to the original settings (system default settings)
  >
  > @return: see the API Code Documentation for details.

- **int save_conf(void)**

  > Save config
  > Note:
  > 1. This interface can record the current settings and will not be lost after the restart.
  > 2. The clean_conf interface can restore system default settings
  >
  > @return: see the API Code Documentation for details.

- **int set_position(fp32 pose[6], fp32 radius=-1, fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool wait=false, fp32 timeout=NO_TIMEOUT, bool relative = false, unsigned char motion_type=0)**
- **int set_position(fp32 pose[6], fp32 radius, bool wait, fp32 timeout=NO_TIMEOUT, bool relative=false, unsigned char motion_type=0)**
- **int set_position(fp32 pose[6], bool wait, fp32 timeout=NO_TIMEOUT, bool relative=false, unsigned char motion_type=0)**

  > Set the position
  > MoveLine: Linear motion
  > MoveArcLine: Linear arc motion with interpolation
  >
  > @param pose: position, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
  > if default_is_radian is true, the value of roll/pitch/yaw should be in radians
  > if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
  > @param radius: move radius, if radius is None or radius less than 0, will MoveLine, else MoveArcLine
  > @param speed: move speed (mm/s, rad/s), default is this.last_used_tcp_speed
  > @param mvacc: move acceleration (mm/s^2, rad/s^2), default is this.last_used_tcp_acc
  > @param mvtime: reserved, 0
  > @param wait: whether to wait for the arm to complete, default is false
  > @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true
  > @param relative: relative move or not
  > Note: only available if firmware_version >= 1.8.100
  > @param motion_type: motion planning type, default is 0

> motion_type == 0: default, linear planning
> motion_type == 1: prioritize linear planning, and turn to IK for joint planning when linear planning is not possible
> motion_type == 2: direct transfer to IK using joint planning
> Note:
> 1. only available if firmware_version >= 1.11.100
> 2. when motion_type is 1 or 2, linear motion cannot be guaranteed
> 3. once IK is transferred to joint planning, the given Cartesian velocity and acceleration are converted into joint velocity and acceleration according to the percentage
> `speed = speed / max_tcp_speed * max_joint_speed`
> `acc = acc / max_tcp_acc * max_joint_acc`
> 4. if there is no suitable IK, a C40 error will be triggered
>
> @return: see the API Code Documentation for details.

- **int set_tool_position(fp32 pose[6], fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool wait=false, fp32 timeout=NO_TIMEOUT, unsigned char motion_type=0)**
- **int set_tool_position(fp32 pose[6], bool wait, fp32 timeout=NO_TIMEOUT, unsigned char motion_type=0)**

> Movement relative to the tool coordinate system
> MoveToolLine: Linear motion
> MoveToolArcLine: Linear arc motion with interpolation
>
> @param pose: the coordinate relative to the current tool coordinate systemion, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
> @param speed: move speed (mm/s, rad/s), default is this.last_used_tcp_speed
> @param mvacc: move acceleration (mm/s^2, rad/s^2), default is this.last_used_tcp_acc
> @param mvtime: reserved, 0
> @param wait: whether to wait for the arm to complete, default is false
> @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true
> @param radius: move radius, if radius less than 0, will MoveToolLine, else MoveToolArcLine
> Note: only available if firmware_version >= 1.11.100
> @param motion_type: motion planning type, default is 0
> motion_type == 0: default, linear planning
> motion_type == 1: prioritize linear planning, and turn to IK for joint planning when linear planning is not possible
> motion_type == 2: direct transfer to IK using joint planning
> Note:
> 1. only available if firmware_version >= 1.11.100
> 2. when motion_type is 1 or 2, linear motion cannot be guaranteed
> 3. once IK is transferred to joint planning, the given Cartesian velocity and acceleration are converted into joint velocity and acceleration according to the percentage
> `speed = speed / max_tcp_speed * max_joint_speed`

> `acc = acc / max_tcp_acc * max_joint_acc`
>
> 4. if there is no suitable IK, a C40 error will be triggered
>
> @return: see the API Code Documentation for details.

- **int set_servo_angle(fp32 angles[7], fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool wait=false, fp32 timeout=NO_TIMEOUT, fp32 radius = -1)**
- **int set_servo_angle(fp32 angles[7], bool wait, fp32 timeout=NO_TIMEOUT, fp32 radius = -1, bool relative = false)**
- **int set_servo_angle(int servo_id, fp32 angle, fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool wait=false, fp32 timeout=NO_TIMEOUT, fp32 radius = -1, bool relative = false)**
- **int set_servo_angle(int servo_id, fp32 angle, bool wait, fp32 timeout=NO_TIMEOUT, fp32 radius = -1, bool relative = false)**

> Set the servo angle
>
> @param angles: angles, like [servo-1, ..., servo-7]
> if default_is_radian is true, the value of servo-1/.../servo-7 should be in radians
> if default_is_radian is false, The value of servo-1/.../servo-7 should be in degrees
> @param servo_id: servo id, 1~7, specify the joint ID to set
> @param angle: servo angle, use with servo_id parameters
> @param speed: move speed (rad/s or °/s), default is this.last_used_joint_speed
> if default_is_radian is true, the value of speed should be in radians
> if default_is_radian is false, The value of speed should be in degrees
> @param acc: move acceleration (rad/s^2 or °/s^2), default is this.last_used_joint_acc
> if default_is_radian is true, the value of acc should be in radians
> if default_is_radian is false, The value of acc should be in degrees
> @param mvtime: reserved, 0
> @param wait: whether to wait for the arm to complete, default is false
> @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true
> @param radius: move radius, if radius less than 0, will MoveJoint, else MoveArcJoint
> The blending radius cannot be greater than the track length.
> @param relative: relative move or not
> Note: only available if firmware_version >= 1.8.100
>
> @return: see the API Code Documentation for details.

- **int set_servo_angle_j(fp32 angles[7], fp32 speed=0, fp32 acc=0, fp32 mvtime=0)**

> Set the servo angle, execute only the last instruction, need to be set to servo motion mode(this.set_mode(1))
>
> @param angles: angles, like [servo-1, ..., servo-7]
> if default_is_radian is true, the value of servo-1/.../servo-7 should be in radians
> if default_is_radian is false, The value of servo-1/.../servo-7 should be in degrees
> @param speed: reserved, move speed (rad/s or °/s)
> if default_is_radian is true, the value of speed should be in radians
> if default_is_radian is false, The value of speed should be in degrees
> @param acc: reserved, move acceleration (rad/s^2 or °/s^2)

> if default_is_radian is true, the value of acc should be in radians
> if default_is_radian is false, The value of acc should be in degrees
> @param mvtime: reserved, 0
>
> @return: see the API Code Documentation for details.

- **int set_servo_cartesian(fp32 pose[6], fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool is_tool_coord = false)**

> Servo cartesian motion, execute only the last instruction, need to be set to servo motion mode(this.set_mode(1))
>
> @param pose: position, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
> @param speed: reserved, move speed (mm/s)
> @param mvacc: reserved, move acceleration (mm/s^2)
> @param mvtime: reserved, 0
> @param is_tool_coord: is tool coordinate or not
>
> @return: see the API Code Documentation for details.

- **int move_circle(fp32 pose1[6], fp32 pose2[6], fp32 percent, fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool wait=false, fp32 timeout=NO_TIMEOUT, bool is_tool_coord=false, bool is_axis_angle=false)**

> The motion calculates the trajectory of the space circle according to the three-point coordinates.
> Note:
> The three-point coordinates are (current starting point, pose1, pose2).
>
> @param pose1: cartesian position, [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
> @param pose2: cartesian position, [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
> @param percent: the percentage of arc length and circumference of the movement
> @param speed: move speed (mm/s, rad/s), default is this.last_used_tcp_speed
> @param mvacc: move acceleration (mm/s^2, rad/s^2), default is this.last_used_tcp_acc
> @param mvtime: 0, reserved
> @param wait: whether to wait for the arm to complete, default is false
> @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true
> @param is_tool_coord: is tool coord or not, default is false, only available if firmware_version >= 1.11.100
> @param is_axis_angle: is axis angle or not, default is false, only available if firmware_version >= 1.11.100

> @return: see the API Code Documentation for details.

