Dobot TCP/IP Remote Control Interface Guide

Shenzhen Yuejiang Technology Co.,Ltd.|China

Table of Contents

Preface

Purpose

This document introduces the TCP/IP secondary development interfaces and their usage of Dobot industrial robot controller (V4), which helps users to understand and develop the robot control software based on TCP/IP.

Intended audience

This document is intended for：

- Customer
- Sales Engineer
- Installation and Commissioning Engineer
- Technical Support Engineer

Revision history

Date	Version	Revised content
2024/08/15	V4.5.1	Fixed ServoJ example , and added ServoJ and ServoP return values.
2024/03/25	V4.5.1	Corresponding to six-axis controller V4.5.1
2023/11/28	V4.5.0	Corresponding to six-axis controller V4.5.0, add CreateTray, GetTrayPoint, ServoJ, ServoP commands, and optimizes descriptions.
2023/07/28	V4.4.0	Corresponding to six-axis controller V4.4.0
2023/06/07	V4.3.0	Corresponding to six-axis controller V4.3.0

1 Overview

As the communication based on TCP/IP has high reliability, strong practicability and high performance with low cost, many industrial automation projects have a wide demand for controlling robots based on TCP/IP protocol. Dobot robots, designed on the basis of TCP/IP protocol, provide rich interfaces for interaction with external devices.

Port description

According to the design, Dobot robots will open 29999, 30004, 30005 and 30006 server ports.

- Server port 29999: The upper computer can send some **control commands** directly to the robot via port 29999, or **acquire** certain status of the robot. These functions are called Dashboard.
- Server port 30004, 30005 and 30006: Port 30004 (real-time feedback port) receives robot information every **8ms**. Port 30005 feeds back robot information every **200ms**. Port 30006 is a **configurable** port to feed back robot information (feed back every **50ms** by default. If you need to modify, please contact technical support). Each packet received through the real-time feedback port has 1440 bytes, which are arranged in a standard format.

Message format

Both message commands and message responses are in ASCII format (string).

The format for **sending messages** is shown below:

```
Message name(Param1,Param2,Param3……ParamN)
```

It consists of a message name and parameters in a bracket. Each parameter is separated by an English comma “,”. A complete message ends up with a right bracket.

TCP/IP remote control commands are not case-sensitive in format, e.g. the three expressions below will all be recognized as enabling the robot:

- ENABLEROBOT()
- enablerobot()
- eNabLErobOt()

When the robot receives a command, it returns a **response message** in the following format:

```
ErrorID,{value,...,valueN},Message name(Param1,Param2,Param3……ParamN);
```

- If `ErrorID` is 0, the command is received successfully. If `ErrorID` is a non-zero value, it refers to an error in the command. See Error Code for details.

- `{value,...,valueN}` refers to the return value. {} means no return value.
- `Message name(Param1,Param2,Param3……ParamN)` refers to the content delivered.

Example:

Send:

```
MovL(-500,100,200,150,0,90)
```

Return:

```
0,{},MovL(-500,100,200,150,0,90);
```

0: received successfully. {}: no return value.

Send:

```
Mov(-500,100,200,150,0,90)
```

Return:

```
-10000,{},Mov(-500,100,200,150,0,90);
```

-10000: command does nor exist. {}: no return value.

The examples in this document are pseudo-code and cannot be run directly, they are only used to illustrate how to use the interface.

2 Dashboard Command

2.1 Control command

Command list

Command	Function
PowerOn	Power on the robot
EnableRobot	Enable the robot
DisableRobot	Disable the robot
ClearError	Clear alarms of robot
RunScript	Run the project
Stop	Stop moving (or stop running the project)
Pause	Pause moving (or pause running the project)
Continue	Continue moving (or continue running the paused project)
EmergencyStop	Stop the robot arm in an emergency
BrakeControl	Control the brake of specified joint
StartDrag	Enter the drag mode
StopDrag	Exit the drag mode

PowerOn

Command

```
PowerOn()
```

Description

Power on the robot. It takes about 10 seconds for the robot to be powered on. Do not send control signals before the robot is powered on and initialized, otherwise it may cause the robot to move abnormally.

