Python

JAKA (https://www.jaka.com) November 4, 2024 About 46 min

This document introduces the data types and APIs defined in the JAKA SDK (Python). It is primarily intended for software developers creating robotic applications that communicate with virtual or real controllers using Python. It can also be helpful for users who want to understand JAKA robot controller applications.

Precautions

- Running environment:
  - Linux: Python 3.5 32 bits
  - Windows: Python 3.7 64 bits
- Units used to parameters: mm, rad.
- In non-specific code examples, the robot is turned on by default and powered on.
- All code samples in the documentation default to no interference in the robot's workspace.
- The examples in the documentation are all default to the user's Python interpreter being able to find the jkrc modul

Use under Linux

Linux needs to libjakaAPI.so and jkrc.so under the same folder and add the current folder path to the environment variable, export LD_LIBRARY_PATH=/xx/xx/

Use under Linux

Linux needs to libjakaAPI.so and jkrc.so under the same folder and add the current folder path to the environment variable, export LD_LIBRARY_PATH=/xx/xx/

Dynamic library version number interrogate method

To use jkrc properly, it is necessary to include the dynamic library files. Below are the methods to check the version of the dynamic library:

- In Windows: Right-click the .dll file, select "Properties" and check the version information under the "Details" tab.
- In Linux: Enter the command strings libjakaAPI.so | grep jakaAPI_version to query the version information.

API

Basic Operation of Robots

Instantiated robots

Instantiate a robot

```py
RC(ip)
```

- Parameters
  - ip: The robot's IP address needs to be filled in with a string only the correct IP address instantiated object to control the robot.
- Return value
  - Success: Return a robot object
  - Failed: The created object is destroyed
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64") # Return a robot object
```

Login

Connect the robot controller

```py
login()
```

- Return value
  - Success: (0,)
  - Failed: Others

Logout

Disconnect the controller

```py
logout()
```

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64") # Return a robot object
robot.login()  #Connect the robot controller
pass
robot.logout() #Disconnect the robot controller
```

Power on

Power on the robot. There will be about 8 seconds' delay.

```py
power_on()
```

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64")   #Return a robot object
robot.login()     #Login
robot.power_on() #Power on
robot.logout()    #Logout
```

Power off

Power off the robot

```py
power_off()
```

- Return value
  - Success: (0,)
  - Failed: Others

Shut down the robot controller

Shut down the robot's control cabinet

```py
shut_down()
```

- Return value
  - Success: (0,)
  - Failed: Others

Enable the robot

```py
enable_robot()
```

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64")#Return a robot object
robot.login()  #Login
robot.enable_robot()#
robot.logout() #Logout
```

Disable the robot

```py
disable_robot()
```

- Return value
  - Success: (0,)
  - Failed: Others

Get SDK version

```py
get_sdk_version()
```

- Return value
  - Success: (0,version)
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64")#
robot.login()  #
ret = robot.get_sdk_version()
print("SDK version  is:",ret[1])
robot.logout()  #Logout
```

Get the controller IP

```py
get_controller_ip ()
```

- Return value
  - Success: (0, ip_list), ip_list: indicates the controller IP list. If the controller name is a specific value, the IP address of the controller corresponding to the controller name is returned. If the controller name is empty, all the IP address of the controllers in the network segment is returned.

- Failed: Others

Enable drag mode

```py
drag_mode_enable(enable)
```

- Parameters
  - enable: TRUE means to enter the drag mode, FALSE means to quit the drag mode
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc
import time
#Coordinate system
COORD_BASE  = 0
COORD_JOINT = 1
COORD_TOOL  = 2
#Motion mode
ABS = 0
INCR= 1
robot = jkrc.RC("192.168.2.160")
robot.login()
robot.power_on()
robot.enable_robot()
robot.drag_mode_enable(True)
ret = robot.is_in_drag_mode()
print(ret)
a = input()
robot.drag_mode_enable(False)
ret = robot.is_in_drag_mode()
print(ret)
robot.logout()
```

Check whether in drag mode

To check if the robot is in drag mode

```py
is_in_drag_mode()
```

- Return value
  - Success: (0,state) state is equal to 1: the robot is in drag mode. state is equal to 0: the robot is not in drag mode.
  - Failed: Others

Set joint friction compensation coefficient

This interface is only compatible with controller of version 1.7.2

To set the joint friction compensation coefficient of joints.

```py
set_drag_friction_compensation_gain(joint, gain)
```

- Parameters
  - joint The number of joints
  - gain The friction compensation coefficient, with a range of 0-200. 200 means to compensate 200% of friction.
- Return Value
  - Success: (0,)
  - Failed: Others

Get joint friction compensation coefficient

This interface is only compatible with controller of version 1.7.2

To get the joint friction compensation coefficient of joints.

```py
get_drag_friction_compensation_gain()
```

- Return value
  - Success: (0,list), list is a list of six elements, representing 6 axises.
  - Failed: Others.

Set whether to turn on the debug mode

```py
set_debug_mode(mode)
```

- Parameters
  - mode Select TRUE to enter the debug mode. At this time, debugging information will be output in the standard output stream. Select False, not enter.
- Return value
  - Success: (0,)
  - Failed: Others

Set SDK log filepath

```py
set_SDK_filepath(filepath)
```

- Parameters
  - filepath： filepath of the log
- Return value
  - Success: (0,)
  - Failed: Others

Robot Motion

JOG

To control the robot to move in manual mode.

```py
jog(aj_num, move_mode, coord_type, jog_vel, pos_cmd)
```

- Parameters
  - aj_num： Represent joint number [0-5] in joint space, and x, y, z, rx, ry, rz-axis in Cartesian space.
  - move_mode： 0 is absolute move, 1 is incremental move, 2 is continuous move.

  - coord_type： Robot move coordinate frame, tool coordinate frame, base coordinate frame (current world/user coordinate frame) or joint space.
  - jog_vel： Command velocity, unit of rotation axis or joint move is rad/s, move axis unit is mm/s. The direction of the velocity determines the direction of the movement.
  - pos_cmd： Command position, unit of rotation axis or joint move is rad, move axis unit is mm. When move_mode is absolute move, parameters can be ignored.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code JOG in joint space

```py
    # -*- coding: utf-8 -*-
    import sys
    sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
    import time
    import jkrc

    #coordinate system
    COORD_BASE  = 0
    COORD_JOINT = 1
    COORD_TOOL  = 2
    #motion mode
    ABS = 0
    INCR= 1
    #joint 1-6 is correspond to 0-5

    robot = jkrc.RC("192.168.2.194")#return robot
    robot.login()#login
    robot.power_on() #power on
    robot.enable_robot()
    print("move1")
    robot.jog(0,INCR,COORD_JOINT,30,90)
    time.sleep(5)#jog is non-blocking instruction, and if jog instruction
received in motion state will be discarded
    print("move2")
    robot.jog(0,INCR,COORD_JOINT,5,-90)
    time.sleep(3)
    robot.jog_stop()
    robot.logout()
```

JOG in Cartesian space

```py
import jkrc
import time
COORD_BASE = 0    # base coordinate system
COORD_JOINT = 1   # joint space
COORD_TOOL = 2    # tool coordinate system
ABS = 0             # absolute motion
INCR = 1            # incremental motion
cart_x = 0          #x direction
cart_y = 1          #y direction
cart_z = 2          #z direction
cart_rx = 3         #rx direction
cart_ry = 4         #ry direction
cart_rz = 5         #rz direction
robot = jkrc.RC("192.168.2.64")#return a robot
robot.login()     #login
robot.jog(aj_num = cart_z,move_mode = INCR,coord_type = COORD_BASE,jog_vel = 5,pos_cmd = 10)   # move 10 mm in z+ direction
robot.jog_stop()
robot.logout()     #logout
```

> **Note:**
>
> If the robot is approaching a singular posture or joint limit, it will not be able to jog the robot using the above example code.

Control the robot to stop in JOG

Control the robot to stop JOG mode.

```py
jog_stop(joint_num)
```

- Parameters
  - joint_num: Robot axis number 0-5, when number is -1, stop all axes.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#Coordinate system
COORD_BASE  = 0
COORD_JOINT = 1
COORD_TOOL  = 2
#Motion mode
ABS = 0
INCR= 1
#Joints 1~6 are correspond to 0~5 in order

robot = jkrc.RC("192.168.2.160")#return a robot object
robot.login()#login
robot.power_on() #power on
robot.enable_robot()
print("move1")
robot.jog(0,INCR,COORD_JOINT,30,PI)
time.sleep(5)#jog is a non-blocking instruction, receiving jog instruction in motion state will be discarded
print("move2")
robot.jog(0,INCR,COORD_JOINT,5,-PI)
time.sleep(0.5)
robot.jog_stop(0)  #Stop after 0.5 seconds of movement, compared with the previous example code
robot.logout()
```

MoveJ

Robot joint move

```py
joint_move(joint_pos, move_mode, is_block, speed)
```

- Parameters
  - joint_pos: Joint move position.
  - move_mode: 0 for absolute move, 1 for incremental move.

- is_block: Set whether the interface is a block interface, TRUE represents a block interface and FALSE represents a non-block interface.Block means there will be no return value until robot completes its move, while non-block means there will be immediately a return value after the interface call is completed.
- speed: Robot joint move speed, unit: rad/s.
- acc: Robot joint acceleration, 90rad/s^2 by default.

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
# import sys
# sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#motion mode
ABS = 0
INCR= 1
joint_pos=[PI,PI/2,0,PI//4,0,0]
robot = jkrc.RC("192.168.2.160")      turn a robot
robot.login()#login
robot.power_on() #power on
robot.enable_robot()
print("move1")
robot.joint_move(joint_pos,ABS,True,1)
time.sleep(3)
robot.logout()
```

Robot extension joint move

Robot extension joint move. Add joint move angular acceleration and endpoint error.

```py
joint_move_extend(joint_pos, move_mode, is_block, speed, acc, tol)
```

- Parameters
  - joint_pos: Angle of each target joint of robot joint move.

  - move_mode: Specified move mode, 0 for absolute movement, 1 for incremental move, 2 for continuous move.
  - is_block: whether a block interface, True means block interface, False means non-block interface.
  - speed: Robot joint move speed, unit: rad/s.
  - acc: Robot joint move angular acceleration.
  - tol: Robot joint move endpoint error.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code:

```py
import jkrc                           #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.joint_move_extend(joint_pos=[1, 1, 1, 1, 1, 1],move_mode=0,
is_block=True, speed=20, acc=5, tol=0.1)
robot.joint_move_extend(joint_pos=[-1, 1, 1, 1, 1, 0],move_mode=0,
is_block=True, speed=20, acc=5, tol=0.1)
robot.logout() #logout
```

Robot end linear move

```py
linear_move(end_pos, move_mode, is_block, speed)
```

- Parameters
  - end_pos: Robot end move target position
  - move_mode: 0 for absolute move, 1for incremental move
  - is_block: Set whether the interface is a block interface, TRUE represents a block interface and FALSE represents a non-block interface. Block means there will be no return value until robot completes its move, while non-block means there will be immediately a return value after the interface call is completed.
  - speed: Robot linear move speed, unit: mm/s.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import time
import jkrc

#motion mode
ABS = 0
INCR= 1
tcp_pos=[0,0,-30,0,0,0]
robot = jkrc.RC("192.168.2.160")#return a robot
robot.login()#login
robot.power_on() #power on
robot.enable_robot()
print("move1")
#Blocking 30mm movement at 10mm/s in the negative direction of z-axis
ret=robot.linear_move(tcp_pos,INCR,True,10)
print(ret[0])
time.sleep(3)
robot.logout()
```

> **Note:**
>
> Due to the difference on robot models, the calculation of kine inverse of the pose value filled in the example above [illegible]d fail, thus Return value being -4.

Robot extension end linear move

Robot extension end linear move. Add space acceleration and space motion endpoint error.