- **int move_gohome(fp32 speed=0, fp32 acc=0, fp32 mvtime=0, bool wait=false, fp32 timeout=NO_TIMEOUT)**
  int move_gohome(bool wait=false, fp32 timeout=NO_TIMEOUT)

  > Move to go home (Back to zero)
  >
  > @param speed: move speed (rad/s or °/s), default is 50 °/s
  > if default_is_radian is true, the value of speed should be in radians
  > if default_is_radian is false, The value of speed should be in degrees
  > @param acc: move acceleration (rad/s^2 or °/s^2), default is 1000 °/s^2
  > if default_is_radian is true, the value of acc should be in radians
  > if default_is_radian is false, The value of acc should be in degrees
  > @param mvtime: reserved, 0
  > @param wait: whether to wait for the arm to complete, default is false
  > @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true
  >
  > @return: see the API Code Documentation for details.

- **void reset(bool wait=false, fp32 timeout=NO_TIMEOUT)**

  > Reset
  >
  > @param wait: whether to wait for the arm to complete, default is false
  > @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true

- **void emergency_stop(void)**

  > Emergency stop

- **int set_tcp_offset(fp32 pose_offset[6])**

  > Set the tool coordinate system offset at the end
  >
  > @param pose_offset: tcp offset, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
  > if default_is_radian is true, the value of roll/pitch/yaw should be in radians
  > if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
  >
  > @return: see the API Code Documentation for details.

- **int set_tcp_load(fp32 weight, fp32 center_of_gravity[3])**

  > Set the tcp load
  >
  > @param weight: load weight (unit: kg)
  > @param center_of_gravity: tcp load center of gravity, like [x(mm), y(mm), z(mm)]
  > @return: see the API Code Documentation for details.

- **int set_tcp_jerk(fp32 jerk)**

  > Set the translational jerk of Cartesian space

> @param jerk: jerk (mm/s^3)
> @return: see the API Code Documentation for details.

- **int set_tcp_maxacc(fp32 acc)**

> Set the max translational acceleration of Cartesian space
>
> @param acc: max acceleration (mm/s^2)
> @return: see the API Code Documentation for details.

- **int set_joint_jerk(fp32 jerk)**

> Set the jerk of Joint space
>
> @param jerk: jerk (°/s^3 or rad/s^3)
> if default_is_radian is true, the value of jerk should be in radians
> if default_is_radian is false, The value of jerk should be in degrees
> @return: see the API Code Documentation for details.

- **int set_joint_maxacc(fp32 acc)**

> Set the max acceleration of Joint space
>
> @param acc: max acceleration (°/s^2 or rad/s^2)
> if default_is_radian is true, the value of jerk should be in radians
> if default_is_radian is false, The value of jerk should be in degrees
> @return: see the API Code Documentation for details.

- **int get_inverse_kinematics(fp32 pose[6], fp32 angles[7])**

> Get inverse kinematics
>
> @param pose: source pose, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
> @param angles: target angles, like [servo-1, ..., servo-7]
> if default_is_radian is true, the value of servo-1/.../servo-7 should be in radians
> if default_is_radian is false, The value of servo-1/.../servo-7 should be in degrees
>
> @return: see the API Code Documentation for details.

- **int get_forward_kinematics(fp32 angles[7], fp32 pose[6])**

> Get forward kinematics
>
> @param angles: source angles, like [servo-1, ..., servo-7]
> if default_is_radian is true, the value of servo-1/.../servo-7 should be in radians
> if default_is_radian is false, The value of servo-1/.../servo-7 should be in degrees
> @param pose: target pose, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
>
> @return: see the API Code Documentation for details.

- **int is_tcp_limit(fp32 pose[6], int *limit)**

  > Check the tcp pose is in limit
  >
  > @param pose: pose, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
  > if default_is_radian is true, the value of roll/pitch/yaw should be in radians
  > if default_is_radian is false, The value of roll/pitch/yaw should be in degrees
  > @param limit: 1: limit, 0: no limit
  >
  > @return: see the API Code Documentation for details.

- **int is_joint_limit(fp32 angles[7], int *limit)**

  > Check the joint is in limit
  >
  > @param angles: angles, like [servo-1, ..., servo-7]
  > if default_is_radian is true, the value of servo-1/.../servo-7 should be in radians
  > if default_is_radian is false, The value of servo-1/.../servo-7 should be in degrees
  > @param limit: 1: limit, 0: no limit
  >
  > @return: see the API Code Documentation for details.

- **int set_gripper_enable(bool enable)**

  > Set the gripper enable
  >
  > @param enable: enable or not
  > @return: see the API Code Documentation for details.

- **int set_gripper_mode(int mode)**

  > Set the gripper mode
  >
  > @param mode:
  > 1: location mode
  > 2: speed mode(no use)
  > 3: torque mode(no use)
  > @return: see the API Code Documentation for details.

- **int get_gripper_position(fp32 *pos)**

  > Get the gripper position
  >
  > @param pos: used to store the results obtained
  > @return: see the API Code Documentation for details.

- **int set_gripper_position(fp32 pos, bool wait=false, fp32 timeout=10, bool wait_motion = true)**

  > Set the gripper position
  >
  > @param pos: gripper position
  > @param wait: wait or not, default is false
  > @param timeout: maximum waiting time(unit: second), default is 10s, only valid if wait is true
  >
  > @return: see the API Code Documentation for details.

- **int set_gripper_speed(fp32 speed)**

  > Set the gripper speed
  >
  > @param speed:
  > @return: see the API Code Documentation for details.

- **int get_gripper_err_code(int *err)**

  > Get the gripper error code
  >
  > @param err: used to store the results obtained
  > @return: see the API Code Documentation for details.

- **int clean_gripper_error(void)**

  > Clean the gripper error
  >
  > @return: see the API Code Documentation for details.

- **int get_tgpio_digital(int *io0_value, int *io1_value)**

  > Get the digital value of the Tool GPIO
  >
  > @param io0_value: the digital value of Tool GPIO-0
  > @param io1_value: the digital value of Tool GPIO-1
  > @return: see the API Code Documentation for details.

- **int set_tgpio_digital(int ionum, int value, float delay_sec=0)**

  > Set the digital value of the specified Tool GPIO
  >
  > @param ionum: ionum, 0 or 1
  > @param value: the digital value of the specified io
  > @param delay_sec: delay effective time from the current start, in seconds, default is 0(effective immediately)
  > @return: see the API Code Documentation for details.

- **int get_tgpio_analog(int ionum, fp32 *value)**

  > Get the analog value of the specified Tool GPIO
  >
  > @param ionum: ionum, 0 or 1
  > @param value: the analog value of the specified tool io
  > @return: see the API Code Documentation for details.

- **int get_cgpio_digital(int *digitals)**

  > Get the digital value of the specified Controller GPIO
  >
  > @param digitals: the values of the controller GPIO
  > @return: see the API Code Documentation for details.

- **int get_cgpio_analog(int ionum, fp32 *value)**

  > Get the analog value of the specified Controller GPIO

> @param ionum: ionum, 0 or 1
> @param value: the analog value of the specified controller io
> @return: see the API Code Documentation for details.

- **int set_cgpio_digital(int ionum, int value, float delay_sec=0)**

> Set the digital value of the specified Controller GPIO
>
> @param ionum: ionum, 0 ~ 15
> @param value: the digital value of the specified io
> @param delay_sec: delay effective time from the current start, in seconds, default is 0(effective immediately)
> @return: see the API Code Documentation for details.

- **int set_cgpio_analog(int ionum, float value)**

> Set the analog value of the specified Controller GPIO
>
> @param ionum: ionum, 0 or 1
> @param value: the analog value of the specified io
> @return: see the API Code Documentation for details.

- **int set_cgpio_digital_input_function(int ionum, int fun)**

> Set the digital input functional mode of the Controller GPIO
>
> @param ionum: ionum, 0 ~ 15
> @param fun: functional mode
> 0: general input
> 1: external emergency stop
> 2: protection reset
> 11: offline task
> 12: teaching mode
> 13: reduced mode
> 14: enable arm
>
> @return: see the API Code Documentation for details.

- **int set_cgpio_digital_output_function(int ionum, int fun)**

> Set the digital output functional mode of the specified Controller GPIO
>
> @param ionum: ionum, 0 ~ 15
> @param fun: functional mode
> 0: general output
> 1: emergency stop
> 2: in motion
> 11: has error
> 12: has warn
> 13: in collision
> 14: in teaching
> 15: in offline task
> 16: in reduced mode

> 17: is enabled
> 18: emergency stop is pressed
>
> @return: see the API Code Documentation for details.