Return

```
ErrorID,{},PowerOn();
```

Example

```
PowerOn()
```

Power on the robot.

EnableRobot

Command

```
EnableRobot(load,centerX,centerY,centerZ)
```

Description

Enable the robot.

Optional parameter

Parameter	Type	Description
load	double	Load weight. The value range should not exceed the load range of corresponding robot models. Unit: kg.
centerX	double	Eccentric distance in X direction. Range: -999 – 999. Unit: mm.
centerY	double	Eccentric distance in Y direction. Range: -999 – 999. Unit: mm.
centerZ	double	Eccentric distance in Z direction. Range: -999 – 999. Unit: mm.
isCheck	int	Whether to check load. 1: check, 0: not check. If it is set to 1, the robot will check whether the actual load is consistent with the set load. If not, the robot will be automatically disabled. 0 by default.

Number of optional parameters:

- 0: no parameter (not set load weight and eccentric parameters when enabling the robot).
- 1: one parameter (load weight).
- 4: four parameters (load weight and eccentric parameters).
- 5: five parameters (load weight, eccentric parameters, and whether to check load).

Return

```
ErrorID,{},EnableRobot(load,centerX,centerY,centerZ,isCheck);
```

Example 1

```
EnableRobot()
```

Enable the robot without setting load weight and eccentric parameters.

Example 2

```
EnableRobot(1.5)
```

Enable the robot and set the load weight to 1.5kg.

Example 3

```
EnableRobot(1.5,0,0,30.5)
```

Enable the robot. Set the load weight to 1.5kg and Z-axis eccentric distance to 30.5mm without checking the load.

Example 4

```
EnableRobot(1.5,0,0,30.5,1)
```

Enable the robot. Set the load weight to 1.5kg and Z-axis eccentric distance to 30.5mm, and check the load.

DisableRobot

Command

```
DisableRobot()
```

Description

Disable the robot.

Return

```
ErrorID,{},DisableRobot();
```

Example

```
DisableRobot()
```

Disable the robot.

ClearError

Command

```
ClearError()
```

Description

Clear the alarms of the robot. After clearing the alarm, you can judge whether the robot is still in the alarm status according to RobotMode. Some alarms cannot be cleared unless you resolve the alarm cause or restart the controller.

Return

```
ErrorID,{},ClearError();
```

Example

```
uint64_t robotMode = parseRobotMode(RobotMode()); // parseRobotMode is used to obtain the value returned by the RobotMode command. Please implement it yourself.
if(robotMode=9){
  ClearError()
}
```

Clear the alarms of the robot.

RunScript

Command

```
RunScript(projectName)
```

Description

Run the project. If you need to pause immediately after running the project, you need to wait at least 1s after delivering the RunScript command before delivering the Pause command.

Required parameter

Parameter	Type	Description
projectName	string	Project name. If the name contains Chinese, the encoding method of the sending side must be set to UTF-8, otherwise it will cause an exception in receiving Chinese. If the name is composed of numbers only, it must be enclosed in double inverted commas.

Return

```
ErrorID,{},RunScript(projectName);
```

Example 1

```
RunScript(demo)
```

Run the script project named “demo”.

Example 2

```
RunScript(“123”)
```

Run the script project named “123”.

Example 3

```
RunScript("blockly_test")
```

DobotStudio Pro automatically adds the prefix "blockly_" when saving a blockly project. For example, if a blockly project named "test“ is saved in DobotStudio Pro, you need to specify the project name as "blockly_test" when executing this command.

Stop

Command

```
Stop()
```

Description

Stop the motion command queue that has been delivered or the project run by RunScript command.

Return

```
ErrorID,{},Stop();
```

Example

```
Stop()
```

Stop moving.

Pause

Command

```
Pause()
```

Description

Pause the motion command queue that has been delivered or the project run by RunScript command.

Return

```
ErrorID,{},Pause();
```

Example

```
Pause()
```

Pause moving.

Continue

Command

```
Continue()
```

Description

Continue the motion command queue that has been delivered or the project run by RunScript command.