```py
linear_move_extend(end_pos, move_mode, is_block, speed, acc, tol)
```

- Parameters
  - end_pos: Robot end move target position.
  - move_mode: Specify the movement mode, 0 is absolute movement, 1 is incremental movement.
  - is_block: Whether a block interface, True means block interface, False means non-block interface.
  - speed: Robot Cartesian space motion speed, unit: mm/s.
  - acc： Robot Cartesian space move acceleration, unit: mm/s^2.

  - tol： Robot joint move endpoint error.
- Return value
  - Success: (0,)
  - Failed: Others

Robot end arc movement

```py
circular_move (end_pos, mid_pos, move_mode, is_block, speed, acc, tol)
```

- Parameters
  - end_pos: Robot end move target position.
  - mid_pos: Middle point of robot end move.
  - move_mode: Specified move mode, 0 for absolute movement, 1 for incremental move, 2 for continuous move.
  - is_block: Whether a block interface, True means block interface, False means non-block interface.
  - speed: Robot joint move speed, unit: mm/s.
  - acc： Robot joint move angular acceleration, unit： mm/s^2
  - tol： Robot joint move endpoint error.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                          #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.circular_move(end_pos = [100, 100, 0, 0, 1, 1],mid_pos = [100, 0, 0, 0, 1, 0],move_mode= 0, is_block = True, speed = 20, acc = 5, tol = 0.1)
robot.logout() #log out
```

```py
circular_move_extend (end_pos, mid_pos, move_mode, is_block, speed, acc, tol, cricle_cnt, opt_cond)
```

- Parameters
  - end_pos: Arc move at the end of the robot.
  - mid_pos: The middle point of robot end move.

- move_mode: Specify move mode, 0 is absolute move, 1 is incremental move, 2 is continuous move.
- is_block: Set whether the interface is a block interface, TRUE represents a block interface and FALSE represents a non-block interface.
- speed: Speed of robot MoveL (unit: mm/s)
- acc: Angular acceleration of robot MoveL (unit: mm/s^2)
- tol: Robot motion endpoint error.
- circle_cnt: Circle number of MoveC.
- opt_cond: Placeholder, enter none.

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc
import math
import traceback


_ABS = 0
_BLOCK = 1


try:
    rc = jkrc.RC("192.168.20.139")
    rc.login()
    rc.power_on()
    rc.enable_robot()

    start_pos = [-406.250, 116.250, 374.000, PI, 0,PI/2]
    mid_pos = [-468.650, 211.450, 374.000, PI, 0,PI/2]
    end_pos = [-295.050, 267.450, 374.000, PI, 0, PI/2]

    rc.joint_move([0] + [PI * 0.5] * 3 + [PI * -0.5, 0], 0, 1, 200)
    rc.linear_move(start_pos, _ABS, _BLOCK, 50)
    rc.circular_move_extend(end_pos, mid_pos, _ABS, _BLOCK, 250, 1200, 5, None,
5)
except Exception:
    traceback.print_exc()
```

Set robot block wait timeout

```py
set_block_wait_timeout (seconds)
```

- Parameters Seconds $>$0.5
- Return value
  - Success: (0,)
  - Failed: Others

Motion abort

To terminate the robot's motion in any situation

```py
motion_abort ()
```

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
    # -*- coding: utf-8 -*-
    import sys
    #sys.path.append('D:\\vs2019ws\PythonCtt\lib')
    import time
    import jkrc
    robot = jkrc.RC("192.168.2.160")#return a robot
    robot.login()#login
    robot.power_on() #power on
    robot.enable_robot()
    print("move1")
    robot.joint_move(joint_pos=
[1,0,0,0,0,0],move_mode=1,is_block=False,speed=0.05)#incremental motion
```

```
    print("wait")
    time.sleep(2)
    print("stop")
    robot.motion_abort()
    robot.logout()#log out
```

Get the robot's motion status

```py
get_motion_status()
```

- Return value
  - Success: (0, data),
    - int motion_line; ///< the executing motion cmd id
    - int motion_line_sdk; ///< reserved
    - BOOL inpos; ///< previous motion cmd is done, should alway check queue info together
    - BOOL err_add_line; ///< previous motion cmd is dropped by controller, like target is already reached
    - int queue; ///< motion cmd number in buffer
    - int active_queue; ///< motion cmd number in blending buffer
    - BOOL queue_full; ///< motic uffer is full and motion cmds reveived at this moment will be dropped
    - BOOL paused; ///< motion cmd is paused and able to resume
  - Failed: Others

To check if the robot is in position

```py
is_in_pos()
```

- Return value
  - Success: (0, state), state is 1 means location reached, 0 means not.
  - Failed: Others

Set and Interrogate Robot Operation Information

Get robot status（simple）

```py
get_robot_status_simple()
```

- Return value
  - Success: (0,data)
    - errcode: error code
    - errmsg: error message
    - powered_on: whether the robot is powered on
    - enabled: whether the robot is enabled
  - Failed: Others

Get the monitoring data of the robot status

```py
get_robot_status ()
```

- Return value
  - Success: (0, robotstatus), where the length of robotstatus data is 24, and the data are returned in an order as shown below:
    - errcode the error code in case of a robot error, with 0 representing normal operation, and others representing abnormal operation
    - inpos whether the robot is in position, 0 means robot still not move to position, 1 means robot has been moved to position
    - powered_on whether the robot is powered on, 0 means not powered on, 1 means powered on
    - enabled whether the robot is enabled, 0 means not enabled, 1 means enabled
    - rapidrate robot speed rate
    - protective_stop whether the robot detects a collision, 0 means no collision detected, 1 means the opposite
    - drag_status whether the robot is in drag status, 0 means not in drag status, 1 means the opposite

    - on_soft_limit whether the robot is on limit, 0 means not triggered limit protection, 1 means triggered limit protection
    - current_user_id the current user coordinate frame ID used by the robot
    - current_tool_id the current tool coordinate frame ID used by the robot
    - dout robot control cabinet digital output signal
    - din robot control cabinet digital input signal
    - aout robot control cabinet analog output signal
    - ain robot control cabinet analog input signal
    - tio_dout robot end tool digital output signal
    - tio_din robot end tool digital input signal
    - tio_ain robot end tool analog input signal
    - extio robot external extension IO module signal
    - cart_position robot end Cartesian space position value
    - joint_position robot joint space position
    - robot_monitor_data robot status monitor data (scb major version, scb minor version, controller temperature, robot average voltage, robot average current, monitor data of 6 robot joints (transient current, voltage and temperature))
    - torq_sensor_monitor_data robot torque sensor status monitor data (torque sensor IP addresses and port numbers, tool payload (payload mass, centroid x-axis, y-axis and z-axis coordinates), torque sensor status and exception error code, actual contact force values of 6 torque sensors and original reading values of 6 torque sensors)
    - is_socket_connect whether the connection channel of SDK and controller is normal, 0 means abnormal connection channel, 1 means normal connection channel
    - emergency_stop whether the robot is emergency stopped, 0 means not pressed the emergency-stop button, 1 means the opposite
    - tio_key Robot end tool button [0] free; [1] point; [2] end light button
  - Failed: Others

> **提示:**
>
> If the robot has no corresponding IO, it will return such a string as "no extio".

```py
import sys
import jkrc
import time

robot = jkrc.RC("192.168.2.160")
```

```
robot.login()
ret = robot.get_robot_status()
if ret[0] == 0:
    print("the joint position is :",ret[1])
    print(len(ret[1]))
    for i in range(len(ret[1])):
        print(ret[1][i])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()
```

Set robot status data update time interval

Set the robot status data update time interval to change the system occupied storage, the default is to send as quick as possible to ensure the data is the newest (4ms).

```py
set_status_data_update_time_interval (millisecond)
```

- Parameters
  - millisecond: time Parameters, unit: ms。
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                              #import modules
robot = jkrc.RC("192.168.2.165")  #return to a robot
robot.login()
robot.set_status_data_update_time_interval(100)
robot.logout()
```

Get robot DH parameters

Get the robot's user coordinate system information

```py
get_dh_param()
```

- Return value

  - Success: (0, {'alpha': (0.0, 1.5707963, 0.0, 0.0, 1.5707963, -1.5707963), 'a': (0.0, 0.0, 897.0, 744.5, 0.0, 0.0), 'd': (196.5, 0.0, 0.0, -188.35000610351562, 138.5, 120.5), 'joint_homeoff': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)})
    - DH Parameter Explanation

Parameter Name	**Type**	**Description**	**Unit**
`alpha` (α)	Link Twist	The angle between adjacent Z-axes (rotation around the X-axis)	Radian (rad)
`a`	Link Length	The length of the link along the X-axis (perpendicular distance between adjacent Z-axes)	Millimeter (mm)
`d`	Link Offset	The translation distance along the Z-axis (offset between adjacent X-axes)	Millimeter (mm)
`joint_homeoff`	Joint Home Offset	The zero-position calibration offset for the joint (correcting the robot arm's zero point)	Radian (rad)

  - Failure: Others

Get joint position

Get the position of the robot's 6 joints

```py
get_joint_position()
```

- Return value
  - Success: (0, joint_pos), joint_pos is a tuple of six elements (j1, j2, j3, j4, j5, j6), j1, j2,j3, j4, j5, j6 the Angle values from joint 1 to joint 6 respectively.
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc

robot = jkrc.RC("192.168.2.160")#return a robot
ret = robot.login()#login
ret = robot.get_joint_position()
if ret[0] == 0:
    print("the joint position is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()  #logout
```

Get TCP pose

```py
get_tcp_position()
```

- Return value
  - Success: (0, cartesian_pose), cartesian_pose is a tuple of six elements. (x, y, z, rx ,ry, rz), x, y, z, rx, ry, rz the pose value of the robot tool end.
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc

robot = jkrc.RC("192.168.2.160")#return a robot
ret = robot.login()#login
ret = robot.get_tcp_position()
if ret[0] == 0:
    print("the tcp position is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()  #logout
```

Set user coordinate system information

```py
set_user_frame_data(id, user_frame, name)
```

- Parameters
  - id： The value range of the user coordinate system number is [1,10], 0 for robot-based coordinate system
  - user_frame： Offset value of user coordinate system [x,y,z,rx,ry,rz]
  - name： Alias of user coordinate system
- Return value
  - Success: (0,)
  - Failed: Others

Get user coordinate system information

```py
get_user_frame_data(id)
```

- Parameters
  - id id of the user coordinate system
- Return value
  - Success: (0, id, tcp), id: The value range of the user coordinate frame number is [1,10], where 0 represents robot basic coordinate frame. tcp: Offset value of user coordinate frame[x,y,z,rx,ry,rz]
  - Failed: Others

Get user coordinate system ID

```py
get_user_frame_id()
```

- Return value
  - Success: (0, id). The value range of the id is [0,10], where 0 represents the world coordinate system.
  - Failed: Others

Set user coordinate system ID

```py
set_user_frame_id(id)
```

- Parameters
  - id: user coordinate system ID
- Return value
  - Success: (0,)
  - Failed: Others

Get the tool ID currently in use

```py
get_tool_id()
```

- Return value
  - Success: (0, id) where the ID values range from 0 to 10, with 0 representing the end flange, which has been used by troller.
  - Failed: Others
- Sample code：