- **int get_cgpio_state(int *state, int *digit_io, fp32 *analog, int *input_conf, int *output_conf, int *input_conf2 = NULL, int *output_conf2 = NULL)**

  > Get the state of the Controller GPIO
  >
  > @param state: contorller gpio module state and controller gpio module error code
  > state[0]: contorller gpio module state
  > state[0] == 0: normal
  > state[0] == 1： wrong
  > state[0] == 6： communication failure
  > state[1]: controller gpio module error code
  > state[1] == 0: normal
  > state[1] != 0： error code
  > @param digit_io:
  > digit_io[0]: digital input functional gpio state
  > digit_io[1]: digital input configuring gpio state
  > digit_io[2]: digital output functional gpio state
  > digit_io[3]: digital output configuring gpio state
  > @param analog:
  > analog[0]: analog-0 input value
  > analog[1]: analog-1 input value
  > analog[2]: analog-0 output value
  > analog[3]: analog-1 output value
  > @param input_conf: digital(0-7) input functional info
  > @param output_conf: digital(0-7) output functional info
  > @param input_conf2: digital(8-15) input functional info
  > @param output_conf2: digital(8-15) output functional info
  >
  > @return: see the API Code Documentation for details.

- **int register_report_location_callback(void(*callback)(const fp32 *pose, const fp32 *angles))**

  > Register the report location callback

- **int register_connect_changed_callback(void(*callback)(bool connected, bool reported))**

  > Register the connect status changed callback

- **int register_state_changed_callback(void(*callback)(int state))**

  > Register the state status changed callback

- **int register_mode_changed_callback(void(*callback)(int mode))**

  > Register the mode changed callback

- **int register_mtable_mtbrake_changed_callback(void(*callback)(int mtable, int mtbrake))**

> Register the motor enable states or motor brake states changed callback

- **int register_error_warn_changed_callback(void(*callback)(int err_code, int warn_code))**

  > Register the error code or warn code changed callback

- **int register_cmdnum_changed_callback(void(*callback)(int cmdnum))**

  > Register the cmdnum changed callback

- **int register_temperature_changed_callback(void(*callback)(const fp32 *temps))**

  > Register the temperature changed callback

- **int register_count_changed_callback(void(*callback)(int count))**

  > Register the value of counter changed callback

- **int register_iden_progress_changed_callback(void(*callback)(int progress))**

  > Register the progress of identification changed callback

- **int release_report_location_callback(void(*callback)(const fp32 *pose, const fp32 *angles)=NULL)**

  > Release the location report callback
  >
  > @param callback: NULL means to release all callbacks

- **int release_connect_changed_callback(void(*callback)(bool connected, bool reported)=NULL)**

  > Release the connect changed callback
  >
  > @param callback: NULL means to release all callbacks for the same event

- **int release_state_changed_callback(void(*callback)(int state)=NULL)**

  > Release the state changed callback
  >
  > @param callback: NULL means to release all callbacks for the same event

- **int release_mode_changed_callback(void(*callback)(int mode)=NULL)**

  > Release the mode changed callback
  >
  > @param callback: NULL means to release all callbacks for the same event

- **int release_mtable_mtbrake_changed_callback(void(*callback)(int mtable, int mtbrake)=NULL)**

  > Release the motor enable states or motor brake states changed callback
  >
  > @param callback: NULL means to release all callbacks for the same event

- **int release_error_warn_changed_callback(void(*callback)(int err_code, int warn_code)=NULL)**

> Release the error warn changed callback
>
> @param callback: NULL means to release all callbacks for the same event

- **int release_cmdnum_changed_callback(void(*callback)(int cmdnum)=NULL)**

> Release the cmdnum changed callback
>
> @param callback: NULL means to release all callbacks for the same event

- **int release_temperature_changed_callback(void(*callback)(const fp32 *temps)=NULL)**

> Release the temperature changed callback
>
> @param callback: NULL means to release all callbacks for the same event

- **int release_count_changed_callback(void(*callback)(int count)=NULL)**

> Release the value of counter changed callback
>
> @param callback: NULL means to release all callbacks for the same event

- **int release_iden_progress_changed_callback(void(*callback)(int progress) = NULL)**

> Release the progress of identification changed callback
>
> @param callback: NULL means to release all callbacks for the same event

- **int get_vacuum_gripper(int *val)**

> Get vacuum gripper state, alias for `get_suction_cup`
>
> @param val:
> 0: vacuum gripper is off
> 1: vacuum gripper is on
> @return: see the API Code Documentation for details.

- **int set_vacuum_gripper(bool on, bool wait=false, float timeout=3, float delay_sec=0)**

> Set vacuum gripper, alias for `set_suction_cup`
>
> @param on: open vacuum gripper or not
> @param wait: wait or not, default is false
> @param timeout: maximum waiting time(unit: second), default is 10s, only valid if wait is true
> @param delay_sec: delay effective time from the current start, in seconds, default is 0(effective immediately)
>
> @return: see the API Code Documentation for details.

- **int get_gripper_version(unsigned char versions[3])**

> Get gripper version, only for debug
>
> @return: see the API Code Documentation for details.

- **int get_servo_version(unsigned char versions[3], int servo_id=1)**

> Get servo version, only for debug

> @return: see the API Code Documentation for details.

- **int get_tgpio_version(unsigned char versions[3])**

> Get tool gpio version, only for debug
>
> @return: see the API Code Documentation for details.

- **int reload_dynamics(void)**

> Reload dynamics, only for debug
>
> @return: see the API Code Documentation for details.

- **int set_reduced_mode(bool on)**

> Turn on/off reduced mode
>
> @param on: on/off
> @return: see the API Code Documentation for details.

- **int set_reduced_max_tcp_speed(float speed)**

> Set the maximum tcp speed of the reduced mode
>
> @param speed: the maximum tcp speed
> @return: see the API Code Documentation for details.

- **int set_reduced_max_joint_speed(float speed)**

> Set the maximum joint speed of the reduced mode
>
> @param speed: the maximum joint speed
> if default_is_radian is true, the value of speed should be in radians
> if default_is_radian is false, The value of speed should be in degrees
> @return: see the API Code Documentation for details.

- **int get_reduced_mode(int *mode)**

> Get reduced mode
>
> @param mode:
> 0: reduced mode is on
> 1: reduced mode is off
> @return: see the API Code Documentation for details.

- **int get_reduced_states(int *on, int *xyz_list, float *tcp_speed, float *joint_speed, float jrange[14]=NULL, int *fense_is_on=NULL, int *collision_rebound_is_on=NULL)**

> Get states of the reduced mode
>
> @param on:
> 0: reduced mode is on
> 1: reduced mode is off
> @param xyz_list: the tcp boundary, like [reduced_x_max, reduced_x_min, reduced_y_max, reduced_y_min, reduced_z_max, reduced_z_min]
> @param tcp_speed: the maximum tcp speed of reduced mode

> @param joint_speed: the maximum joint speed of reduced mode
> if default_is_radian is true, the value of speed should be in radians
> if default_is_radian is false, The value of speed should be in degrees
> @param jrange: the joint range of the reduced mode, like [joint-1-min, joint-1-max, ..., joint-7-min, joint-7-max]
> if default_is_radian is true, the value of speed should be in radians
> if default_is_radian is false, The value of speed should be in degrees
> @param fense_is_on:
> 0: safety mode is on
> 1: safety mode is off
> @param collision_rebound_is_on:
> 0: collision rebound is on
> 1: collision rebound is off
>
> @return: see the API Code Documentation for details.

- **int set_reduced_tcp_boundary(int boundary[6])**

> Set the boundary of the safety boundary mode
>
> @param boundary: like [x_max(mm), x_min(mm), y_max(mm), y_min(mm), z_max(mm), z_min(mm)]
> @return: see the API Code Documentation for details.

- **int set_reduced_joint_range(float jrange[14])**

> Set the joint range of the reduced mode
>
> @param jrange: like [joint-1-min, joint-1-max, ..., joint-7-min, joint-7-max]
> if default_is_radian is true, the value of speed should be in radians
> if default_is_radian is false, The value of speed should be in degrees
> @return: see the API Code Documentation for details.

- **int set_fence_mode(bool on)**

> Turn on/off safety mode, alias for `set_fense_mode`
>
> @param on: on/off
> @return: see the API Code Documentation for details.

- **int set_collision_rebound(bool on)**

> Turn on/off collision rebound
>
> @param on: on/off
> @return: see the API Code Documentation for details.