Return

```
ErrorID,{},Continue();
```

Example

```
Continue()
```

Continue moving.

EmergencyStop

Command

```
EmergencyStop(mode)
```

Description

Stop the robot arm in an emergency. After the emergency stop, the robot arm will be disabled and then alarm. You need to release the E-Stop switch and clear the alarm to re-enable the robot arm.

Required parameter

Parameter	Type	Description
mode	int	Emergency stop mode. 1: press the E-Stop switch, 0: release the E-Stop switch.

Return

```
ErrorID,{},EmergencyStop(mode);
```

Example

```
EmergencyStop(1)
```

Stop the robot arm in an emergency.

BrakeControl

Command

```
BrakeControl(axisID,value)
```

Description

Control the brake of specified joint. The joints automatically brake when the robot is stationary. If you need to drag the joints, you can switch on the brake, i.e. hold the joint manually in the disabled status and deliver the command to switch on the brake.

Joint brake can be controlled only when the robot arm is disabled, otherwise, Error ID will return -1.

Required parameter

Parameter	Type	Description
axisID	int	joint ID, 1: J1, 2: J2, and so on
value	int	Status of brake. 0: switch off brake (joints cannot be dragged). 1: switch on brake (joints can be dragged).

Return

```
ErrorID,{},BrakeControl(axisID,value);
```

Example

```
BrakeControl(1,1)
```

Switch on the brake of Joint 1.

StartDrag

Command

```
StartDrag()
```

Description

Enter the drag mode. The robot cannot enter the drag mode through this command in error status.

Return

```
ErrorID,{},StartDrag();
```

Example

```
StartDrag()
```

The robot enters the drag mode.

StopDrag

Command

```
StopDrag()
```

Description

Exit the drag mode. This command is used to exit both Joint drag and Force-control drag modes.

Return

```
ErrorID,{},StopDrag();
```

Example

```
StopDrag()
```

The robot arm exits the drag mode.

2.2 Settings command

Command list

Command	Function
SpeedFactor	Set global speed ratio
User	Set global user coordinate system
SetUser	Modify specified user coordinate system
CalcUser	Calculate user coordinate system
Tool	Set global tool coordinate system
SetTool	Modify specified tool coordinate system
CalcTool	Calculate tool coordinate system
SetPayload	Set payload
AccJ	Set acceleration ratio of joint motion
AccL	Set acceleration ratio of linear and arc motion
VelJ	Set speed ratio of joint motion
VelL	Set speed ratio of linear and arc motion
CP	Set continuous path (CP) ratio
SetCollisionLevel	Set collision detection level
SetBackDistance	Set backoff distance of collision
SetPostCollisionMode	Set post-collision handling mode
DragSensivity	Set drag sensitivity
EnableSafeSkin	Enable or disable SafeSkin
SetSafeSkin	Set the sensitivity for each part of the SafeSkin
SetSafeWallEnable	Switch on/off specified safety wall
SetWorkZoneEnable	Switch on/off specified safety area

ℹ NOTE

Unless particularly stated, the parameters set by TCP commands only take effect in the current TCP/IP mode.

SpeedFactor

Command

```
SpeedFactor(ratio)
```

Description

Set the global speed ratio.

- Actual robot acceleration/speed ratio in jogging = value in Jog settings × global speed ratio.

  Example: If the joint speed set in the software is 12°/s and the global speed ratio is 50%, then the actual jog speed is 12°/s x 50% = 6°/s.

- Actual robot acceleration/speed ratio in playback = ratio set in motion command × value in Playback settings × global speed ratio.

  Example: If the coordinate system speed set in the software is 2000mm/s, the global speed ratio is 50%, and the speed set in the motion command is 80%, then the actual speed is 2000mm/s x 50% x 80%= 800mm/s.

If it is not set, the value set in the software before entering TCP/IP mode will be adopted.

Required parameter

Parameter	Type	Description
ratio	int	Global speed ratio, range: [1, 100]

Return

```
ErrorID,{},SpeedFactor(ratio);
```

Example

```
SpeedFactor(80)
```

Set the global speed ratio to 80%.

