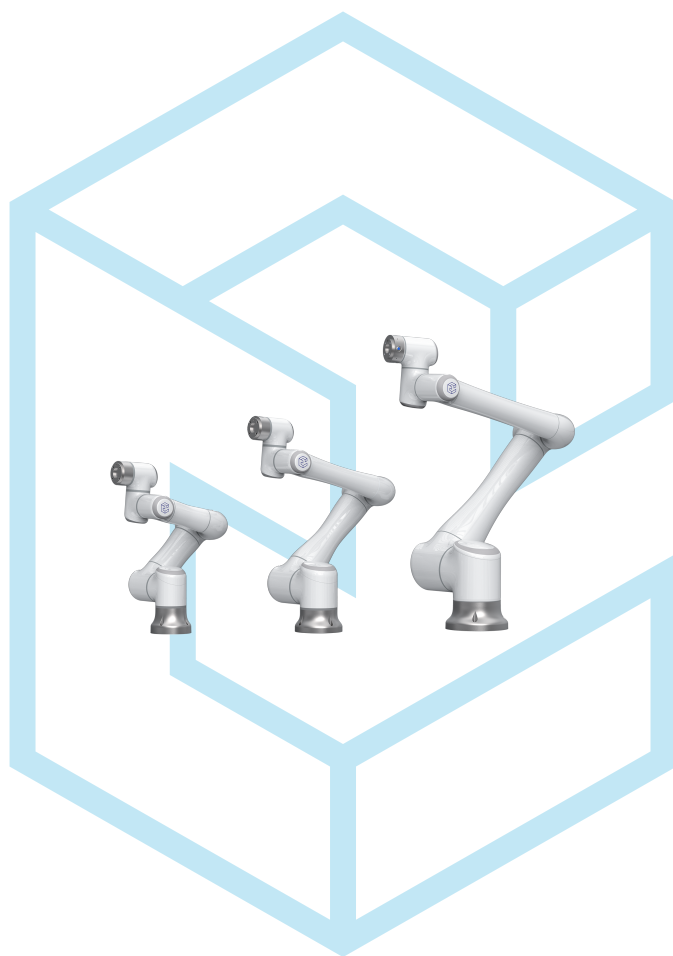


ELITE ROBOTS **EC** Series Programming Manual



SDK-Socket Script Manual

Suzhou Elite Robot Co., Ltd

2022-10-17

Version: Ver3.6.2

Contents

1	Introduction	1
2	Control Interface	2
2.1	Python data processing example	3
2.2	Interface service	5
2.2.1	Servo Service(ServoService)	5
2.2.1.1	Get the servo status of the robotic arm	5
2.2.1.2	Set servo status of robotic arm	5
2.2.1.3	Synchronous Servo Encoder Data	6
2.2.1.4	Clear alarm	6
2.2.1.5	Get synchronization status	7
2.2.2	Parameter Service(ParamService)	7
2.2.2.1	Get robot status	7
2.2.2.2	Get Robot Mode	8
2.2.2.3	Obtain the joint position information of the robot output terminal .	8
2.2.2.4	Get the current pose information of the robot	8
2.2.2.5	Get robot motor speed	9
2.2.2.6	Get the current coordinate system of the robot	10
2.2.2.7	Get robot cycle mode	11
2.2.2.8	Get the current job running line number of the robot	11
2.2.2.9	Get the current encoder value list of the robot	12
2.2.2.10	Get the current tool number of the robot	13
2.2.2.11	Switch the current tool number of the robot	13
2.2.2.12	Get the current user coordinate number of the robot	14
2.2.2.13	Switch the current user coordinate number of the robot	14
2.2.2.14	Get the current torque information of the robot	15
2.2.2.15	Get the current running point number of the trajectory	15

2.2.2.16	Specify coordinate system	16
2.2.2.17	Drag the teaching switch	17
2.2.2.18	Set robot arm load and center of gravity	17
2.2.2.19	Set Robotic Arm Tool Center	18
2.2.2.20	Get collision status	19
2.2.2.21	Get user coordinate system data	19
2.2.2.22	Specify cycle mode	20
2.2.2.23	Set user coordinate system data	21
2.2.2.24	Get tool coordinate system data	22
2.2.2.25	Get tool load quality	23
2.2.2.26	Get tool cog	23
2.2.2.27	Get robot type	24
2.2.2.28	Get robot DH parameters	24
2.2.2.29	Set collision enable	25
2.2.2.30	Set collision sensitivity	25
2.2.2.31	Get the automatically generated encrypted string	26
2.2.2.32	Set security parameters	26
2.2.2.33	Get robot running speed	27
2.2.2.34	Clear collision status	28
2.2.2.35	Get the current tool number of the robot in remote mode	28
2.2.2.36	Set the current tool number of the robot in remote mode	29
2.2.2.37	Get the center pose of flange in Base coordinate system	29
2.2.2.38	Get the center pose of the flange in the user coordinate system	30
2.2.2.39	Get robot subtype	31
2.2.2.40	Get security parameter enable status	31
2.2.2.41	Get safe power	32
2.2.2.42	Get Safe Momentum	32
2.2.2.43	Access to safety tools	33
2.2.2.44	Get the "external" force and torque in the current TCP coordinate	33
2.2.2.45	Get the current joint torque	34
2.2.2.46	Get safe elbow force	33
2.2.2.47	Get speed percentage	34

2.2.2.48	Get the maximum starting speed of dragging	34
2.2.2.49	Get the maximum torque error percentage	35
2.2.2.50	Set end button state	35
2.2.2.51	Get end button status	36
2.2.2.52	Get collision detection enable state	36
2.2.2.53	Get collision sensitivity	37
2.2.2.54	Get the pose of the current tcp in the current user coordinate system	37
2.2.2.55	Get the serial number of the robot alarm information	38
2.2.2.56	Get joint motion speed	38
2.2.2.57	Get tcp acceleration	39
2.2.2.58	Get joint acceleration	39
2.2.2.59	Get tcp movement speed	40
2.2.2.60	Get the emergency stop status of the robot	40
2.2.2.61	Get analog output voltage value	41
2.2.2.62	Get tool load and center of mass	42
2.2.2.63	Get joint position information of robot input	42
2.2.2.64	Get the precise state of the robot servo encoder	43
2.2.2.65	Get the state of the robot servo alarm.	43
2.2.2.66	Clear the booking queue.	44
2.2.2.67	Get the actual tcp pose	44
2.2.2.68	Get the target interpolation tcp pose	45
2.2.2.69	Get the actual joint	45
2.2.2.70	Get the target interpolation joint.	46
2.2.2.71	Get the linear interpolation pose.	46
2.2.2.72	Get the joint temperature.	48
2.2.3	Movement Service(MovementService)	48
2.2.3.1	Joint Movement	48
2.2.3.2	Linear motion	49
2.2.3.3	Circular Movement	51
2.2.3.4	Rotation Movement	52

2.2.3.5	Add waypoint info 2.0	53
2.2.3.6	Clear waypoint information 2.0	55
2.2.3.7	Track Movement 2.0	55
2.2.3.8	Jog Movement	56
2.2.3.9	Stop robot operation	57
2.2.3.10	Robot runs automatically	58
2.2.3.11	Robot stop	58
2.2.3.12	Check if the jbi file exists	59
2.2.3.13	Run jbi file	59
2.2.3.14	Get jbi file running status	60
2.2.3.15	Set robot running speed	60
2.2.3.16	Joint motion at uniform speed	61
2.2.3.17	Stop joint motion at uniform speed	62
2.2.3.18	Linear uniform motion	62
2.2.3.19	Stop linear motion at constant speed	63
2.2.3.20	Linear motion under specified coordinate system	64
2.2.3.21	Encoder zero calibration	66
2.2.4	Kinematics Service(KinematicsService)	66
2.2.4.1	Inverse solution function	66
2.2.4.2	Positive solution function	67
2.2.4.3	Base coordinate to user coordinate pose conversion	68
2.2.4.4	User coordinate to base coordinate pose conversion	68
2.2.4.5	Position multiplication	69
2.2.4.6	Pose inversion	70
2.2.5	IO Service(IOService)	71
2.2.5.1	Get input IO status	71
2.2.5.2	Get output IO status	71
2.2.5.3	Set output IO status	72
2.2.5.4	Get virtual input IO status	72
2.2.5.5	Get virtual output IO status	73

2.2.5.6	Set virtual output IO status	73
2.2.5.7	Read multiple M virtual IO	74
2.2.5.8	Get analog input	74
2.2.5.9	Set analog output	75
2.2.6	Variable service(VarService)	75
2.2.6.1	Get system B variable value	75
2.2.6.2	Set system B variable value	76
2.2.6.3	Get system I variable value	76
2.2.6.4	Set system I variable value	77
2.2.6.5	Get system D variable value	77
2.2.6.6	Set system D variable value	78
2.2.6.7	Get whether the system P variable is enabled	78
2.2.6.8	Get the value of P variable	79
2.2.6.9	Set the value of the P variable	79
2.2.6.10	Set the scope of the V variable	80
2.2.6.11	Get the value of V variable	80
2.2.6.12	Set the value of the V variable	80
2.2.6.13	Save variable data	81
2.2.7	Transparent Transmission Service	82
2.2.7.1	Initialize transparent transmission service	82
2.2.7.2	Set the current transparent transmission servo target joint point	82
2.2.7.3	Get whether the current robot is in a transparent transmission state	83
2.2.7.4	Add transparent transmission of servo target joint point information to the cache	83
2.2.7.5	Clear the transparent transmission cache	85
2.2.7.6	Example 1	85
2.2.7.7	Example 2	88
2.2.8	System Service(SystemService)	91
2.2.8.1	Get the controller software version number	91
2.2.8.2	Get servo version number	92

2.2.9	TrajectoryService(TrajectoryService)	92
2.2.9.1	Initial movement	92
2.2.9.2	Add movement points	93
2.2.9.3	Stop adding movement points	94
2.2.9.4	Check execution status	95
2.2.9.5	Start a time-stamped movement	96
2.2.9.6	Pause motion	97
2.2.9.7	Resume motion	98
2.2.9.8	Stop motion	98
2.2.9.9	Empty the cache	99
2.2.9.10	Example 1	99
2.2.9.11	Example 2	101
2.2.10	ProfinetService(ProfinetService)	104
2.2.10.1	Get the value of the profinet int input register	104
2.2.10.2	Get the value of the profinet int output register	105
2.2.10.3	Get the value of the profinet float type input register	106
2.2.10.4	Get the value of the profinet float type output register	106
2.2.10.5	Set the value of the profinet int output register	107
2.2.10.6	Set the value of the profinet float output register	108
2.2.11	Backdrive service	108
2.2.11.1	Get the opening of the servo brake	108
2.2.11.2	Get whether it is in reverse drive mode	109
2.2.11.3	Enter backdrive mode	109
2.2.11.4	Exit backdrive mode	110
2.2.11.5	Reset controller state	110
2.2.12	Ethernet/IP	112
2.2.12.1	Get the value of the Ethernet/IP int input register	112
2.2.12.2	Get the value of the Ethernet/IP int output register	113
2.2.12.3	Get the value of the Ethernet/IP float type input register	114
2.2.12.4	Get the value of the Ethernet/IP float type output register	115

2.2.12.5	Set the value of the Ethernet/IP int output register	116
2.2.12.6	Set the value of the Ethernet/IP float output register	117
2.2.13	External force sensor service	118
2.2.13.1	Mark the start of the torque data transfer	118
2.2.13.2	Transfer the torque data	119
2.2.13.3	Stop the transfer of the current torque data	120
2.2.13.4	Get the source of the current torque data	121
2.2.13.5	Example	122
2.2.14	Force Control Service	125
2.2.14.1	Start the force control mode	125
2.2.14.2	End the force control mode	126
2.2.14.3	Get the state of the force control mode	126
2.3	Examples	127
2.3.1	Example 1	127
2.3.2	Example 2	129
2.3.3	Example 3	132
3	Monitor Interface	135
3.1	Data description list of monitor interface	135
3.2	Example	137
4	Log Interface	145
4.1	Example	145
5	Raw Log Interface	146
5.1	Example	146

Chapter 1 Introduction

CAUTION



The applicable software version of this manual is: Ver3.3.2.

Elite Robot has opened robot controller ports to support users in secondary development, as shown in **Table 1-1** .

Table 1-1 . Controller port

Port Number	Name	Function
8055	Control Interface	receive json string in specified format
8056	Monitor Interface	output robot information

The user can connect to the corresponding controller port through socket communication to perform some operations to realize the corresponding function.

Chapter 2 Control Interface

The user can send a json string in the specified format to the control interface of the controller through socket communication to achieve related functions, as shown below.

```
1  send
2
3  {"jsonrpc":"2.0","method":"method name","params":parameters,"id":id}
4
5  receive
6
7  Normal
8      {"jsonrpc":"2.0","result":"result","id":id}
9  Error
10     {"jsonrpc":"2.0","error":{"code":error code,"message":"error
        message"},"id":id}
```

CAUTION



This function is applicable to 2.13.0 and above versions.

The id when sending the json string is the same as the id when receiving the result, as shown below.

```
1  Send
2      {"jsonrpc":"2.0","method":"cmd_set_payload","params":{"cog"
        : [1,2,3],"tool_num":1,"m":12},"id":1}
3
4      {"jsonrpc":"2.0","method":"checkJbiExist","params":{"filename":"
        123123"},"id":1}
5
6      {"jsonrpc":"2.0","method":"getRobotState","params":[],"id":1}
7
8
9  Receive
10     Normal
11         {"jsonrpc":"2.0","result":"false","id":1}
12
13     Error
14         {"jsonrpc":"2.0","error":{"code":-32601,"message":"Method not
        found."},"id":1}
```

NOTICE

Both sending and receiving end with \n

Currently, there are two common exceptions returned by the json protocol:

JRPC_METHOD_NOT_FOUND -32601, JRPC_INTERNAL_ERROR -32693.

- 32601 means that the corresponding interface is not found. You need to check whether the interface name is correct or whether the current version supports the interface.
- 32693 is an exception defined inside the interface. Corresponding parameters are not found, parameters are out of range, and execution conditions are not met. Such exceptions are reported. Such errors only need to check whether the parameters and their ranges and execution conditions are satisfied according to the error information.

2.1 Python data processing example

The examples in this chapter are all in Python. Users can modify the code according to the examples in this section.

CAUTION

The Python language version must be Python3.

```
1 import socket
2 import json
3 import time
4
5 def connectETController(ip,port=8055):
6     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7     try:
8         sock.connect((ip,port))
9         return (True,sock)
10    except Exception as e:
11        sock.close()
12        return (False,)
13
14 def disconnectETController(sock):
15     if(sock):
```



```
16     sock.close()
17     sock=None
18 else:
19     sock=None
20
21 def sendCMD(sock,cmd,params=None,id=1):
22     if(not params):
23         params=[]
24     else:
25         params=json.dumps(params)
26     sendStr="{\method\": \"{0}\", \"params\": {1}, \"jsonrpc\": \"2.0\", \"id
        \": {2}}".format(cmd,params,id)+"\n"
27     try:
28         sock.sendall(bytes(sendStr,"utf-8"))
29         ret=sock.recv(1024)
30         jdata=json.loads(str(ret,"utf-8"))
31         if("result" in jdata.keys()):
32             return (True,json.loads(jdata["result"]),jdata["id"])
33         elif("error" in jdata.keys()):
34             return (False,jdata["error"],jdata["id"])
35         else:
36             return (False,None,None)
37     except Exception as e:
38         return (False,None,None)
39
40 if __name__ == "__main__":
41     # Robot IP address
42     robot_ip="192.168.1.200"
43     conSuc,sock=connectETController(robot_ip)
44     if(conSuc):
45         # Get robot status
46         suc, result, id = sendCMD(sock, "getRobotState")
47         # Print results
48         print(result)
```

2.2 Interface service

2.2.1 Servo Service(ServoService)

2.2.1.1 Get the servo status of the robotic arm

```
{"jsonrpc": "2.0", "method": "getServoStatus", "id": id}
```

Function: Get the servo status of the robotic arm

Parameter: None

Return: Enable true, not enable false

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc):  
        # Get the servo status of the robotic arm  
        suc, result ,id=sendCMD(sock,"getServoStatus")
```

2.2.1.2 Set servo status of robotic arm

```
{"jsonrpc": "2.0", "method": "set_servo_status", "params": {"status": status}, "  
    id": id}
```

Function: Set servo enable state

Parameter: status: servo switch, range: int[0,1], 1 is on, 0 is off

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc):  
        # Get the servo status of the robotic arm  
        suc, result ,id=sendCMD(sock,"getServoStatus")  
        if (result == 0):  
            # Set the servo status of the robot arm to ON  
            suc, result ,id=sendCMD(sock,"set_servo_status",{"status":1})  
            time.sleep(1)
```

Note: This command is only supported in remote mode.

2.2.1.3 Synchronous Servo Encoder Data

```
{"jsonrpc": "2.0", "method": "syncMotorStatus", "id": id}
```

Function: Synchronize servo encoder data

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get synchronization status
        suc, result, id = sendCMD(sock, "getMotorStatus")
        if(result == 0):
            # Synchronize servo encoder data
            suc, result, id = sendCMD(sock, "syncMotorStatus")
            time.sleep(0.5)
```

Note: This command is only supported in remote mode.

2.2.1.4 Clear alarm

```
{"jsonrpc": "2.0", "method": "clearAlarm", "id": id}
```

Function: Clear alarm

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Clear alarm
        ret, result, id = sendCMD(sock, "clearAlarm")
```

Note: This command is only supported in remote mode.

2.2.1.5 Get synchronization status

```
{"jsonrpc": "2.0", "method": "getMotorStatus", "id": id}
```

Function: Get synchronization status

Parameter: None

Return: True for synchronization, false for unsynchronization

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc):  
        # Get synchronization status  
        suc, result, id = sendCMD(sock, "getMotorStatus")
```

2.2.2 Parameter Service(ParamService)

2.2.2.1 Get robot status

```
{"jsonrpc": "2.0", "method": "getRobotState", "id": id}
```

Function: Get robot status

Parameter: None

Return: Stop state 0, pause state 1, emergency stop state 2, running state 3, alarm state 4, collision state 5

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc):  
        # Get robot status  
        suc, result, id = sendCMD(sock, "getRobotState")
```

Note: The emergency stop status obtained by this instruction will only exist for a short time and will be covered by the alarm soon. If you need to get the emergency stop status, please refer to Subsection 2.2.2.58.

2.2.2.2 Get Robot Mode

```
{"jsonrpc": "2.0", "method": "getRobotMode", "id": id}
```

Function: Get Robot Mode

Parameter: None

Return: Teaching mode 0, operating mode 1, remote mode 2

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get Robot Mode
        suc, result, id = sendCMD(sock, "getRobotMode")
```

2.2.2.3 Obtain the joint position information of the robot output terminal

```
{"jsonrpc": "2.0", "method": "get_joint_pos", "id": id}
```

Function: Obtain the joint position information of the robot output terminal

Parameter: None

Return: The current position information of the robot double pos[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get information about the current position of the robot
        suc, result, id = sendCMD(sock, "get_joint_pos")
```

Note: getRobotPos is deprecated, please use get_joint_pos to obtain the joint position information of the robot output terminal

2.2.2.4 Get the current pose information of the robot

```
{"jsonrpc": "2.0", "method": "get_tcp_pose", "params": {"coordinate_num":
    coordinate_num, "tool_num": tool_num, "unit_type": unit_type}, "id": id}
```


Function: Get the current pose information of the robot

Parameter: coordinate_num: coordinate number; int[-1,7], -1: base coordinate system, 0~7: corresponding user coordinate system
tool_num: tool number: int[-1,7], -1: current tool number, 0~7: corresponding tool number
unit_type: int[0,1], optional parameter, returns the unit type of pose rx, ry, rz, 0: angle, 1: radians, if not written, the default value is radians.

Return: The current pose information of the robot double pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current pose information of the robot
        suc, result ,id=sendCMD(sock,"get_tcp_pose",{ "coordinate_num": 0,"tool_num": 0})
```

Note: If the parameter coordinate_num and the parameter tool_num are used at the same time, the corresponding tool number and the user coordinate in the corresponding user coordinate system are returned; if neither or only the parameter tool_num is used, the robot pose in the base coordinate system is returned. The parameter coordinate_num cannot be used alone.

getRobotPose and getTcpPose are deprecated, please use get_tcp_pose to get the current robot pose.

2.2.2.5 Get robot motor speed

```
{"jsonrpc": "2.0", "method": "get_motor_speed", "id": id}
```

Function: Get robot motor speed

Parameter: None

Return: Robot motor speed double speed[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    point = []
    point.append([0.0065,-103.9938,102.2076,-88.2138,
    90.0000,0.0013])
    point.append([-16.2806,-82.4996,81.9848,-89.4851,
    90.0000,-16.2858])
    point.append([3.7679, -71.7544, 68.7276, -86.9732,
    90.0000, 3.7627])
    point.append([12.8237,-87.3028,87.2361,-89.9333,
    90.0000,12.8185])
    if(conSuc):
        for i in range(0, 4, 1):
            # Joint movement
            suc, result ,id=sendCMD(sock,"moveByJoint",{ "targetPos":point[i ], "speed":30})
            while(True):
                # Get robot motor speed
                suc, result ,id =sendCMD(sock,"get_motor_speed")
                print ( result )
                # Get robot status
                suc, result ,id=sendCMD(sock,"getRobotState")
                if ( result == 0):
                    break
```

Note: getMotorSpeed is deprecated, please use get_motor_speed to obtain the motor speed of the robot.