- **int set_world_offset(float pose_offset[6])**

> Set the base coordinate system offset at the end
>
> @param pose_offset: tcp offset, like [x(mm), y(mm), z(mm), roll(rad or °), pitch(rad or °), yaw(rad or °)]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, The value of roll/pitch/yaw should be in degrees

> @return: see the API Code Documentation for details.

- **int start_record_trajectory(void)**

> Start trajectory recording, only in teach mode, so you need to set joint teaching mode before.
>
> @return: see the API Code Documentation for details.

- **int stop_record_trajectory(char* filename=NULL)**

> Stop trajectory recording
>
> @param filename: the name to save
> If the filename is NULL, just stop recording, do not save, you need to manually call `save_record_trajectory` save before changing the mode. otherwise it will be lost the trajectory is saved in the controller box.
> This action will overwrite the trajectory with the same name
> empty the trajectory in memory after saving, so repeated calls will cause the recorded trajectory to be covered by an empty trajectory.
>
> @return: see the API Code Documentation for details.

- **int save_record_trajectory(char* filename, float timeout=10)**

> Save the trajectory you just recorded
>
> @param filename: the name to save
> The trajectory is saved in the controller box.
> This action will overwrite the trajectory with the same name
> empty the trajectory in memory after saving, so repeated calls will cause the recorded trajectory to be covered by an empty trajectory.
>
> @return: see the API Code Documentation for details.

- **int load_trajectory(char* filename, float timeout=10)**

> Load the trajectory
>
> @param filename: the name of the trajectory to load
> @param timeout: the maximum timeout waiting for loading to complete, default is 10 seconds.
>
> @return: see the API Code Documentation for details.

- **int playback_trajectory(int times=1, char* filename=NULL, bool wait=false, int double_speed=1)**

> Playback trajectory
>
> @param times: number of playbacks.
> @param filename: the name of the trajectory to play back
> if filename is None, you need to manually call the `load_trajectory` to load the trajectory.
> @param wait: whether to wait for the arm to complete, default is false.

> @param double_speed: double speed, only support 1/2/4, default is 1, only available if version > 1.2.11
>
> @return: see the API Code Documentation for details.

- **int get_trajectory_rw_status(int *status)**

> Get trajectory read/write status
>
> @param status:
> 0: no read/write
> 1: loading
> 2: load success
> 3: load failed
> 4: saving
> 5: save success
> 6: save failed
> @return: see the API Code Documentation for details.

- **int set_counter_reset(void)**

> Reset counter value
>
> @return: see the API Code Documentation for details.

int set_counter_increase(void)

> Set counter plus 1
>
> @return: see the API Code Documentation for details.

- **int set_tgpio_digital_with_xyz(int ionum, int value, float xyz[3], float tol_r)**

> Set the digital value of the specified Tool GPIO when the robot has reached the specified xyz position
>
> @param ionum: 0 or 1
> @param value: value
> @param xyz: position xyz, as [x, y, z]
> @param tol_r: fault tolerance radius
>
> @return: see the API Code Documentation for details.

- **int set_cgpio_digital_with_xyz(int ionum, int value, float xyz[3], float tol_r)**

> Set the digital value of the specified Controller GPIO when the robot has reached the specified xyz position
>
> @param ionum: 0 ~ 7
> @param value: value
> @param xyz: position xyz, as [x, y, z]
> @param tol_r: fault tolerance radius
>
> @return: see the API Code Documentation for details.

- **int set_cgpio_digital_with_xyz(int ionum, int value, float xyz[3], float tol_r)**

> Set the analog value of the specified Controller GPIO when the robot has reached the specified xyz position
>
> @param ionum: 0 ~ 1
> @param value: value, 0~10.0
> @param xyz: position xyz, as [x, y, z]
> @param tol_r: fault tolerance radius
>
> @return: see the API Code Documentation for details.

- **int config_tgpio_reset_when_stop(bool on_off)**

> Config the Tool GPIO reset the digital output when the robot is in stop state
>
> @param on_off: true/false
>
> @return: see the API Code Documentation for details.

- **int config_cgpio_reset_when_stop(bool on_off)**

> Config the Controller GPIO reset the digital output when the robot is in stop state
>
> @param on_off: true/false
>
> @return: see the API Code Documentation for details.

- **int set_position_aa(fp32 pose[6], fp32 speed = 0, fp32 acc = 0, fp32 mvtime = 0, bool is_tool_coord = false, bool relative = false, bool wait = false, fp32 timeout = NO_TIMEOUT, unsigned char motion_type = 0)**
- **int set_position_aa(fp32 pose[6], bool is_tool_coord, bool relative = false, bool wait = false, fp32 timeout = NO_TIMEOUT, unsigned char motion_type = 0)**

> Set the pose represented by the axis angle pose
> MoveLineAA: Linear motion
> MoveArcLineAA: Linear arc motion with interpolation
>
> @param pose: the axis angle pose, like [x(mm), y(mm), z(mm), rx(rad or °), ry(rad or °), rz(rad or °)]
> Note: if default_is_radian is true, the value of rx/ry/rz should be in radians
> Note: if default_is_radian is false, The value of rx/ry/rz should be in degrees
> @param speed: move speed (mm/s, rad/s), default is this.last_used_tcp_speed
> @param mvacc: move acceleration (mm/s^2, rad/s^2), default is this.last_used_tcp_acc
> @param mvtime: reserved, 0
> @param is_tool_coord: is tool coordinate or not, if it is true, the relative parameter is no longer valid
> @param relative: relative move or not
> @param wait: whether to wait for the arm to complete, default is false
> @param timeout: maximum waiting time(unit: second), default is no timeout, only valid if wait is true
> @param radius: move radius, if radius less than 0, will MoveLineAA, else MoveArcLineAA
> Note: only available if firmware_version >= 1.11.100
> @param motion_type: motion planning type, default is 0
> motion_type == 0: default, linear planning

> motion_type == 1: prioritize linear planning, and turn to IK for joint planning when linear planning is not possible
> motion_type == 2: direct transfer to IK using joint planning
> Note:
> 1. only available if firmware_version >= 1.11.100
> 2. when motion_type is 1 or 2, linear motion cannot be guaranteed
> 3. once IK is transferred to joint planning, the given Cartesian velocity and acceleration are converted into joint velocity and acceleration according to the percentage
>
> `speed = speed / max_tcp_speed * max_joint_speed`
>
> `acc = acc / max_tcp_acc * max_joint_acc`
>
> 4. if there is no suitable IK, a C40 error will be triggered
>
> @return: see the API Code Documentation for details.

- **int set_servo_cartesian_aa(fp32 pose[6], fp32 speed = 0, fp32 acc = 0, bool is_tool_coord = false, bool relative = false)**
- **int set_servo_cartesian_aa(fp32 pose[6], bool is_tool_coord, bool relative = false)**

> Set the servo cartesian represented by the axis angle pose, execute only the last instruction, need to be set to servo motion mode(self.set_mode(1))
> Note: only available if firmware_version >= 1.4.7
>
> @param pose: the axis angle pose, like [x(mm), y(mm), z(mm), rx(rad or °), ry(rad or °), rz(rad or °)]
> Note: if default_is_radian is true, the value of rx/ry/rz should be in radians
> Note: if default_is_radian is false, The value of rx/ry/rz should be in degrees
> @param speed: reserved, move speed (mm/s)
> @param mvacc: reserved, move acceleration (mm/s^2)
> @param is_tool_coord: is tool coordinate or not
> @param relative: relative move or not
>
> @return: see the API Code Documentation for details.

- **int get_pose_offset(float pose1[6], float pose2[6], float offset[6], int orient_type_in = 0, int orient_type_out = 0)**

> Calculate the pose offset of two given points
>
> @param pose1: position, like [x(mm), y(mm), z(mm), roll/rx(rad or °), pitch/ry(rad or °), yaw/rz(rad or °)]
> Note: if default_is_radian is true, the value of roll/rx/pitch/ry/yaw/rz should be in radians
> Note: if default_is_radian is false, The value of roll/rx/pitch/ry/yaw/rz should be in degrees
> @param pose2: position, like [x(mm), y(mm), z(mm), roll/rx(rad or °), pitch/ry(rad or °), yaw/rz(rad or °)]
> Note: if default_is_radian is true, the value of roll/rx/pitch/ry/yaw/rz should be in radians
> Note: if default_is_radian is false, The value of roll/rx/pitch/ry/yaw/rz should be in degrees
> @param offset: the offset between pose1 and pose2

> @param orient_type_in: input attitude notation, 0 is RPY (default), 1 is axis angle
> @param orient_type_out: notation of output attitude, 0 is RPY (default), 1 is axis angle
>
> @return: see the API Code Documentation for details.