```py
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc
robot = jkrc.RC("192.168.2.160") #return a robot
robot.login()                        #login
ret = robot.get_tool_data(1)    #get tool coordinate system data
if ret[0] == 0:
    print("the tool data is :",ret)
else:
    print("some things happend,the errcode is: ",ret[0])
robot.set_tool_data(1,[0,0,1,0,0,0],'testlx') #set tool coordinate system data
time.sleep(0.5)
ret = robot.get_tool_data(1)    #get tool coordinate system data
if ret[0] == 0:
    print("the tool data is :",ret)
else:
    print("some things happend,the errcode is: ",ret[0])
ret = robot.get_tool_id()     #get tool coordinate system ID
print("tool_id",ret)
robot.set_tool_id(1)          #set tool coordinate system data
time.sleep(0.5)
ret = robot.get_tool_id()     #get tool coordinate system ID
print("tool_id",ret)
robot.logout()
```

Set the tool information for the specified number

```py
set_tool_data(id, tcp, name)
```

- Parameters
  - id： Set the tool ID, optional ID is 1 to 10, 0 means end flange, already used by controller.
  - tcp： Set tool coordinate system parameters [x,y,z,rx,ry,rz].
  - name： Alias of user coordinate system
- Return value
  - Success: (0,)
  - Failed: Others

Set tool ID currently in use

```py
set_tool_id(id)
```

- Parameters
  - id: tool coordinate system ID
- Return value
  - Success: (0,)
  - Failed: Others

Get target tool coordinate system ID

```py
get_tool_data(id)
```

- Return value
  - Success: (0, id, tcp), where the ID values range from 0 to 10, with 0 representing the end flange, which has been used by controller.tcp: tool coordinate frame parameter [x,y,z,rx,ry,rz]
  - Failed: Others

Set digital output value

```py
set_digital_output(iotype = a_type,  index = a_number,  value = a_value)
```

- Parameters
  - iotype： DO type
  - index： DO index
  - value： DO value
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code： Set the value of DO3 to 1

```py
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc
IO_CABINET  = 0  #control cabinet panel IO
IO_TOOL = 1     #tool IO
IO_EXTEND = 2   #extension IO

robot = jkrc.RC("192.168.2.160")
robot.login()
ret = robot.get_digital_output(0,2)
if ret[0] == 0:
    print("1the DO2 is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.set_digital_output(IO_CABINET, 2, 1)#set the pin output value of DO2 to 1
time.sleep(0.1)
ret = robot.get_digital_output(0, 2)
if ret[0] == 0:
    print("2the DO2 is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()  #logout
```

Set analog output value

```py
set_analog_output(iotype = a_type,  index = a_number,  value = a_value)
```

- Parameters
  - iotype： AO type
  - index： AO index
  - value： AO value
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code： set the value of AO4 to 1.55

```py
import jkrc
IO_CABINET  =0  #control cabinet panel IO
IO_TOOL = 1     #tool IO
IO_EXTEND = 2   #extension IO
robot = jkrc.RC("192.168.2.64")
robot.login()
robot.set_analog_output(iotype = IO_CABINET,index = 3,value = 1.55)#
robot.logout()  #logout
```

Set digital output in motion

```py
set_motion_digital_output(iotype = a_type,  index = a_number,  value = a_value,
rel = a_rel, distance = a_distance)
```

- Parameters
  - iotype: DO type
  - index: DO index (starting from 0)
  - value: DO value
  - rel
    - 0 means the starting point of movement
    - 1 means the ending point of movement
  - distance means according to the rel parameters, the distance compared to the staring point and ending point to trigger DO
- Return value
  - Success: (0,)
  - Failed: Others

### Set analog output in motion

```py
set_motion_analog_output(iotype = a_type,  index = a_number,  value = a_value,
rel = a_rel, distance = a_distance)
```

- Parameters
  - iotype: AO type
  - index: AO index (starting from ‿,
  - value: AO value
  - rel
    - 0 means the starting point of movement
    - 1 means the ending point of movement
  - distance means according to the rel parameters, the distance compared to the staring point and ending point to trigger AO
- Return value
  - Success: (0,)
  - Failed: Others

Get digital input status

```py
get_digital_input(iotype, index)
```

- Parameters
  - iotype: DI type

  - index: DI index
- Return value
  - Success: (0, value), value: result of DI status
  - Failed: Others

Get digital output status

```py
get_digital_output(iotype, index)
```

- Parameters
  - iotype: DO type
  - index: DO index
- Return value
  - Success: (0, value), value: result of DO status
  - Failed: Others

Get analog input status

```py
get_analog_input(iotype, index)
```

- Parameters
  - iotype： AI type
  - index： AI index
- Return value
  - Success: (0, value), value: the result of AI status, expressed as a floating point
  - Failed: Others

Get analog output

```py
get_analog_output(type, index)
```

- Parameters
  - type： AO type
  - index： AO index
- Return value
  - Success: (0, value), value: the result of AO status, expressed as a floating point

  - Failed: Others
- Sample code：查询AO4的值为

```py
import jkrc
IO_CABINET  =0  #control cabinet panel IO
IO_TOOL = 1     #tool IO
IO_EXTEND = 2   #extension IO
robot = jkrc.RC("192.168.2.64")
robot.login()
robot.set_analog_output(iotype = IO_CABINET,index = 3,value = 1.55)
ret = robot.get_analog_output(iotype = IO_CABINET,index = 3)
print("AO value is:",ret[1])
robot.logout()  #logout
```

Whether the extension IO is running

```py
is_extio_running()
```

- Return value
  - Success: (0, status), status: 1 m running, 0 means not running
  - Failed: Others

Set robot payload

```py
set_payload(mass = m, centroid = [x,y,z])
```

- Parameters
  - mass： payload mass, unit: kg
  - centroid： payload centroid coordinates [x, y ,z], unit: mm
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                                #import module
robot = jkrc.RC("192.168.2.226")      #return a robot
robot.login()
robot.set_payload(mass= 1, centroid =[0.01,0.02,0.03])
robot.logout()              #logout
```

Get robot payload data

```py
get_payload()
```

- Return value
  - Success: (0, payload)The expression of payload is (mass, (x, y, z)). Payload is a tuple,whose length is 2.The first element of tuple is the mass of payload.The second element of tuple is the centroid of payload.
  - Failed: Others
- Sample code：

```py
import jkrc
robot = jkrc.RC("192.168.2.64")     :urn a robot
robot.login()                        #login
robot.set_payload(mass= 1,centroid =[0.01,0.02,0.03])
ret = robot.get_payload()
if ret[0] == 0:
    print("the payload is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()
```

Set TIO V3 voltage parameters

```py
set_tio_vout_param(vout_enable,vout_vol)
```

- Parameters
  - vout_enable 0 means off, 1 means on
  - vout_vol voltage volume 0:24v 1:12v
- Return value

- Success: (0)
- Failed: Others

Get TIO V3 Voltage Parameters

```py
get_tio_vout_param(vout_enable,vout_vol)
```

- Parameters
  - vout_enable 0 means off, 1 means on
  - vout_vol voltage volume 0:24v 1:12v
- Return value
  - Success: (0,(vout_enable,vout_vol))
  - Failed: Others

Get robot state

```py
get_robot_state()
```

- Return value
  - Success: (0,(estoped, power_on, servo_enabled))
    - estoped： emergency stop. 0: off, 1: on
    - power_on： power on. 0: off, 1: on
    - servo_enabled： servo enable. 0: off, 1: on
  - Failed: Others
- Sample code：

```py
def example_get_robot_state():
    _RC_ADDRESS = "192.168.2.64"
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret, (estoped, power_on, servo_enabled) = rc.get_robot_state()
    print('ret is {}, estop: {}, power_on {}, servo_enabled: {}'.format(ret,
estoped, power_on, servo_enabled))
```

Add or modify TIO signal

```py
add_tio_rs_signal(sign_info)
```

- Parameters
  - sign_info： dict signal name
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
def example_add_tio_rs_signal():
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.add_tio_rs_signal({
        'sig_name': 'signal_tmp',  //signal name
        'chn_id': 0,  //RS485 chanel ID
        'sig_type': 0,  //semapl     type
        'sig_addr': 0x1,  //reg      address
        'value': 5,  //value  invalid when setting
        'frequency': 5  //The refresh frequency of semaphore in the controller
is not more than 10.
    })
```

Delete TIO signal

```py
del_tio_rs_signal(sign_name)
```

- Parameters
  - sign_info: str Signal identification name
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
def example_del_tio_rs_signal():
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.del_tio_rs_signal('signal_tmp')
    print('ret is {}'.format(ret))
```

Send TIO RS485 command

```py
send_tio_rs_command(chn_id, cmd)
```

- Parameters
  - chn_id: int channel number
  - data: str data bit
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
def example_send_tio_rs_command(
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.send_tio_rs_command(0x1, "test_cmd")
    print('ret is {}'.format(ret))
```

Update TIO signal

```py
update_tio_rs_signal(sign_info)
```

- Parameters
  - sign_info: dict the attribute of signal, a dictionary-type data
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
def example_update_tio_rs_signal():
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.update_tio_rs_signal({
        'sig_name': 'signal_tmp',  //signal name
        'frequency': 5  //the update frequency of signal in controller is less
than 20.
    })
```

Get TIO signal information

```py
get_rs485_signal_info()
```

- Return value
  - Success: (0,sign_info_list) sign_info_list: list semaphore information array
  - Failed: Others
- Sample code

```py
def example_get_rs485_signal_inf
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret, sign_info_list = rc.get_rs485_signal_info()
    print('ret is: {}, sign_info_list: {}'.format(ret, sign_info_list))
    # [{'value': 0, 'chn_id': 0, 'sig_addr': 0, 'sig_name': '', 'sig_type': 0,
'frequency': 0}, ...]  -> 4.3.26TIO add or modify semaphore
```

Set TIO mode

```py
set_tio_pin_mode(pin_type, pin_mode)
```

- Parameters
  - pin_type: tio type. 0 for DI Pins, 1 for DO Pins, 2 for AI Pins
  - pin_mode: tio mode.
    - DI Pins: 0:0x00 DI2 is NPN, DI1 is NPN, 1:0x01 DI2 is NPN, DI1 is PNP, 2:0x10 DI2 is PNP, DI1 is NPN, 3:0x11 DI2 is PNP, DI1 is PNP

    - DO Pins: The lower 8 bits of data and the upper 4 bits are configured as DO2, The lower four bits are DO1 configuration, 0x0 DO is NPN output, 0x1 DO is PNP output, 0x2 DO is push-pull output, 0xF RS485H interface
    - AI Pins: Analog input function is enabled, RS485L is disabled, 1:RS485L interface is enabled and analog input function is disabled.
- Return value
  - Success: (0,)
  - Failed: Others

Get TIO mode

```py
get_tio_pin_mode(pin_type)
```

- Parameters pin_type: tio type. 0 for DI Pins, 1 for DO Pins, 2 for AI Pins.
- Return value
  - Success: (0,pin_mode)
  - Failed: Others

Set TIO RS485 Communication Parameters

```py
set_rs485_chn_comm(chn_id, slave_id, baudrate, databit, stopbit, parity)
```

- Parameters
  - chn_id: The RS485 channel ID, used as an input parameter when querying the channel ID.
  - slaveId: When the channel mode is set to Modbus RTU, you need to specify the Modbus slave node ID. This is not required for other modes.
  - baudrate: Baud rate options include 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400.
  - databit: Data bits: 7, 8
  - stopbit: Stop bits: 1, 2
  - parity: Parity: 78→no parity 79→odd parity 69→even parity.