2.2.2.6 Get the current coordinate system of the robot

```
{"jsonrpc": "2.0", "method": "getCurrentCoord", "id": id}
```

Function: Get the current coordinate system of the robot

Parameter: None

Return: Joint 0, Base 1, Tool 2, User 3, Cylinder 4

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the current coordinate system of the robot
        suc, result, id = sendCMD(sock,"getCurrentCoord")
```

2.2.2.7 Get robot cycle mode

```
{"jsonrpc":"2.0","method":"getCycleMode","id":id}
```

Function: Get robot cycle mode

Parameter: None

Return: Single step 0, single loop 1, continuous loop 2

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get robot cycle mode
        suc, result, id = sendCMD(sock,"getCycleMode")
```

2.2.2.8 Get the current job running line number of the robot

```
{"jsonrpc":"2.0","method":"getCurrentJobLine","id":id}
```

Function: Get the current job running line number of the robot

Parameter: None

Return: JBI line number

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the servo status of the robotic arm
        suc, result ,id=sendCMD(sock,"getServoStatus")
        if( result == 0):
            # Set the servo status of the robot arm to ON
            suc, result ,id=sendCMD(sock,"set_servo_status",{ "status ":1})
            time.sleep(1)
        # Check if the jbi file exists
        suc, result ,id=sendCMD(sock,"checkJbiExist",{ "filename": jbi_filename })
        if(suc and result ==1):
            # Run jbi file
            suc, result ,id=sendCMD(sock,"runJbi",{ "filename": jbi_filename })
            if(suc and result ):
                checkRunning=3
                while(checkRunning==3):
                    # Get jbi file running status
                    suc, result ,id=sendCMD(sock,"getJbiState")
                    checkRunning=result["runState"]
                    # Get the current job running line number of the robot
                    # The line number needs to count the line number of the point information, not
                    # the line number of the teach pendant program
                    suc, result ,id=sendCMD(sock,"getCurrentJobLine")
                    print ( result )
                    time.sleep(0.1)
```

2.2.2.9 Get the current encoder value list of the robot

```
{"jsonrpc": "2.0", "method": "getCurrentEncode", "id": id}
```

Function: Get the current encoder value list of the robot

Parameter:

Return: The current encoder value list of the robot double encode[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current encoder value list of the robot
        suc, result, id = sendCMD(sock,"getCurrentEncode")
```

2.2.2.10 Get the current tool number of the robot

```
{"jsonrpc": "2.0", "method": "getToolNumber", "id": id}
```

Function: Get the current tool number of the robot in the teaching mode

Parameter: None

Return: The current tool number of the robot, range: int[0,7]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current tool number of the robot
        suc, result, id = sendCMD(sock,"getToolNumber")
```

2.2.2.11 Switch the current tool number of the robot

```
{"jsonrpc": "2.0", "method": "setToolNumber", "params": {"tool_num": tool_num}, "id": id}
```

Function: Switch the current tool number

Parameter: tool_num: Tool number, range: int[0,7]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Switch the current tool number of the robot
        suc, result ,id=sendCMD(sock,"setToolNumber",{ "tool_num":7})
        time.sleep(0.5)
        # Get the current tool number of the robot
        suc, result , id = sendCMD(sock,"getToolNumber")
```

Note: This command can only switch the current tool number in teaching mode.

This command is only supported in remote mode.

2.2.2.12 Get the current user coordinate number of the robot

```
{"jsonrpc": "2.0", "method": "getUserNumber", "id": id}
```

Function: Get the current user coordinate numberr of the robot

Parameter: None

Return: Coordinate number of the current user of the robot, range: int[0,7]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the current user coordinate number of the robot
        suc, result , id = sendCMD(sock,"getUserNumber")
```

2.2.2.13 Switch the current user coordinate number of the robot

```
{"jsonrpc": "2.0", "method": "setUserNumber", "params": {"user_num": user_num},
  "id": id}
```

Function: Switch the current user coordinate number of the robot

Parameter: user_num: user coordinate number, range: int[0,7]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc)::
        # Switch the current user coordinate number of the robot
        suc, result ,id=sendCMD(sock,"setUserNumber",{ "user_num":7})
        time.sleep(0.5)
        # Get the current user coordinate number of the robot
        suc, result , id = sendCMD(sock,"getUserNumber")
```

Note: This command is only supported in remote mode.

2.2.2.14 Get the current torque information of the robot

```
{"jsonrpc": "2.0", "method": "get_motor_torque", "id": id}
```

Function: Get the current torque information of the robot

Parameter: None

Return: Robot current torque information double torques[6], The ratio per thousand of the ratedtorque of the joint, unit ‰.

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the current torque information of the robot
        suc, result , id = sendCMD(sock,"get_motor_torque")
```

Note: getRobotTorques is deprecated, please use get_motor_torque to get the current torque information of the robot

2.2.2.15 Get the current running point number of the trajectory

```
{"jsonrpc": "2.0", "method": "getPathPointIndex", "id": id}
```

Function: Get the current running point number of the robot

Parameter: None

Return: Store the serial number of the current running point, -1 is non-waypoint movement

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    C000 = [0.0065,-103.9938,102.2076,-88.2138,
    90.0000,0.0013]
    C001 = [-16.2806,-82.4996,81.9848,-89.4851,
    90.0000,-16.2858]
    C002 = [3.7679, -71.7544, 68.7276, -86.9732,
    90.0000, 3.7627]
    if(conSuc):
        # Clear waypoint information 2.0
        suc, result, id = sendCMD(sock,"clearPathPoint")
        if( result == True):
            # Add waypoint information 2.0
            suc, result, id = sendCMD(sock,"addPathPoint", {"wayPoint": C000,"moveType": 0, "
            speed": 50, " circular_radius ":20})
            suc, result, id = sendCMD(sock,"addPathPoint", {"wayPoint": C001,"moveType":0, "
            speed": 50, " circular_radius ":20})
            suc, result, id = sendCMD(sock,"addPathPoint", {"wayPoint": C002,"moveType": 0, "
            speed": 50, " circular_radius ":0})
            # Trajectory movement 2.0
            suc, result, id = sendCMD(sock,"moveByPath")
            while(True):
                # Get the line number of trackfile file running ( consistent with the line number
                displayed by the teach pendant)
                suc, result, id = sendCMD(sock,"getPathPointIndex")
                print( result )
                # Get robot status
                suc, result, id = sendCMD(sock,"getRobotState")
                if( result == 0):
                    break
```

2.2.2.16 Specify coordinate system

```
{"jsonrpc": "2.0", "method": "setCurrentCoord", "params": {"coord_mode":
    coord_mode}, "id": id}
```


Function: Specify coordinate system

Parameter: coord_mode: coordinate system, range int[0,4]. joint: 0, right angle: 1, tool: 2, user: 3, cylinder: 4

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        for i in range(0, 5, 1)
            # Specify coordinate system
            suc, result ,id=sendCMD(sock,"setCurrentCoord",{ "coord_mode":i})
            time.sleep(0.5)
            # Specify coordinate system
            suc, result ,id=sendCMD(sock,"getCurrentCoord")
            print ( result )
```

Note: This command is only supported in remote mode.

2.2.2.17 Drag the teaching switch

```
{"jsonrpc": "2.0", "method": "drag_teach_switch", "params": {"switch": switch
}, "id": id}
```

Function: Drag the teaching switch

Parameter: switch: switch, range: int[0,1], 0 is off, 1 is on

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Drag the teaching switch
        suc, result ,id=sendCMD(sock,"drag_teach_switch",{ "switch":1})
```

Note: This command is only supported in remote mode.

2.2.2.18 Set robot arm load and center of gravity

```
{"jsonrpc": "2.0", "method": "cmd_set_payload", "params": {"tool_num": tool_num
, "m": m, "point": point}, "id": id} or {"jsonrpc": "2.0", "method": "
cmd_set_payload", "params": {"tool_num": tool_num, "m": m, "cog": cog}, "id":
id}
```

Function: Set robot arm load and center of gravity

Parameter: tool_num: Tool number, range: int[0,7]
m: load weight, unit Kg, double, range: EC63 [0,3.6] EC66 [0,7.2] EC612 [0,14.4]
point or cog: center of gravity, x, y, z, unit mm, range: double[-5000,5000]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Set robot arm load and center of gravity
        suc, result, id = sendCMD(sock,"cmd_set_payload",{"tool_num":0,"m":5,"cog":[10, 20, 30]})
```

Note: Version 2.16.2 and above support the selection of tool number. The value of tool_num is set as the load and center of gravity of the tool.
The parameter point is deprecated.
This command is only supported in remote mode.

2.2.2.19 Set Robotic Arm Tool Center

```
{"jsonrpc": "2.0", "method": "cmd_set_tcp", "params": {"point": point, "tool_num": tool_num, "unit_type": unit_type}, "id": id}
```

Function: Set Robotic Arm Tool Center

Parameter: tool_num: tool number, range: int[0,7]
point: tool center, the first three items are in millimeters, the range: double[-500,500], the last three units: radian, range: double[- π , π] or angle, range: double[-180,180]
unit_type: optional parameter, int[0,1], sets the unit type of rx, ry, rz in the tool center, 0: angle, 1: radian, if not written, the default value is radians.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Set up the robotic arm tool center
        suc, result, id = sendCMD(sock, "cmd_set_tcp", {"point": [10, 0, 0, 30, 0, 0], "tool_num": 1, "unit_type": 0})
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode.

2.2.2.20 Get collision status

```
{"jsonrpc": "2.0", "method": "getCollisionState", "id": id}
```

Function: Get collision status

Parameter: None

Return: Collision: 1, No collision: 0

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.200"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get collision status
        suc, result, id = sendCMD(sock, "getCollisionState")
```

2.2.2.21 Get user coordinate system data

```
{"jsonrpc": "2.0", "method": "getUserFrame", "params": {"user_num": user_num, "unit_type": unit_type}, "id": id}
```

Function: Get user coordinate system data

Parameter: user_num: User coordinate number, range: int[0,7]

unit_type: int[0,1], optional parameter, returns the unit type of pose rx, ry, rz, range int [0,1]

Default: radians, 0: angle, 1: radians

Return: User coordinate system data double pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        for i in range(8):
            suc, result ,id=sendCMD(sock,"getUserFrame",{"user_num":i,"unit_type":1})
            print ("User coordinate number = ",i)
            print ("suc=",suc,"", "id =",id)
            if (suc):
                print ("result =", result )
            else :
                print ("err_msg=", result ["message"])
```

Note: The unit_type parameter is only applicable to v2.15.2 and above.

2.2.2.22 Specify cycle mode

```
{"jsonrpc": "2.0", "method": "setCycleMode", "params": {"cycle_mode": cycle_mode}, "id": id}
```

Function: Specify cycle mode

Parameter: cycle_mode: cycle mode, range: int[0,2] single step: 0, single cycle: 1, continuous cycle: 2

Return: Success true, failure false

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        #Set the loop mode to single loop
        ret , result , id = sendCMD(sock,"setCycleMode",{"cycle_mode":1})
        if ret :
            print ("result =", result )
        else :
            print ("err_msg=", result ["message"])
```

Note: This command is only supported in remote mode.

2.2.2.23 Set user coordinate system data

```
{"jsonrpc":"2.0","method":"setUserFrame","params":{"user_num":user_num,"user_frame":user_frame,"unit_type":unit_type},"id":id}
```

Function: Set user coordinate system data

Parameter: user_num: user number, range int [0,7]
user_frame: user coordinate system data, double user_frame[6], range: [-1e+9,1e+9],
x, y, z unit: millimeter, rx, ry, rz unit: degree/rad
unit_type: the unit type of rx, ry, rz of the user coordinate system, int[0,1], optional
parameters, the unit type of rx, ry, rz, 0: angle, 1: radians, if not written, the default is
radians value.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        user_frame=[499.011212,570.517817,
                    247.082805, -3.141593, -0.000000, -0.773067]
        ret , result ,id=sendCMD(sock,"setUserFrame",{ "user_num":0,"user_frame":user_frame,"
                                                         unit_type":1})
        if ret :
            print ("result =", result )
        else :
            print ("err_msg=", result ["message"])
```

Note: This command is only supported in remote mode.
The unit_type parameter is only applicable to v2.15.2 and above.

2.2.2.24 Get tool coordinate system data

```
{"jsonrpc": "2.0", "method": "getTcpPos", "params": {"tool_num": tool_num, "
                                                    unit_type": unit_type}, "id": id}
```

Function: Get tool coordinate system data

Parameter: tool_num: tool coordinate number, range int [0,7]
unit_type: int[0,1], return the unit type of rx, ry, rz of the coordinate system, 0:angle, 1:radian, optional parameter, if not written, the default value is radians.

Return: Tool coordinate system data double pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        ret, result, id = sendCMD(sock, "getTcpPos", {"tool_num": 0})
        if ret:
            print("result=", result)
        else:
            print("err_msg=", result["message"])
```

2.2.2.25 Get tool load quality

```
{"jsonrpc": "2.0", "method": "getPayload", "params": {"tool_num": tool_num}, "id": id}
```

Function: Get tool load quality

Parameter: tool_num: tool coordinate number, range int [0,7]

Return: Tool load quality, double m

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        ret, result, id = sendCMD(sock, "getPayload", {"tool_num": 0})
        if ret:
            print("result=", result)
        else:
            print("err_msg=", result["message"])
```

Note: This command is deprecated.

2.2.2.26 Get tool cog

```
{"jsonrpc": "2.0", "method": "getCentreMass", "params": {"tool_num": tool_num}, "id": id}
```

Function: Get tool centroid

Parameter: tool_num: tool coordinate number, range int [0,7]

Return: Tool load cog double cog

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        ret, result, id = sendCMD(sock, "getCentreMass", {"tool_num": 0})
        if ret:
            print("result =", result)
        else:
            print("err_msg=", result["message"])
```

Note: This command is deprecated.

2.2.2.27 Get robot type

```
{"jsonrpc": "2.0", "method": "getRobotType", "id": id}
```

Function: Get robot type

Parameter: None

Return: Robot type int 62 (six-axis collaborative robot), 60 (vertical multi-joint series robot), 41 (four-axis rotary joint robot), 40 (palletizing robot), 43 (SCARA robot), 30 (Delta parallel robot)

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        ret, result, id = sendCMD(sock, "getRobotType")
        if ret:
            print("result =", result)
        else:
            print("err_msg=", result["message"])
```

2.2.2.28 Get robot DH parameters

```
{"jsonrpc": "2.0", "method": "getDH", "params": {"index": index}, "id": id}
```


Function: Get robot DH parameters

Parameter: index: range int [0,11], corresponding to connecting rod parameter d1~d12

Return: DH parameters

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the value of connecting rod parameter d1
        ret, result, id = sendCMD(sock, "getDH", {"index":0})
        if ret:
            print("result =", result)
        else:
            print("err_msg=", result["message"])
```

2.2.2.29 Set collision enable

```
{"jsonrpc": "2.0", "method": "setCollisionEnable", "params": {"enable":
    enable}, "id": id}
```

Function: Set collision enable

Parameter: enable:int[0,1], 1: turn on the collision switch, 0: turn off the collision switch

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Turn on the collision switch
        ret, result, id = sendCMD(sock, "setCollisionEnable", {"enable": 1})
        if ret:
            print("result =", result)
        else:
            print("err_msg=", result["message"])
```

Note: This command is deprecated.

2.2.2.30 Set collision sensitivity

```
{"jsonrpc": "2.0", "method": "setCollisionSensitivity", "params": {"value":
    value}, "id": id}
```

Function: Set collision sensitivity

Parameter: value: sensitivity range int [10,100]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Set collision sensitivity to 50%
        ret, result, id = sendCMD(sock,"setCollisionSensitivity", {"value": 50})
        if ret:
            print("result=", result)
        else:
            print("err_msg=", result["message"])
```

Note: This command is only supported in remote mode.
This command is deprecated.

2.2.2.31 Get the automatically generated encrypted string

```
{"jsonrpc": "2.0", "method": "get_remote_sys_password", "id": id}
```

Function: Get the automatically generated encrypted string

Parameter: None

Return: Automatically generated encrypted string

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.200"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret, str, id = sendCMD(sock, "get_remote_sys_password")
        print(str)
    else:
        print("Connection failed")
        disconnectETController(sock)
```

2.2.2.32 Set security parameters

```
{"jsonrpc": "2.0", "method": "setSafetyParams", "params": {"password": password,
    "enable": enable, "mode": mode, "power": power, "momentum": momentum, "
    tool_force": tool_force, "elbow_force": elbow_force, "speed": speed, "
    collision_enable": collision_enable, "collision_sensitivity":
    collision_sensitivity}, "id": id}
```

Function: Set security parameters

Parameter: password: When the remote mode password is not set on the interface: the default password is "123456"; after the remote mode user password is set on the interface: firstly, you need to use "get_remote_sys_password" to obtain the encrypted string generated by the system, and the remote The user password combination of the mode, calculate its MD5 value as the password required for json security parameter setting
enable: safety restriction parameter enable, int[0,1], 1: enable, 0: disable
mode: mode, int[0,1], 0: normal mode, 1: reduced mode
power: power, range: double [80,1500], unit:W
momentum: momentum, range: double [5,90], unit:kg·m/s
tool_force: Tool force, range: double [100,400], unit:N
elbow_force: elbow force: double [100,400], unit:N
speed: speed percentage, double [0-100], unit:%
collision_enable: int[0,1], optional parameter, set the collision detection switch, 0: turn off the collision detection switch, 1: turn on the collision detection switch
collision_sensitivity: optional parameter, set collision detection sensitivity, range int[10,100], unit:%

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret, str1, id = sendCMD(sock, "get_remote_sys_password")
        word = hashlib.md5()
        str2 = "123456"
        word.update(str1.encode("utf8"))
        word.update(str2.encode("utf8"))
        password = word.hexdigest()
        print(password)
        ret, result, id = sendCMD(sock, "setSafetyParams", {"password": password, "enable": 1,
            "mode": 1, "power": 400, "momentum": 90, "tool_force": 400, "elbow_force": 400,
            "speed": 0.5, "collision_enable": 0, "collision_sensitivity": 10})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is supported in remote and play mode.

2.2.2.33 Get robot running speed

```
{"jsonrpc": "2.0", "method": "getSpeed", "id": id}
```

Function: Get the robot's automatic speed

Parameter: no

Return: Automatic speed double

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        ret , result , id = sendCMD(sock, "getSpeed")
        if ret :
            print ("result =", result )
        else :
            print ("err_msg=", result ["message"])
```

2.2.2.34 Clear collision status

```
{"jsonrpc": "2.0", "method": "resetCollisionState", "id": id}
```

Function: Clear collision status

Parameter: None

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        ret , result , id = sendCMD(sock, "resetCollisionState ")
        if ret :
            print ("result =", result )
        else :
            print ("err_msg=", result ["message"])
```

Note: This command is only supported in remote mode.

2.2.2.35 Get the current tool number of the robot in remote mode

```
{"jsonrpc": "2.0", "method": "getAutoRunToolNumber", "id": id}
```

Function: Get the current tool number of the robot in remote mode

Parameter: None

Return: The current tool number of the robot in remote mode, range: int[0,7]

Note: The tool number in automatic mode is the same as the tool number in remote mode.

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    # print(conSuc)
    if conSuc:
        ret, result, id = sendCMD(sock, "getAutoRunToolNumber")
        if ret:
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
```

Note: This command is applicable to v2.14.4 and above.

2.2.2.36 Set the current tool number of the robot in remote mode

```
{"jsonrpc": "2.0", "method": "setAutoRunToolNumber", "params": {"tool_num": tool_num}, "id": id}
```

Function: Set the current tool number of the robot in remote mode

Parameter: tool_num: tool number, range: int[0,7]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        ret, result, id = sendCMD(sock, "setAutoRunToolNumber", {"tool_num": 0})
        if ret:
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.14.4 and above.

2.2.2.37 Get the center pose of flange in Base coordinate system

```
{"jsonrpc": "2.0", "method": "get_base_flange_pose", "params": {"unit_type":  
    unit_type}, "id": id}
```

Function: Get the center pose of the flange in the Base coordinate system

Parameter: unit_type: int[0,1], return the unit type of the pose rx, ry, rz, 0: return angle, 1: return radian, optional parameter, if not written, the default value is radians.

Return: The center pose of the flange in the Base coordinate system, Double[6]

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.202"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc):  
        # Get the center pose of the flange in the Base coordinate system  
        ret, result, id = sendCMD(sock,"get_base_flange_pose",{"unit_type": 0})  
        print("result =", result)  
    else:  
        print("Connection failed")  
    disconnectETController(sock)
```

Note: This command is applicable to v2.14.4 and above.

2.2.2.38 Get the center pose of the flange in the user coordinate system

```
{"jsonrpc": "2.0", "method": "get_user_flange_pose", "params": {"unit_type":  
    unit_type}, "id": id}
```

Function: Get the center pose of the flange in the user coordinate system

Parameter: unit_type: int[0,1], return the unit type of the pose rx, ry, rz, 0: return angle, 1: return radian, optional parameter, if not written, the default value is radians.

Return: Center pose of flange in user coordinate system, Double[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the center pose of the flange in the user coordinate system
        ret , result ,id = sendCMD(sock,"get_user_flange_pose",{ "unit_type": 0})
        print ("result =", result )
    else :
        print ("Connection failed ")
    disconnectETController (sock)
```

Note: This command is applicable to v2.14.4 and above.