- **int get_position_aa(fp32 pose[6])**

> Get the pose represented by the axis angle pose
>
> @param pose: the pose represented by the axis angle pose of xArm, like [x(mm), y(mm), z(mm), rx(rad or °), ry(rad or °), rz(rad or °)]
> Note: if default_is_radian is true, the value of rx/ry/rz should be in radians
> Note: if default_is_radian is false, The value of rx/ry/rz should be in degrees
>
> @return: see the API Code Documentation for details.

- **int robotiq_reset(unsigned char ret_data[6] = NULL)**

> Reset the robotiq gripper (clear previous activation if any)
>
> @param ret_data: the response from robotiq
>
> @return: see the API Code Documentation for details.

- **int robotiq_set_activate(bool wait = true, fp32 timeout = 3, unsigned char ret_data[6] = NULL)**
- **int robotiq_set_activate(bool wait = true, unsigned char ret_data[6] = NULL)**
- **int robotiq_set_activate(unsigned char ret_data[6] = NULL)**

> If not already activated. Activate the robotiq gripper
>
> @param wait: whether to wait for the robotiq activate complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 3, only available if wait=true
> @param ret_data: the response from robotiq
>
> @return: see the API Code Documentation for details.

- **int robotiq_set_position(unsigned char pos, unsigned char speed = 0xFF, unsigned char force = 0xFF, bool wait = true, fp32 timeout = 5, unsigned char ret_data[6] = NULL)**
- **int robotiq_set_position(unsigned char pos, bool wait = true, fp32 timeout = 5, unsigned char ret_data[6] = NULL)**
- **int robotiq_set_position(unsigned char pos, bool wait = true, unsigned char ret_data[6] = NULL)**
- **int robotiq_set_position(unsigned char pos, unsigned char ret_data[6] = NULL)**

> Go to the position with determined speed and force.
>
> @param pos: position of the gripper. Integer between 0 and 255. 0 being the open position and 255 being the close position.
> @param speed: gripper speed between 0 and 255
> @param force: gripper force between 0 and 255
> @param wait: whether to wait for the robotion motion complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 5, only available if

> wait=true
> @param ret_data: the response from robotiq
>
> @return: see the API Code Documentation for details.

- **int robotiq_open(unsigned char speed = 0xFF, unsigned char force = 0xFF, bool wait = true, fp32 timeout = 5, unsigned char ret_data[6] = NULL)**
- **int robotiq_open(bool wait = true, fp32 timeout = 5, unsigned char ret_data[6] = NULL)**
- **int robotiq_open(bool wait = true, unsigned char ret_data[6] = NULL)**
- **int robotiq_open(unsigned char ret_data[6] = NULL)**

> Open the robotiq gripper
>
> @param speed: gripper speed between 0 and 255
> @param force: gripper force between 0 and 255
> @param wait: whether to wait for the robotion motion complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 5, only available if wait=true
> @param ret_data: the response from robotiq
>
> @return: see the API Code Documentation for details.

- **int robotiq_close(unsigned char speed = 0xFF, unsigned char force = 0xFF, bool wait = true, fp32 timeout = 5, unsigned char ret_data[6] = NULL)**
- **int robotiq_close(bool wait = true, fp32 timeout = 5, unsigned char ret_data[6] = NULL)**
- **int robotiq_close(bool wait = true, unsigned char ret_data[6] = NULL)**
- **int robotiq_close(unsigned char ret_data[6] = NULL)**

> Close the robotiq gripper
>
> @param speed: gripper speed between 0 and 255
> @param force: gripper force between 0 and 255
> @param wait: whether to wait for the robotion motion complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 5, only available if wait=true
> @param ret_data: the response from robotiq
>
> @return: see the API Code Documentation for details.

- **int robotiq_get_status(unsigned char ret_data[9], unsigned char number_of_registers = 3)**

> Reading the status of robotiq gripper
>
> @param ret_data: the response from robotiq
> @param number_of_registers: number of registers, 1/2/3, default is 3
> number_of_registers=1: reading the content of register 0x07D0
> number_of_registers=2: reading the content of register 0x07D0/0x07D1
> number_of_registers=3: reading the content of register 0x07D0/0x07D1/0x07D2

> Note:
>
> register 0x07D0: Register GRIPPER STATUS
> register 0x07D1: Register FAULT STATUS and register POSITION REQUEST ECHO
> register 0x07D2: Register POSITION and register CURRENT
>
> @return: see the API Code Documentation for details.

- **int set_bio_gripper_enable(bool enable, bool wait = true, fp32 timeout = 3)**

> If not already enabled. Enable the bio gripper
>
> @param enable: enable or not
> @param wait: whether to wait for the bio gripper enable complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 3, only available if wait=true
>
> @return: See the API Code Documentation for details.

- **int set_bio_gripper_speed(int speed)**

> Set the speed of the bio gripper
>
> @param speed: speed
>
> @return: See the API Code Documentation for details.

- **int open_bio_gripper(int speed = 0, bool wait = true, fp32 timeout = 5)**
- **int open_bio_gripper(bool wait = true, fp32 timeout = 5)**

> Open the bio gripper
>
> @param speed: speed value, default is 0 (not set the speed)
> @param wait: whether to wait for the bio gripper motion complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 5, only available if wait=true
>
> @return: See the API Code Documentation for details.

- **int close_bio_gripper(int speed = 0, bool wait = true, fp32 timeout = 5)**
- **int close_bio_gripper(bool wait = true, fp32 timeout = 5)**

> Close the bio gripper
>
> @param speed: speed value, default is 0 (not set the speed)
> @param wait: whether to wait for the bio gripper motion complete, default is true
> @param timeout: maximum waiting time(unit: second), default is 5, only available if wait=true
>
> @return: See the API Code Documentation for details.

- **int get_bio_gripper_status(int *status)**

> Get the status of the bio gripper

> @param status: the result of the bio gripper status value
>
> status & 0x03 == 0: stop
>
> status & 0x03 == 1: motion
>
> status & 0x03 == 2: catch
>
> status & 0x03 == 3: error
>
> (status >> 2) & 0x03 == 0: not enabled
>
> (status >> 2) & 0x03 == 1: enabling
>
> (status >> 2) & 0x03 == 2: enabled
>
> return: See the API Code Documentation for details.

- **int get_bio_gripper_error(int *err)**

> Get the error code of the bio gripper
>
> @param err: the result of the bio gripper error code
>
> @return: See the API Code Documentation for details.

- **int clean_bio_gripper_error(void)**

> Clean the error code of the bio gripper
>
> @return: See the API Code Documentation for details.

- **int set_tgpio_modbus_timeout(int timeout, bool is_transparent_transmission = false)**

> Set the modbus timeout of the tool gpio
>
> @param timeout: timeout, milliseconds
> @param is_transparent_transmission: whether the set timeout is the timeout of transparent transmission
>
> Note: only available if firmware_version >= 1.11.0
>
> @return: See the API Code Documentation for details.

- **int set_tgpio_modbus_baudrate(int baud)**

> Set the modbus baudrate of the tool gpio
>
> @param baud: baudrate, 4800/9600/19200/38400/57600/115200/230400/460800/921600/1000000/1500000/2000000/2500000
>
> return: See the API Code Documentation for details.

- **int get_tgpio_modbus_baudrate(int *baud)**

> Get the modbus baudrate of the tool gpio
>
> @param baud: the result of baudrate
>
> @return: See the API Code Documentation for details.

- **int getset_tgpio_modbus_data(unsigned char *modbus_data, int modbus_length, unsigned char *ret_data, int ret_length, unsigned char host_id = 9, bool is_transparent_transmission = false)**

> Send the modbus data to the tool gpio
>
> @param modbus_data: send data
> @param modbus_length: the length of the modbus_data
> @param ret_data: the response data of the modbus
> @param ret_length: the length of the response data
> @param host_id: host id
> 9: END RS485
> 10: Controller RS485
> @param is_transparent_transmission: whether to choose transparent transmission, default is false
> Note: only available if firmware_version >= 1.11.0
> @param use_503_port: whether to use port 503 for communication, default is false
> Note: if it is true, it will connect to 503 port for communication when it is used for the first time, which is generally only useful for transparent transmission
> Note: only available if firmware_version >= 1.11.0
>
> @return: See the API Code Documentation for details.