User

Command

```
User(index)
```

Description

Set the global user coordinate system. You can select a user coordinate system while delivering motion commands. If you do not specify the user coordinate system, the global user coordinate system will be used.

If it is not set, the default global user coordinate system is User coordinate system 0.

Required parameter

Parameter	Type	Description
index	int	Index of the calibrated user coordinate system, which needs to be calibrated by software before it can be selected here.

Return

```
ErrorID,{ResultID},User(index);
```

-1 indicates that the index of the set user coordinate system does not exist. ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
User(1)
```

Set the User coordinate system 1 to the global user coordinate system.

SetUser

Command

```
SetUser(index,table,type)
```

Description

Modify the specified user coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Index of the calibrated user coordinate system, which needs to be calibrated by software before it can be selected here.
table	string	User coordinate system after modification (format: {x, y, z, rx, ry, rz}), which is recommended to obtain through "CalcUser" command.

Optional parameter:

Parameter	Type	Description
type	int	Whether to save globally. 0: The coordinate system modified by this command only takes effect when the project is running, and it will be restored to the original value after exiting TCP mode. 1: The coordinate system modified by this command is saved globally, and the modified value is still maintained after exiting TCP mode.

Return

```
ErrorID,{},SetUser(index,table,type);
```

Example

```
SetUser(1,{10,10,10,0,0,0})
```

Modify user coordinate system 1 to X=10, Y=10, Z=10, RX=0, RY=0, RZ=0.

CalcUser

Command

```
CalcUser(index,matrix_direction,table)
```

Description

Calculate the user coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Index of the calibrated user coordinate system, which needs to be calibrated by software before it can be selected here.
matrix_direction	int	Calculation method. 1: left multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along the base coordinate system. 0: right multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along itself.
table	string	User coordinate system offset (format: {x, y, z, rx, ry, rz}).

Return

```
ErrorID,{x,y,z,rx,ry,rz},CalcUser(index,matrix_direction,table);
```

{x, y, z, rx, ry, rz} is the user coordinate system after calculation.

Example 1:

```
newUser = CalcUser(1,1,{10,10,10,10,10,10})
```

Calculate: User coordinate system 1 left-multiplies {10,10,10,0,0,0}.
The calculation process can be equivalent to: A coordinate system with the same initial posture as User coordinate system 1, moves {x=10, y=10, z=10} along the base coordinate system and rotates {rx=10, ry=10, rz=10}, and the new coordinate system is newUser.

Example 2:

```
newUser = CalcUser(1,0,{10,10,10,10,10,10})
```

Calculate: User coordinate system 1 right-multiplies {10,10,10,0,0,0}.
The calculation process can be equivalent to: A coordinate system with the same initial posture as User coordinate system 1, moves {x=10, y=10, z=10} along user coordinate system 1 and rotates {rx=10, ry=10, rz=10}, and the new coordinate system is newUser.

Tool

Command

```
Tool(index)
```

Description

Set the global tool coordinate system. You can select a tool coordinate system while delivering motion commands. If you do not specify the tool coordinate system, the global tool coordinate system will be used.

If it is not set, the default global tool coordinate system is Tool coordinate system 0.

Required parameter

Parameter	Type	Description
index	int	Index of the calibrated tool coordinate system, which needs to be calibrated by software before it can be selected here.

Return

```
ErrorID,{ResultID},Tool(index);
```

-1 indicates that the index of the set tool coordinate system does not exist. ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
Tool(1)
```

Set the Tool coordinate system 1 to the global tool coordinate system.

SetTool

Command

```
SetTool(index,table,type)
```

Description

Modify the specified tool coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Index of the calibrated tool coordinate system, which needs to be calibrated by software before it can be selected here.
table	string	Tool coordinate system after modification (format: {x, y, z, rx, ry, rz}), represents the offset of the coordinate system relative to the default tool coordinate system.