2.2.2.39 Get robot subtype

```
{"jsonrpc": "2.0", "method": "getRobotSubtype", "id": id}
```

Function: Get robot subtype

Parameter: None

Return: Robot subtype int

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc,sock=connectETController(ip)
    if conSuc:
        # Get robot subtype
        suc, result ,id=sendCMD(sock,"getRobotSubtype")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.40 Get security parameter enable status

```
{"jsonrpc": "2.0", "method": "getRobotSafetyParamsEnabled", "id": id}
```

Function: Get security parameter enable status

Parameter: None

Return: Close 0, open 1

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get security parameter enable status
        suc, result, id = sendCMD(sock, "getRobotSafetyParamsEnabled")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.41 Get safe power

```
{"jsonrpc": "2.0", "method": "getRobotSafetyPower", "id": id}
```

Function: Get safe power

Parameter: None

Return: The power value in normal mode and reduced mode double

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get robot safety power
        suc, result, id = sendCMD(sock, "getRobotSafetyPower")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.42 Get Safe Momentum

```
{"jsonrpc": "2.0", "method": "getRobotSafetyMomentum", "id": id}
```

Function: Gain safety momentum

Parameter: None

Return: Momentum value double in normal mode and reduced mode

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
```



```
if conSuc:
    # Get robot safety momentum
    suc, result ,id =sendCMD(sock,"getRobotSafetyMomentum")
    print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.43 Access to safety tools

```
{"jsonrpc":"2.0","method":"getRobotSafetyToolForce","id":id}
```

Function: Acquire the power of safety tools

Parameter: None

Return: Tool force double in normal mode and reduced mode

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Acquire robot safety tool power
        suc, result ,id =sendCMD(sock,"getRobotSafetyToolForce")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.44 Get safe elbow force

```
{"jsonrpc":"2.0","method":"get_tcp_force","params":{"ref_tcp":ref_tcp},"id":id}
```

Function: Get the "external" force and torque of the current TCP coordinate system

Parameter: ref_tcp: reference coordinate system, int[0,1], optional parameter, 0: base coordinate system, 1: tcp coordinate system, If not written, the default is tcp coordinate system.

Return: The "external" force and torque of the current tcp double force[6], the first three are external force and the last three are torque

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the force and torque of the current TCP in the base coordinate system
        suc, result , id = sendCMD(sock,"get_tcp_force",{"get_tcp_force":0})
```

```
print ( suc, result, id )
else :
    print ("Connection failed ")
    disconnectETController (sock)
```

Note: The default direction of force and torque is TCP.

2.2.2.45 Get the current joint torque

```
{"jsonrpc": "2.0", "method": "get_joint_torques", "id": id}
```

Function: Get the current joint torque

Parameter: None

Return: The current joint torque double torques[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.248"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the current joint torque of the robot
        suc, result, id = sendCMD(sock, "get_joint_torques")
        print ( suc, result, id )
    else :
        print ("Connection failed ")
        disconnectETController (sock)
```

Note: The joint torque is the motor torque minus the "torque required to drive itself", reflecting the "external" torque.

2.2.2.46 Get safe elbow

```
{"jsonrpc": "2.0", "method": "getRobotSafetyElbowForce", "id": id}
```

Function: Obtain safe elbow force

Parameter: None

Return: Elbow force in normal mode and reduced mode double

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Obtain the robot safety elbow force
        suc, result, id = sendCMD(sock, "getRobotSafetyElbowForce")
```

```
print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.47 Get speed percentage

```
{"jsonrpc": "2.0", "method": "getRobotSpeedPercentage", "id": id}
```

Function: Get the robot's speed percentage

Parameter: None

Return: Speed percentage in normal mode and reduced mode double

Example:

```
if __name__ == "__main__":  
    ip = "192.168.1.202"  
    conSuc, sock = connectETController(ip)  
    if conSuc:  
        # Get robot speed percentage  
        suc, result, id = sendCMD(sock, "getRobotSpeedPercentage")  
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.48 Get the maximum starting speed of dragging

```
{"jsonrpc": "2.0", "method": "getRobotDragStartupMaxSpeed", "id": id}
```

Function: Get the maximum start speed of drag

Parameter: None

Return: The maximum starting speed of the robot during dragging double

Example:

```
if __name__ == "__main__":  
    ip = "192.168.1.202"  
    conSuc, sock = connectETController(ip)  
    if conSuc:  
        # Get the maximum start speed of the drag  
        suc, result, id = sendCMD(sock, "getRobotDragStartupMaxSpeed")  
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.49 Get the maximum torque error percentage

```
{"jsonrpc": "2.0", "method": "getRobotTorqueErrorMaxPercents", "id": id}
```

Function: Get the maximum torque error percentage of the machine

Parameter: None

Return: The maximum torque error percentage in the robot force control parameters double

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get the maximum torque error percentage of the machine
        suc, result, id = sendCMD(sock, "getRobotTorqueErrorMaxPercents")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.50 Set end button state

```
{"jsonrpc": "2.0", "method": "setFlangeButton", "params": {"button_num":
    button_num, "state": state}, "id": id}
```

Function: Set the end button state

Parameter: button_num: button, int[0,1], 0: blue button, 1: green button
state: state, int[0,2], 0: disable, 1: drag, 2: mark

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Set the end button state
        suc, result, id = sendCMD(sock, "setFlangeButton", {"button_num": 0, "state": 1})
        print ( result )
```

Note: This command is only supported in remote mode.
This command is applicable to v2.16.2 and above.

2.2.2.51 Get end button status

```
{"jsonrpc": "2.0", "method": "checkFlangeButton", "params": {"button_num":
    button_num}, "id": id}
```

Function: Get the end button state

Parameter: button_num: button, int[0,1], 0: blue button, 1: green button

Return: Disable 0, drag 1, mark 2

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get the status of the end button
        suc, result, id = sendCMD(sock, "checkFlangeButtonFlangeButton", {"button_num": 0})
        print (suc, result, id)
```

Note: This command is applicable to v2.16.2 and above.

2.2.2.52 Get collision detection enable state

```
{"jsonrpc": "2.0", "method": "get_collision_enable_status", "id": id}
```

Function: Get the status of collision detection

Parameter: None

Return: 0: not enabled, 1: enabled

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get collision detection enable state
        suc, result, id = sendCMD(sock, "get_collision_enable_status")
        print (result)
    else:
        print ("Connection failed")
    disconnectETController(sock)
```

2.2.2.53 Get collision sensitivity

```
{"jsonrpc": "2.0", "method": "getCollisionSensitivity", "id": id}
```

Function: Get collision sensitivity

Parameter: None

Return: Collision sensitivity int

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
```

```
# Get collision sensitivity
suc, result, id = sendCMD(sock, "getCollisionSensitivity")
print(result)
else:
    print("Connection failed")
disconnectETController(sock)
```

2.2.2.54 Get the pose of the current tcp in the current user coordinate system

```
{"jsonrpc": "2.0", "method": "getTcpPoseInUser", "params": {"unit_type":
    unit_type}, "id": id}
```

Function: Get the pose of the current tcp in the current user coordinate system

Parameter: unit_type: int[0,1], optional parameter, returns the unit type of pose rx, ry, rz, 0: angle, 1: radians, if not written, the default value is radians.

Return: The current pose of tcp in the user coordinate system

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if conSuc:
        # Get the pose information of the current tcp in the current user coordinate system
        suc, result, id = sendCMD(sock, "getTcpPoseInUser", {"unit_type": 0})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.2.55 Get the serial number of the robot alarm information

```
{"jsonrpc": "2.0", "method": "getAlarmNum", "id": id}
```

Function: Get the serial number of the robot alarm information

Parameter: None

Return: Returns the serial numbers of the last 5 robot alarm messages successfully, otherwise returns false

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the abnormality of the robot arm body
        suc, result, id = sendCMD(sock, "getAlarmNum")
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.2.56 Get joint motion speed

```
{"jsonrpc": "2.0", "method": "get_joint_speed", "id": id}
```

Function: Get joint speed

Parameter: None

Return: Joint movement speed double speed[6], unit: degree/s

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        while 1:
            suc, result, id = sendCMD(sock, "get_joint_speed")
            print(suc, result, id)
            time.sleep(0.001)
        else:
            print("Connection failed")
    disconnectETController(sock)
```

2.2.2.57 Get tcp acceleration

```
{"jsonrpc": "2.0", "method": "get_tcp_acc", "id": id}
```

Function: Get tcp acceleration

Parameter: None

Return: tcp motion acceleration double tcp_acc, unit mm/s^2

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        while 1:
            suc, result, id = sendCMD(sock, "get_tcp_acc")
            print(suc, result, id)
            time.sleep(0.01)
        else:
            print("Connection failed")
            disconnectETController(sock)
```

2.2.2.58 Get joint acceleration

```
{"jsonrpc": "2.0", "method": "get_joint_acc", "id": id}
```

Function: Get joint acceleration

Parameter: None

Return: Joint motion acceleration double joint_acc[6], unit: degree/s²

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        while 1:
            suc, result, id = sendCMD(sock, "get_joint_acc")
            print(suc, result, id)
            time.sleep(0.01)
        else:
            print("Connection failed")
            disconnectETController(sock)
```

2.2.2.59 Get tcp movement speed

```
{"jsonrpc": "2.0", "method": "get_tcp_speed", "id": id}
```

Function: Get tcp movement speed

Parameter: None

Return: Current tcp movement speed double tcp_speed, unit: mm/sec

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get tcp movement speed
        suc, result, id = sendCMD(sock, "get_tcp_speed")
        print (result)
    else:
        print ("Connection failed")
    disconnectETController(sock)
```

2.2.2.60 Get the emergency stop status of the robot

```
{"jsonrpc": "2.0", "method": "get_estop_status", "id": id}
```

Function: Get the emergency stop status of the robot

Parameter: None

Return: int[0,1], whether the robot is in an emergency stop state, 1: emergency stop, 0: non-emergency stop

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "get_estop_status")
        print (result)
```

2.2.2.61 Get analog output voltage value

```
{"jsonrpc": "2.0", "method": "get_analog_output", "params": {"addr": addr}, "id": id}
```

Function: Get analog output voltage value

Parameter: addr: analog output address, range: int [0,4]

Return: Analog output voltage value, double

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        for i in range(0,5):
            # Get analog output voltage value
            suc, result, id = sendCMD(sock,"get_analog_output",{"addr":i})
            print ( result )
```

2.2.2.62 Get tool load and center of mass

```
{"jsonrpc": "2.0", "method": "get_tool_payload", "params": {"tool_num":
    tool_num}, "id": id}
```

Function: Get tool load and centroid

Parameter: tool_num: tool number, range int[0,7]

Return: m: tool load quality, double
tool_cog: tool load centroid, double cog[3]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        for i in range(0,8):
            # Get tool load and centroid
            suc, result, id = sendCMD(sock,"get_tool_payload",{"tool_num": i})
            print ( result )
        else :
            print ("Connection failed")
    disconnectETController(sock)
```

2.2.2.63 Get joint position information of robot input

```
{"jsonrpc": "2.0", "method": "get_motor_pos", "id": id}
```

Function: Obtain the joint position information of the robot input

Parameter: None

Return: double pos[6]: joint position information of the robot input

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the joint position information of the robot input
        suc, result , id = sendCMD(sock,"get_motor_pos")
        print ( result )
    else :
        print ("Connection failed ")
    disconnectETController (sock)
```

2.2.2.64 Get the precise state of the robot servo encoder

```
{"jsonrpc":"2.0","method":"get_servo_precise_position_status","id":id}
```

Function: Get the precise status of the robot servo encoder

Parameter: None

Return: 1: exact, 0: inexact

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the precise status of the robot servo encoder
        suc, result , id = sendCMD(sock," get_servo_precise_position_status ")
        print ( result )
    else :
        print ("Connection failed ")
    disconnectETController (sock)
```

2.2.2.65 Get the state of the robot servo alarm

```
{"jsonrpc":"2.0","method":" get_servo_alarm_state","id":id }
```

Function: Get the state of the robot servo alarm

Parameter: None

Return: 1: in servo alarm state, 0: not in servo alarm state

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.248"
```

```
conSuc, sock = connectETController(robot_ip)
if (conSuc):
    # Get the state of the robot servo alarm
    suc, result, id = sendCMD(sock, "get_servo_alarm_state")
    print(result)
else:
    print("Connection failed")
disconnectETController(sock)
```

2.2.2.66 Clear the booking queue

```
{"jsonrpc":"2.0","method":"book_program_clear","id":id}
```

Function: Clear the booking queue

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.248"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Clear the booking queue
        suc, result, id = sendCMD(sock, "book_program_clear")
        print(suc, result, id)
        time.sleep(0.001)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: The command is only supported in the remote mode and it can clear the queue only when the robot stops or gives an alarm (under the static condition).

2.2.2.67 Get the actual tcp pose

```
{"jsonrpc":"2.0","method":"get_actual_tcp","params":{"tool_num":tool_num,
"user_num":user_num},"id":id}
```

Function: Get the actual tcp pose data in the base coordinate system or the specified user coordinate system

Parameter: tool_num: tool coordinate number, optional parameter, int[0,7], when it is not entered, the current tool is used. Otherwise, the specified tool is used
user_num: user coordinate number, optional parameter, int[0,7], when it is not entered, the user gets the pose in the base coordinate system. Otherwise, the user gets the pose

Return: in the specified user coordinate

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the actual tcp pose of the tool 1 in the user 1 coordinate system
        suc, result ,id=sendCMD(sock,"get_actual_tcp",{"tool_num":1,"user_num":1})
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

2.2.2.68 Get the target interpolation tcp pose

```
{"jsonrpc":"2.0","method":"get_target_tcp","params":{"tool_num":tool_num,
"user_num":user_num},"id":id}
```

Function: Get the target interpolation tcp pose data in the base coordinate system or the specified user coordinate system

Parameter: tool_num: tool coordinate number, optional parameter, int[0,7], when it is not entered, the current tool is used. Otherwise, the specified tool is used
user_num: user coordinate number, optional parameter, int[0,7], when it is not entered, the user gets the pose in the base coordinate system. Otherwise, the user gets the pose in the specified user coordinate

Return: Robot pose information double pose [6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController (robot_ip)
    if (conSuc):
        # Get the target interpolation tcp pose of the tool 1 in the user 1 coordinate system
        suc, result ,id=sendCMD(sock,"get_target_tcp",{"tool_num":1,"user_num":1})
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

2.2.2.69 Get the actual joint

```
{"jsonrpc":"2.0","method":"get_actual_joint","id":id}
```

Function: Get the current actual joint data

Parameter: none

Return: Robot joint information double joint[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current actual joint
        suc, result ,id=sendCMD(sock,"get_actual_joint")
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

2.2.2.70 Get the target interpolation joint

```
{"jsonrpc": "2.0", "method": "get_target_joint", "id": id}
```

Function: Get the current target interpolation joint data

Parameter: none

Return: Robot joint information double joint [6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current target interpolation joint
        suc, result ,id=sendCMD(sock,"get_target_joint")
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

2.2.2.71 Get the linear interpolation pose

```
{"jsonrpc": "2.0", "method": "get_interp_pose", "params": {"data1": data1,
    "data2": data2, "ratio": ratio}, "id": id}
```

Function: Get the linear interpolation pose data between two given poses

Parameter: data1: pose data, standard parameter, double pose [6], the first three stand for position, unit x, y, z is mm, range is $[-\infty, +\infty]$, the last three stand for pose, unit Rx, Ry, Rz is radian, range is $[-\pi, \pi]$
data2: pose data, standard parameter, double pose[6], the first three stand for position, unit x, y, z is mm, range is $[-\infty, +\infty]$, the last three stand for pose, unit Rx, Ry, Rz is radian, range is $[-\pi, \pi]$
ratio: floating-point data, standard parameter, it stands for the proportional value. The range is [0,1]. When the value is equal to 0, the robot will return to the first pose. When the value is equal to 1, the robot will return to the second pose

Return: Robot pose information double pose [6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    point1=[371.533, 101.636, 3.038, 0, -0.174, 2.861]
    point2=[346.312, -256.945, -91.131, -0.014, 0.521, 1.903]
    if (conSuc):
        # Get the linear interpolation pose data between two given poses
        suc, result , id=sendCMD(sock,"get_interp_pose",{"data1":point1,"data2":point2,"ratio":0.5})
        print ( result )
    else:
        print ("Connection failed")
    disconnectETController (sock)
```

2.2.2.72 Get the joint temperature

```
ret get_joint_temp()
```

Function: Get the joint temperature

Parameter: none

Return: double joint_temp[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        suc, result ,id=sendCMD(sock,"get_joint_temp")
        print ( suc, result, id )
    else:
        print ("Connection failed")
    disconnectETController(sock)
```

2.2.3 Movement Service(MovementService)

2.2.3.1 Joint Movement

```
{"jsonrpc": "2.0", "method": "moveByJoint", "params": {"targetPos": targetPos, "speed": speed, "acc": acc, "dec": dec, "cond_type": cond_type, "cond_num": cond_num, "cond_value": cond_value}, "id": id}
```

Function: Joint movement

Parameter: targetpos: the target joint point double pos[6], the range is [-360,360]

speed: operating speed, range: double[0.01,100]

cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range: int [0,2]

cond_num: IO address, optional parameter, range int[0,63]

cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, the unfinished movement is immediately abandoned and the next instruction is executed.

cond_judgment: conditional judgment, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.

acc: acceleration percentage, range: int [1,100], optional parameter, the default values 80 if not written.

dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc, sock = connectETController(robot_ip)
    point = []
    point.append([0.0065,-103.9938,102.2076,-88.2138,
        90.0000,0.0013])
    point.append([-16.2806,-82.4996,81.9848,-89.4851,
        90.0000,-16.2858])
    point.append([3.7679,-71.7544,68.7276,-86.9732,
        90.0000,3.7627])
    point.append([12.8237,-87.3028,87.2361,-89.9333,
        90.0000,12.8185])
    if(conSuc):
        # Get the servo status of the robotic arm
        suc, result, id = sendCMD(sock,"getServoStatus")
        if(result == 0):
            # Set the servo status of the robotic arm to ON
            suc, result, id = sendCMD(sock,"set_servo_status",{"status":1})
            time.sleep(1)
            for i in range(4):
                # Joint movement
                suc, result, id=sendCMD(sock,"moveByJoint",{"targetPos":point[i],"speed":30,"acc":10,"dec":10,"cond_type":0,"cond_num":7,"cond_value":1})
            while(True):
                # Get robot status
                suc, result, id = sendCMD(sock,"getRobotState")
                if(result == 0):
                    break
```

Note: When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command. Except for the keyword UNTIL, the length of the parsed data cannot exceed 1024 characters.

2.2.3.2 Linear motion

```
{"jsonrpc": "2.0", "method": "moveByLine", "params": {"targetPos": targetPos, "speed_type": speed_type, "speed": speed, "acc": acc, "dec": dec, "cond_type": cond_type, "cond_num": cond_num, "cond_value": cond_value}, "id": id}
```

Parameter: targetpos: the target joint point double pos[6], the range is [-360,360]
speed: operating speed. double Type is linear speed, range: 1-3000; is rotation angular speed, range: 1-300; is absolute linear speed, range: linear minimum speed parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maximum speed parameter value
speed_type: Speedtype, 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed). Optional.
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range: int [0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, immediately abandon the unfinished movement and execute the next instruction
cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.
acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.
dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```

if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.205"
    conSuc, sock = connectETController(robot_ip)
    point = []
    point.append([0.0065,-103.9938,102.2076,-88.2138,
    90.0000,0.0013])
    point.append([-16.2806,-82.4996,81.9848,-89.4851,
    90.0000,-16.2858])
    point.append([3.7679,-71.7544,68.7276,-86.9732,
    90.0000,3.7627])
    point.append([12.8237,-87.3028,87.2361,-89.9333,
    90.0000,12.8185])
    if(conSuc):
        # Set the servo status of the robotic arm to ON
        suc, result, id = sendCMD(sock,"set_servo_status",{ "status":1})
        time.sleep(1)
        for i in range(4):
            # Linear motion
            suc, result, id=sendCMD(sock,"moveByLine",{ "targetPos":point[i],"speed_type":0,"speed":200,"cond_type":0,"cond_num":7,"cond_value":1})
        while(True):
            # Get robot status
            suc, result, id = sendCMD(sock,"getRobotState")
            if (result == 0):
                break

```

Note: When there is no speed_type parameter, speed means absolute linear speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command. Except for the keyword UNTIL, the length of the parsed data cannot exceed 1024 characters.