Get TIO RS485 Communication Parameters

```py
get_rs485_chn_comm(chn_id)
```

- Return value
  - Success: (0, (chn_id, slave_id,baudrate, databit, stopbit, parity))
  - Failed: Others

Set TIO RS485 communication mode

```py
set_rs485_chn_mode(chn_id, chn_mode)
```

- Parameters
  - chn_id 0: RS485H, channel 1; 1:RS485L, channel 2
  - chn_mode: 0: Modbus RTU, 1: Raw RS485, 2, torque sensor

Get TIO RS485 communication mode

```py
get_rs485_chn_mode(chn_id)
```

- Parameters
  - chn_id 0: RS485H, channel 1; 1:RS485L, channel 2
- Return value
  - Success: (0, chn_mode)
    - chn_mode: 0: Modbus RTU, 1: Raw RS485, 2, torque sensor
  - Failed: Others

Set robot mounting angle

```py
set_installation_angle(anglex, angley)
```

- Parameters
  - anglex： the mounting angle is as the x-axis direction, range [0, PI] rad
  - anglez： the mounting angle is as the z-axis direction, range [0, 360] rad
- Return value
  - Success: (0)
  - Failed: Others
- Sample code：

```py
import jkrc
rc = jkrc.RC("192.168.137.152")
res = rc.login()
if res[0] != 0:
    raise "rc login failed."

anglex = 3.14
angley = 0
res = rc.set_installation_angle(anglex, angley)
if res[0] != 0:
    raise "set installation angle failed."

res = rc.get_installation_angle()
if res[0] != 0:
    raise "get installation angle failed."

print("installation angle:")
print("quat: [{x}, {y}, {z}, {s}]".format(s=res[1][0], x=res[1][1], y=res[1]
[2], z=res[1][3]))
print("rpy: [{rx}, {ry}, {rz}]".format(rx=res[1][4], ry=res[1][5], rz=res[1]
[6]))

rc.logout()
```

Get robot mounting angle

```py
get_installation_angle()
```

- Parameters
  - anglex： the mounting angle is as the x-axis direction, range [0, PI] rad
  - anglez： the mounting angle is as the z-axis direction, range [0, 360] rad
- Return value
  - Success: (0, [qs, qx, qy, qz, rx, ry, rz])。 Quaternion representing the mounting angle [qx, qy, qz, qs], Installation angles in Roll-Pitch-Yaw (RPY) format [rx, ry, rz], ry is fixed to be 0.
  - Failed: Others。
- Sample code

py
```python
import jkrc
rc = jkrc.RC("192.168.137.152")
res = rc.login()
if res[0] != 0:
    raise "rc login failed."

anglex = 3.14
angley = 0
res = rc.set_installation_angle(anglex, angley)
if res[0] != 0:
    raise "set installation angle failed."

res = rc.get_installation_angle()
if res[0] != 0:
    raise "get installation angle failed."

print("installation angle:")
print("quat: [{x}, {y}, {z}, {s}]".format(s=res[1][0], x=res[1][1], y=res[1]
[2], z=res[1][3]))
print("rpy: [{rx}, {ry}, {rz}]".format(rx=res[1][4], ry=res[1][5], rz=res[1]
[6]))

rc.logout()
```

Set user variables

py
```python
demo.set_user_var(id, value, name)
```

- Parameters
  - id： system variable ID
  - value: system variable value
  - name： system variable name
- Return value
  - Success: (0,)
  - Failed: Others

Get user variable

```py
get_user_var
```

- Return value
  - Success: (0, data)
  - Failed: Others

Robot safety status settings

To inquire robot status

```py
is_on_limit()
```

- Return value
  - Success: (0, state), state为1代表 器人超出限位, 0则相反
  - Failed: Others

To inquire whether in collision protection mode

```py
is_in_collision()
```

- Return value
  - Success: (0, state), state: 1 means in, 0 means not.
  - Failed: Others

To recover from the collision protection mode

```py
collision_recover()
```

- Return value

  - Success: (0,)
  - Failed: Others
- Sample code

```py
from typing import Counter
import jkrc
import time


robot = jkrc.RC("192.168.2.160")
robot.login()
robot.power_on()
robot.enable_robot()
ret = robot.get_collision_level()#get the collision protection level
print(ret)
robot.set_collision_level(1)#set collision protection level
ret = robot.get_collision_level()
print(ret)
num = 0
while(1):
    ret = robot.is_in_collision()  #To inquire whether in collision protection
mode
    collision_status = ret[1]
    if collision_status == 1:
        time.sleep(5)
        robot.collision_recover()   #to recover from collision protection mode
if collision happened
        print(" in collision "+ str(num))
    else:
        print("the robot is not in collision "+ str(num))
    time.sleep(1)
    num=num+1

robot.logout()
```

Set collision level

```py
set_collision_level(level)
```

- Parameters

- level: the range of collision value is [0,5],
  - 0: close collision,
  - 1: collision threshold 25N,
  - 2: collision threshold 50N,
  - 3: collision threshold 75N,
  - 4: collision threshold 100N,
  - 5: collision threshold 125N

- Return value
  - Success: (0,)
  - Failed: Others

Get collision level

```py
get_collision_level()
```

- Return value
  - Success: level: the collision level.
    - 0: close collision,
    - 1: collision threshold 25N,
    - 2: collision threshold 50N,
    - 3: collision threshold 75N,
    - 4: collision threshold 100N,
    - 5: collision threshold 125N
  - Failed: Others

Get the last error code

Get the last error code in the robot running process, when clear_error is called, the last error code will be cleared. If you need to use the get_last_error interface, set the error code file path. If you just need to get the error code, there is no need to call set_errorcode_file_path.

```py
get_last_error()
```

- Return value
  - Success: (0, error)
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.194")#return a robot
robot.login()   #login
robot.program_load("not_exist") #Intentionally load a non-existent program, causing an error
ret = robot.get_last_error ()#Without setting the error code file path, you can only get the error code, not the specific error information
print(ret[1])
robot.set_errorcode_file_path("D:\\JAKA_ERROR_CODE.csv") #The error file path cannot contain Chinese
ret = robot.get_last_error ()   #Get the error code and specific error information after setting error code file path
robot.logout()  #logout
```

Set error code file path

Set the error code file path. If you need to use the get_last_error interface, set the error code file path. If no need to use the get_last_error interface, do not set the interface.

> **Note**
>
> Note: The path can not contain any Chinese characters, otherwise it will not work.

```py
set_errorcode_file_path (errcode_file_path)
```

- Parameters
  - errcode_file_path: where the error code file is stored.
- Return value
  - Success: (0,)
  - Failed: Others

Clear error status

```py
clear_error()
```

- Return value
  - Success: (0,)
  - Failed: Others

Set the robot automatic move termination types upon network exceptions

Set the network exception control handle to control the robot motion status upon network exceptions.

```py
set_network_exception_handle(millisecond,mnt)
```

- Parameters
  - millisecond: time parameters, unit： ms。
  - mnt: the motion types the robot needs to perform upon network exceptions. 0 means that the robot should keep its current move, 1 means that the robot should pause its move and 2 means the robot should stop its move.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
# import sys
# sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#motion mode
ABS = 0
INCR= 1
robot = jkrc.RC("192.168.2.160")#return a robot
```

```
robot.login()#login
robot.power_on() #power on
robot.enable_robot()
robot.set_network_exception_handle(100,2)#set 100ms, pause motion
print("move1")
num=0
while(1):
    robot.joint_move([1,1,1,1,1,1],ABS,False,0.5)
    robot.joint_move([-1,1,1,1,1,1],ABS,False,0.5)
    num = num +1
    print(num)
    time.sleep(6)
robot.logout()
```

Use App Script Program

Load the specified program

```py
program_load(file_name)
```

- Parameters
  - file_name： program name
- Return value
  - Success: (0,)
  - Failed: Others

Get the loaded program

```py
get_loaded_program()
```

- Return value
  - Success: (0, file_name)
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64")#return a robot
robot.login()  #login
ret = robot.program_load("program_test")#load the program_test script written
through the app
ret = robot.get_loaded_program()
print("the loaded program is:",ret[1])
robot.logout() #logout
```

Get current line

```py
get_current_line()
```

- Return value
  - Success: (0, curr_line), curr_line: the current line number.
  - Failed: Others

Run the loaded program

```py
program_run()
```

- Return value
  - Success: (0,)
  - Failed: Others

Pause the running program

```py
program_pause()
```

- Return value
  - Success: (0,)
  - Failed: Others

Resume the program

```py
program_resume()
```

- Return value
  - Success: (0,)
  - Failed: Others

Abort the program

```py
program_abort()
```

- Return value
  - Success: (0,)
  - Failed: Others

Get the program state

```py
get_program_state()
```

- Return value
  - Success: (0, state), state:
    - 0 stopped
    - 1 running
    - 2 paused
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc
import _thread
```

```
def print_state(name,robot):
    while(1):
        ret = robot.get_program_state() #Interrogate the program running status, 0 represents for Program terminated or no program running, 1 represents for Program running, 2 represents for Pause
        print("the robot program state is:",ret[1])
        time.sleep(1)

robot = jkrc.RC("192.168.2.160")#return a robot
robot.login()  #login
ret = robot.program_load("simple")#Load the script program_test written through the app
ret = robot.get_loaded_program()
print("the loaded program is:",ret[1])
robot.program_run()
_thread.start_new_thread( print_state,("p1_state", robot))#Open a "p1" thread to interrogate the program status
time.sleep(10)
robot.program_pause() #pause
time.sleep(10)
robot.program_resume() #resume
time.sleep(10)
robot.program_abort()   #stop
time.sleep(3)
robot.logout()            #logout
```

Set rapid rate

```py
set_rapidrate(rapid_rate)
```

- Parameters
  - rapid_rate： set robot rapid rate
- Return value
  - Success: (0,)
  - Failed: Others

Get rapid rate

```py
get_rapidrate()
```

- Return value
  - Success: (0, rapidrate), rapid rate is the speed rate, and Returned value is within a closed interval between 0 and 1
  - Failed: Others

Trajectory Reproduction

Set trajectory configuration

In setting the trajectory track configuration parameters, you can set the space position sample accuracy, pose sample accuracy, script execution running speed and script execution running acceleration.

```py
set_traj_config(xyz_interval, rpy_interval, vel, acc)
```

- Return value
  - xyz_interval： Space position acquisition speed
  - rpy_interval： Orientation capture accuracy
  - vel： Script execution running speed
  - acc： Script execution running acceleration
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc
import time

#coordinate system
COORD_BASE  = 0
COORD_JOINT = 1
```

```
COORD_TOOL  = 2
#motion mode
ABS = 0
INCR= 1

robot = jkrc.RC("192.168.2.160")
robot.login()
robot.power_on()
robot.enable_robot()
robot.joint_move(joint_pos =[1,1,1,1,1,1] ,move_mode = 0 ,is_block = True
,speed = 10 )
print("joint")
robot.set_traj_config([0.1, 0.1, 25, 100]) #Set track recurrence parameters,
only the recording process is valid
time.sleep(0.1)
ret = robot.get_traj_config()#get trajectory recurrence parameters
print("traj_config:")
print(ret)
robot.set_traj_sample_mode(True, 'pythonTrack')#enable trajectory recurrence
capture
time.sleep(0.1)
robot.joint_move(joint_pos =[-1,1,1,1,1,1] ,move_mode = 0 ,is_block = True
,speed = 30*3.14/180 )#blocking motion
robot.joint_move(joint_pos =[1,[illegible],1,1] ,move_mode = 0 ,is_block = True
,speed = 30*3.14/180 )
# robot.jog(2,INCR,COORD_BASE,10,-2)
# robot.jog(2,INCR,COORD_BASE,10,2)
robot.set_traj_sample_mode(False, 'pythonTrack')#disable trajectory recurrence
capture
time.sleep(1)
res = robot.generate_traj_exe_file('pythonTrack')#convert the captured
trajectory files into scripts
print(res)
robot.program_load("track/pythonTrack")#loading trajectory programs
time.sleep(0.1)
robot.program_run()
```

Get trajectory configuration

By getting the trajectory track configuration parameters, you can get the space position sample accuracy, pose sample accuracy, script execution running speed, and script execution running acceleration.