Optional parameter:

Parameter	Type	Description
type	int	Whether to save globally. 0: The coordinate system modified by this command only takes effect when the project is running, and it will be restored to the original value after exiting TCP mode. 1: The coordinate system modified by this command is saved globally, and the modified value is still maintained after exiting TCP mode.

Return

```
ErrorID,{},SetTool(index,table,type);
```

Example

```
SetTool(1,{10,10,10,0,0,0})
```

Modify tool coordinate system 1 to X=10, Y=10, Z=10, RX=0, RY=0, RZ=0.

CalcTool

Command

```
CalcTool(index,matrix_direction,table)
```

Description

Calculate the tool coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Index of the calibrated tool coordinate system, which needs to be calibrated by software before it can be selected here.
matrix_direction	int	Calculation method. 1: left multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along the flange coordinate system. 0: right multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along itself.
table	string	Tool coordinate system offset (format: {x, y, z, rx, ry, rz}).

Return

```
ErrorID,{x,y,z,rx,ry,rz},CalcTool(index,matrix_direction,table);
```

{x, y, z, rx, ry, rz} is the tool coordinate system after calculation.

Example 1:

```
CalcTool(1,1,{10,10,10,0,0,0})
```

Calculate: Tool coordinate system 1 left-multiplies {10,10,10,0,0,0}.
The calculation process can be equivalent to: A coordinate system with the same initial posture as Tool coordinate system 1, moves {x=10, y=10, z=10} along the flange coordinate system and rotates {rx=10, ry=10, rz=10}, and the new coordinate system is newTool.

Example 2:

```
CalcTool(1,0,{10,10,10,0,0,0})
```

Calculate: Tool coordinate system 1 right-multiplies {10,10,10,0,0,0}.
The calculation process can be equivalent to: A coordinate system with the same initial posture as Tool coordinate system 1, moves {x=10, y=10, z=10} along Tool coordinate system 1 and rotates {rx=10, ry=10, rz=10}, and the new coordinate system is newTool.

SetPayload

Command

```
SetPayload(load,x,y,z)
SetPayload(name)
```

Description

Set the payload of the robot arm, supporting two ways of settings.

Method 1: Set the load parameters directly.

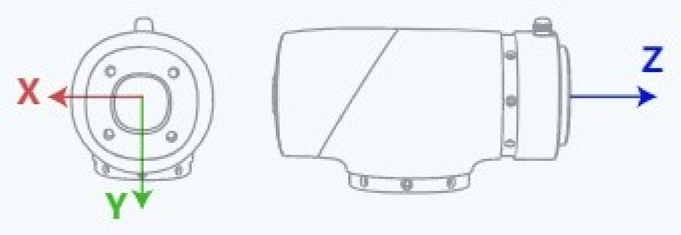
Required parameter 1

Parameter	Type	Description
load	double	Load weight. The value range should not exceed the load range of corresponding robot models. Unit: kg.

Optional parameter 1

Parameter	Type	Description
x	double	X-axis eccentric coordinates of payload. Range: -500 – 500. Unit: mm.
y	double	Y-axis eccentric coordinates of payload. Range: -500 – 500. Unit: mm.
z	double	Z-axis eccentric coordinates of payload. Range: -500 – 500. Unit: mm.

The three parameters need to be set or not set at the same time. The eccentric coordinates are the coordinates of the mass centre of the load (including the fixture) under the default tool coordinate system, as shown below.