2.2.3.3 Circular Movement

```
{ "jsonrpc": "2.0", "method": "moveByArc", "params": { "midPos": midPos, "targetPos": targetPos, "speed_type": speed_type, "speed": speed, "acc": acc, "dec": dec, "cond_type": cond_type, "cond_num": cond_num, "cond_value": cond_value }, "id": id }
```

Function: Circular motion

Parameter: midpos: middle joint point double pos[6], the range is [-360,360]

targetpos: the target joint point double pos[8], the range is [-360,360]

speed: operating speed. double Type is linear speed, range: [1,3000]; is rotation angular speed, range: [1,300]; is absolute linear speed, range: linear minimum speed parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maximum speed parameter value

speed_type: speed type, int[0,3], 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed). Optional.

cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]

cond_num: IO address, optional parameter, range int[0,63]

cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, the unfinished movement is immediately abandoned and the next instruction is executed.

cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.

acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.

dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    P000 = [0.0065,-103.9938,102.2076,-88.2138,
```

```

90.0000,0.0013]
P001 = [-16.2806,-82.4996,81.9848,-89.4851,
90.0000,-16.2858]
if (conSuc):
    # Get the servo status of the robotic arm
    suc, result, id=sendCMD(sock,"getServoStatus")
    if ( result == 0):
        # Set the servo status of the robotic arm to ON
        suc, result, id=sendCMD(sock,"set_servo_status",{ " status ":1})
        time.sleep(1)
    #Circular movement
    suc, result, id=sendCMD(sock,"moveByArc",{ "midPos":P000,"targetPos":P001,"speed_type"
:0,"speed":20,"cond_type":0,"cond_num":7,"cond_value":1})

```

Note: When there is no speed_type parameter, speed means absolute linear speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command. Except for the keyword UNTIL, the length of the parsed data cannot exceed 1024 characters.

2.2.3.4 Rotation Movement

```

{"jsonrpc": "2.0", "method": "moveByRotate", "params": {"targetPos": targetPos
, "speed_type": speed_type, "speed": speed, "acc": acc, "dec": dec, "cond_type
": cond_type, "cond_num": cond_num, "cond_value": cond_value}, "id": id}

```

Function: Rotational movement

Parameter: targetpos: the target joint point double pos[6], the range is [-360,360]
speed: operating speed. double Type is linear speed, range: [1,3000]; is rotation an-gular speed, range: [1,300]; is absolute linear speed, range: linear minimum speed parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maxi-mum speed parameter value
speed_type: speed type, int[0,1], 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed). Optional.
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, the unfinished movement is immediately abandoned and the next instruction is executed.
cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and

the next instruction will be executed.

acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.

dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ ==
    # Robot IP
    robot_ip="192.168.1.200
    conSuc,sock=connectETController(robot_ip
    P000 = [0.0065,-103.9938,102.2076,-88.2138,
    90.0000,0.0013]
    if(conSuc):
        # Get the servo status of the robotic arm
        suc, result ,id=sendCMD(sock,"getServoStatus")
        if ( result == 0):
            # Set the servo status of the robotic arm to ON
            suc, result ,id=sendCMD(sock,"set_servo_status",{ " status ":1})
            time.sleep(1)
        # Rotational movement
        suc, result ,id=sendCMD(sock,"moveByRotate",{ "targetPos":P000,"speed_type":0,"speed":20,
            "cond_type":0,"cond_num":7,"cond_value":1 })
```

Note: When there is no speed_type parameter, speed represents the absolute rotational angular speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command. Except for the keyword UNTIL, the length of the parsed data cannot exceed 1024 characters. This command is deprecated.

CAUTION



The above commands are only supported in remote mode.

Before executing the above commands, please make sure that the robot is in a stopped state. If the robot is running, send the stop command first and wait for the robot to stop.

2.2.3.5 Add waypoint info 2.0

```
{"jsonrpc":"2.0","method":"addPathPoint","params":{"wayPoint":wayPoint,"
    moveType":moveType,"speed_type":speed_type,"speed":speed,"acc":acc,"
    dec":dec,"smooth":smooth,"cond_type":cond_type,"cond_num":cond_num,"
    cond_value":cond_value},"id":id} or {"jsonrpc":"2.0","method":"
    addPathPoint","params" :{"wayPoint":wayPoint,"moveType":moveType,"
    speed_type":speed_type,"speed":speed,"acc":acc,"dec":dec,"
    circular_radius":circular_radius,"cond_type":cond_type,"cond_num":
    cond_num,"cond_value": cond_value},"id":id}
```

Function: Add Waypoint Info 2.0

Parameter: waypoint: target joint point double pos[6], the range is [-360,360]
 moveType: 0 joint motion, 1 linear motion, 2 rotational motion around the tool tip point, 3 arc motion
 speed: running speed. double, joint speed range: [1,100] when the joint moves. For linear, rotary and circular motion, the type is linear speed range: [1,3000]; it is rotational angular speed, range: [1,300]; is the absolute linear speed, the range: the minimum linear speed parameter value - the linear maximum speed parameter value; is the absolute rotational angular velocity, the range: the rotational angle minimum speed parameter value - the rotational angle maximum speed parameter value When there is no speed_type parameter, it is expressed as: motion speed, joint motion speed range [1,100], linear and arc speed range [1,3000], rotational motion speed range [1,300]
 speed_type: Speed type, int[0,3], 0 is V (linear velocity), 1 is VR (rotational angular velocity), 2 is AV (absolute linear velocity), 3 is AVR (absolute rotational angular velocity). Optional.
 smooth: smoothness, range: int[0,7], optional parameter, the smoothness of the last point must be 0. deprecated.
 circular_radius: blending radius, range: double[0, 2147483647], unit millimeter, optional parameter, default is 0 if not written. The blend radius of the last point must be 0.
 cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]
 cond_num: IO address, optional parameter, range int[0,63]
 cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, immediately give up the unfinished motion and execute the next instruction
 cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.
 acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.
 dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, False for failure

Example:

```

if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    C000 = [0.0065,-103.9938,102.2076,-88.2138,
    90.0000, 0.0013]
    C001 = [-16.2806,-82.4996,81.9848,-89.4851,
    90.0000,-16.2858]
    # clear waypoint info 2.0
    suc, result, id = sendCMD(sock,"clearPathPoint")
    if (result == True):
        # Add waypoint info 2.0
        suc, result, id=sendCMD(sock,"addPathPoint",{ "wayPoint":C000,"moveType":0,"speed":50,"
        circular_radius":50})
        suc, result, id=sendCMD(sock,"addPathPoint",{ "wayPoint":C001,"moveType":1,"speed_type":
  
```

```
0,"speed":50,"circular_radius":0})
```

Note: When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command. Except for the keyword UNTIL, the length of the parsed data cannot exceed 1024 characters.

CAUTION



This command is only supported in remote mode.

If the motion type is joint motion, the speed_type parameter is invalid and not recommended.

The parameter circular_radius and the parameter smooth can be used either. It is recommended to use the parameter circular_radius.

2.2.3.6 Clear waypoint information 2.0

```
{"jsonrpc":"2.0","method":"clearPathPoint","id":id}
```

Function: Clear waypoint information 2.0

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the servo status of the robotic arm
        suc, result ,id=sendCMD(sock,"getServoStatus")
        if (result == 0):
            # Clear waypoint information 2.0
            suc, result , id = sendCMD(sock,"clearPathPoint")
```

Note: This command is only supported in remote mode.

2.2.3.7 Track Movement 2.0

```
{"jsonrpc":"2.0","method":"moveByPath","id":id}
```

Function: Track Movement 2.0

Parameter: None

Return: Failure: -1, Success: the total number of waypoints

Example: if __name__ == "__main__":

```
# Robot IP address
robot_ip="192.168.1.200"
conSuc,sock=connectETController(robot_ip)
C000 = [0.0065,-103.9938,102.2076,-88.2138,
90.0000,0.0013]
C001 = [-16.2806,-82.4996,81.9848,-89.4851,
90.0000,-16.2858]
C002 = [3.7679, -71.7544, 68.7276, -86.9732,
90.0000, 3.7627]
if(conSuc):
    # Clear waypoint information 2.0
    suc, result, id = sendCMD(sock, "clearPathPoint")
    if( result == True):
        # Add waypoint information 2.0
        suc, result, id = sendCMD(sock, "addPathPoint", {"wayPoint": C000,"moveType": 0, "
speed": 50, " circular_radius ":20})
        suc, result, id = sendCMD(sock, "addPathPoint", {"wayPoint": C001,"moveType":0, "
speed": 50, " circular_radius ":20})
        suc, result, id = sendCMD(sock, "addPathPoint", {"wayPoint": C002,"moveType": 0, "
speed": 50, " circular_radius ":0})
        # Trajectory movement 2.0
        suc, result, id = sendCMD(sock, "moveByPath")
        while(True):
            # Get the line number of trackfile file running ( consistent with the line number
            displayed by the teach pendant)
            suc, result, id = sendCMD(sock, "getPathPointIndex")
            print ( result )
            # Get robot status
            suc, result, id = sendCMD(sock, "getRobotState")
            if ( result == 0):
                break
```

Note: This command is only supported in remote mode.
Before executing this command, make sure that the robot is in a stopped state. If the robot is running, send the stop command first and wait for the robot to stop.

2.2.3.8 Jog Movement

```
{"jsonrpc": "2.0", "method": "jog", "params": {"index": index, "speed": speed}, "
id": id}
```

Function: Jog movement

Parameter: index: axis direction or coordinate system direction number, range: int[0,11]
speed: manual speed percentage, range double [0.05,100] (optional parameter, not required)

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the servo status of the robotic arm
        suc, result, id = sendCMD(sock, "getServoStatus")
        if (result == 0):
            # Set the servo status of the robot arm to ON
            suc, result, id=sendCMD(sock, "set_servo_status", {"status":1})
            time.sleep(1)
        # Specify the coordinate system
        suc, result, id=sendCMD(sock, "setCurrentCoord", {"coord_mode":1})
        for i in range(0, 10, 1):
            # x-axis negative direction jog movement
            suc, result, id = sendCMD(sock, "jog", {"index":0, "speed":10})
            print(suc, result, id)
            time.sleep(0.1)
        suc, result, id = sendCMD(sock, "stop")
```

Note: After stopping sending the jog command, the robot will not stop immediately. Instead, the robot needs to be stopped immediately by the "stop robot operation" command below.

This command is only supported in remote mode.

If the next jog motion instruction is not received for more than 1 second, stop receiving the jog instruction, and the robot jog motion will stop.

2.2.3.9 Stop robot operation

```
{"jsonrpc": "2.0", "method": "stop", "id": id}
```

Function: Stop robot operation

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Robot stop
        suc, result, id = sendCMD(sock, "stop")
```

Note: This command is only supported in remote mode.

2.2.3.10 Robot runs automatically

```
{"jsonrpc": "2.0", "method": "run", "id": id}
```

Function: Robot runs automatically

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Robot stop
        suc, result, id = sendCMD(sock, "pause")
        time.sleep(1)
        # Robot start
        suc, result, id = sendCMD(sock, "run")
```

Note: This command is only supported in remote mode.

2.2.3.11 Robot stop

```
{"jsonrpc": "2.0", "method": "pause", "id": id}
```

Function: Robot stop

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Robot stop
        suc, result, id = sendCMD(sock, "pause")
        time.sleep(1)
```

Note: This command is only supported in remote mode.

2.2.3.12 Check if the jbi file exists

```
{"jsonrpc": "2.0", "method": "checkJbiExist", "params": {"filename": filename}, "id": id}
```

Function: Check if the jbi file exists

Parameter: filename: the name of the file to be checked, string

Return: 0 means not present, 1 means present

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    jbi_filename="test"
    if (conSuc):
        # Check if the jbi file exists
        suc, result, id=sendCMD(sock, "checkJbiExist", {"filename": jbi_filename })
```

2.2.3.13 Run jbi file

```
{"jsonrpc": "2.0", "method": "runJbi", "params": {"filename": filename}, "id": id}
```

Function: Run jbi file

Parameter: filename the name of the file to be run, string

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    jbi_filename="test"
    if (conSuc):
        # Check if the jbi file exists
```

```
suc, result, id=sendCMD(sock,"checkJbiExist",{"filename": jbi_filename })
if (suc and result ==1):
    # Run jbi file
    suc, result, id=sendCMD(sock,"runJbi",{"filename": jbi_filename })
```

Note: This command is only supported in remote mode.
Before executing this command, make sure that the robot is in a stopped state. If the robot is running, send the stop command first and wait for the robot to stop.

2.2.3.14 Get jbi file running status

```
{"jsonrpc": "2.0", "method": "getJbiState", "id": id}
```

Function: Get jbi file running status

Parameter: None

Return: jbiName: file name
runState: 0 stop state, 1 pause state, 2 emergency stop state, 3 running state, 4 error state

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    jbi_filename="test"
    if (conSuc):
        # Check if the jbi file exists
        suc, result, id=sendCMD(sock,"checkJbiExist",{"filename": jbi_filename })
        if (suc and result ==1):
            # Run jbi file
            suc, result, id=sendCMD(sock,"runJbi",{"filename": jbi_filename })
            if (suc and result ):
                checkRunning=3
                while (checkRunning==3):
                    # Get jbi file running status
                    suc, result, id=sendCMD(sock,"getJbiState")
                    checkRunning=result["runState"]
                    time.sleep(0.1)
```

2.2.3.15 Set robot running speed

```
{"jsonrpc": "2.0", "method": "setSpeed", "params": {"value": value}, "id": id}
```

Function: Set the robot running speed

Parameter: value: speed, range: double [0.05,100]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Set the robot running speed to 30%
        suc, result, id = sendCMD(sock,"setSpeed",{ "value": 30})
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v2.13.1 and above.
This command is only supported in remote mode.

2.2.3.16 Joint motion at uniform speed

```
{"jsonrpc": "2.0", "method": "moveBySpeedj", "params": {"vj": vj, "acc": acc, "t": t}, "id": id}
```

Function: Uniform motion of joints

Parameter: vj: double type, speed value of 6 joints, six-digit unit: degree/second
acc: joint acceleration, int, range: greater than 0, unit: degree/s²
t: SPEEDJ execution time, double, range: greater than 0, unit: second

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    speed_j = [1.0,1.0,1.0,1.0,1.0,1.0]
    if(conSuc):
        # Uniform motion of joints
        suc, result, id=sendCMD(sock,"moveBySpeedj",{ "vj":speed_j,"acc":20,"t":5})
        print(suc, result, id)
```

Note: This command is applicable to v2.16.2 and above.
This command is only supported in remote mode.
During moveBySpeedj movement, when multiple moveBySpeedj instructions are sent, or stopj instruction is sent, the robot will not decelerate after executing the moveBySpeedj instruction, and continue to execute the last instruction sent by the user during the movement.

2.2.3.17 Stop joint motion at uniform speed

```
{"jsonrpc": "2.0", "method": "stopj", "params": {"acc": acc}, "id": id}
```

Function: Stop joint movement at a constant speed

Parameter: acc: int, joint acceleration, stop motion at this acceleration, unit: degree/s^2 , range: greater than 0

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc):  
        suc, result ,id=sendCMD(sock,"moveBySpeedj",{"vj": [20,0,0,0,0,0,0,0], "acc":50, "t":2})  
        print (suc, result ,id)  
        time.sleep(1)  
        suc, result ,id=sendCMD(sock,"stopj",{"acc":10})  
        print ( result )
```

2.2.3.18 Linear uniform motion

```
{"jsonrpc": "2.0", "method": "moveBySpeedl", "params": {"v": v, "acc": acc, "arot": arot, "t": t}, "id": id}
```

Function: Linear uniform motion

Parameter: v: double type, the speed value of moving along 6 directions, unit: the first three are mm/sec, the last three are degrees/sec

acc: int displacement acceleration, range: greater than 0, unit: mm/s^2

arot: int optional parameter, attitude acceleration, range: greater than 0, unit: degree/s^2

t: SPEEDJ execution time, double, range: greater than 0, unit: second

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    speed_l =[1.0,1.0,1.0,1.0,1.0,1.0]
    if (conSuc):
        # Linear uniform motion
        suc, result ,id=sendCMD(sock,"moveBySpeedl",{ "v":speed_l,"acc":100,"arot":10,"t":3.0})
        print (suc, result ,id)
```

Note: This command is applicable to v2.16.2 and above.

This command is only supported in remote mode.

During moveBySpeedl movement, when multiple moveBySpeedj instructions are sent, or stopl instruction is sent, the robot will not decelerate after executing the moveBySpeedl instruction, and continue to execute the last instruction sent by the user during the movement.

2.2.3.19 Stop linear motion at constant speed

```
{"jsonrpc": "2.0", "method": "stopl", "params": {"acc": acc, "arot": arot}, "id": id}
```

Function: Stop linear motion at constant speed

Parameter: acc: int, acceleration, stop motion at this acceleration, unit: mm/sec^2 , range: greater than 0

arot: int optional parameter, attitude acceleration, range: greater than 0, unit: degree/s^2

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        suc, result ,id=sendCMD(sock,"moveBySpeedI",{ "v":[20,0,0,0,0,0],"acc":50,"arot":10,"t":2
        })
        print (suc, result ,id)
        time.sleep(1)
        suc, result ,id=sendCMD(sock,"stopI",{ "acc":10})
        print ( result )
```

2.2.3.20 Linear motion under specified coordinate system

```
{"jsonrpc": "2.0", "method": "moveByLineCoord", "params": {"targetUserPose":
    targetUserPose, "speed_type": speed_type, "speed": speed, "acc": acc, "dec":
    dec, "user_coord": user_coord, "cond_type": cond_type, "cond_num": " :
    cond_num:", "cond_value": cond_value, "unit_type": unit_type}, "id": id}
```

Function: Linear motion in the specified coordinate system

Parameter: targetUserPose: specify the pose in the user coordinate system, where rx, ry, rz are radians, range: double $[-\pi, \pi]$ or angle, range: double $[-180, 180]$

speed: operating speed. double Type is linear speed, range: $[1, 3000]$; is rotation

angular speed, range: $[1, 300]$; is absolute linear speed, range: linear minimum speed

parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maximum speed parameter value

speed_type: speed type, optional parameter, int [0,3], 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed).

user_coord: user coordinate system data, double[6], where rx, ry, rz are radians, range: double $[-\pi, \pi]$ or angle, range: double $[-180, 180]$, do not write the current coordinate system.

cond_type: optional, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]

cond_num: IO address, optional, range int[0,63]

cond_value: IO status, optional, range int[0,1], when the actual IO status is consistent with this value, immediately abandon the unfinished movement and execute the next instruction.

cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately.

acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.

dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

unit_type: the unit type of rx, ry, rz of user coordinate and user coordinate system, int [0,1], 0: angle, 1: radians, optional parameters, if not written, the default value is radians.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    point =[211,126,343,-2.58,-0.013,-1.813]
    if (conSuc):
        # Specify the linear motion in the coordinate system
        suc, result ,id=sendCMD(sock,"moveByLineCoord",{ "targetUserPose": point,"user_coord"
            :[0,0,0,0,0,0], "speed_type":1,"speed":30,"unit_type":1})
    print (suc, result , id)
```

Note: This command is applicable to v2.16.2 and above. This command is only supported in remote mode. When there is no speed_type parameter, speed means absolute linear speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command. Except for the keyword UNTIL, the length of the parsed data cannot exceed 1024 characters.

DN:EC

2.2.3.21 Encoder zero calibration

```
{"jsonrpc": "2.0", "method": "calibrate_encoder_zero_position", "id": id}
```

Function: Encoder Zero Calibration

Parameter: None

Return: success true, failure false

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.0.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "getServoStatus")
        print(suc, result, id)
        time.sleep(0.5)
        if result == 0:
            # Set the arm servo state
            ret, result, id = sendCMD(sock, "set_servo_status", {"status": 1})
            print(result)
            time.sleep(1)
            # encoder zero calibration
            suc, result, id = sendCMD(sock, "calibrate_encoder_zero_position")
            print(suc, result, id)
        else:
            print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode.

2.2.4 Kinematics Service(KinematicsService)

2.2.4.1 Inverse solution function

```
{"jsonrpc": "2.0", "method": "inverseKinematic", "params": {"targetPose": targetPose, "referencePos": referencePos}, "id": id}
```

Function: Inverse solution function, inverse solution with reference point position, according to the pose information to get the corresponding manipulator joint angle information

Parameter: targetPose: target pose information
referencePos: inverse solution reference point joint angle information double pos[6], the range is [-360,360], optional parameter
unit_type: the unit type of the rx, ry, rz of the input pose, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Joint coordinates double pos[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    # Reference point
    P000 = [0, -90, 90, -90, 90, 0]
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current pose information of the robot
        suc, result, id = sendCMD(sock, "get_tcp_pose")
        # Inverse solution function 2.0, with reference point position
        suc, result, id=sendCMD(sock, "inverseKinematic", {"targetPose": result, "referencePos": P000})
```

2.2.4.2 Positive solution function

```
{"jsonrpc": "2.0", "method": "positiveKinematic", "params": {"targetPos": targetPos, "unit_type": unit_type}, "id": id}
```

Function: Positive solution function, get the corresponding pose information according to the joint angle information of the manipulator

Parameter: targetpos: target joint angle information double pos[6], the range is [-360,360]
unit_type: return the unit type of the pose rx, ry, rz, int [0,1], 0: return angle, 1: return radian, optional parameter, if not written, the default value is radians.