- **int set_report_tau_or_i(int tau_or_i = 0)**

> Set the reported torque or electric current
>
> @param tau_or_i:
> 0: torque
> 1: electric current
>
> @return: See the API Code Documentation for details.

- **int get_report_tau_or_i(int *tau_or_i)**

> Get the reported torque or electric current
>
> @param tau_or_i: the result of the tau_or_i
>
> @return: See the API Code Documentation for details.

- **int set_self_collision_detection(bool on)**

> Set whether to enable self-collision detection
>
> @param on: enable or not
>
> @return: See the API Code Documentation for details.

- **int set_collision_tool_model(int tool_type, int n = 0, ...)**

> Set the geometric model of the end effector for self collision detection
>
> @param tool_type: the geometric model type
> 0: No end effector, no additional parameters required
> 1: xArm Gripper, no additional parameters required
> 2: xArm Vacuum Gripper, no additional parameters required
> 3: xArm Bio Gripper, no additional parameters required
> 4: Robotiq-2F-85 Gripper, no additional parameters required
> 5: Robotiq-2F-140 Gripper, no additional parameters required

21: Cylinder, need additional parameters radius, height
arm->set_collision_tool_model(21, 2, radius, height)
@param radius: the radius of cylinder, (unit: mm)
@param height: the height of cylinder, (unit: mm)
22: Cuboid, need additional parameters x, y, z
arm->set_collision_tool_model(22, 3, x, y, z)
@param x: the length of the cuboid in the x coordinate direction, (unit: mm)
@param y: the length of the cuboid in the y coordinate direction, (unit: mm)
@param z: the length of the cuboid in the z coordinate direction, (unit: mm)

@param n: the count of the additional parameters
@param ...: additional parameters

@return: See the API Code Documentation for details.

- **int vc_set_joint_velocity(fp32 speeds[7], bool is_sync = true, fp32 duration = -1.0)**

Joint velocity control, need to be set to joint velocity control mode(this.set_mode(4))

@param speeds: [spd_J1, spd_J2, ..., spd_J7]
if default_is_radian is true, the value of spd_J1/.../spd_J1 should be in radians
if default_is_radian is false, the value of spd_J1/.../spd_J1 should be in degrees
@param is_sync: whether all joints accelerate and decelerate synchronously, default is true
@param duration: the maximum duration of the speed, over this time will automatically set the speed to 0.
duration > 0: seconds, indicates the maximum number of seconds that this speed can be maintained
duration == 0: always effective, will not stop automatically
duration < 0: default value, only used to be compatible with the old protocol, equivalent to 0
Note: only available if firmware_version >= 1.8.0

@return: See the API Code Documentation for details.

- **int vc_set_cartesian_velocity(fp32 speeds[6], bool is_tool_coord = false, fp32 duration = -1.0)**

Cartesian velocity control, need to be set to cartesian velocity control mode(self.set_mode(5))

@param speeds: [spd_x, spd_y, spd_z, spd_rx, spd_ry, spd_rz]
if default_is_radian is true, the value of spd_rx/spd_ry/spd_rz should be in radians
if default_is_radian is false, the value of spd_rx/spd_ry/spd_rz should be in degrees
@param is_tool_coord: is tool coordinate or not, default is false
@param duration: the maximum duration of the speed, over this time will automatically set the speed to 0.
duration > 0: seconds, indicates the maximum number of seconds that this speed can be maintained
duration == 0: always effective, will not stop automatically
duration < 0: default value, only used to be compatible with the old protocol, equivalent to 0
Note: only available if firmware_version >= 1.8.0

> @return: See the API Code Documentation for details.

- **int calibrate_tcp_coordinate_offset(float four_points[4][6], float ret_xyz[3])**

> Four-point method to calibrate tool coordinate system position offset
>
> @param four_points: a list of four teaching coordinate positions [x, y, z, roll, pitch, yaw]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, the value of roll/pitch/yaw should be in degrees
> @param ret_xyz: the result of the calculated xyz(mm) TCP offset, [x, y, z]
>
> @return: See the API Code Documentation for details.

- **int calibrate_tcp_orientation_offset(float rpy_be[3], float rpy_bt[3], float ret_rpy[3])**

> An additional teaching point to calibrate the tool coordinate system attitude offset
>
> @param rpy_be: the rpy value of the teaching point without TCP offset [roll, pitch, yaw]
> @param rpy_bt: the rpy value of the teaching point with TCP offset [roll, pitch, yaw]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, the value of roll/pitch/yaw should be in degrees
> @param ret_rpy: the result of the calculated rpy TCP offset, [roll, pitch, yaw]
>
> @return: See the API Code Documentation for details.

- **int calibrate_user_orientation_offset(float three_points[3][6], float ret_rpy[3], int mode = 0, int trust_ind = 0)**

> Three-point method teaches user coordinate system posture offset
>
> @param four_points: a list of teaching TCP coordinate positions [x, y, z, roll, pitch, yaw]
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, the value of roll/pitch/yaw should be in degrees
> @param ret_rpy: the result of the calculated rpy user offset, [roll, pitch, yaw]
>
> @return: See the API Code Documentation for details.

- **int calibrate_user_coordinate_offset(float rpy_ub[3], float pos_b_uorg[3], float ret_xyz[3])**

> An additional teaching point determines the position offset of the user coordinate system.
>
> @param rpy_ub: the confirmed offset of the base coordinate system in the user coordinate system [roll, pitch, yaw], which is the result of calibrate_user_orientation_offset()
> if default_is_radian is true, the value of roll/pitch/yaw should be in radians
> if default_is_radian is false, the value of roll/pitch/yaw should be in degrees
> @param pos_b_uorg: the position of the teaching point in the base coordinate system [x, y, z], if the arm cannot reach the target position, the user can manually input the position of the target in the base coordinate.
> @param ret_xyz: the result of the calculated xyz user offset, [x, y, z]
>
> @return: See the API Code Documentation for details.

- **int set_impedance(int imp_coord, int imp_c_axis[6], float M[6], float K[6], float B[6])**

> Set all parameters of impedance control through the Six-axis Force Torque Sensor.
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param imp_coord: task frame. 0: base frame. 1: tool frame.
> @param imp_c_axis: a 6d vector of 0s and 1s. 1 means that robot will be impedance in the corresponding axis of the task frame.
> @param M: mass. (kg)
> @param K: stiffness coefficient.
> @param B: damping coefficient. invalid.
>
> Note: the value is set to 2*sqrt(M*K) in controller.
>
> @return: See the API Code Documentation for details.

- **int set_impedance_mbk(float M[6], float K[6], float B[6])**

> Set mbk parameters of impedance control through the Six-axis Force Torque Sensor.
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param M: mass. (kg)
> @param K: stiffness coefficient.
> @param B: damping coefficient. invalid.
>
> Note: the value is set to 2*sqrt(M*K) in controller.
>
> @return: See the API Code Documentation for details.

- **int set_impedance_config(int imp_coord, int imp_c_axis[6])**

> Set impedance control parameters of impedance control through the Six-axis Force Torque Sensor.
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param imp_coord: task frame. 0: base frame. 1: tool frame.
> @param imp_c_axis: a 6d vector of 0s and 1s. 1 means that robot will be impedance in the corresponding axis of the task frame.
>
> @return: See the API Code Documentation for details.

- **int config_force_control(int f_coord, int f_c_axis[6], float f_ref[6], float f_limits[6])**

> Set force control parameters through the Six-axis Force Torque Sensor.
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)

> @param f_coord: task frame. 0: base frame. 1: tool frame.
> @param f_c_axis: a 6d vector of 0s and 1s. 1 means that robot will be impedance in the corresponding axis of the task frame.
> @param f_ref: the forces/torques the robot will apply to its environment. The robot adjusts its position along/about compliant axis in order to achieve the specified force/torque.
> @param f_limits: for compliant axes, these values are the maximum allowed tcp speed along/about the axis.
>
> @return: See the API Code Documentation for details.

- **int set_force_control_pid(float kp[6], float ki[6], float kd[6], float xe_limit[6])**

> Set force control pid parameters through the Six-axis Force Torque Sensor.
> Note:
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param kp: proportional gain
> @param ki: integral gain.
> @param kd: differential gain.
> @param xe_limit: 6d vector. for compliant axes, these values are the maximum allowed tcp speed along/about the axis. mm/s
>
> @return: See the API Code Documentation for details.

- **int ft_sensor_set_zero(void)**

> Set the current state to the zero point of the Six-axis Force Torque Sensor
> Note:
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @return: See the API Code Documentation for details.