```py
get_traj_config()
```

- Return value
  - Success: (0 , (xyz_interval, rpy_interval, vel, acc))
    - xyz_interval： Space position acquisition speed
    - rpy_interval： Orientation capture accuracy
    - vel： Script execution running speed
    - acc： Script execution running acceleration
  - Failed: Others

Set trajectory sample mode

```py
set_traj_sample_mode(mode, filename)
```

- Parameters
  - mode： Control mode, True means starting data collection, False means ending data collection.
  - filename： The name of the file where data are stored.
- Return value
  - Success: (0,)
  - Failed: Others

Get trajectory sample state

Tip：

It is not allowed to turn on the data collection switch again during the data collection process.

```py
get_traj_sample_status()
```

- Return value
  - Success: (0, sample_status), sample_status： Data status, True means data are being collected, False means data collection has been finished.
  - Failed: Others

Get exist trajectory file name

```py
get_exist_traj_file_name ()
```

- Return value
  - Success: (0,)
  - Failed: Others

Rename exist trajectory file name

```py
rename_traj_file_name (src, dest)
```

- Parameters
  - src： source file name
  - dest： target file name, the length of the file name cannot exceed 100 characters, the file name cannot be empty, the target file name does not support Chinese.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                                 #import module
robot = jkrc.RC("192.168.2.226")       #return a robot
robot.login()
robot.rename_traj_file_name('/home/src', '/home/dst')
robot.logout() #logout
```

Remove the trajectory file in the controller

```py
remove_traj_file(filename)
```

- Parameters
  - filename： The name of the file to be deleted

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                             #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.remove_traj_file('test')
robot.logout() #logout
```

Generate the trajectory execution script

```py
generate_traj_exe_file(filename)
```

- Parameters
  - filename： The filename of data
- Return value
  - Success: (0,)
  - Failed: Others

Robot Kinematics

Convert Rpy to rotation matrix

```py
rpy_to_rot_matrix(rpy = [rx,ry,rz])
```

- Parameters
  - rpy： rpy parameters to be converted[rx,ry,rz]
- Return value
  - Success: (0, rot_matrix), rot_matrix is a 3X3 rotation matirx
  - Failed: Others

Convert rotation matrix to rpy

```py
rot_matrix_to_rpy(rot_matrix)
```

- Parameters
  - rot_matrix: Rot matrix data to be converted
- Return value
  - Success: (0, rpy), rpy is a tuple, whose length is 3. The expression of rpy is (rx, ry, rz).
  - Failed: Others

Convert quaternion to rotation matrix

```py
quaternion_to_rot_matrix (quaternion = [w,x,y,z])
```

- Parameters
  - quaternion： Quaternion data to be converted
- Return value
  - Success: (0, rot_matrix), rot_matrix is a 3*3 rotation matirx
  - Failed: Others

Inverse kinematics

Calculate the kine inverse of the specified pose under the current tool, current installation angle, and current user coordinate system settings.

```py
kine_inverse(ref_pos, cartesian_pose)
```

- Parameters
  - ref_pos： Reference joint position for kine inverse
  - cartesian_pose： Cartesian space pose value.
- Return value
  - Success: (0 , joint_pos) joint_pos is a tuple with 6 elements (j1, j2, j3, j4, j5, j6), J1, J2, J3, J4, J5, and j6 represent the Angle values from joint 1 to joint 6 respectively.
  - Failed: Others

Forward Kinematics

Calculate the pose value of the specified joint position under the current tool, current installation angle and current user coordinate system settings.

```py
kine_forward(joint_pos)
```

- Parameters
  - joint_pos: Joint space position.
- Return value
  - Success: (0, cartesian_pose), Cartesian_pose is a tuple containing 6 elements (x,y,z,rx,ry,rz), with x,y,z,rx,ry,rz representing the pose value of robot tool end.
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.64") #return a robot
robot.login()                            gin
ret = robot.get_joint_position(
if ret[0] == 0:
    print("the joint position is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
joint_pos = ret[1]
robot.kine_forward(joint_pos)   #forward kinematics
robot.logout()  #logout
```

Kine forward expanded

Calculate kine forward of the specified pose in specified under the current tool, current installation angle and current user coordinate frame settings.

```py
kine_forward(joint_pos, tool, userFrame)
```

- Parameters
  - joint_pos Joint space position calculation result when calculation is successful

  - tool The specified tool coordinate system. Using the current tool coordinate system by default if not specified.
  - userFrame The specified user coordinate system. Using the current user coordinate system by default if not specified.
- Return value
  - Success: (0, cartesian_pose)， cartesian_pose is an array of six elements (x, y, z, rx, ry, rz). x, y, z, rx, ry, rz representing the pose value of robot's tool end.
  - Failed: Others

Kine inverse expanded

Calculate kine inverse of the specified pose in specified under the current tool, current installation angle and current user coordinate frame settings.

```py
kine_inverse(ref_pos, cartesian_pose, tool , userFrame)
```

- Parameters
  - ref_pos Reference joint position for kine inverse
  - cartesian_pose Cartesian space pose value
  - joint_pos Joint space position calculation result when calculation is successful
  - tool The specified tool coordin [illegible] system. Using the current tool coordinate system by default if not specified.
  - userFrame The specified user coordinate system. Using the current user coordinate system by default if not specified.
- Return value
  - Success：(0 , joint_pos) joint_pos is an array of six elements (j1, j2, j3, j4, j5, j6)， j1, j2, j3, j4, j5, j6 representing the angle of robot joint 1 to joint 6.
  - Failed: Others

Convert rotation matrix to quaternion

```py
rot_matrix_to_quaternion(rot_matrix)
```

- Parameters
  - rot_matrix：3x3 rot matrix to be converted
- Return value
  - Success: (0, quaternion), quaternion is a tuple, whose length is 4. The expression of quaternion is (w, x, y, z).

  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.160")
robot.login()
ret = robot.get_tcp_position()
print(ret)
rpy = [ret[1][3], ret[1][4], ret[1][5]]#get rpy
ret = robot.rpy_to_rot_matrix(rpy)#convert rpy to rot matrix
print(ret)
rot_matrix = ret[1]#get rot matrix
ret = robot.rot_matrix_to_rpy(rot_matrix)#convert rot matrix to rpy
print(ret)
ret = robot.rot_matrix_to_quaternion(rot_matrix)#convert rot matrix to
quaternion
print(ret)
quaternion = ret[1]
ret = robot.quaternion_to_rot_matrix(quaternion)#convert rot matrix to
quaternion
print(ret)
robot.logout()
```

Robot Servo Mode

Enable robot servo move

```py
servo_move_enable(enable)
```

- Parameters
  - enable: TRUE means to enter the servo move control mode, FALSE means to quit the mode.
- Return value
  - Success: (0,)
  - Failed: Others

Robot servo MoveJ

```py
servo_j(joint_pos, move_mode)
```

Tips:

- Before using this interface the user needs to call servo_move_enable(True) first to enter the position control mode.
- This command is generally used in trajectory planning in university research.
- When users use this mode to control the robot motion, the controller planner will not be involved in the motion interpolation, and the position command will be sent to the servo directly. Therefore users need to do the trajectory planning by themselves. Otherwise the robot motion effectiveness can be poor, like violent shaking, which can not meet the user's expectation.
- Since the control cycle time of the controller is 8ms, it is recommended that the user should send the command with a period of 8ms too, and continuously. There will be no effect if the command is sent only once. In case of a poor network, the command can be sent with a p [illegible] d less than 8ms.
- The upper limit on the Jaka robot joint speed is 180 radrees per second. If the joint speed exceeds this limit due to the joint angle that is sent, this command will then fail. For example, if the joint angle that is sent is [1.5,0.5,0.5,0.5,0.5,0.5] (here the unit is radree) and the sending period is 8ms, thus 1.5/0.008 = 187.5 radrees per second, which exceeds the upper limit on the joint speed. Then the command will be invalid.
- After using this command, the user needs to use servo_move_enable(False) to exit the position control mode.
- There is a big difference between this command and the aforementioned joint_move(), which interpolation is processed by controller, and the user does not need to care about it. When using servo_j command, users need to make trajectory planning in advance. Otherwise the effect will be poor and can not meet the expectation. **If there is no special requirement, it is recommended to use joint_move instead of servo_j on robot joint move control.**

- Parameters
  - joint_pos: target robot joint move position.

- move_mode: specified move mode: 0 for absolute move, 1 for incremental move

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc
ABS = 0           # absolute motion
INCR = 1          # incremental motion
Enable = True
Disable = False
robot = jkrc.RC("192.168.2.160")#return a robot
robot.login()#login
robot.power_on() #power on
robot.enable_robot()
robot.servo_move_enable(Enable)  #enter position control mode
print("enable")
for i in range(200):
    robot.servo_j(joint_pos =[0     ,0,0,0,0,0.001],move_mode = INCR)#
for i in range(200):
    robot.servo_j(joint_pos =[-0.001,0,0,0,0,-0.001],move_mode = INCR)
robot.servo_move_enable(Disable)#exit position control mode
print("disable")
robot.logout()  #logout
```

Robot servo MoveJ extension

Tips:

- Before using this interface the user needs to call servo_move_enable(True) first to enter the position control mode.
- This command is generally used in trajectory planning in university research.
- When users use this mode to control the robot motion, the controller planner will not be involved in the motion interpolation, and the position command will be sent to the servo directly. Therefore users need to do the trajectory planning by themselves. Otherwise the robot motion effectiveness can be poor, like violent shaking, which can not meet the user's expectation.
- Since the control cycle time of the controller is 8ms, it is recommended that the user should send the command with a period of 8ms too, and continuously. There will be no effect if the command is sent only once. In case of a poor network, the command can be sent with a period less than 8ms.
- The upper limit on the Jaka robot joint speed is 180 radrees per second. If the joint speed exceeds this limit due to the joint angle that is sent, this command will then fail. For example, if the jo[illegible]ngle that is sent is [1.5,0.5,0.5,0.5,0.5,0.5] (here the unit is radree) and the sending period is 8ms, thus 1.5/0.008 = 187.5 radrees per second, which exceeds the upper limit on the joint speed. Then the command will be invalid.
- After using this command, the user needs to use servo_move_enable(False) to exit the position control mode.
- There is a big difference between this command and the aforementioned joint_move(), which interpolation is processed by controller, and the user does not need to care about it. When using servo_j command, users need to make trajectory planning in advance. Otherwise the effect will be poor and can not meet the expectation. **If there is no special requirement, it is recommended to use joint_move instead of servo_j on robot joint move control.**

- Parameters
  - joint_pos: target robot joint move position
  - move_mode: specified move mode: 0 for incremental move, 1 for absolute move
  - step_num: multiplying period, servo_j move period is step_num*8ms, where step_num>=1

- Return value
  - Success: (0,)
  - Failed: Others

Robot Cartesian space servo move

Tips:

- Before using this interface the user needs to call servo_move_enable(True) first to enter the position control mode.
- This command is generally used in trajectory planning in university research.
- When users use this mode to control the robot motion, the controller planner will not be involved in the motion interpolation, and after the controller kine inverse is calculated, the position command will be sent to the servo directly. Therefore users need to do the trajectory planning himself. Otherwise the robot motion effectiveness can be poor, like violent shaking, which cannot meet users' expectation.
- Since the control cycle time of the controller is 8ms, it is recommended that the user should send the command with a period of 8ms too, and continuously. There will be no effect if the command is sent only once. In case of a poor network, the command can be sent with a p [illegible] less than 8ms.
- The upper limit on the JAKA robot joint speed is 180 radrees per second. If the joint speed exceeds this limit due to the position that is sent, this command will become invalid.
- After using this command, the user needs to use servo_move_enable(False) to exit the position control mode.
- There is a big difference between this command and the aforementioned linear_move(), which interpolation is processed by controller, and the user does not need to care about it. When using servo_p command, users need to make trajectory planning in advance. Otherwise the effect will be poor and can not meet the expectation. **If there is no special requirement, it is recommended to use linear_move instead of servo_p on robot Cartesian space move control.**

- Parameters
  - cartesian_pose： Target robot Cartesian space move position.
  - move_mode： Specified move mode, 0 for absolute move, 1 for incremental move.
- Return value
  - Success: (0,)
  - Failed: Others

- Sample code

```py
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

PI = 3.14
ABS = 0          # absolute motion
INCR = 1         # incremental motion
Enable = True
Disable = False

robot = jkrc.RC("192.168.2.160")#return a robot
robot.login()#login
robot.power_on() #power on
robot.enable_robot()
joint_pos=[PI/3,PI/3,PI/3,PI/4,PI/4,0]
robot.joint_move(joint_pos,ABS,True,1)
robot.servo_move_enable(Enable)  #enter position control mode
print("enable")
for i in range(200):
    robot.servo_p(end_pos = [1, 0, 0, 0, 0, 0],move_mode = INCR)
for i in range(200):
    robot.servo_p(end_pos = [-1,0, 0, 0, 0, 0],move_mode = INCR)
robot.servo_move_enable(Disable)#exit position control mode
print("disable")
robot.logout()  #logout
```

Robot Cartesian space servo move extension

Tips:

- Before using this interface the user needs to call servo_move_enable(True) first to enter the position control mode.
- This command is generally used in trajectory planning in university research.
- When users use this mode to control the robot motion, the controller planner will not be involved in the motion interpolation, and after the controller kine inverse is calculated, the position command will be sent to the servo directly. Therefore users need to do the trajectory planning himself. Otherwise the robot motion effectiveness can be poor, like violent shaking, which cannot meet users' expectation.
- Since the control cycle time of the controller is 8ms, it is recommended that the user should send the command with a period of 8ms too, and continuously. There will be no effect if the command is sent only once. In case of a poor network, the command can be sent with a period less than 8ms.
- The upper limit on the JAKA robot joint speed is 180 radrees per second. If the joint speed exceeds this limit c o the position that is sent, this command will become invalid.
- (g) After using this command, the user needs to use servo_move_enable(False) to exit the position control mode.
- (h) There is a big difference between this command and the aforementioned linear_move(), which interpolation is processed by controller, and the user does not need to care about it. When using servo_p command, users need to make trajectory planning in advance. Otherwise the effect will be poor and can not meet the expectation. **If there is no special requirement, it is recommended to use linear_move instead of servo_p on robot Cartesian space move control.**

```py
servo_p_extend(cartesian_pose, move_mode, step_num)
```

- Parameters
  - cartesian_pose: Target robot Cartesian space move position.
  - move_mode: Specify the movement mode, 0 is absolute movement, 1 is incremental movement.

  - step_num: multiplying period, servo_p move period is step_num*8ms, where step_num>=1.
- Return value
  - Success: (0,)
  - Failed: Others

No filters in servo mode

It is not allowed to use any filter in the robot SERVO mode. This command cannot be set in the SERVO mode. It can be set only after exiting the SERVO mode.

```py
servo_move_use_none_filter()
```

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                                  #import module
robot = jkrc.RC("192.168.2.226")        eturn a object
robot.login()
robot.servo_move_use_none_filter()
robot.logout() #logout
```

Use joint first-order low pass filter in SERVO mode

Use Robot joint space first-order low-pass filter in SERVO mode. This command cannot be set in the SERVO mode. It can be set only after exiting the SERVO mode.

```py
servo_move_use_joint_LPF(cutoffFreq)
```

- Parameters
  - cutoffFreq: First-order low-pass filter cut-off frequency.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                          #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.servo_move_use_joint_LPF(0.5)
robot.logout() #logout
```

Use joint nonlinear filter in SERVO mode

Use Robot joint space nonlinear filter in SERVO mode. This command can not be set in SERVO mode. It can be set only after exiting the SERVO mode.

```py
servo_move_use_joint_NLF(max_vr, max_ar, max_jr)
```

- Parameters
  - max_vr: The upper limit of Cartesian space orientation change speed (absolute value) °/s
  - max_ar: The upper limit of accelerated speed of Cartesian space orientation change speed (absolute value)°/s^2
  - max_jr: The upper limit value of jerk (absolute value) of Cartesian space orientation change speed °/s^3
- Sample code

```py
import jkrc                          #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.servo_move_use_joint_NLF(max_vr=2, max_ar=2, max_jr=4)
robot.logout() #logout
```

Use Cartesian space nonlinear filter in SERVO mode

Use Cartesian space nonlinear filter in the Robot Servo mode. This command can not be set in the SERVO mode. It can be set only after exiting the SERVO mode.

```py
servo_move_use_carte_NLF(max_vp, max_ap, max_jp, max_vr, max_ar, max_jr)
```

- Parameters

- max_vp: The upper limit (absolute value) of the move command speed in Cartesian space mm/s
- max_ap: The upper limit (absolute value) of the move command acceleration in Cartesian space mm/s^2
- max_jp: The upper limit (absolute value) of the move command accelerated acceleration in Cartesian space mm/s^3
- max_vr: The upper limit of Cartesian space orientation change speed (absolute value) °/s
- max_ar: The upper limit of accelerated speed of Cartesian space orientation change speed (absolute value)°/s^2
- max_jr: The upper limit value of jerk (absolute value) of Cartesian space orientation change speed °/s^3

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                               #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.servo_move_use_carte_NLF(max_vp=2, max_ap=2, max_jp=4,max_vr=2, max_ar=2,
max_jr=4)
robot.logout() #logout
```

Use joint multi-order mean filter in SERVO mode

Use joint space multi-order mean filter in the Robot SERVO mode. This command can not be set in the SERVO mode. It can be set only after exiting the SERVO mode.

```py
servo_move_use_joint_MMF(max_buf, kp, kv, ka)
```

- Parameters
  - max_buf: The buffer size of the mean filter.
  - kp: Acceleration filter factor.
  - kv: Speed filter factor.
  - ka: Position filter factor.
- Return value
  - Success: (0,)

  - Failed: Others
- Sample code

```py
import jkrc                           #import module
robot = jkrc.RC("192.168.2.226")  #return a robot
robot.login()
robot.servo_move_use_joint_MMF(max_buf=20 , kp=0.2 , kv=0.4 ,ka=0.2)
robot.logout() #logout
```

Set the speed foresight parameters in SERVO mode

```py
servo_speed_foresight (max_buf, kp)
```

- Parameters
  - max_buf: The buffer size of the mean filter.
  - kp： Acceleration filter factor.
- Return value
  - Success: (0,)
  - Failed: Others

Force Control Functions

JAKA provides a set of force control interfaces based on torque sensors. Users can utilize these interfaces to implement advanced force control functions as compliance control and more. This enables the realization of complex application scenarios, such as Cartesian space-directed dragging and force-controlled assembly.

But these interfaces require additional configuration of end force sensors.

For better understanding of the content below and the use of related functions, we suggest our users to read User Manual for Force Control Products (https://www.jakarobotics.com/wp-content/uploads/2023/09/JAKA%E5%8A%9B%E6%8E%A7%E4%BA%A7%E5%93%81%E4%BD%BF%E7%94%A8%E6%89%8B%E5%86%8CV3.1.0_ENG.pdf) and (Click to download quick start). Also, in this chapter, **force control** refers to **force compliant control**. **End force sensor**, **Force sensor** or **Sensor** refers to the 1/6 dimensions force/torque sensor mounted at the end of the robot.

Set sensor mode

Set the sensor brand. The number input represents the corresponding sensor brand, with the optional values of 1, 2 and 3, representing the different F/T sensors, respectively.

```py
set_torsensor_brand(sensor_brand)
```

- Parameters
  - sensor_brand： the mode of sensor, with a value range from 1-6 or a set value 10, and needs to be in consistent with the type number engraved on the sensor shell. 10 means the sensor embeded within the robots, which is managed by the system automatically, therefore it does not need to be configured by this interface.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                              #import module
robot = jkrc.RC("192.168.2.165")  #return a robot
robot.login()
robot.set_torsenosr_brand(2)
robot.logout() #logout
```

Get sensor mode

Get the mode of the sensor currently being used.

```py
get_torsensor_brand()
```

- Return value
  - Success: (0, sensor_brand), sensor_brand: different sensor modes.
  - Failed: Others
- Sample code

```py
import jkrc                          #import module
robot = jkrc.RC("192.168.2.165")  #return a robot
robot.login()
ret=robot.get_torsenosr_brand()
if ret[0] == 1:
    print("the sensor_band is", ret[1])
robot.logout() #logout
```

Turn on or turn end force sensor

```py
set_torque_sensor_mode(sensor_mode)
```

- Parameters
  - sensor_mode： The sensor working mode, 0 means turning off, 1 means turning on.
- Return value
  - Success: (0,)
  - Failed: Others

Start to identify the tool e payload

Start to identify the tool end payload, and input a tuple of 6 elements, representing the 6 joint angles of end position.

This command would control the robot to move to the position specified by joint_pos.

```py
start_torq_sensor_payload_identify(joint_pos)
```

- Parameters
  - joint_pos： The last position when the torque sensor is used for automatic payload. Refer to User Manual for Force Control Products (https://www.jakarobotics.com/wp-content/uploads/2023/09/JAKA%E5%8A%9B%E6%8E%A7%E4%BA%A7%E5%93%81%E4%BD%BF%E7%94%A8%E6%89%8B%E5%86%8CV3.1.0_ENG.pdf) and (Click to download quick start) for details.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc
PI = 3.1415926

robot = jkrc.RC("10.5.5.100")#return a robot
ret = robot.login()#login
ret = robot.power_on()
ret = robot.enable_robot()
robot.set_torsenosr_brand(2)
robot.set_torque_sensor_mode(1)
robot.set_compliant_type(1, 1)
print("inint sensor comple")
print("ready to run")
ret = robot.get_joint_position()
joint_pos_origin = ret[1]
joint_pos = ret[1]
print(joint_pos)
joint_pos[3] += PI / 4
if (joint_pos[3] > 265 * PI / 180):
    joint_pos[3] -= 90
joint_pos[4] += PI / 4
if (joint_pos[4] > 320 * PI / 180):
    joint_pos[4] -= 90
joint_pos[5] += PI / 4
if (joint_pos[5] > 265 * PI / 180):
    joint_pos[5] -= PI
print(joint_pos)
ret = robot.start_torq_sensor_payload_identify(joint_pos)
time.sleep(1)
flag = 1
while (1 == flag):
    ret = robot.get_torq_sensor_identify_staus()
    print(ret)
    time.sleep(1)
    flag = ret[1]
print("identy_finish")
ret = robot.get_torq_sensor_payload_identify_result()
print(ret)
ret = robot.set_torq_sensor_payload()
print(ret)
```

```
ret = robot.get_torq_sensor_payload_identify_result()
print(ret)
robot.joint_move(joint_pos_origin,0,1,10)
print("back")
robot.logout()  #logout
```

Get end payload identification status

```py
get_torq_sensor_identify_status()
```

- Parameters
  - identify_status： The identification status, 0 means identification completed, 1 means in process, 2 means failed.
- Return value
  - Success: (0,identify_status)
  - Failed: Others

Get end payload identification result

Get the end load identification resul d input load quality and centroid coordinates.