Return: Response pose information: double pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current position information of the robot
        suc, result, id = sendCMD(sock, "get_joint_pos")
        # Positive solution function
        suc, result, id=sendCMD(sock, "positiveKinematic", {"targetPos": result, "unit_type": 1})
```

2.2.4.3 Base coordinate to user coordinate pose conversion

```
{"jsonrpc": "2.0", "method": "convertPoseFromCartToUser", "params": {"targetPose": targetPose, "userNo": userNo, "unit_type": unit_type}, "id": id}
```

Function: Base coordinate to user coordinate pose conversion function, in the current user coordinate system, the pose information in the corresponding user coordinate system is obtained according to the pose information of the base coordinate

Parameter: targetPose: pose information in the base coordinate system, double pose[6], rx, ry, rz range: radian is $[-\pi, \pi]$, angle is $[-180, 180]$
userNo: User coordinate number, range: int[0,7]
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Pose information under the user standard system: double user_pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc, sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current pose information of the robot
        suc, result, id = sendCMD(sock, "get_tcp_pose")
        # Base coordinate to user coordinate pose conversion
        suc, result, id=sendCMD(sock, "convertPoseFromCartToUser", {"targetPose": result, "userN": 0, "unit_type": 1})
```

2.2.4.4 User coordinate to base coordinate pose conversion

```
{"jsonrpc": "2.0", "method": "convertPoseFromUserToCart", "params": {"targetPose": targetPose, "userNo": userNo, "unit_type": unit_type}, "id": id}
```

Function: User coordinate to base coordinate pose conversion, in the current user coordinate system, the pose information in the corresponding base coordinate system is obtained according to the pose information of the user coordinate

Parameter: targetPose: the pose information in the user coordinate system, double pose[6], the range of rx, ry, rz: radian is $[-\pi, \pi]$, angle is $[-180, 180]$
userNo: User coordinate number, range: int[0,7]
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Pose information in the base coordinate system: double base_pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current pose information of the robot
        suc, result, id = sendCMD(sock, "get_tcp_pose")
        # User coordinate to base coordinate pose conversion
        suc, result, id=sendCMD(sock, "convertPoseFromUserToCart", {"targetPose":result, "userNo":0})
```

2.2.4.5 Position multiplication

```
{"jsonrpc": "2.0", "method": "poseMul", "params": {"pose1": pose1, "pose2": pose2, "unit_type": unit_type}, "id": id}
```

Function: Position multiplication

Parameter: pose1: pose information, double pose[6], the range of rx, ry, rz: radian is $[-\pi, \pi]$, angle is $[-180, 180]$
pose2: pose information, double pose[6], the range of rx, ry, rz is: $[-\pi, \pi]$ in radians, and the angle is $[-180, 180]$
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Posture multiplication result information: :double response_pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    # pose1
    V000 = [10, -10, 10, 0, 0, 0]
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the current pose information of the robot
        suc, result, id = sendCMD(sock, "get_tcp_pose")
        # Position multiplication
        suc, result, id = sendCMD(sock, "poseMul", {"pose1": V000, "pose2": result, "unit_type": 1})
```

2.2.4.6 Pose inversion

```
{"jsonrpc": "2.0", "method": "poseInv", "params": {"pose": pose, "unit_type": unit_type}, "id": id}
```

Function: Pose inversionPose inversion

Parameter: pose: pose information, double pose[6], the range of rx, ry, rz: radian is $[-\pi, \pi]$, angle is $[-180, 180]$
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Pose inversion result information: double response_pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the current pose information of the robot
        suc, result, id = sendCMD(sock, "get_tcp_pose")
        # Pose inversion
        suc, result, id = sendCMD(sock, "poseInv", {"pose": result, "unit_type": 1})
```

2.2.5 IO Service(IOService)

2.2.5.1 Get input IO status

```
{"jsonrpc": "2.0", "method": "getInput", "params": {"addr": addr}, "id": id}
```

Function: Get input IO status

Parameter: addr: input IO address, range: int[0,19][48,51]

Return: Input IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        for i in range(0, 64 ,1):
            # Get input IO status
            suc, result, id = sendCMD(sock, "getInput", {"addr":i})
            print ( result )
```

2.2.5.2 Get output IO status

```
{"jsonrpc": "2.0", "method": "getOutput", "params": {"addr": addr}, "id": id}
```

Function: Get output IO status

Parameter: addr: output IO address, range: int[0,19][48,51]

Return: Output IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        for n in range(0, 20 ,1):
            # Get output IO status
            suc, result, id = sendCMD(sock, "getOutput", {"addr":n})
            print ( result )
```

2.2.5.3 Set output IO status

```
{"jsonrpc": "2.0", "method": "setOutput", "params": {"addr": addr, "status": status}, "id": id}
```

Function: Set output IO status

Parameter: addr: output IO address, range: int[0,19][48,49]
status: IO status, int[0,1], 0 is off, 1 is on

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc):  
        for i in range(0, 20 ,1):  
            # Set output IO status  
            suc, result ,id=sendCMD(sock,"setOutput",{"addr":i,"status":1})  
            print ( result )
```

Note: This command is only supported in remote mode.

2.2.5.4 Get virtual input IO status

```
{"jsonrpc": "2.0", "method": "getVirtualInput", "params": {"addr": addr}, "id": id}
```

Function: Get virtual input IO status

Parameter: addr: virtual IO address, range: int[0,399]

Return: Input IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc):  
        for i in range(0, 400 ,1):  
            # Get virtual input IO status  
            suc, result ,id=sendCMD(sock,"getVirtualInput",{"addr":i})  
            print ( result )
```


2.2.5.5 Get virtual output IO status

```
{"jsonrpc": "2.0", "method": "getVirtualOutput", "params": {"addr": addr}, "id": id}
```

Function: Get virtual output IO status

Parameter: addr: virtual IO address, range: int [400,1535]

Return: Output IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        for n in range(528, 800 ,1):
            # Get virtual output IO status
            suc, result ,id=sendCMD(sock,"getVirtualOutput",{"addr":n})
            print ( result )
```

2.2.5.6 Set virtual output IO status

```
{"jsonrpc": "2.0", "method": "setVirtualOutput", "params": {"addr": addr, "status": status}, "id": id}
```

Function: Set virtual output IO status

Parameter: addr: output IO address, range: int[528,799]
status: output IO status, int[0,1], 0 is off, 1 is on

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        for i in range(528, 800 ,1):
            # Set virtual output IO status
            suc, result ,id=sendCMD(sock,"setVirtualOutput",{"addr":i,"status":1})
```

Note: This command is only supported in remote mode.

2.2.5.7 Read multiple M virtual IO

```
{ "jsonrpc": "2.0", "method": "getRegisters", "params": { "addr": addr, "len": len }, "id": id }
```

Function: Read multiple M virtual IO

Parameter: addr: virtual IO address range int [0,1535]

len: The starting address starts to read backward and the length is (16*len) virtual IO range int [1,96]

The range of addr+16*len is int[0,1535]

Return: List of virtual IO values (each 16 virtual IO values are represented by a decimal integer, and the length of the list is len)

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the value of M0~M16
        ret, result, id = sendCMD(sock, "getRegisters", { "addr": 0, "len": 1 })
        if ret:
            print("result=", result)
        else:
            print("err_msg=", result["message"])
```

2.2.5.8 Get analog input

```
{ "jsonrpc": "2.0", "method": "getAnalogInput", "params": { "addr": addr }, "id": id }
```

Function: Get analog input

Parameter: addr: analog address, range: int[0,2]

Return: Analog value, range: double[-10,10]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        for i in range(0, 2, 1):
            # Get analog input
            suc, result, id = sendCMD(sock, "getAnalogInput", { "addr": i })
```

2.2.5.9 Set analog output

```
{ "jsonrpc": "2.0", "method": "setAnalogOutput", "params": { "addr": addr, "value": value }, "id": id }
```

Function: Set analog output

Parameter: addr: analog address, range: int[0,4]
value: analog value, when addr is 0-3, range: double[-10,10]; when addr is 4, range: double[0,10]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Set analog output
        suc, result ,id=sendCMD(sock,"setAnalogOutput",{ "addr":0,"value":-10})
        suc, result ,id=sendCMD(sock,"setAnalogOutput",{ "addr":1,"value":-3.5})
        suc, result ,id=sendCMD(sock,"setAnalogOutput",{ "addr":2,"value":0})
        suc, result ,id=sendCMD(sock,"setAnalogOutput",{ "addr":3,"value":0.5})
        suc, result ,id=sendCMD(sock,"setAnalogOutput",{ "addr":4,"value":0.5})
```

Note: This command is only supported in remote mode.

2.2.6 Variable service(VarService)

2.2.6.1 Get system B variable value

```
{ "jsonrpc": "2.0", "method": "getSysVarB", "params": { "addr": addr }, "id": id }
```

Function: Get system B variable value

Parameter: addr: variable address, range: int [0,255]

Return: Variable value, range: int[0,2147483647]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc)
        for n in range(0, 11 ,1):
            # Get system B variable value
            suc, result , id = sendCMD(sock, "getSysVarB", { "addr":n})
            print ( result )
```

2.2.6.2 Set system B variable value

```
{"jsonrpc": "2.0", "method": "setSysVarB", "params": {"addr": addr, "value": value}, "id": id}
```

Function: Set system B variable value

Parameter: addr: variable address, range: int [0,255]
value: variable value, range: int [0,2147483647]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc)  
        for n in range(0, 11,1):  
            # Set system B variable value  
            suc, result, id=sendCMD(sock,"setSysVarB",{"addr":i,"value":100})
```

Note: This command is only supported in remote mode.

2.2.6.3 Get system I variable value

```
{"jsonrpc": "2.0", "method": "getSysVarI", "params": {"addr": addr}, "id": id}
```

Function: Get system I variable value

Parameter: addr: variable address, range: int [0,255]

Return: Variable value, range: int[-32767,32767]

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc)  
        for n in range(0, 11,1):  
            # Get system I variable value  
            suc, result, id = sendCMD(sock,"getSysVarI", {"addr":n})  
            print ( result )
```

2.2.6.4 Set system I variable value

```
{"jsonrpc": "2.0", "method": "setSysVarI", "params": {"addr": addr, "value": value}, "id": id}
```

Function: Set system I variable value

Parameter: addr: variable address, range: int [0,255]
value: variable value, range: int[-32767,32767]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc)  
        for n in range(0, 11,1):  
            # Set system I variable value  
            suc, result ,id=sendCMD(sock,"setSysVarI",{"addr":i,"value":100})
```

Note: This command is only supported in remote mode.

2.2.6.5 Get system D variable value

```
{"jsonrpc": "2.0", "method": "getSysVarD", "params": {"addr": addr}, "id": id}
```

Function: Get system D variable value

Parameter: addr: variable address, range: int [0,255]

Return: Variable value, range: double[-1e+09,1e+09]

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if(conSuc)  
        for n in range(0, 11,1):  
            # Get system D variable value  
            suc, result , id = sendCMD(sock,"getSysVarD",{"addr":n})  
            print ( result )
```

2.2.6.6 Set system D variable value

```
{"jsonrpc": "2.0", "method": "setSysVarD", "params": {"addr": addr, "value": value}, "id": id}
```

Function: Set system D variable value

Parameter: addr: variable address, range: int [0,255]
value Variable value, range: double[-1e+09,1e+09]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc)  
        for n in range(0, 11,1):  
            # Set system D variable value  
            suc, result ,id=sendCMD(sock,"setSysVarD",{"addr":i,"value":100})
```

Note: This command is only supported in remote mode.

2.2.6.7 Get whether the system P variable is enabled

```
{"jsonrpc": "2.0", "method": "getSysVarPState", "params": {"addr": addr}, "id": id}
```

Function: Get whether the system P variable is enabled

Parameter: addr: variable address, range: int [0,255]

Return: Store the enabled state of the P variable, 0: not enabled, 1: enabled

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc)  
        for i in range(0, 101,1):  
            # Get whether the system P variable is enabled  
            suc, result , id = sendCMD(sock,"getSysVarPState", {"addr": i})
```

2.2.6.8 Get the value of P variable

```
{"jsonrpc": "2.0", "method": "getSysVarP", "params": {"addr": addr}, "id": id}
```

Function: Get the value of P variable

Parameter: addr: variable address, range: int [0,255]

Return: System P variable value double pos[8]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc)
        for i in range(0, 101,1):
            # Get whether the system P variable is enabled
            suc, result, id = sendCMD(sock, "getSysVarPState", {"addr": i})
            if (result == 1):
                # Get system P variable value
                suc, result, id = sendCMD(sock, "getSysVarP", {"addr":i})
                print (result)
```

2.2.6.9 Set the value of the P variable

```
{"jsonrpc": "2.0", "method": "setSysVarP", "params": {"addr": addr, "pos": pos}, "id": id}
```

Function: Set the value of the system P variable

Parameter: addr: variable address, range int[0,255]

pos: the value of the p variable, double pos[6], range [-360,360]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    point = [0, -90, 0, -90, 90, 0]
    if conSuc:
        ret, result, id=sendCMD(sock,"setSysVarP",{"addr":0,"pos":point})
        if ret:
            print (result)
        else:
            print ("err_msg = ", result ["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.15.2 and above.

2.2.6.10 Set the scope of the V variable

```
{"jsonrpc": "2.0", "method": "setSysVarV", "params": {"addr": addr, "pose": pose}, "id": id}
```

Function: Set the value of the system V variable

Parameter: addr: variable address: range int[0,255]
pose: double pose[6], the range of rx,ry,rz is $[-\pi, \pi]$

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    pos = [200, 125.5, -50, 1.57, -1.57, 3.14]
    if conSuc:
        ret, result, id = sendCMD(sock, "setSysVarV", {"addr": 0, "pos": pos})
    if ret:
        print(result)
    else:
        print("err_msg = ", result["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.15.2 and above.

2.2.6.11 Get the value of V variable

```
{"jsonrpc": "2.0", "method": "getSysVarV", "params": {"addr": addr}, "id": id}
```

Function: Get the value of V variable

Parameter: addr: variable address, range: int [0,255]

Return: System V variable value double pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.200"
    conSuc, sock = connectETController(robot_ip)
    if conSuc:
        for i in range(0, 101, 1):
            # Get the value of V variable
            suc, result, id = sendCMD(sock, "getSysVarV", {"addr": i})
            print(result)
```

2.2.6.12 Set the value of the V variable


```
{"jsonrpc": "2.0", "method": "setSysVarV", "params": {"addr": addr, "pose": pose}, "id": id}
```

Function: Set the value of the system V variable

Parameter: addr: variable address, range int[0,255]
pose: the value of the V variable, double pose[6], the range of rx,ry,rz is $[-\pi, \pi]$

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.205"
    conSuc, sock = connectETController(robot_ip)
    pose = [200, 125.5, -50, 0, 0, 0]
    if (conSuc):
        # Set the system V variable value
        suc, result, id = sendCMD(sock, "setSysVarV", {"addr": 0, "pose": [243.5, -219.4, 169.578000, 3.139376, -0.002601, 0.106804]})
        print(suc, result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode.

2.2.6.13 Save variable data

```
{"jsonrpc": "2.0", "method": "save_var_data", "id": id}
```

Function: Save system variable data

Parameter: no

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    pos = [200, 125.5, -50, 1.57, -1.57, 3.14]
    if conSuc:
        ret, result, id = sendCMD(sock, "save_var_data")
        if ret:
            print(result)
        else:
            print("err_msg = ", result["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.15.2 and above.

2.2.7 Transparent Transmission Service

2.2.7.1 Initialize transparent transmission service

```
{"jsonrpc": "2.0", "method": "transparent_transmission_init", "params": {"lookahead": lookahead, "t": t, "smoothness": smoothness, "response_enable": response_enable}, "id": id}
```

Function: Initialize transparent transmission service of the robot

Parameter: lookahead: lookahead time, unit ms, range: int [10,1000]

t: sampling time, unit ms, range: int [2,100]

smoothness: gain, unit percentage, range: double [0,1].

Note: The current version of smoothness is not applicable

response_enable: optional parameter, if not written, there will be a return value by default. int[0,1], whether the add point instruction returns a value, 0: no return value, 1: return value

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Initialize transparent transmission service
        suc, result, id = sendCMD(sock, "transparent_transmission_init", {"lookahead": 400, "t": 10, "smoothness": 0.1, "response_enable": 0})
```

Note: This command is only supported in remote mode.

2.2.7.2 Set the current transparent transmission servo target joint point

```
{"jsonrpc": "2.0", "method": "tt_set_current_servo_joint", "params": {"targetPos": targetPos}, "id": id}
```

Function: Set the current transparent transmission servo target joint point

Parameter: targetpos: target joint point double pos[6], the range is [-360,360]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    # Transparent transmission starting point
    P0 = [0, -90, 0, -90, 90, 0]
    if(conSuc):
        # Initialize transparent transmission service
        suc, result ,id=sendCMD(sock," transparent_transmission_init ",{"lookahead":400,"t":10,"smoothness":0.1})
        # Set the current transparent transmission target joint point
        suc, result ,id=sendCMD(sock," tt_set_current_servo_joint ",{"targetPos": P0})
```

Note: This command is only supported in remote mode. This command is deprecated.

2.2.7.3 Get whether the current robot is in a transparent transmission state

```
{"jsonrpc":"2.0","method":"get_transparent_transmission_state","id":id}
```

Function: Get whether the current robot is in a transparent transmission state

Parameter: None

Return: Current transparent transmission status. 0: non-transparent transmission state, 1: transparent transmission state

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get whether the current robot is in a transparent transmission state
        suc, result ,id=sendCMD(sock," get_transparent_transmission_state ")
```

2.2.7.4 Add transparent transmission of servo target joint point information to the cache

```
{"jsonrpc":"2.0","method":"tt_put_servo_joint_to_buf","params":{"targetPos":targetPos},"id":id} or {"jsonrpc":"2.0","method":"tt_put_servo_joint_to_buf","params":{"targetPose":targetPose},"id":id}
```

Function: Add transparent transmission of servo target joint point information to the cache

Parameter: targetpos: target joint point double pos[6], the range is [-360,360] or targetPose: target pose point is double pos[6]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(ip)
    i = 0
    if(conSuc):
        # Get whether the current robot is in a transparent transmission state
        suc, result ,id=sendCMD(sock," get_transparent_transmission_state ")
        print (suc, result , id)
        if( result == 1):
            # Clear the transparent transmission cache
            suc, result , id = sendCMD(sock,"tt_clear_servo_joint_buf")
            time.sleep(0.5)
        # open a file
        file_name ='D:\\ tttest8 .txt '
        fo = open(file_name, "r")
        while 1:
            # Read each line of the file in turn (point information)
            line = fo.readline ()
            if not line: break
            # Remove the blanks at the beginning and end of each line
            line_list = line.strip ()
            line_list = list(map(float, line_list.split(',') ))
            if (i == 0):
                # Joint movement to the starting point
                suc, result ,id=sendCMD(sock,"moveByJoint",{ "targetPos": line_list ,"speed":30})
                wait_stop() # Wait for the robot to stop
                # Initialize the transparent transmission service
                suc, result ,id=sendCMD(sock," transparent_transmission_init",{ "lookahead":400,
                    "t":10,"smoothness":0.1,"response_enable":1})
                print (suc, result , id)
            # Add transparent transmission of servo target joint point information to the cache
            suc, result , id = sendCMD(sock,"tt_put_servo_joint_to_buf",{ "targetPos ":
                line_list })
            time.sleep(0.01)
            i = i + 1
```

Note: This command is only supported in remote mode.

Choose one of the two parameters targetPos and targetPose, and only one parameter can be sent in a command.

2.2.7.5 Clear the transparent transmission cache

```
{"jsonrpc": "2.0", "method": "tt_clear_servo_joint_buf", "id": id}
```

Function: Clear the transparent transmission cache

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get whether the current robot is in a transparent transmission state
        suc, result ,id =sendCMD(sock," get_transparent_transmission_state ")
        if ( result == 1):
            # Clear the transparent transmission cache
            suc, result ,id=sendCMD(sock," tt_clear_servo_joint_buf")
            time.sleep(0.5)
```

Note: This command is only supported in remote mode.