- **int ft_sensor_iden_load(float result[10])**

> Identification the tcp load with the the Six-axis Force Torque Sensor
> Note: only available if firmware_version >= 1.8.3
> Note: the Six-axis Force Torque Sensor is required (the third party is not currently supported)
> Note: tstarting from SDK v1.11.0, the centroid unit is millimeters (originally meters)
>
> @param result: the result of identification, ([mass(kg)，x_centroid(mm)，y_centroid(mm)，z_centroid(mm)，Fx_offset，Fy_offset，Fz_offset，Tx_offset，Ty_offset，Tz_ffset])
>
> @return: See the API Code Documentation for details.

- **int ft_sensor_cali_load(float load[10], bool association_setting_tcp_load = false, float m = 0.325, float x = -17, float y = 9, float z = 11.8)**

> Write the load offset parameters identified by the Six-axis Force Torque Sensor
>
> Note: only available if firmware_version >= 1.8.3
>
> Note: the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> Note: tstarting from SDK v1.11.0, the centroid unit is millimeters (originally meters)
>
> @param load: iden result([mass(kg)，x_centroid(mm)，y_centroid(mm)，z_centroid(mm)，Fx_offset，Fy_offset，Fz_offset，Tx_offset，Ty_offset，Tz_ffset])
> @param association_setting_tcp_load: whether to convert the paramster to the corresponding tcp load and set, default is false
>
> if true, the value of tcp load will be modified
>
> @return: See the API Code Documentation for details.

- **int ft_sensor_enable(int on_off)**

> Used for enabling and disabling the use of the Six-axis Force Torque Sensor measurements in the controller.
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param on_off: enable or disable F/T data sampling.
>
> @return: See the API Code Documentation for details.

- **int ft_sensor_app_set(int app_code)**

> Set robot to be controlled in force mode. (Through the Six-axis Force Torque Sensor)
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param app_code: force mode.
>
> 0: non-force mode
> 1: impendance control
> 2: force control
>
> @return: See the API Code Documentation for details.

- **int ft_sensor_app_get(int *app_code)**

> Get force mode
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param app_code: the result of force mode.
>
> 0: non-force mode
> 1: impendance control
> 2: force control

> @return: See the API Code Documentation for details.

- **int get_ft_sensor_data(float ft_data[6])**

> Get the data of the Six-axis Force Torque Sensor
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param ft_data: the result of the Six-axis Force Torque Sensor.
>
> @return: See the API Code Documentation for details.

- **int get_ft_sensor_config(int *ft_app_status = NULL, int *ft_is_started = NULL, int *ft_type = NULL, int *ft_id = NULL, int *ft_freq = NULL, float *ft_mass = NULL, float *ft_dir_bias = NULL, float ft_centroid[3] = NULL, float ft_zero[6] = NULL, int *imp_coord = NULL, int imp_c_axis[6] = NULL, float M[6] = NULL, float K[6] = NULL, float B[6] = NULL, int *f_coord = NULL, int f_c_axis[6] = NULL, float f_ref[6] = NULL, float f_limits[6] = NULL, float kp[6] = NULL, float ki[6] = NULL, float kd[6] = NULL, float xe_limit[6] = NULL)**

> Get the config of the Six-axis Force Torque Sensor
>
> Note:
>
> 1. only available if firmware_version >= 1.8.3
> 2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)
>
> @param ft_app_status: force mode
>
> 0: non-force mode
>
> 1: impendance control
>
> 2: force control
>
> @param ft_is_started: ft sensor is enable or not
> @param ft_type: ft sensor type
> @param ft_id: ft sensor id
> @param ft_freq: ft sensor frequency
> @param ft_mass: load mass
> @param ft_dir_bias:
> @param ft_centroid: [x_centroid，y_centroid，z_centroid]
> @param ft_zero: [Fx_offset，Fy_offset，Fz_offset，Tx_offset，Ty_offset，Tz_ffset]
> @param imp_coord: task frame of impendance control mode.
>
> 0: base frame.
>
> 1: tool frame.
>
> @param imp_c_axis: a 6d vector of 0s and 1s. 1 means that robot will be impedance in the corresponding axis of the task frame.
> @param M: mass. (kg)
> @param K: stiffness coefficient.
> @param B: damping coefficient. invalid.
>
> Note: the value is set to 2*sqrt(M*K) in controller.
>
> @param f_coord: task frame of force control mode.
>
> 0: base frame.

1: tool frame.
@param f_c_axis: a 6d vector of 0s and 1s. 1 means that robot will be impedance in the corresponding axis of the task frame.
@param f_ref: the forces/torques the robot will apply to its environment. The robot adjusts its position along/about compliant axis in order to achieve the specified force/torque.
@param f_limits: for compliant axes, these values are the maximum allowed tcp speed along/about the axis.
@param kp: proportional gain
@param ki: integral gain.
@param kd: differential gain.
@param xe_limit: 6d vector. for compliant axes, these values are the maximum allowed tcp speed along/about the axis. mm/s

@return: See the API Code Documentation for details.

- **int get_ft_sensor_error(int *err)**

Get the error code of the Six-axis Force Torque Sensor
Note:
1. only available if firmware_version >= 1.8.3
2. the Six-axis Force Torque Sensor is required (the third party is not currently supported)

@param err: the result of ft sensor error code

@return: See the API Code Documentation for details.

- **int iden_tcp_load(float result[4], float estimated_mass = 0.0)**

Identification the tcp load with current
Note: only available if firmware_version >= 1.8.0

@param result: the result of identification. [mass，x_centroid，y_centroid，z_centroid]
@param estimated_mass: estimated mass
Note: this parameter is only available on the Lite6 model manipulator, and this parameter must be specified for the Lite6 model manipulator

@return: See the API Code Documentation for details.

- **int get_linear_track_error(int *err)**

Get the error code of the linear track
Note: only available if firmware_version >= 1.8.0

@param err: the result of linear track error

@return: See the API Code Documentation for details.

- **int get_linear_track_status(int *status)**

Get the status of the linear track
Note: only available if firmware_version >= 1.8.0

> @param status: the result of linear track status
> status & 0x00: motion finish.
> status & 0x01: in motion
> status & 0x02: has stop
>
> @return: See the API Code Documentation for details.

- **int get_linear_track_pos(int *pos)**

> Get the pos of the linear track
> Note: only available if firmware_version >= 1.8.0
>
> @param pos: the result of linear track position
>
> @return: See the API Code Documentation for details.

- **int get_linear_track_is_enabled(int *status)**

> Get the linear track is enabled or not
> Note: only available if firmware_version >= 1.8.0
>
> @param status: the result of linear track status
> status == 0: linear track is not enabled
> status == 1: linear track is enabled
>
> @return: See the API Code Documentation for details.

- **int get_linear_track_on_zero(int *status)**

> Get the linear track is on zero positon or not
> Note: only available if firmware_version >= 1.8.0
>
> @param status: the result of linear track status
> status == 0: linear track is not on zero
> status == 1: linear track is on zero
>
> @return: See the API Code Documentation for details.

- **int get_linear_track_sci(int *sci1)**

> Get the sci1 value of the linear track
> Note: only available if firmware_version >= 1.8.0
>
> @param sci1: the result of linear track sci1
>
> @return: See the API Code Documentation for details.

- **int get_linear_track_sco(int sco[2])**

> Get the sco value of the linear track
> Note: only available if firmware_version >= 1.8.0
>
> @param sco: the result of linear track sco0 and sco1
>
> @return: See the API Code Documentation for details.

- **int clean_linear_track_error(void)**

> Clean the linear track error
>
> Note: only available if firmware_version >= 1.8.0
>
> @return: See the API Code Documentation for details.

- **int set_linear_track_enable(bool enable)**

> Set the linear track enable/disable
>
> Note: only available if firmware_version >= 1.8.0
>
> @param enable: enable or not
>
> @return: See the API Code Documentation for details.

- **int set_linear_track_speed(int speed)**

> Set the speed of the linear track
>
> Note: only available if firmware_version >= 1.8.0
>
> @param speed: Integer between 1 and 1000mm/s.
>
> @return: See the API Code Documentation for details.

- **int set_linear_track_back_origin(bool wait = true, bool auto_enable = true)**

> Set the linear track go back to the origin position
>
> Note:
>
> 1. only available if firmware_version >= 1.8.0
> 2. only useful when powering on for the first time
> 3. this operation must be performed at the first power-on
>
> @param wait: wait to motion finish or not, default is true
> @param auto_enable: enable after back to origin or not, default is true
>
> @return: See the API Code Documentation for details.