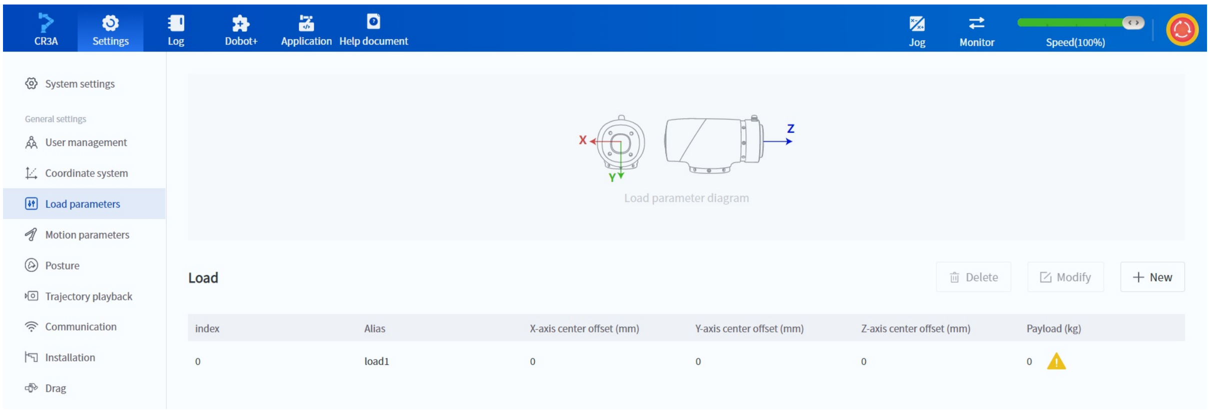


Method 2: Set through the preset load parameter group saved in the software.

Required parameter 2

Parameter	Type	Description

name	string	Name of the preset load parameter group saved in the software.



Load

index	Alias	X-axis center offset (mm)	Y-axis center offset (mm)	Z-axis center offset (mm)	Payload (kg)
0	load1	0	0	0	0 ⚠

Return

```
ErrorID,{ResultID},SetPayload(load,x,y,z);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example 1

```
SetPayload(3,10,10,10)
```

Set the load weight to 3kg, and eccentric coordinates to {10,10,10}.

Example 2

```
SetPayload("Load1")
```

Load the preset parameter group "Load1".

AccJ

Command

```
AccJ(R)
```

Description

Set acceleration ratio of joint motion.

Defaults to 100 if not set.

Required parameter

Parameter	Type	Description

R	int	Acceleration ratio. Range: [1,100].

Return

```
ErrorID,{},AccJ(R);
```

Example

```
AccJ(50)
```

Set the acceleration ratio of joint motion to 50%.

AccL

Command

```
AccL(R)
```

Description

Set acceleration ratio of linear and arc motion.

Defaults to 100 if not set.

Required parameter

Parameter	Type	Description
R	int	Acceleration ratio. Range: [1,100].

Return

```
ErrorID,{},AccL(R);
```

Example

```
AccL(50)
```

Set the acceleration ratio of linear and arc motion to 50%.

VelJ

Command

```
VelJ(R)
```

Description

Set the speed ratio of joint motion.

Defaults to 100 if not set.

Required parameter

Parameter	Type	Description
R	int	Speed ratio. Range: [1,100].

Return

```
ErrorID,{},VelJ(R);
```

Example

```
VelJ(50)
```

Set the speed ratio of joint motion to 50%.

VelL

Command

```
VelL(R)
```

Description

Set the speed ratio of linear and arc motion.

Defaults to 100 if not set.

Required parameter

Parameter	Type	Description
R	int	Speed ratio. Range: [1,100].

Return

```
ErrorID,{},VelL(R);
```

Example

```
VelL(50)
```

Set the speed ratio of linear and arc motion to 50%.

CP

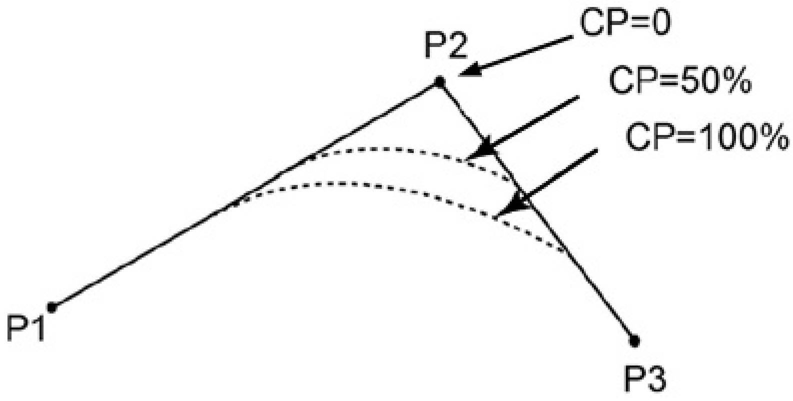
Command