```py
get_torq_sensor_payload_identify_result()
```

- Parameters
  - mass： payload mass, unit: kg
  - centroid： load centroid coordinates [x,y,z], unit: mm
- Return value
  - Success: (0,)
  - Failed: Others

Set sensor end payload

Set the sensor end load, and input load quality and centroid coordinates.

```py
set_torq_sensor_tool_payload (mass, centroid)
```

- Parameters

- mass： payload mass, unit: kg
- centroid： load centroid coordinates [x,y,z], unit: mm

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                                 #import module
robot = jkrc.RC("192.168.2.226")       #return a robot
robot.login()
robot.set_torq_sensor_tool_payload(mass= 1, centroid =[10,10,0])
robot.logout() #logout
```

Get sensor end payload

Get sensor end payload and centroid coordinates.

```py
get_torq_sensor_tool_payload ()
```

- Return value
  - Success: (0,(mass, centroid))
    - mass： payload mass, unit: kg
    - centroid： load centroid coordinates [x,y,z], unit: mm
  - Failed: Others

Force-control admittance enable control

Note:

This interface is adapted to controller of version 1.7.1. In 1.7.2, this interface will be divided into different interfaces. :::

The sensor needs to be activated and appropriate compliance control parameters should be set. Additionally, it is recommended to perform at least one force sensor zero calibration before enabling tool drive.

```py
enable_admittance_ctrl (enable_flag)
```

- Parameters
  - enable_flag： The flag, 0 means turning off force control drag enable, 1 means turning on.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc


robot = jkrc.RC("10.5.5.100")#return a robot
ret = robot.login()#login
ret = robot.power_on()
ret = robot.enable_robot()
robot.set_torsenosr_brand(2)
robot.set_torque_sensor_mode(1)
robot.set_compliant_type(1, 1)
print("inint sensor comple")
print("ready to run")
#set compliance control parameters
```

```
ret = robot.set_admit_ctrl_config(0, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(1, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(2, 1, 20, 10, 0,0)
ret = robot.set_admit_ctrl_config(3, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(4, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(5, 0, 20, 5, 0, 0)
#Set force control drag, 1 means on and 0 means off
ret = robot.enable_admittance_ctrl(1)
print("enable_admittance_ctrl open！")
print("input any word to quit:")
a = input()
ret = robot.enable_admittance_ctrl(0)
ret = robot.set_admit_ctrl_config(2, 0, 20, 5, 0, 0)
robot.set_torque_sensor_mode(0)
robot.logout()  #lougout
```

Set Force Control Type and Zero Calibration (Initialization)

Note:

This interface is adapted to controller of version 1.7.1. In 1.7.2, this interface will be divided into different ir aces.

When the 1.7.2 controller calls this interface, it is only used to trigger zero calibration and set the force control type. Note that when sensor_compensation is set to 1, the compliance_type parameter must be set to 0. Additionally, you must wait 1 second before calling this interface again to set the compliance_type parameter.

```py
set_compliant_type(sensor_compensation, compliance_type)
```

- Parameters
  - sensor_compensation: 1 indicates that a sensor zero calibration will be performed immediately, and the real-time force curve on the app as well as the values returned from port 10000 for torqsensor[1][2] will switch to represent the actual external force. 0 means no calibration will be performed, and when not in force control mode, the real-time force curve on the app and the values returned from port 10000

for torqsensor[1][2] will switch to the raw sensor readings (if in force control mode, they will still represent the actual external force).
  - compliance_type: The force control type. 0 means not using any kind of compliance control method, 1 means using constant compliance control, and 2 means using speed compliance control.
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code

```py
import jkrc                                   #import module
robot = jkrc.RC("192.168.2.226")      #return a robot
robot.login()
robot.set_compliant_type(1,1)
robot.logout() #logout
```

Get the force control type and sensor initial state

```py
get_compliant_type()
```

- Return value
  - Success: (0, sensor_compensation, compliance_type)
    - sensor_compensation: whether to turn on sensor compensation. 1 means turning on. 0 means no.
    - compliance_type: The force control type. 0 means not using any kind of compliance control, 1 means using constant compliance control, and 2 means using speed compliance control.
  - Failed: Others

Set force control coordinate system

> **Note:**
>
> This interface is adapted to controller of version 1.7.1. In controller version 1.7.2, this interface is divided into different interfaces based on functions. When calling this interface in controller 1.7.2, all its replaced interfaces would be called, and parameters will be set accordingly.

```py
set_ft_ctrl_frame(ftFrame)
```

- Parameters
  - ftFrame 0 means tools, 1 means world
- Return value
  - Success: (0,)
  - Failed: Others

Get force control coordinate system

> **Note:**
>
> This interface is adapted to controller of version 1.7.1. In controller version 1.7.2, this interface is divided into different interfaces based on functions. When calling this interface in controller 1.7.2, it will return the force control coordinate system.

```py
get_ft_ctrl_frame()
```

- Return value
  - Success: (0, ftFrame)
    - ftFrame: 0 means tools, 1 means world
  - Failed: Others

Set compliance control parameters

> **Note:**
>
> This interface is adapted to controller of version 1.7.1. In controller version 1.7.2, this interface is divided into different interfaces based on functions. When calling this interface in controller 1.7.2, all its replaced interfaces would be called, and parameters will be set accordingly.

```py
set_admit_ctrl_config(axis, opt, ftUser, ftReboundFK, ftConstant,
ftNormalTrack)
```

Set parameters such as joint axis numbers, compliance direction, damping force, rebound force, constant force and normal tracking.

- Parameters
  - axis: represents the axis numb Cartesian space, with a value range of 0 to 5, representing the corresponding directions of fx, fy, fz, mx, my and mz, respectively.
  - opt: The compliance direction, 0 for turning off, 1 for turning on.
  - ftUser: The damping force, indicating that how much force the user needs to apply to make the robot move along a certain direction at the maximum speed.
  - ftReboundFK: The rebound force, indicating the ability of the robot to return to its initial state.
  - ftConstant: The constant force, all of which are set to 0 when manual operation.
  - ftNormalTrack: The normal tracking, all of which are set to 0 when manual operation.
  - Users can refer to User Manual for Force Control Products (https://www.jakarobotics.com/wp-content/uploads/2023/09/JAKA%E5%8A%9B%E6%8E%A7%E4%BA%A7%E5%93%81%E4%BD%BF%E7%94%A8%E6%89%8B%E5%86%8CV3.1.0_ENG.pdf) and (Click to download quick start) for detailed descriptions on above parameters.
- Return value
  - Success: (0,)
  - Failed: Others

Get compliance control parameters

> **Note:**
>
> This interface is adapted to controller of version 1.7.1. In controller version 1.7.2, this interface is divided into different interfaces based on functions. When calling this interface in controller 1.7.2, all its replaced interfaces would be called, and parameters will be set accordingly.

By getting the force-control compliance control parameters, you can get the compliance direction, damping force, rebound force, constant force and normal tracking which are corresponding to the 6 joints.

```py
get_admit_ctrl_config()
```

- Return value
  - Success: (0, [[opt, ftUser, ftReb dFK, ftConstant, ftNormanlTrack], ……])
    - opt: The compliance directic.., ..ith an optional value range of 1 to 6, corresponding to the directions of fx, fy fz, mx, my and mz, respectively, and 0 means no check.
    - ftUser: The damping force, indicating that how much force the user needs to apply to make the robot move along a certain direction at the maximum speed.
    - ftReboundFK: The rebound force, indicating the ability of the robot to return to its initial state.
    - ftConstant: The constant force, all of which are set to 0 when manual operation.
    - ftNormalTrack: The normal tracking, all of which are set to 0 when manual operation.
    - An array made of 6 AdmitCtrlType parameters, corresponding to the compliant force control parameters of x, y, z, rx, ry, rz directions.
  - Failed: Others

Set end sensor IP address

```py
set_torque_sensor_comm(type, ip_addr, port)
```

- Parameters
  - type： The sensor type
  - ip_addr： The sensor IP address
  - port： The port number
  - ip_addr and port are invalid when set to 1. Using the default parameters given in example will do fine.
- Return value
  - Success: (0,)
  - Failed: Others

Get end sensor IP address

```py
get_torque_sensor_comm()
```

- Return value
  - Success: (0, type, ip_addr, port),
    - type: 0 means using the network port and USB, 1 means using TIO.
    - ip_addr: the sensor IP address.
    - port: the sensor's port number when using the network port.
    - ip_addr and port are invalid n using TIO or USB. The return value doesn't have a real meaning.
  - Failed: Others

Disable force control

This only takes effect on constant compliant control and speed compliant control, excluding tool drive.

```py
disable_force_control ()
```

- Return value
  - Success: (0,)
  - Failed: Others

Set speed compliance control parameters

> **Note:**
>
> Deprecated in controller of version 1.7.2. Not suggested to use.

Set the speed compliance control parameters. There are 3 speed compliance control levels and 4 rate levels.

```py
set_vel_compliant_ctrl (level, rate1, rate2, rate3, rate4)
```

- Parameters
  - level: compliance control level: 1, 2, 3
  - rate1: rate 1
  - rate2: rate 2
  - rate3: rate 3
  - rate4: rate 4

> **Tips:**
>
> Relations between rate levels: 0<rate4<rate3<rate2<rate1<1;
>
> - When level = 1, only the values of rate1 and rate 2 can be set, and the values of rate 3 and rates 4 are all zero.
> - When level = 2, only the values of rate1, rate 2 and rate 3 can be set, and the value of rate 4 is zero.
> - When level = 3, all the values of rate 1, rate 2, rate 3 and rate 4 can be set.

- Return value
  - Success: (0,)
  - Failed: Others

Set the compliance control torque conditions

> **Note:**
>
> Deprecated in controller of version 1.7.2. Not suggested to use.

```py
set_compliance_condition (fx, fy, fz, tx, ty, tz)
```

- Parameters
  - fx: the force along X-axis, unit: N
  - fy: the force along Y-axis, unit: N
  - fz: the force along Z-axis, unit: N
  - tx: the torque around X-axis, unit: Nm
  - ty: the torque around Y-axis, unit: Nm
  - tz: the torque around Z-axis, unit: Nm
  - Parameters above are torque l[illegible]s of compliant control. Robots would stop once these parameters are being ex[illegible]ed.
- Return value
  - Success: (0,)
  - Failed: Others

Set Cutoff Frenquency of the Torque Sensor Low-pass Filter

Set the value of the low-pass filter for force control.

```py
set_torque_sensor_filter(torque_sensor_filter)
```

- Parameters
  - torque_sensor_filter: low-pass filter parameters, unit： HZ
- Return value
  - Success: (0,)
  - Failed: Others

Get Cutoff Frenquency of the Torque Sensor Low-pass Filter

Get the value of the low-pass filter for force control.