2.2.7.6 Example 1

```
1 import socket
2 import json
3 import time
4 import random
5
6 def connectETController(ip,port=8055):
7     sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
8     try:
9         sock.connect((ip,port))
10        return (True,sock)
11    except Exception as e:
12        sock.close()
13        return (False,None)
14
15 def disconnectETController(sock):
16     if(sock):
17         sock.close()
18         sock=None
19     else:
```



```

20         sock=None
21
22     def sendCMD(sock,cmd,params=None,id=1):
23         if(not params):
24             params=[]
25         else:
26             params=json.dumps(params)
27         sendStr="{\"method\": \"{0}\", \"params\": {1}, \"jsonrpc\": \"2.0\", \"id\": {2}}\".format(cmd,params,id)+"\n"
28         try:
29             sock.sendall(bytes(sendStr,"utf-8"))
30             ret =sock.recv(1024)
31             jdata=json.loads(str(ret,"utf-8"))
32             if("result" in jdata.keys()):
33                 return (True,json.loads(jdata["result"]),jdata["id"])
34             elif("error" in jdata.keys()):
35                 return (False,jdata["error"],jdata["id"])
36             else:
37                 return (False,None,None)
38         except Exception as e:
39             return (False,None,None)
40
41     def wait_stop():
42         while True:
43             time.sleep(0.01)
44             ret1, result1, id1 = sendCMD(sock, "getRobotState")
45             if (ret1):
46                 if result1 == 0 or result1 == 4:
47                     break
48             else:
49                 print("getRobotState failed")
50                 break
51
52     if __name__ == "__main__":
53         # Robot IP address
54         robot_ip="192.168.1.202"
55         conSuc,sock=connectETController(robot_ip)
56         print(conSuc)
57         if(conSuc):
58             # Get robot status
59             suc, result, id = sendCMD(sock, "getRobotState")
60             if(result == 4):
61                 # Clear alarm

```

```

62         suc, result, id = sendCMD(sock, "clearAlarm")
63         time.sleep(0.5)
64     # Get synchronization status
65     suc, result, id = sendCMD(sock, "getMotorStatus")
66     if(result == 0):
67         # Synchronize servo encoder data
68         suc, result, id = sendCMD(sock, "syncMotorStatus")
69         time.sleep(0.5)
70     # Get the servo status of the robotic arm
71     suc, result, id = sendCMD(sock, "getServoStatus")
72     if (result == 0):
73         # Set the servo status of the robotic arm to ON
74         suc, result, id = sendCMD(sock, "set_servo_status",{"status
75             ":1})
76         time.sleep(1)
77     # Get whether the current robot is in a transparent transmission
78     state
79     suc, result, id = sendCMD(sock, "
80         get_transparent_transmission_state")
81     print(suc, result, id)
82     if(result == 1):
83         # Clear the transparent transmission cache
84         suc, result, id = sendCMD(sock,"tt_clear_servo_joint_buf")
85         time.sleep(0.5)
86     # open a file
87     file_name = 'D:\\tttest8.txt'
88     fo = open(file_name, "r")
89     while 1:
90         # Read each line of the file in turn (point information)
91         line = fo.readline()
92         if not line: break
93         # Remove the blanks at the beginning and end of each line
94         line_list = line.strip()
95         line_list = list(map(float, line_list.split(',')))
96         print(i,line_list)
97         if (i == 0):
98             # Joint movement to the starting point
99             suc, result, id = sendCMD(sock, "moveByJoint", {"
100                 targetPos": line_list, "speed": 30})
101             wait_stop() # Wait for the robot to stop
102             # Initialize the transparent transmission service
103             suc, result, id = sendCMD(sock, "
104                 transparent_transmission_init", {"lookahead": 400, "

```

```

100         t": 10, "smoothness": 0.1, "response_enable": 1})
101         print(suc, result, id)
102         # Add transparent transmission of servo target joint point
103         # information to the cache
104         suc, result, id = sendCMD(sock, "tt_put_servo_joint_to_buf"
105         ,{"targetPos": line_list})
106         time.sleep(0.01)
107         i = i + 1
108         # Close file
109         fo.close()
110         # Clear the transparent transmission cache
111         suc, result, id = sendCMD(sock, "tt_clear_servo_joint_buf")
112         print("clear_ret = ", suc)
else:
    print("Connection failed")
disconnectETController(sock)

```

2.2.7.7 Example 2

```

1  import socket
2  import json
3  import time
4
5  def connectETController(ip,port=8055):
6      sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
7      try:
8          sock.connect((ip,port))
9          return (True,sock)
10     except Exception as e:
11         sock.close()
12         return (False,None)
13
14  def disconnectETController(sock):
15      if(sock):
16          sock.close()
17          sock=None
18      else:
19          sock=None
20
21  def sendCMD(sock,cmd,params=None,id=1):
22      if(not params):
23          params=[]

```




```
24     else:
25         params=json.dumps(params)
26         sendStr="{\\\"method\\\":\\\"{0}\\\",\\\"params\\\":{1},\\\"jsonrpc\\\":\\\"2.0\\\",\\\"
            id\\\":{2}}\".format(cmd,params,id)+"\n"
27     try:
28         sock.sendall(bytes(sendStr,"utf-8"))
29         # print(sock.recv)
30         ret =sock.recv(1024)
31         jdata=json.loads(str(ret,"utf-8"))
32         if("result" in jdata.keys()):
33             return (True,json.loads(jdata["result"]),jdata["id"])
34         elif("error" in jdata.keys()):
35             return (False,jdata["error"],jdata["id"])
36         else:
37             return (False,None,None)
38     except Exception as e:
39         return (False,None,None)
40
41 def send_Point(sock,cmd,params=None,id=1):
42     if(not params):
43         params=[]
44     else:
45         params=json.dumps(params)
46         sendStr="{\\\"method\\\":\\\"{0}\\\",\\\"params\\\":{1},\\\"jsonrpc\\\":\\\"2.0\\\",\\\"
            id\\\":{2}}\".format(cmd,params,id)+"\n"
47         sock.sendall(bytes(sendStr,"utf-8"))
48
49 def wait_stop():
50     while True:
51         time.sleep(0.01)
52         ret1, result1, id1= sendCMD(sock, "getRobotState") #
53         getRobotstate
54         if (ret1):
55             if result1 == 0 or result1 == 4:
56                 break
57         else:
58             print("getRobotState failed")
59             break
60
61 if __name__ == "__main__":
62     # Robot IP address
63     robot_ip="192.168.1.202"
```

```
64     conSuc, sock = connectETController(robot_ip)
65
66     point = []
67     i = 0
68     if(conSuc):
69         # Get robot status
70         suc, result, id = sendCMD(sock, "getRobotState")
71         if(result == 4):
72             # Clear alarm
73             suc, result, id = sendCMD(sock, "clearAlarm")
74             time.sleep(0.5)
75         # Get synchronization status
76         suc, result, id = sendCMD(sock, "getMotorStatus")
77         if(result == 0):
78             # Synchronize servo encoder data
79             suc, result, id = sendCMD(sock, "syncMotorStatus")
80             time.sleep(0.5)
81         # Get the servo status of the robotic arm
82         suc, result, id = sendCMD(sock, "getServoStatus")
83         if (result == 0):
84             # Set the servo status of the robotic arm to ON
85             suc, result, id = sendCMD(sock, "set_servo_status", {"status": 1})
86             time.sleep(1)
87         # Get whether the current robot is in a transparent
            transmission state
88         suc, result, id = sendCMD(sock, "
            get_transparent_transmission_state")
89         print(result)
90         if(result == 1):
91             # Clear the transparent transmission cache
92             suc, result, id = sendCMD(sock, "tt_clear_servo_joint_buf")
93             time.sleep(0.5)
94         # open a file
95         file_name = 'D:\\tttest8.txt'
96         fo = open(file_name, "r")
97         while 1:
98             # Read each line of the file in turn (point information)
99             line = fo.readline()
100             if not line: break
101             # Remove the blanks at the beginning and end of each line
102             line_list = line.strip()
103             line_list = list(map(float, line_list.split(',')))
```

```

104
105         if (i == 0):
106             # Joint movement to the starting point
107             suc, result, id = sendCMD(sock, "moveByJoint", {"
                targetPos": line_list, "speed": 30})
108             print(result)
109             wait_stop() # Wait for the robot to stop
110             print(1)
111             # Initialize the transparent transmission service
112             suc, result, id = sendCMD(sock, "
                transparent_transmission_init", {"lookahead": 400, "
                t": 10, "smoothness": 0.1, "response_enable":0})
113             print(result)
114             # Add transparent transmission of servo target joint point
                information to the cache
115             send_Point(sock, "tt_put_servo_joint_to_buf",{"targetPos":
                line_list})
116             print(result)
117             time.sleep(0.01)
118             i = i + 1
119             # Close file
120             fo.close()
121             # Clear the transparent transmission cache
122             suc, result, id = sendCMD(sock, "tt_clear_servo_joint_buf")
123             print("clear_ret = ", suc)
124         else:
125             print("Connection failed")
126         disconnectETController(sock)

```

2.2.8 System Service(SystemService)

2.2.8.1 Get the controller software version number

```
{ "jsonrpc": "2.0", "method": "getSoftVersion", "id": id }
```

Function: Get the controller software version number

Parameter: None

Return: The controller software version number

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the controller software version number
        suc, result, id = sendCMD(sock,"getSoftVersion")
        print ( result )
```

2.2.8.2 Get servo version number

```
{" jsonrpc ":"2.0" ," method ":" getJointVersion ","params":{"axis":axis
}, "id":id}
```

Function: Get the servo version number

Parameter: axis: range int [0,7], corresponding to axis number 1~8

Return: Servo version number

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get 1 axis servo version number
        ret, result, id = sendCMD(sock,"getJointVersion", {"axis":0})
        if ret:
            print ("result =", result)
        else:
            print ("err_msg=", result ["message"])
```

Note: The servo version for this function is 11 and above

2.2.9 TrajectoryService(TrajectoryService)

2.2.9.1 Initial movement

```
{"jsonrpc":"2.0","method":"start_push_pos","params":{"path_lenth":
    path_lenth,"pos_type":pos_type,"ref_joint_pos":ref_joint_pos,"
    ref_frame":ref_frame,"ret_flag": ret_flag},"id":id}
```

Function: Initial movement

Parameter: path_lenth: the number of transmitted points, int, range: greater than or equal to 3
pos_type: point type, int[0,1], 0: joint, 1: pose
ref_joint_pos: reference point, double pos[6], if the input is a pose point, this reference point is the inverse solution reference point of the first point
ref_frame: coordinate system, double pose[6], if it is based on the base coordinate system, all 0; if the input coordinate is a pose point, this parameter is the coordinate system of the point.
ret_flag: int[0,1], whether the add point instruction returns a value, 0: no return value, 1: return value

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip = "192.168.1.202"  
    conSuc, sock = connectETController(robot_ip)  
    pos = [0, -90, 0, -90, 90, 0]  
    frame = [0, 0, 0, 0, 0, 0]  
    if(conSuc):  
        suc, result ,id=sendCMD(sock,"start_push_pos",{ "path_lenth":10,"pos_type":0,"  
            ref_joint_pos ":pos,"ref_frame":frame," ref_flag ":1})  
        print ( result )
```

Note: The timestamp of the first point of the transmission must be 0.

2.2.9.2 Add movement points

```
{"jsonrpc": "2.0", "method": "push_pos", "params": {"timestamp": timestamp, "pos": pos}, "id": id}
```

Function: Add exercise points

Parameter: timestamp: double, the timestamp of the point (at which time point in the point sequence), unit: s, range: greater than or equal to 0, and increasing
pos: double pose[6], point data

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if(conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print(result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print(result, i)
            joint[0] += 0.01
            res = res + 0.002
```

2.2.9.3 Stop adding movement points

```
{"jsonrpc": "2.0", "method": "stop_push_pos", "id": id}
```

Function: Stop adding timestamp points

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if(conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print (result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print (result, i)
            joint[0] += 0.01
            res = res + 0.002
        ret, result, id = sendCMD(sock, "stop_push_pos")
        print (result)
```

Note: Stop_push_pos and push_pos are corresponding relations. Only when all the points sent by the corresponding push_pos are correct, will it return True, otherwise, it will return False.

2.2.9.4 Check execution status

```
{"jsonrpc": "2.0", "method": "check_trajectory", "id": id}
```

Function: Check the execution status

Parameter: None

Return: int[-3,0], 0: transmission point and time stamp are correct, -1: point length does not match, -2: point format error, -3: time stamp is not standardized.

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if(conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print (result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint })
            print (result, i)
            joint [0] += 0.01
            res = res + 0.002
        ret, result, id = sendCMD(sock, "stop_push_pos")
        print (result)
        ret, result, id = sendCMD(sock, "check_trajectory")
        print (result)
```

2.2.9.5 Start a time-stamped movement

```
{"jsonrpc": "2.0", "method": "start_trajectory", "params": {"speed_percent": speed_percent}, "id": id}
```


Function: Start a time-stamped movement

Parameter: speed_percent: double, trajectory speed percentage, that is, the original speed multiplied by the percentage of speed movement, unit: %, range: greater than or equal to 0.1

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if(conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print (result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print (result, i)
            joint[0] += 0.01
            res = res + 0.002
        ret, result, id = sendCMD(sock, "stop_push_pos")
        print (result)
        ret, result, id = sendCMD(sock, "check_trajectory")
        print (result)
        ret, result, id = sendCMD(sock, "start_trajectory", {"speed_percent": 50})
        print (result)
```

Note: As long as flush_trajectory, or start_push_pos is not executed, the current trajectory can run in a loop without repeated transmission

2.2.9.6 Pause motion

```
{"jsonrpc": "2.0", "method": "pause_trajectory", "id": id}
```

Function: Pause motion

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret, result, id = sendCMD(sock, "pause_trajectory")
        print (result)
```

2.2.9.7 Resume motion

```
{"jsonrpc": "2.0", "method": "resume_trajectory", "id": id}
```

Function: Resume motion

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret, result, id = sendCMD(sock, "resume_trajectory")
        print (result)
```

2.2.9.8 Stop motion

```
{"jsonrpc": "2.0", "method": "stop_trajectory", "id": id}
```

Function: Stop motion

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret, result, id = sendCMD(sock, "stop_trajectory")
        print (result)
```

2.2.9.9 Empty the cache

```
{"jsonrpc": "2.0", "method": "flush_trajectory", "id": id}
```

Function: Empty the cache

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if(conSuc):
        ret, result, id = sendCMD(sock, "flush_trajectory")
        print(result)
```

2.2.9.10 Example 1

```
1 import socket
2 import json
3 import time
4
5 def connectETController(ip, port=8055):
6     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7     try:
8         sock.connect((ip, port))
9         return (True, sock)
10    except Exception as e:
11        sock.close()
12        return (False, None)
13
14 def disconnectETController(sock):
15     if(sock):
16         sock.close()
17         sock=None
18     else:
19         sock=None
20
21 def sendCMD(sock, cmd, params=None, id=1):
22     if(not params):
23         params=[]
24     else:
```



```
25     params=json.dumps(params)
26     sendStr="{\"method\": \"{0}\", \"params\": {1}, \"jsonrpc\": \"2.0\", \"id\": {2} }\".format(cmd,params,id)+\"\\n\"
27     try:
28         sock.sendall(bytes(sendStr,\"utf-8"))
29         # print(sock.recv)
30         ret =sock.recv(1024)
31         jdata=json.loads(str(ret,\"utf-8"))
32         if("result" in jdata.keys()):
33             return (True,json.loads(jdata["result"]),jdata["id"])
34         elif("error" in jdata.keys()):
35             return (False,jdata["error"],jdata["id"])
36         else:
37             return (False,None,None)
38     except Exception as e:
39         return (False,None,None)
40
41 def wait_stop():
42     while True:
43         time.sleep(0.01)
44         ret1, result1, id1 = sendCMD(sock, "getRobotState")
45         if (ret1):
46             if result1 == 0 or result1 == 4:
47                 break
48             else:
49                 print("getRobotState failed")
50                 break
51
52 if __name__ == "__main__":
53     ip = "192.168.1.200"
54     conSuc, sock = connectETController(ip)
55     start_pos = [0, -90, 0, -90, 90, 0]
56     ref_pos = [0, 0, 0, 0, 0, 0]
57     res = 0
58     if conSuc:
59         suc, result, id = sendCMD(sock, "getRobotState")
60         if (result == 4):
61             # Clear alarm
62             suc, result, id = sendCMD(sock, "clearAlarm")
63             time.sleep(0.5)
64             # Get synchronization status
65             suc, result, id = sendCMD(sock, "getMotorStatus")
66             if result != True:
```

```
67         # Synchronize servo encoder data
68         suc, result, id = sendCMD(sock, "syncMotorStatus")
69         time.sleep(0.5)
70     time.sleep(0.5)
71     # Get the servo status of the robotic arm
72     suc, result, id = sendCMD(sock, "getServoStatus")
73     if result == 0:
74         # Set the servo status of the robotic arm to ON
75         ret, result, id = sendCMD(sock, "set_servo_status", {"
            status": 1})
76     suc, result, id = sendCMD(sock, "moveByJoint", {"targetPos":
        start_pos, "speed": 50})
77     wait_stop()
78     suc, result, id = sendCMD(sock, "start_push_pos", {"path_lenth"
        : 10, "pos_type": 0, "ref_joint_pos": start_pos, "ref_frame"
        : ref_pos, "ret_flag": 1})
79     print(result)
80     time.sleep(0.2)
81     for i in range(0, 10):
82         suc, result, id = sendCMD(sock, "push_pos", {"timestamp":
            res, "pos": start_pos})
83         start_pos[0] += 0.02
84         res = res + 0.002
85     time.sleep(1)
86     suc, result, id = sendCMD(sock, "stop_push_pos")
87     print(result)
88     suc, result, id = sendCMD(sock, "check_trajectory")
89     print(result)
90     suc, result, id = sendCMD(sock, "start_trajectory", {"
        speed_percent": 50})
91     print(result)
92     time.sleep(5)
93     suc, result, id = sendCMD(sock, "flush_trajectory")
94     print(result)
95     else:
96         print("Connection failed")
97     disconnectETController(sock)
```

2.2.9.11 Example 2

```
1 import socket
2 import json
3 import time
4
5 def connectETController(ip,port=8055):
6     sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
7     try:
8         sock.connect((ip,port))
9         return (True,sock)
10    except Exception as e:
11        sock.close()
12        return (False,None)
13
14 def disconnectETController(sock):
15     if(sock):
16         sock.close()
17         sock=None
18     else:
19         sock=None
20
21 def sendCMD(sock,cmd,params=None,id=1):
22     if(not params):
23         params=[]
24     else:
25         params=json.dumps(params)
26     sendStr="{\"method\":\"{0}\",\"params\":{\"1},\"jsonrpc\":\"2.0\", \"id\":{\"2} }}\".format(cmd,params,id)+"\n"
27     try:
28         sock.sendall(bytes(sendStr,"utf-8"))
29         # print(sock.recv)
30         ret =sock.recv(1024)
31         jdata=json.loads(str(ret,"utf-8"))
32         if("result" in jdata.keys()):
33             return (True,json.loads(jdata["result"]),jdata["id"])
34         elif("error" in jdata.keys()):
35             return (False,jdata["error"],jdata["id"])
36         else:
37             return (False,None,None)
38     except Exception as e:
39         return (False,None,None)
40
41 def send_Point(sock,cmd,params=None,id=1):
42     if(not params):
```

```
43     params=[]
44     else:
45         params=json.dumps(params)
46         sendStr="{\"method\":\"{0}\",\"params\":{\"1},\"jsonrpc\":\"2.0\", \"
            id\":{\"2} }\".format(cmd,params,id)+"\n"
47         sock.sendall(bytes(sendStr,"utf-8"))
48
49 def wait_stop():
50     while True:
51         time.sleep(0.01)
52         ret1, result1, id1 = sendCMD(sock, "getRobotState")
53         if (ret1):
54             if result1 == 0 or result1 == 4:
55                 break
56             else:
57                 print("getRobotState failed")
58                 break
59
60 if __name__ == "__main__":
61     ip = "192.168.1.200"
62     conSuc, sock = connectETController(ip)
63     start_pos = [0, -90, 0, -90, 90, 0]
64     ref_pos = [0, 0, 0, 0, 0, 0]
65     res = 0
66     if conSuc:
67         suc, result, id = sendCMD(sock, "getRobotState")
68         if (result == 4):
69             # Clear alarm
70             suc, result, id = sendCMD(sock, "clearAlarm")
71             time.sleep(0.5)
72             # Get synchronization status
73             suc, result, id = sendCMD(sock, "getMotorStatus")
74             if result != True:
75                 # Synchronize servo encoder data
76                 suc, result, id = sendCMD(sock, "syncMotorStatus")
77                 time.sleep(0.5)
78             time.sleep(0.5)
79             # Get the servo status of the robotic arm
80             suc, result, id = sendCMD(sock, "getServoStatus")
81             if result == 0:
82                 # Set the servo status of the robotic arm to ON
83                 ret, result, id = sendCMD(sock, "set_servo_status", {"
                    status": 1})
```

```

84         suc, result, id = sendCMD(sock, "moveByJoint", {"targetPos":
            start_pos, "speed": 50})
85     wait_stop()
86     suc, result, id = sendCMD(sock, "start_push_pos", {"path_lenth"
            : 10, "pos_type": 0, "ref_joint_pos": start_pos, "ref_frame"
            : ref_pos, "ret_flag": 0})
87     print(result)
88     time.sleep(0.2)
89     for i in range(0, 10):
90         send_Point(sock, "push_pos", {"timestamp": res, "pos":
            start_pos})
91         start_pos[0] += 0.02
92         res = res + 0.002
93     time.sleep(1)
94     suc, result, id = sendCMD(sock, "stop_push_pos")
95     print(result)
96     suc, result, id = sendCMD(sock, "check_trajectory")
97     print(result)
98     suc, result, id = sendCMD(sock, "start_trajectory", {"
            speed_percent": 50})
99     print(result)
100    time.sleep(5)
101    suc, result, id = sendCMD(sock, "flush_trajectory")
102    print(result)
103    else:
104        print("Connection failed")
105    disconnectETController(sock)

```

2.2.10 ProfinetService(ProfinetService)

2.2.10.1 Get the value of the profinet int input register

```

{"jsonrpc": "2.0", "method": "get_profinet_int_input_registers", "params": {"
    addr": addr, "length": length}, "id": id}

```


Function: Get the value of the profinet int input register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the profinet int input register
        suc, result, id = sendCMD(sock, "get_profinet_int_input_registers", {"addr":0, "length":1})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.10.2 Get the value of the profinet int output register

```
{"jsonrpc": "2.0", "method": "get_profinet_int_output_registers", "params": {"addr": addr, "length": length}, "id": id}
```

Function: Get the value of the profinet int output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the profinet int output register
        suc, result, id = sendCMD(sock, "get_profinet_int_output_registers", {"addr":1, "length":2})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.10.3 Get the value of the profinet float type input register

```
{"jsonrpc":"2.0","method":"get_profinet_float_input_registers","params": {"addr":addr,"length":length},"id":id}
```

Function: Get the value of the profinet float type input register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the profinet float type input register
        suc, result, id = sendCMD(sock, "get_profinet_float_input_registers", {"addr":0,"length":1})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.10.4 Get the value of the profinet float type output register

```
{"jsonrpc":"2.0","method":"get_profinet_float_output_registers","params": {"addr":addr,"length":length},"id":id}
```

Function: Get the value of the profinet float type output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the profinet float type output register
        suc, result, id = sendCMD(sock, "get_profinet_float_output_registers", {"addr": 0, "length": 1})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.10.5 Set the value of the profinet int output register

```
{"jsonrpc": "2.0", "method": "set_profinet_int_output_registers", "params": {"addr": addr, "length": length, "value": value}, "id": id}
```

Function: Set the value of the profinet int output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

value: list of register values, type int[length], element range [-2147483648, 2147483647]

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.0.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Set the profinet int output register
        suc, result, id = sendCMD(sock, "set_profinet_int_output_registers", {"addr": 1, "length": 2, "value": [1, 1]})
        print(result)
    else:
        print("Connection failed")
```

Note: This command is only supported in remote mode.