- **int set_linear_track_pos(int pos, int speed = 0, bool wait = true, fp32 timeout = 100, bool auto_enable = true)**

> Set the position of the linear track
>
> Note: only available if firmware_version >= 1.8.0
>
> @param pos: position. Integer between 0 and 700/1000/1500.
>
> If the SN of the linear track is start with AL1300, the position range is 0~700mm.
>
> If the SN of the linear track is start with AL1301, the position range is 0~1000mm.
>
> If the SN of the linear track is start with AL1302, the position range is 0~1500mm.
>
> @param speed: auto set the speed of the linear track if the speed is changed, Integer between of 1 and 1000mm/s, default is -1(not set)
> @param wait: wait to motion finish or not, default is true
> @param timeout: wait timeout, seconds, default is 100s.
> @param auto_enable: auto enable if not enabled, default is true
>
> @return: See the API Code Documentation for details.

- **int set_linear_track_stop(void)**

> Set the linear track to stop
>
> Note: only available if firmware_version >= 1.8.0
>
> @return: See the API Code Documentation for details.

- **int set_baud_checkset_enable(bool enable)**

> Enable auto checkset the baudrate of the end IO board or not
>
> Note: only available in the API of gripper/bio/robotiq/linear_track.
>
> @return: See the API Code Documentation for details.

- **int set_checkset_default_baud(int type, int baud)**

> Set the checkset baud value
>
> Note: do not modify at will, only use when the baud rate of the corresponding peripheral is changed
>
> @param type: checkset type
>
> 1: xarm gripper
>
> 2: bio gripper
>
> 3: robotiq gripper
>
> 4: linear track
>
> @param baud: checkset baud value, less than or equal to 0 means disable checkset
>
> @return: See the API Code Documentation for details.

- **int get_checkset_default_baud(int type, int *baud)**

> Get the checkset baud value
>
> @param type: checkset type
>
> 1: xarm gripper
>
> 2: bio gripper
>
> 3: robotiq gripper
>
> 4: linear track
>
> @param baud: checkset baud value, less than or equal to 0 means disable checkset
>
> @return: See the API Code Documentation for details.

- **int set_cartesian_velo_continuous(bool on_off)**

> Set cartesian motion velocity continuous
>
> Note: only available if firmware_version >= 1.9.0
>
> @param on_off: continuous or not, default is false
>
> @return: See the API Code Documentation for details.

- **int set_allow_approx_motion(bool on_off)**

> Set allow to avoid overspeed near some singularities using approximate solutions
>
> Note: only available if firmware_version >= 1.9.0
>
> @param on_off: allow or not, default is false
>
> @return: See the API Code Documentation for details.

- **int get_joint_states(fp32 position[7], fp32 velocity[7], fp32 effort[7])**

> Get the joint states
>
> Note: only available if firmware_version >= 1.9.0
>
> @param position: the angles of the joints, like [angle-1, ..., angle-7]
>
> if default_is_radian is true, the value of angle-1/.../angle-7 should be in radians
>
> if default_is_radian is false, The value of angle-1/.../angle-7 should be in degrees
>
> @param velocity: the velocities of the joints, like [velo-1, ..., velo-7]
>
> if default_is_radian is true, the value of velo-1/.../velo-7 should be in radians
>
> if default_is_radian is false, The value of velo-1/.../velo-7 should be in degrees
>
> @param effort: the efforts of the joints, like [effort-1, ..., effort-7]
>
> @return: see the API Code Documentation for details.

- **int iden_joint_friction(int *result, unsigned char *sn = NULL)**

> Identification the friction
>
> Note: only available if firmware_version >= 1.9.0
>
> @param result: the result of identification
>
> 0: success
>
> -1: failure
>
> @param sn: robot sn
>
> @return: See the API Code Documentation for details.

- **int open_lite6_gripper(void)**

> Open the gripper of Lite6 series robotics arms
>
> Note:
>
> 1. only available if firmware_version >= 1.10.0
> 2. this API can only be used on Lite6 series robotic arms
>
> @return: See the API Code Documentation for details.

- **int close_lite6_gripper(void)**

> Close the gripper of Lite6 series robotics arms
>
> Note:
>
> 1. only available if firmware_version >= 1.10.0
> 2. this API can only be used on Lite6 series robotic arms
>
> @return: See the API Code Documentation for details.

- **int stop_lite6_gripper(void)**

> Stop the gripper of Lite6 series robotics arms
>
> Note:
>
> 1. only available if firmware_version >= 1.10.0
> 2. this API can only be used on Lite6 series robotic arms
>
> @return: See the API Code Documentation for details.

- **int set_only_check_type(unsigned char only_check_type = 0)**

Set the motion process detection type (valid for all motion interfaces of the current SDK instance)

Note1: only available if firmware_version >= 1.11.100

Note2: This interface is a global configuration item of the current SDK, and affects all motion-related interfaces

Note3: Generally, you only need to call when you don't want to move the robotic arm and only check whether some paths will have self-collision/angle-limit/cartesian-limit/overspeed. motion-related interfaces

Note4: Currently only self-collision/angle-limit/cartesian-limit/overspeed are detected

Note5: If only_check_type is set to be greater than 0, and the return value of calling the motion interface is not 0, you can view `arm.only_check_result` to view the specific error code

Example: (Common scenarios, here is an example of the set_position interface)

1. Check whether the process from point A to point B is normal (no self-collision and overspeed triggered)

1.1 Move to point A

`arm->set_only_check_type(0)`

`code = arm->set_position(A)`

1.2 Check if the process from point A to point B is normal (no self-collision and overspeed triggered)

`arm->set_only_check_type(1)`

`code = arm->set_position(B)`

// If code is not equal to 0, it means that the path does not pass. You can check the specific error code through arm->only_check_result

`arm->set_only_check_type(0)`

2. Check whether the process from point A to point B, C, and D to point E is normal (no self-collision and overspeed are triggered)

2.1 Move to point A

`arm->set_only_check_type(0)`

`code = arm->set_position(A)`

2.2 Check whether the process of point A passing through points B, C, D to point E is normal (no self-collision and overspeed are triggered)

`arm->set_only_check_type(3)`

`code = arm->set_position(B)`

// If code is not equal to 0, it means that the path does not pass. You can check the specific error code through arm->only_check_result

`code = arm->set_position(C)`

// If code is not equal to 0, it means that the path does not pass. You can check the specific error code through arm->only_check_result

`code = arm->set_position(D)`

// If code is not equal to 0, it means that the path does not pass. You can check the specific error code through arm->only_check_result

`code = arm->set_position(E)`

// If code is not equal to 0, it means that the path does not pass. You can check the specific error code through arm->only_check_result

`arm->set_only_check_type(0)`

@param only_check_type: Motion Detection Type

only_check_type == 0: Restore the original function of the motion interface, it will move, the default is 0

only_check_type == 1: Only check the self-collision without moving, take the actual state of the manipulator as the initial planned path, and check whether the path has self-collision (the intermediate state will be updated at this time)

only_check_type == 2: Only check the self-collision without moving, use the intermediate state as the starting planning path, check whether the path has self-collision (the intermediate state will be updated at this time), and restore the intermediate state to the actual state after the end

only_check_type == 3: Only check the self-collision without moving, use the intermediate state as the starting planning path, and check whether the path has self-collision (the intermediate state will be updated at this time)

@return: See the API Code Documentation for details.

- **int get_dh_params(fp32 dh_params[28])**

Stop the gripper of Lite6 series robotics arms

Note:

1. only available if firmware_version >= 2.0.0

@param dh_params: the result of DH parameters

dh_params[0:4]: DH parameters of Joint-1

dh_params[4:8]: DH parameters of Joint-2

...

dh_params[24:28]: DH parameters of Joint-7

@return: See the API Code Documentation for details.

- **int set_dh_params(fp32 dh_params[28], unsigned char flag = 0)**

Set the DH parameters

Note:

1. only available if firmware_version >= 2.0.0
2. this interface is only provided for users who need to use external DH parameters, ordinary users should not try to modify DH parameters.

@param dh_params: DH parameters

@param flag:

0: Use the set DH parameters, but do not write to the configuration file

1: Use the set DH parameters and write to the configuration file

2: Use the set DH parameters and delete the DH parameters of the configuration file

3: Use the default DH parameters, but will not delete the DH parameters of the configuration file

4: Use the default DH parameters and delete the DH parameters of the configuration file

@return: See the API Code Documentation for details.