```py
get_torque_sensor_filter()
```

- Return value
  - Success: (0,torque_sensor_filter)
    - torque_sensor_filter: low-pass filter parameters, unit: HZ
  - Failed: Others

Set the soft limit of the sensor

```py
set_torque_sensor_soft_limit(fx, fy, fz, tx, ty, tz)
```

- Parameters
  - fx: force along x-axis, unit: N
  - fy: force along the y-axis, unit: N
  - fz: force along the z-axis, unit: N
  - tx: torque around the x-axis, unit: Nm
  - ty: torque around y-axis, unit: Nm
  - tz: torque around z-axis, unit: Nm
- Return value
  - Success: (0,)
  - Failed: Others

Get the soft limit of the sensor

> **Note:**
>
> Deprecated in controller of version 1.7.2. Not suggested to use.

Set condition of compliance control torque.

```py
get_torque_sensor_soft_limit()
```

- Return value
  - Success: (0,(fx,fy,fz,tx,ty,tz))
    - fx: force along x-axis, unit: N
    - fy: force along the y-axis, unit: N
    - fz: force along the z-axis, unit: N
    - tx: torque around the x-axis, unit: Nm
    - ty: torque around y-axis, unit: Nm
    - tz: torque around z-axis, unit: Nm
  - Failed: Others

> **In this chapter, interfaces below are adapted to controllers of version 1.7.2. If your controller is of this version and above, we suggest you to use this interface.**

Zero calibration of sensor

```py
zero_end_sensor()
```

- Return value
  - Success: (0,)
  - Failed: Others

Get tool drive state

```py
get_tool_drive_state()
```

- Return value
  - Success: (0,enable_flag, drive_state)
    - enable_flag: 0 is to turn off force control dragging, 1 is to turn it on
    - drive_state: whether the current state of dragging triggers singularity point, speed, joint limit warning
  - Failed: Others

Get the force sensor's coordinate system in drag mode

```py
get_tool_drive_frame()
```

- Return value
  - Success: (0,ftFrame)
    - ftFrame: 0 Tool; 1 World
  - Failed: Others

Set the force sensor's coordinate system in drag mode

```py
set_tool_drive_frame(ftFrame)
```

- Parameters
  - ftFrame: 0 Tool; 1 World
- Return value
  - Success: (0,)
  - Failed: Others

Get fusion drive sensitivity

```py
get_fusion_drive_sensitivity_level()
```

- Return value
  - Success: (0,level)
    - level: sensitivity level. value: 0-5, 0 means not turned on
  - Failed: Others

Set fusion drive sensitivity

```py
set_fusion_drive_sensitivity_level(level)
```

- Parameters
  - level: sensitivity level. value: 0-5, 0 means not turned on
- Return value
  - Success: (0,)
  - Failed: Others

Get motion limit (singularity and joint limit) warning range

```py
get_motion_limit_warning_range(warningRange)
```

- Return value
  - Success: (0,warningRange) warningRange: range that will trigger warnings
  - Failed: Others

Set motion limit (singularity and joint limit) warning range

```py
set_motion_limit_warning_range(warningRange)
```

- Parameters
  - warningRange: range that will trigger warnings
- Return value
  - Success: (0,)
  - Failed: Others

Get compliant speed limit

```py
get_compliant_speed_limit(vel, angularvel)
```

- Return value
  - Success: (0,vel,angularvel)
    - vel: linear velocity limit, mm/s
    - angularvel: angular velocity limit, rad/s
  - Failed: Others

Set compliant speed limit

```py
set_compliant_speed_limit(vel, angularvel)
```

- Parameters
  - vel: linear velocity limit, mm/s
  - angularvel: angular velocity limit, rad/s
- Return value
  - Success: (0,)
  - Failed: Others

Get torque reference center

```py
get_torque_ref_point()
```

- Return value
  - Success: (0,refpoint)
    - refpoint: 0 torque sensor center, 1 TCP center
  - Failed: Others

Set torque reference center

```py
set_torque_ref_point(refpoint)
```

- Parameters
  - refpoint: 0 torque sensor center, 1 TCP center
- Return value
  - Success: (0,)
  - Failed: Others

Get end sensor sensitivity

```py
get_end_sensor_sensitivity_threshold ()
```

- Return value
  - Success: (0,data)
    - Fx: force on x axis, unit: N
    - Fy: force on y axis, unit: N
    - Fz: force on z axis, unit: N
    - Tx: torque around x axis, unit: Nm
    - Ty: torque around y axis, unit: Nm
    - Tz: torque around z axis, unit: Nm
  - Failed: Others

Set end sensor sensitivity

```py
set_end_sensor_sensitivity_thres    d (fx, fy, fz, tx, ty, tz)
```

- Parameters
  - Fx: force on x axis, unit: N
  - Fy: force on y axis, unit: N
  - Fz: force on z axis, unit: N
  - Tx: torque around x axis, unit: Nm
  - Ty: torque around y axis, unit: Nm
  - Tz: torque around z axis, unit: Nm An array of six element, the larger the value is, the less sensitive the sensor is.
- Return value
  - Success: (0,data)
  - Failed: Others

FTP Services

Initialize FTP

Initialize the FTP client, establish connection with control cabinet, and import and export `program` and `track`

```py
init_ftp_client()
```

- Return value
  - Success: (0,)
  - Failed: Others

Close FTP

```py
close_ftp_client()
```

- Return value
  - Success: (0,)
  - Failed: Others

Interrogate the directory of controller FTP

```py
get_ftp_dir(remotedir, type)
```

- Parameters
  - remotedir： The controller internal folder name
  - type： 0 for file and folder, 1 for file, 2 for folder
- Return value
  - Success: (0,ret) ret is a string
  - Failed: Others
- Sample code

```py
import jkrc
robot = jkrc.RC("192.168.2.26")
robot.login()
dir= "/program/"
#log in the controller and replace 192.168.2.26 with the IP address of your
controller
robot.init_ftp_client()
result = robot.get_ftp_dir("/program/", 0)
print(result)
robot.close_ftp_client()
robot.logout()
```

Download FTP File

Download the file or folder from Robot control cabinet FTP to your local directory, and Interrogate track "/track/" and script program "/program/"

```py
download_file(local, remote, opt)
```

- Parameters
  - remote: the controller internal filename absolute path. Whether the folder name should be ended with a "" or "/" should depend on the system. For example, a path of "/program/test/test.jks" for a single file, or a path of "/program/test/" for a folder
  - local: the absolute filename path to download the file to local directory
  - opt 1 is single file, 2 is folder
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code： Download the program folder on the ftp to the program folder on the desktop

```py
import jkrc
robot = jkrc.RC("192.168.2.26")#VMmodel
robot.login()
remote = "/program/"
local= "C:\\Users\\Administrator\\Desktop\\program\\track\\"
```

```py
robot.init_ftp_client()
result = robot.download_file(local, remote, 2)
print(result)
robot.close_ftp_client()
robot.logout()
```

Upload files to FTP

Upload files with specified types and names from your local directory to controller.

```py
upload_file(local, remote, opt)
```

- Parameters
  - remote: the absolute filename path to upload the file to controller. Whether a folder name should be ended with a "" or "/" should depend on the system
  - local: the local filename absolute path
  - opt: 1 for single file, 2 for folder
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code： Upload all files and folders from the lxxpro folder on the desktop to the program/folder of the ftp

```py
import jkrc
robot = jkrc.RC("192.168.2.26")#VMmodel
robot.login()
remote = "/program/"
local= "C:\\Users\\Administrator\\Desktop\\lxxpro\\"
robot.init_ftp_client()
result = robot.upload_file(local, remote, 2)
print(result)
robot.close_ftp_client()
robot.logout()
```

Rename FTP files

Rename controller files with specified types and names.

```py
upload_file(local, remote,  opt)
```

- Parameters
  - remote the controller internal filename absolute path. Whether the folder name should be ended with a "" or "/" should depend on the system.
  - des the name to be used as rename
  - opt 1 for single file, 2 for folder. When renaming files, all files in the folder will be renamed to be used by /track/ easily
- Return value
  - Success: (0,)
  - Failed: Others
- Sample code： Rename all files and folders in the ftp's lxxpro folder to lxx

```py
import jkrc

robot = jkrc.RC("192.168.2.26")#VMmodel
robot.login()
remote = "/lxxpro/"
des = "lxx"
robot.init_ftp_client()
result = robot.rename_ftp_file(remote, des, 2)
print(result)
robot.close_ftp_client()
robot.logout()
```

Delte FTP files

Delte the files of specified types and names in the controller.

```py
del_ftp_file(remote, opt)
```

- Parameters

- remote the controller internal filename absolute path. Whether the folder name should be ended with a "" or "/" should depend on the system.
- opt 1 for single file, 2 for folder

- Return value
  - Success: (0,)
  - Failed: Others
- Sample code (**Attention: this sample code would delete all programs. Please use it with caution**)

```py
import jkrc

robot = jkrc.RC("192.168.2.26")#VMmodel
robot.login()
dir= "/program/"
robot.init_ftp_client()
result = robot.del_ftp_file("/program/", 2)
print(result)
robot.close_ftp_client()
robot.logout()
```

Feedback## List of Ir rface Call Return Values and Troubleshooting

Error Code	Description	Suggested Action
0	success	No action needed.
2	Interface error or controller not supported	Verify controller and SDK versions; consider upgrading or using another interface.
-1	Invalid handler	Ensure you call the login interface before using it.
-2	Invalid parameter	Check if the parameters are correct.
-3	Fail to connect	Verify network status or ensure the robot IP is correct.
-4	Kinematic inverse error	Check the current coordinate system or reference joint angles for validity.

Error Code	Description	Suggested Action
-5	Emergency stop	E-stop status; reserved state.
-6	Not powered on	Power the robot.
-7	Not enabled	Enable the robot.
-8	Not in servo mode	Ensure servo mode is enabled before calling `servoJP` .
-9	Must turn off enable before power off	Disable before powering off.
-10	Cannot operate, program is running	Stop the running program first.
-11	Cannot open file, or file doesn't exist	Check if the file exists or can be accessed.
-12	Motion abnormal	Check for singularities or movements exceeding robot limits.
-14	FTP error	Investigate FTP connection issues.
-15	Socket message or va oversize	Ensure parameters are within acceptable limits.
-16	Kinematic forward error	Verify inputs for forward kinematics.
-17	Not support empty folder	Avoid using empty folders.
-20	Protective stop	Investigate and address protective stop conditions.
-21	Emergency stop	Check and reset the emergency stop.
-22	On soft limit	Use the teaching feature in the robot app to exit soft limits.
-30	Fail to encode command string	Likely an error in parsing controller messages.
-31	Fail to decode command string	Check for errors in controller message parsing.
-32	Fail to uncompress port 10004 string	Possible network fluctuation or data overflow.

Error Code	Description	Suggested Action
-40	Move linear error	Verify the path avoids singularity regions.
-41	Move joint error	Ensure joint angles are valid.
-42	Move circular error	Check the circular motion parameters.
-50	Block wait timeout	Verify and increase wait timeout if needed.
-51	Power on timeout	Check the power-on process.
-52	Power off timeout	Check the power-off process.
-53	Enable timeout	Review the enabling process.
-54	Disable timeout	Review the disabling process.
-55	Set user frame timeout	Check the user frame settings.
-56	Set tool timeout	Verify the tool coordinate settings.
-60	Set IO timeout	Ensure IO configurations are correct.

Let me know if you'd like additional ____ils or assistance!

Feedback

For any inaccurate descriptions or errors in the document, we would like to invite the readers to correct and criticize. In case of any questions during your reading or any comments you want to make, please send an email to <support@jaka.com,> and our colleagues will reply.

Last update: 4/27/2025, 8:10:19 AM