2.2.10.6 Set the value of the profinet float output register

```
{"jsonrpc": "2.0", "method": "set_profinet_int_output_registers", "params": {"addr": addr, "length": length, "value": value}, "id": id}
```

Function: Set the value of the profinet float type output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

value: list of register values, type double[length], element range [-3.40E+38,3.40E+38]

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.0.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Set the profinet float type output register
        suc, result, id = sendCMD(sock, "set_profinet_float_output_registers", {"addr": 0, "length": 2, "value": [1,1]})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode.

2.2.11 Backdrive service

2.2.11.1 Get the opening of the servo brake

```
{"jsonrpc": "2.0", "method": "get_servo_brake_off_status", "id": id}
```

Function: Get the opening of the servo brake

Parameter: none

Return: Servo brake open condition array

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "get_servo_brake_off_status")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in reverse drive mode.

2.2.11.2 Get whether it is in reverse drive mode

```
{"jsonrpc": "2.0", "method": "get_backdrive_status", "id": id}
```

Function: Get whether it is in backdrive mode

Parameter: none

Return: 1: Backdrive mode, 0: Not in backdrive mode

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "get_backdrive_status")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

2.2.11.3 Enter backdrive mode

```
{"jsonrpc": "2.0", "method": "enter_backdrive", "id": id}
```

Function: Enter backdrive mode

Parameter: none

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "enter_backdrive")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode, and the robot must be in the reset state.

2.2.11.4 Exit backdrive mode

```
{"jsonrpc": "2.0", "method": "exit_backdrive", "id": id}
```

Function: Exit backdrive mode

Parameter: none

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "exit_backdrive")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode, and the robot must be in reverse drive mode.

2.2.11.5 Reset controller state

```
{"jsonrpc": "2.0", "method": "reset_robot_status", "id": id}
```

Function: Reset controller state

Parameter: none

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "reset_robot_status")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: The robot must be in a stopped or error state (excluding emergency stop alarm).
This command is only supported in remote mode.

2.2.12 Ethernet/IP

2.2.12.1 Get the value of the Ethernet/IP int input register

```
{"jsonrpc": "2.0", "method": "get_eip_int_input_registers", "params": {"addr": addr, "length": length}, "id": id}
```

Function: Get the value of the Ethernet/IP int input register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the Ethernet/IP int input register
        suc, result, id = sendCMD(sock, "get_eip_int_input_registers", {"addr": 0, "length": 1})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

2.2.12.2 Get the value of the Ethernet/IP int output register

```
{"jsonrpc": "2.0", "method": "get_eip_int_output_registers", "params": {"addr": addr, "length": length}, "id": id}
```

Function: Get the value of the Ethernet/IP int output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the Ethernet/IP int output register
        suc, result, id = sendCMD(sock, "get_eip_int_output_registers", {"addr": 1, "length": 2})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

2.2.12.3 Get the value of the Ethernet/IP float type input register

```
{"jsonrpc": "2.0", "method": "get_eip_float_input_registers", "params": {"addr": addr, "length": length}, "id": id}
```

Function: Get the value of the Ethernet/IP float input register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values float [length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the Ethernet/IP float input register
        suc, result, id = sendCMD(sock, "get_eip_float_input_registers", {"addr": 0, "length": 1})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

2.2.12.4 Get the value of the Ethernet/IP float type output register

```
{"jsonrpc":"2.0","method":"get_eip_float_output_registers","params":{"addr":addr,"length":length},"id":id}
```

Function: Get the value of the Ethernet/IP float output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

Return: list of register values float [length]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the value of the Ethernet/IP float output register
        suc, result, id = sendCMD(sock, "get_eip_float_output_registers", {"addr":0, "length":1})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

2.2.12.5 Set the value of the Ethernet/IP int output register

```
{"jsonrpc": "2.0", "method": "set_eip_int_output_registers", "params": {"addr": addr, "length": length}, "id": id}
```

Function: Set the value of the Ethernet/IP int output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

value: list of register values, type int[length], element range [-2147483648, 2147483647]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Set the Ethernet/IP int output register
        suc, result, id = sendCMD(sock, "set_eip_int_output_registers", {"addr": 1, "length": 2, "value": [1,1]})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

2.2.12.6 Set the value of the Ethernet/IP float output register

```
{"jsonrpc": "2.0", "method": "set_eip_float_output_registers", "params": {"addr": addr, "length": length}, "id": id}
```

Function: Set the value of the Ethernet/IP float output register

Parameter: addr: register start address, range int[0,31]

length: number of registers, range int[1,32]

Note: the sum of addr and length should be less than or equal to 32

value: list of register values, type double[length], element range [-3.40E+38, 3.40E+38]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Set the Ethernet/IP float output register
        suc, result, id = sendCMD(sock, "set_eip_float_output_registers", {"addr": 1, "length": 2, "value": [1.1, 1.2]})
        print(result)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command can be used only in the remote mode and is applicable to v3.5.2 and above.

2.2.13 External force sensor service

2.2.13.1 Mark the start of the torque data transfer

```
{"jsonrpc": "2.0", "method": "start_push_force", "params": {"return_flag":  
    rerurn_flag}, "id": id}
```

Function: Mark the start of the torque data transfer

Parameter: return_flag: optional parameter, if it is not entered, then there is a return value by default. The range is int[0,1]. If the value of the parameter start_push_force is set to 0, then there is no return value. If the value of the parameter start_push_foce is set to 1, then there is a return value.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":  
    # Robot IP address  
    robot_ip="192.168.1.200"  
    conSuc,sock=connectETController(robot_ip)  
    if (conSuc):  
        # Mark the start of the torque data transfer  
        suc, result , id=sendCMD(sock,"start_push_force",{"return_flag":1})
```

Note: The function can be used only when the data source of the force control is SDK.

2.2.13.2 Transfer the torque data

```
{"jsonrpc": "2.0", "method": "push_external_force", "params": {"index": index, "torque_array": torque_array}, "id": id}
```

Function: Transfer the torque data

Parameter: index: serial no., indicates the transfer sequence, range is [0, 65535]

torque_array*: array of torque data, double torques[6]

*The value of the parameter index shall be gradually increased from 0 to 65535 after a complete transfer process is started. When it reaches the maximum, the counter will restart from 0. The parameter torque_array will be marked as the latest data by repeating this process. The definitions of the parameter torque_array are given as below: if the data name is the force in the X-axis direction and the data type is double, it indicates the force in the X-axis direction under the output coordinate system of the force sensor; if the data name is the force in the Y-axis direction and the data type is double, it indicates the force in the Y-axis direction under the output coordinate system of the force sensor; if the data name is the force in the Z-axis direction and the data type is double, it indicates the force in the Z-axis direction under the output coordinate system of the force sensor; if the data name is the torque on the X axis and the data type is double, it indicates the torque on the X axis under the output coordinate system of the force sensor; if the data name is the torque on the Y axis and the data type is double, it indicates the torque on the Y axis under the output coordinate system of the force sensor; if the data name is the torque on the Z axis and the data type is double, it indicates the torque on the Z axis under the output coordinate system of the force sensor. The unit of the force in the X, Y, Z-axis direction is kg and the unit of the torque is kgM. If the unit of the original data and the coordinate system are different from the defined ones, please change the parameter first and then pass in them.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Mark the start of the torque data transfer
        suc, result ,id=sendCMD(sock,"start_push_force",{ "return_flag":1})
        if result==True:
            # Transfer the torque data
            suc, result ,id=sendCMD(sock,"push_external_force",{ "index":0,"torque_array":[1,2,3,4,5,6]})
```

Note: Please call the parameter start_push_force first to mark the start of the external torque data first before using this function. The function can be used only when the data source of the force control is SDK.

2.2.13.3 Stop the transfer of the current torque data

```
{"jsonrpc": "2.0", "method": "stop_push_force", "id": id}
```

Function: End the transfer of the current torque data

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Mark the start of the torque data transfer
        suc, result ,id=sendCMD(sock,"start_push_force",{ "return_flag":1})
        if result==True:
            # Transfer the torque data
            suc, result ,id=sendCMD(sock,"push_external_force",{ "index":0,"torque_arry":[1,2,3,4,5,6]})
            # Stop the transfer of the torque data
            suc, result ,id=sendCMD(sock,"stop_push_force")
```

Note: The function can be used only when the data source of the force control is SDK.

2.2.13.4 Get the source of the current torque data

```
{"jsonrpc": "2.0", "method": "get_force_ctrl_mode", "id": id}
```

Function: Get the source of the current torque data

Parameter: none

Return: int[0,4], 0 and 1 indicate that the torque data is from the inside, 2 indicates that the torque data is from SDK, 3 indicates that the torque data is from LUA 4 indicates that the torque data is from the terminal end

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the source of the current torque data
        suc, result ,id=sendCMD(sock,"get_force_ctrl_mode")
        print ( result )
```

2.2.13.5 Example

```
1 import socket
2 import json
3 import os
4 import time import sleep
5
6 def connectETController(ip="192.168.1.200", port=8055):
7     sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8     try:
9         sock1.connect((Ip, port))
10        return True, sock1
11    except Exception as e:
12        print('error is',e)
13        sock1.close()
14        return False, None
15
16 def disconnectETController(sock1):
17     if (sock1):
18         sock1.close()
19         sock1 = None
20     else:
21         sock1 = None
22
23 def sendCMD(sock1, cmd, params=None, id=1):
24     if (not params):
25         params = []
26     else:
27         params = json.dumps(params)
28     sendStr = "{\\"method\\":\\"{0}\\",\\"params\\":{1},\\"jsonrpc\\":\\"2.0\\",\\"id\\":{2}}".format(cmd, params, id) + "\n"
29     try:
30         # print(sendStr)
31         sock1.sendall(bytes(sendStr, "utf-8"))
32         ret = sock1.recv(1024)
33         jData = json.loads(str(ret, "utf-8"))
34         # print("raw data:",jData)
35         if "result" in jData.keys():
36             return True, json.loads(jData["result"]), jData["id"]
37         elif "error" in jData.keys():
38             return False, jData["error"], jData["id"]
```

```
38         else:
39             return False, None, None
40     except Exception as e:
41
42         return False, None, None
43
44 if __name__ == "__main__":
45     robot_ip = "192.168.1.200"
46     return_flag = True
47
48     conSuc, sock = connectETController(robot_ip)
49     if conSuc:
50         ret, result, id = sendCMD(sock, "get_force_ctrl_mode")
51         if result != 2: # sdk mode = 2
52             print("please change the mode to sdk.")
53             exit()
54
55         ret, result, id = sendCMD(sock, "start_push_force",
56                                   {"return_flag":return_flag})
57         if result != True:
58             exit()
59
60         # torque data array
61         array = [1.2,2.3,3.2,4.4,5.6,6.9]
62         index = 0
63         cycle_time = 0.005 #interval(ms)
64         host_time = 5 #last time(s)
65         while loopCnt > 0:
66             ret, result, id = sendCMD(sock, "push_external_force",)
67             {"index":index,"torque_array":array})
```

```
66         if result != True:
67             print("push_external_force failed!", result)
68             break
69
70         sleep(cycle_time)
71         loopCnt -= 1
72         array[0] += 1
73         index += 1
74
75         ret, result, id = sendCMD(sock, "stop_push_force")
76     else:
77         print("Connection failed")
78
79     disconnectETController(sock)
```

2.2.14 Force Control Service

2.2.14.1 Start the force control

```
{ "jsonrpc": "2.0", "method": "start_force_mode", "params": { "mode": 3,
    "arr_frame": frame, "arr_optional": [6, 1, 5, 1, 0, 1], "arr_torque":
    [-10, -10, -10, 1.3, 1.30, 1.30], "arr_speed": speed }, "id": id }
```

Function: Start the force control mode

Parameter: mode: mode, optional parameter, int[0,4], The numbers mean the fixed mode, the point mode, the motion mode, the TCP mode and the pose mode. The default is 0.
arr_frame: the user-specified force control coordinate system, optional parameter, unit of x, y and z is mm, range is [-10000,10000], unit of Rx, Ry and Rz is radian, range is $[-\pi, \pi]$. The default is [0,0,0,0,0,0] and the the force control coordinate system will be concentric with the TCP coordinate system.
arr_optional: force control method (mask of the DOFs), optional parameter, the value range is int[0,6] and the numbers mean the motion control, the force tracking, the fixed mode, the floating mode, the spring mode, the floating mode & motion control, the spring mode & motion control. The default is [0,0,0,0,0,0] and it indicates the motion control.
arr_torque: target torque, optional parameter, double arr_torque[6], the first three numbers mean the force, the torque range of EC63 is [-30,30]N, the torque range of EC66 is [-60,60]N, the torque range of EC612 is [-120,120]N and the torque range of EC616 is [-160,160]N; the last three numbers mean the torque and the range is [-1.5,1.5]N. The default is [0,0,0,0,0,0]
arr_speed: speed limit, optional parameter, unit is mm/s and °/s, double arr_speed[6], the first three numbers mean the linear speed, range is [0,200]; the last three numbers mean the angular speed, range is [0,11]. The default is [100, 100,100, 5.73, 5.73, 5.73]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        suc, result , id=sendCMD(sock,"start_force_mode",{ "mode":0, "arr_frame":[0,0,0,0,0,0],
            "arr_optional":[1,0,0,0,0,0], "arr_torque":[1,0,0,0,0,0], "arr_speed":[100,100,100,5.73,
            5.73,5.73]})
        print (suc, result , id)
    else:
        print ("Connection failed")
    disconnectETController (sock)
```

Note: The force control function can be enabled by using this command. In the process, users can dynamically configure the related parameters.

When the force control mode is started, the jog movement is executed and the force control mode will be automatically ended.

It is impossible to start the force control mode and configure the parameters when the robot is moving.

The drag function and the force control function are mutually exclusive. They cannot work together.

Currently, users can only select the fixed mode and the TCP mode. This command is applicable to v3.6.2 and above.

2.2.14.2 End the force control mode

```
{"jsonrpc": "2.0", "method": "end_force_mode", "id": id}
```

Function: End the force control mode

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        suc, result ,id=sendCMD(sock,"end_force_mode")
        print (suc, result ,id)
    else:
        print ("Connection failed")
    disconnectETController (sock)
```

Note: Once the force control mode is started, it is impossible to end the mode during the robotic movement and the robot will stop running after sending this command.

This command is applicable to v3.6.2 and above.

2.2.14.3 Get the state of the force control mode

```
{"jsonrpc": "2.0", "method": "get_force_mode_state", "id": id}
```

Function: End the force control mode

Parameter: none

Return: True means that the robot is in the force control mode, false means that it is not in the force control mode

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        suc, result ,id=sendCMD(sock,"get_force_mode_state")
        print(suc, result ,id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.6.2 and above.

2.3 Examples

2.3.1 Example 1

```
1 import socket
2 import json
3 import time
4 import random
5
6 def connectETController(ip,port=8055):
7     sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
8     try:
9         sock.connect((ip,port))
10        return (True,sock)
11    except Exception as e:
12        sock.close()
13        return (False,None)
14
15 def disconnectETController(sock):
16     if(sock):
17         sock.close()
18         sock=None
19     else:
20         sock=None
```



```
21
22 def sendCMD(sock,cmd,params=None,id=1):
23     if(not params):
24         params=[]
25     else:
26         params=json.dumps(params)
27     sendStr="{\"method\":\"{0}\",\"params\":{\"1},\"jsonrpc\":\"2.0\", \"
        id\":{\"2}}\".format(cmd,params,id)+"\n"
28     try:
29         sock.sendall(bytes(sendStr,"utf-8"))
30         ret =sock.recv(1024)
31         jdata=json.loads(str(ret,"utf-8"))
32         if("result" in jdata.keys()):
33             return (True,json.loads(jdata["result"]),jdata["id"])
34         elif("error" in jdata.keys()):
35             return (False,jdata["error"],jdata["id"])
36         else:
37             return (False,None,None)
38     except Exception as e:
39         return (False,None,None)
40
41 if __name__ == "__main__":
42     # Robot IP address
43     robot_ip="192.168.1.202"
44     conSuc,sock=connectETController(robot_ip)
45     print(conSuc)
46     if(conSuc):
47         # Clear alarm
48         ret, result, id = sendCMD(sock, "clearAlarm")
49         print("Clear alarm")
50         print("ret = ", ret, "", "id = ", id)
51         if (ret == True):
52             print("result = ", result)
53             time.sleep(1)
54         else:
55             print("err_msg = ", result["message"])
56         # Get synchronization status
57         ret, result, id = sendCMD(sock, "getMotorStatus")
58         print("Get synchronization status")
59         print("ret = ", ret, "", "id = ", id)
60         if (ret == True):
61             print("result = ", result)
62             if(result != 1):
```



```

63         # Synchronize
64         ret1, result1, id = sendCMD(sock, "syncMotorStatus")
65         print("synchronization")
66         print("ret = ", ret1, "", "id = ", id)
67         if (ret1 == True):
68             print("result = ", result1)
69             time.sleep(0.5)
70         else:
71             print("err_msg = ", result1["message"])
72     else:
73         print("err_msg = ", result["message"])
74
75
76     # Turn on the servo
77     ret, result, id = sendCMD(sock, "set_servo_status", {"status"
78         :1})
79     print("Turn on the servo")
80     print("ret = ", ret, "", "id = ", id)
81     if (ret == True):
82         print("result = ", result)
83         time.sleep(1)
84     else:
85         print("err_msg = ", result["message"])
86     # Get servo status
87     ret, result, id = sendCMD(sock, "getServoStatus")
88     print("ret = ", ret, "", "id = ", id)
89     if (ret == True):
90         print("result = ", result)
91     else:
92         print("err_msg = ", result["message"])
93 else:
94     print("Connection failed")
95     disconnectETController(sock)

```

2.3.2 Example 2

```

1  import socket
2  import json
3  import time
4
5
6  # v1.2

```



```
7
8 def connectETController(ip, port=8055):
9     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10    try:
11        sock.connect((ip, port))
12        return (True, sock)
13    except Exception as e:
14        sock.close()
15        return (False, None)
16
17
18 def disconnectETController(sock):
19     if (sock):
20         sock.close()
21         sock = None
22     else:
23         sock = None
24
25
26 def sendCMD(sock, cmd, params=None, id=1):
27     if (not params):
28         params = []
29     else:
30         params = json.dumps(params)
31     sendStr = "{ \"method\": \"{0}\", \"params\": {1}, \"jsonrpc\"
32               \": \"2.0\", \"id\": {2} }\".format(cmd, params, id) + \"\n\"
33     try:
34         # print(sendStr)
35         sock.sendall(bytes(sendStr, \"utf-8\"))
36         ret = sock.recv(1024)
37         jdata = json.loads(str(ret, \"utf-8\"))
38         if (\"result\" in jdata.keys()):
39             return (True, json.loads(jdata[\"result\"]), jdata[\"id\"])
40         elif (\"error\" in jdata.keys()):
41             return (False, jdata[\"error\"], jdata[\"id\"])
42         else:
43             return (False, None, None)
44     except Exception as e:
45         return (False, None, None)
46
47 if __name__ == \"__main__\":
48     ip = \"192.168.1.205\"
```

```
49     conSuc, sock = connectETController(ip)
50     # print(conSuc)
51     if (conSuc):
52         # Get robot status
53         ret, result, id = sendCMD(sock, "getRobotState")
54         print("Get robot status")
55         print("ret = ", ret, " ", "id = ", id)
56         if (ret == True):
57             print("result = ", result)
58         else:
59             print("err_msg = ", result["message"])
60         # Get robot mode
61         ret, result, id = sendCMD(sock, "getMotorStatus")
62         print("Get robot mode")
63         print("ret = ", ret, " ", "id = ", id)
64         if (ret == True):
65             print("result = ", result)
66         else:
67             print("err_msg = ", result["message"])
68         # Get the current position information of the robot
69         ret, result, id = sendCMD(sock, "get_joint_pos")
70         print("Get the current position information of the robot")
71         print("ret = ", ret, " ", "id = ", id)
72         if (ret == True):
73             print("result = ", result)
74         else:
75             print("err_msg = ", result["message"])
76         # Get the current pose information of the robot
77         print("Get the current pose information of the robot")
78         ret, result, id = sendCMD(sock, "get_tcp_pose")
79         print("ret = ", ret, " ", "id = ", id)
80         if (ret == True):
81             print("result = ", result)
82         else:
83             print("err_msg = ", result["message"])
84         # Get the value of analog input
85         ret, result, id=sendCMD(sock, "getAnalogInput", {"addr":1})
86         print("Get the value of analog input")
87         print("ret = ", ret, " ", "id = ", id)
88         if (ret == True):
89             print("result = ", result)
90         else:
91             print("err_msg = ", result["message"])
```

```
92     else:
93         print("Connection failed")
94     disconnectETController(sock)
```

2.3.3 Example 3

```
1  import socket
2  import json
3  import time
4
5
6
7  def connectETController(ip, port=8055):
8      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9      try:
10         sock.connect((ip, port))
11         return (True, sock)
12     except Exception as e:
13         sock.close()
14         return (False, None)
15
16
17 def disconnectETController(sock):
18     if (sock):
19         sock.close()
20         sock = None
21     else:
22         sock = None
23
24
25 def sendCMD(sock, cmd, params=None, id=1):
26     if (not params):
27         params = []
28     else:
29         params = json.dumps(params)
30     sendStr = "{\\"method\\":\\"{0}\\",\\"params\\":{1},\\"jsonrpc\\":\\"2.0\\",\\"id\\":{2}}".format(cmd, params, id) + "\n"
31     try:
32         # print(sendStr)
33         sock.sendall(bytes(sendStr, "utf-8"))
34         ret = sock.recv(1024)
35         jdata = json.loads(str(ret, "utf-8"))
```



```
36         if ("result" in jdata.keys()):
37             return (True, json.loads(jdata["result"]), jdata["id"])
38         elif ("error" in jdata.keys()):
39             return (False, jdata["error"], jdata["id"])
40         else:
41             return (False, None, None)
42     except Exception as e:
43         return (False, None, None)
44
45
46 if __name__ == "__main__":
47     ip = "192.168.1.205"
48     conSuc, sock = connectETController(ip)
49     # print(conSuc)
50     if (conSuc):
51         # Switch the current tool number to 0
52         ret, result, id = sendCMD(sock, "setToolNumber", {"tool_num":
53             0})
54         print("Switch the current tool number to 0")
55         print("ret = ", ret, " ", "id = ", id)
56         if (ret == True):
57             print("result = ", result)
58             time.sleep(3)
59         else:
60             print("err_msg = ", result["message"])
61             # Set the load and center of gravity of the robotic arm
62             ret, result, id = sendCMD(sock, "cmd_set_payload", {"tool_num":
63                 0, "m": 6, "cog": [20.2, 40, 30.5]})
64             print("Set the load and center of gravity of the robotic arm")
65             print("ret = ", ret, " ", "id = ", id)
66             if (ret == True):
67                 print("result = ", result)
68             else:
69                 print("err_msg = ", result["message"])
70                 # Set up the tool center of the robotic arm
71                 point1 = [1.002, -2.5, 5.0, 0.74, -1.57, 0]
72                 ret, result, id = sendCMD(sock, "cmd_set_tcp", {"tool_num": 0,
73                     "point": point1}) # cmd_set_tcp
74                 print("ret = ", ret, " ", "id = ", id)
75                 if (ret == True):
76                     print("result = ", result)
77                 else:
78                     print("err_msg = ", result["message"])
```

```
76     else:
77         print("Connection failed")
78     disconnectETController(sock)
```

Chapter 3 Monitor Interface

Users can get robot information by connecting to monitor interface of the robot through the socket client.

WARNING



On the teach pendant interface, select “Run Prepare > Robot Configuration” and check “Remote” under “Common sys” to use this function.

CAUTION



This function is applicable to 2.13.1 and above versions.

CAUTION



8056 is a (tcp) long-connection interface. If the access time exceeds 8ms, the data will pile up in the buffer zone.

3.1 Data description list of monitor interface

Name	Type	Byte	Description
Message Size	unsigned int32	4*1	The current packet length is 1024, the effective field length is 799, and the reserved field length is 22
timestamp	unsigned int64	8*1	timestamp, the number of milliseconds since January 1, 1970
autorun_cycleMode	unsigned char	1*1	cycle mode, 0: single step, 1: single cycle, 2: continuous
machinePos	double[AXIS_COUNT]	8*8	joint angle, unit degree
machinePose	Double[6]	8*6	Base coordinates, the first three items are in millimeters, the last three items are in radians
machineUserPose	Double[6]	8*6	Current user coordinates, the first three items are in millimeters, the last three items are in radians

Name	Type	Byte	Description
torque	double[AXIS_COUNT]	8*8	The ratio per thousand of the rated torque of the joint, unit ‰,
robotState	int32_t	4*1	Robot state: 0: stop, 1: pause, 2: emergency stop, 3: run, 4: alarm
servoReady	int32_t	4*1	Brake status: 0: not open, 1: open.
can_motor_run	int32_t	4*1	Synchronization status: 0: not synchronized, 1: synchronized
motor_speed	int[AXIS_COUNT]	4*8	Motor speed, unit: rpm
robotMode	int32_t	4*1	Robot mode: 0: teaching mode, 1: automatic mode, 2: remote mode
analog_ioInput	double[ANALOG_IN_NUM]	8*3	Analog input port data, unit V
analog_ioOutput	double[ANALOG_OUT_NUM]	8*5	Analog output port data, unit V
digital_ioInput	unsigned int64	8*1	digital input port data
digital_ioOutput	unsigned int64	8*1	digital output port data
collision	unsigned char	1*1	collision alarm state, 0: non-collision alarm state, 1: collision alarm state.
machineFlangePose	Double[6]	8*6	The pose of the flange center in the Base coordinate system, the first three items are in millimeters, and the last three are in radians.
machineUserFlange Pose	Double[6]	8*6	The pose of the flange center in the user coordinate system, the first three items are in millimeters, and the last three are in radians. v2.14.4 added
emergencyStopState	unsigned char	1*1	Whether it is an emergency stop state, v2.16.2 added
tcpSpeed	Double	8*1	tcp movement speed, unit mm/s, v2.16.2 added
jointSpeed	Double[AXIS_COUNT]	8*8	joint speed, unit degree/s, v2.16.2 added
tcpAcc	Double	8*1	tcp acceleration, unit: mm/s ²

Name	Type	Byte	Description
jointAcc	Double[ANALOG_OUT_NUM]	8*8	joint acceleration, unit: degree/s ²
jointTemperature	Double[6]	8*6	joint temperature, unit: °C
jointTorque	Double[6]	8*6	joint output torque, unit: Nm
Reserved	unsigned char	1*321	The reserved length of the data packet, the length is 321

3.2 Example

```

1  import socket
2  import struct
3  import collections
4  import time
5  import math
6  HOST = "192.168.1.202"
7  PORT = 8056
8
9  while 1:
10     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11     s.settimeout(8)
12     s.connect((HOST, PORT))
13     index = 0
14     lost = 0
15
16     while True:
17         dic = collections.OrderedDict()
18         dic['MessageSize'] = 'I'
19         dic['TimeStamp'] = 'Q'
20         dic['autorun_cycleMode'] = 'B'
21         dic['machinePos01'] = 'd'
22         dic['machinePos02'] = 'd'
23         dic['machinePos03'] = 'd'
24         dic['machinePos04'] = 'd'
25         dic['machinePos05'] = 'd'
26         dic['machinePos06'] = 'd'
27         dic['machinePos07'] = 'd'

```

```
28         dic['machinePos08'] = 'd'
29         dic['machinePose01'] = 'd'
30         dic['machinePose02'] = 'd'
31         dic['machinePose03'] = 'd'
32         dic['machinePose04'] = 'd'
33         dic['machinePose05'] = 'd'
34         dic['machinePose06'] = 'd'
35         dic['machineUserPose01'] = 'd'
36         dic['machineUserPose02'] = 'd'
37         dic['machineUserPose03'] = 'd'
38         dic['machineUserPose04'] = 'd'
39         dic['machineUserPose05'] = 'd'
40         dic['machineUserPose06'] = 'd'
41         dic['torque01'] = 'd'
42         dic['torque02'] = 'd'
43         dic['torque03'] = 'd'
44         dic['torque04'] = 'd'
45         dic['torque05'] = 'd'
46         dic['torque06'] = 'd'
47         dic['torque07'] = 'd'
48         dic['torque08'] = 'd'
49         dic['robotState'] = 'i'
50         dic['servoReady'] = 'i'
51         dic['can_motor_run'] = 'i'
52         dic['motor_speed01'] = 'i'
53         dic['motor_speed02'] = 'i'
54         dic['motor_speed03'] = 'i'
55         dic['motor_speed04'] = 'i'
56         dic['motor_speed05'] = 'i'
57         dic['motor_speed06'] = 'i'
58         dic['motor_speed07'] = 'i'
59         dic['motor_speed08'] = 'i'
60         dic['robotMode'] = 'i'
61         dic['analog_ioInput01'] = 'd'
62         dic['analog_ioInput02'] = 'd'
63         dic['analog_ioInput03'] = 'd'
64         dic['analog_ioOutput01'] = 'd'
65         dic['analog_ioOutput02'] = 'd'
66         dic['analog_ioOutput03'] = 'd'
67         dic['analog_ioOutput04'] = 'd'
68         dic['analog_ioOutput05'] = 'd'
69         dic['digital_ioInput'] = 'Q'
70         dic['digital_ioOutput'] = 'Q'
```

```
71         dic['collision'] = 'B'
72         dic['machineFlangePose01'] = 'd'
73         dic['machineFlangePose02'] = 'd'
74         dic['machineFlangePose03'] = 'd'
75         dic['machineFlangePose04'] = 'd'
76         dic['machineFlangePose05'] = 'd'
77         dic['machineFlangePose06'] = 'd'
78         dic['machineUserFlangePose01'] = 'd'
79         dic['machineUserFlangePose02'] = 'd'
80         dic['machineUserFlangePose03'] = 'd'
81         dic['machineUserFlangePose04'] = 'd'
82         dic['machineUserFlangePose05'] = 'd'
83         dic['machineUserFlangePose06'] = 'd'
84         dic["emergencyStopState"] = "B"
85         dic["tcp_speed"] = "d"
86         dic["joint_speed01"] = "d"
87         dic["joint_speed02"] = "d"
88         dic["joint_speed03"] = "d"
89         dic["joint_speed04"] = "d"
90         dic["joint_speed05"] = "d"
91         dic["joint_speed06"] = "d"
92         dic["joint_speed07"] = "d"
93         dic["joint_speed08"] = "d"
94         dic["tcpacc"] = "d"
95         dic["jointacc01"] = "d"
96         dic["jointacc02"] = "d"
97         dic["jointacc03"] = "d"
98         dic["jointacc04"] = "d"
99         dic["jointacc05"] = "d"
100        dic["jointacc06"] = "d"
101        dic["jointacc07"] = "d"
102        dic["jointacc08"] = "d"
103        dic["joint_temperature01"] = "d"
104        dic["joint_temperature02"] = "d"
105        dic["joint_temperature03"] = "d"
106        dic["joint_temperature04"] = "d"
107        dic["joint_temperature05"] = "d"
108        dic["joint_temperature06"] = "d"
109        dic["joint_torque01"] = "d"
110        dic["joint_torque02"] = "d"
111        dic["joint_torque03"] = "d"
112        dic["joint_torque04"] = "d"
113        dic["joint_torque05"] = "d"
```

```

114         dic["joint_torque06"] = "d"
115         dic["extjoint_torques01"] = "d"
116         dic["extjoint_torques02"] = "d"
117         dic["extjoint_torques03"] = "d"
118         dic["extjoint_torques04"] = "d"
119         dic["extjoint_torques05"] = "d"
120         dic["extjoint_torques06"] = "d"
121         dic["exttcpforceintool01"] = "d"
122         dic["exttcpforceintool02"] = "d"
123         dic["exttcpforceintool03"] = "d"
124         dic["exttcpforceintool04"] = "d"
125         dic["exttcpforceintool05"] = "d"
126         dic["exttcpforceintool06"] = "d"
127
128         print("index =", index)
129         data = s.recv (1024)
130         if len(data) != 1024:
131             lost += 1
132             print(str(lost))
133             continue
134
135         names = []
136         ii=range(len(dic))
137         for key ,i in zip(dic ,ii):
138             fmtsize = struct. calcsize (dic[key ])
139             data1 , data = data [0: fmtsize], data[fmtsize :]
140             fmt="!" + dic[key]
141             names.append(struct.unpack(fmt ,data1))
142             dic[key] = dic[key], struct.unpack(fmt , data1)
143         output = ""
144         for key in dic.keys ():
145             output += str(key) + ":" + str(dic[key ][1][0]) + ";\n"
146
147         output = "lost : " + str(lost) + " index : " + str(index) +
148             ";" + output + "\n"
149         if dic['MessageSize'] != ('I', (1024,)):
150             s.close()
151             break
152
153         if index %10 == 0:
154             # Print all information
155             print(output)

```

```

154         # Print packet length
155         print(dic['MessageSize'])
156         # Print timestamp
157         timestamp01_value = dic['TimeStamp'][1][0] // 1000
158         timeValue = time.gmtime(int(timestamp01_value))
159         print(time.strftime("%Y-%m-%d %H:%M:%S", timeValue))
160         # Print joint coordinates
161         print(dic['machinePos01'][1][0], dic['machinePos02'
162             ][1][0],
163             dic['machinePos03'][1][0], dic['machinePos04'
164             ][1][0],
165             dic['machinePos05'][1][0], dic['machinePos06'
166             ][1][0],
167             dic['machinePos07'][1][0], dic['machinePos08'
168             ][1][0])
169         # Print base coordinates
170         print(dic['machinePose01'][1][0], dic['machinePose02'
171             ][1][0], dic['machinePose03'][1][0],
172             dic['machinePose04'][1][0],
173             dic['machinePose05'][1][0], dic['machinePose06'
174             ][1][0])
175         # Print user coordinates
176         print(dic['machineUserPose01'][1][0], dic['
177             machineUserPose02'][1][0], dic['machineUserPose03'
178             ][1][0],
179             dic['machineUserPose04'][1][0],
180             dic['machineUserPose05'][1][0], dic['
181             machineUserPose06'][1][0])
182         # Print the percentage of joint rated torque
183         print(dic['torque01'][1][0], dic['torque02'][1][0],
184             dic['torque03'][1][0], dic['torque04'][1][0],
185             dic['torque05'][1][0],
186             dic['torque06'][1][0], dic['torque07'][1][0],
187             dic['torque08'][1][0])
188         # Print robot status
189         print(dic['robotState'][1][0])
190         # Print servo enable status
191         print(dic['servoReady'][1][0])
192         # Print synchronization status
193         print(dic['can_motor_run'][1][0])
194         # Print the motor speed of each axis
195         print(dic['motor_speed01'][1][0], dic['motor_speed02'
196             ][1][0], dic['motor_speed03'][1][0],
197             dic['motor_speed04'][1][0], dic['motor_speed05'

```

```

186         ][1][0],
            dic['motor_speed06'] [1][0], dic['motor_speed07
            '][1][0], dic['motor_speed08'] [1][0])
187     # Print robot mode
188     print(dic['robotMode'] [1][0])
189     # Print analog input port data
190     print(dic['analog_ioInput01'] [1][0], dic['
            analog_ioInput02'] [1][0], dic['analog_ioInput03'
            '][1][0])
191     # Print analog output port data
192     print(dic['analog_ioOutput01'] [1][0], dic['
            analog_ioOutput02'] [1][0], dic['analog_ioOutput03'
            '][1][0],
193         dic['analog_ioOutput04'] [1][0], dic['
            analog_ioOutput05'] [1][0])
194     # Print the binary form of the digital input port
            data
195     print(bin(dic['digital_ioInput'] [1][0]) [2:].zfill(64)
            )
196     # Print the binary form of the digital output port
            data
197     print(bin(dic['digital_ioOutput'] [1][0]) [2:].zfill
            (64))
198     # Print collision alarm status
199     print(dic["collision"] [1][0])
200     # Print the center pose of the flange in the base
            coordinate system
201     print(dic['machineFlangePose01'] [1][0], dic['
            machineFlangePose02'] [1][0], dic['
            machineFlangePose03'] [1][0],
202         dic['machineFlangePose04'] [1][0],
203         dic['machineFlangePose05'] [1][0], dic['
            machineFlangePose06'] [1][0])
204     # Print the center pose of the flange in the user
            coordinate system
205     print(dic['machineUserFlangePose01'] [1][0], dic['
            machineUserFlangePose02'] [1][0],
206         dic['machineUserFlangePose03'] [1][0], dic['
            machineUserFlangePose04'] [1][0],
207         dic['machineUserFlangePose05'] [1][0], dic['
            machineUserFlangePose06'] [1][0])
208     # Print whether it is currently in emergency stop
            state

```

```
209         print(dic["emergencyStopState"][1][0])
210         # Print tcp movement speed
211         print(dic["tcp_speed"][1][0])
212         # Print the speed of each joint under joint motion
213         print(dic['joint_speed01'][1][0], dic['joint_speed02 '
214             ] [1][0],
215             dic['joint_speed03'][1][0], dic['joint_speed04 '
216             ] [1][0],
217             dic['joint_speed05'][1][0], dic['joint_speed06 '
218             ] [1][0],
219             dic['joint_speed07'][1][0], dic['joint_speed08 '
220             ] [1][0])
221         # Print tcp acceleration
222         print(dic["tcpacc"][1][0])
223         # Print the acceleration of each joint under joint
224         movement
225         print(dic['jointacc01'][1][0], dic['jointacc02 '
226             ] [1][0],
227             dic['jointacc03'][1][0], dic['jointacc04 '
228             ] [1][0],
229             dic['jointacc05'][1][0], dic['jointacc06 '
230             ] [1][0],
231             dic['jointacc07'][1][0], dic['jointacc08 '
232             ] [1][0])
233         # Print temperature
234         print(dic['joint_temperature01'][1][0], dic['
235             joint_temperature02 '][1][0],
236             dic['joint_temperature03'][1][0], dic['
237             joint_temperature04 '][1][0],
238             dic['joint_temperature05'][1][0], dic['
239             joint_temperature06 '][1][0])
240         # Print output torque
241         print(dic['joint_torque01'][1][0], dic['
242             joint_torque02 '][1][0],
243             dic['joint_torque03'][1][0], dic['
244             joint_torque04 '][1][0],
245             dic['joint_torque05'][1][0], dic['joint_torque06 '
246             ] [1][0])
```

```
232         # Print the estimated external joint torque value
233         print(dic['exjoint_torques01'][1][0], dic['
234             exjoint_torques02'][1][0],
235             dic['exjoint_torques03'][1][0], dic['
236             exjoint_torques04'][1][0],
237             dic['exjoint_torques05'][1][0], dic['
238             exjoint_torques06'][1][0])
239
240         # Print the estimated external end force/moment of force
241         in the current tool coordinate system
242         print(dic['extcpforceintool01'][1][0], dic['
243             extcpforceintool02'][1][0],
244             dic['extcpforceintool03'][1][0], dic['
245             extcpforceintool04'][1][0],
246             dic['extcpforceintool05'][1][0], dic['
247             extcpforceintool06'][1][0])
248
249         index = index +1
250         output = ""
251         dic = {}
252         data = ""
253         time.sleep(0.008)
254     s.close ()
```


Chapter 4 Log Interface

Users can connect to the robot log interface through the socket client.

The log types are: Error, Warning, Info. If the Error type is entered, Error information will be obtained; If the Warning type is entered, the log information of Error and Warning types will be obtained; If you enter Info, all types of log information will be obtained.

After connecting, enter all to enter all logs; enter a number, such as 10, to output the last 10 lines of logs; enter exit to exit the connection.

4.1 Example

```
1 import socket
2 HOST = "192.168.1.202"
3 PORT = 8058
4
5 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 s.settimeout(2)
7 s.connect((HOST,PORT))
8 # New file or empty file content
9 # file = open(r'D:\205\log\all_err_log.md','w').close()
10 # Get all log information
11 str1 = "Type=Info\n"
12 s.send(str1.encode())
13 str2 = "all\n"
14 s.send(str2.encode())
15 while True:
16     try:
17         data = s.recv(128000)
18         # with open(r'D:\205\log\all_err_log.md','a+') as f:
19         # f.write(data.decode())
20         print(data.decode())
21     except (Exception):
22         break
23 s.close()
```

Chapter 5 Raw Log Interface

Users can connect to the raw log interface of the robot through the socket client.

After connecting, enter all to enter all logs; enter a number, such as 10, to output the last 10 lines of logs; enter exit to exit the connection.

CAUTION



This function is applicable to 2.14.0 and above.

5.1 Example

```
1 import socket
2 HOST = "192.168.1.205"
3 PORT = 8059
4
5 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 s.settimeout(2)
7 s.connect((HOST, PORT))
8 # New file or empty file content
9 file = open(r'D:\205\log\err_log_1.md', 'w').close()
10
11 # Get the last 10 log entries, the number sent corresponds to the
    number of log entries obtained
12 str1 = "10\n"
13 s.send(str1.encode())
14 while True:
15     try:
16         data = s.recv(1024)
17         with open(r'D:\205\log\err_log_1.md', 'a+') as f:
18             f.write(data.decode())
19             print(data.decode())
20     except (Exception):
21         break
22 s.close()
```

ALWAYS EASIER THAN BEFORE

- **Contact Us**

Sales & Service: market@elibot.cn

Technical Support: tech@elibot.cn

- **Shanghai**

Building 18, Lane 36,
Xuelin Road, Shanghai

- **Suzhou**

1F, Building 4,
No 259 Changyang Street, Suzhou
+86-400-189-9358
+86-0512-83951898

- **Beijing**

Room 1102, Building 6, No. 2,
Ronghua South Road, Beijing

- **Shenzhen**

Room 202, Building 1A,
Hangkong Road, Shenzhen



WeChat
Official Account