```
CP(R)
```

Description

Set the continuous path (CP) ratio, that is, when the robot arm moves continuously via multiple points, whether it transitions at a right angle or in a curved way when passing through the intermediate point.

Defaults to 0 if not set.



Required parameter

Parameter	Type	Description
R	int	Continuous path ratio. Range: [0, 100].

Return

```
ErrorID,{},CP(R);
```

Example

```
CP(50)
```

Set the continuous path ratio to 50.

SetCollisionLevel

Command

```
SetCollisionLevel(level)
```

Description

Set the collision detection level.

If it is not set, the value set in the software before entering TCP/IP mode will be adopted.

Required parameter

Parameter	Type	Description
level	int	Collision detection level. 0: switching off collision detection. 1 – 5: the larger the number, the higher the sensitivity.

Return

```
ErrorID,{ResultID},SetCollisionLevel(level);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
SetCollisionLevel(1)
```

Set the collision detection level to 1.

SetBackDistance

Command

```
SetBackDistance(distance)
```

Description

Set the backoff distance after the robot detects collision.

If it is not set, the value set in the software before entering TCP/IP mode will be adopted.

Required parameter:

Parameter	Type	Description
distance	double	Collision backoff distance, range: [0,50], unit: mm.

Return

```
ErrorID,{ResultID},SetBackDistance(distance)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
SetBackDistance(20)
```

Set the collision backoff distance to 20mm.

SetPostCollisionMode

Command

```
SetPostCollisionMode(mode)
```

Description

Set the robot to enter the specified status after detecting collision.

If it is not set, the value set in the software before entering TCP/IP mode will be adopted.

Required parameter:

Parameter	Type	Description
mode	int	Post-collision handling mode. 0: enter stop status after detecting collision. 1: enter pause status after detecting collision.

Return

```
ErrorID,{ResultID},SetPostCollisionMode(mode)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
SetPostCollisionMode(0)
```

Set the robot to enter the stop status after detecting collision.

DragSensivity

Command

```
DragSensivity(index,value)
```

Description

Set the drag sensitivity.

If it is not set, the value set in the software before entering TCP/IP mode will be adopted.

Required parameter

Parameter	Type	Description
index	int	Axis No., 1 – 6: J1 – J6, 0: set all axes at the same time.
value	int	Drag sensitivity. The smaller the value, the larger the force when dragging. Range: [1, 90].

Return

```
ErrorID,{},DragSensivity(index,value);
```

Example

```
DragSensivity(0,50)
```

Set the drag sensitivity of all axes to 50.

EnableSafeSkin

Command

```
EnableSafeSkin(status)
```

Description

Enable or disable the SafeSkin. This command is only valid for robots installed with SafeSkin.

Required parameter

Parameter	Type	Description
status	int	SafeSkin switch. 0: OFF, 1: ON

Return

```
ErrorID,{ResultID},EnableSafeSkin(status);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
EnableSafeSkin(1)
```

Switch on the SafeSkin.

SetSafeSkin

Command

```
SetSafeSkin(part,status)
```

Description

Set the sensitivity for each part of the SafeSkin. This command is only valid for robots installed with SafeSkin.

If it is not set, the value set in the software before entering TCP/IP mode will be adopted.

Required parameter

Parameter	Type	Description
part	int	Part of the robot. 3: forearm, 4 – 6: J4 – J6 joints
status	int	Sensitivity. 0: OFF, 1: low, 2: medium, 3: high

Return

```
ErrorID,{ResultID},SetSafeSkin(part,status);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
SetSafeSkin(3.1)
```

Set the SafeSkin sensitivity of the forearm to 1.

SetSafeWallEnable

Command

```
SetSafeWallEnable(index,value)
```

Description

Switch on/off the specified safety wall.

Required parameter:

Parameter	Type	Description
index	int	Safety wall index, which needs to be added in the software first. Range: [1.8].
value	int	ON/OFF status of safety wall. 0: OFF, 1: ON

Return

```
ErrorID,{ResultID},SetSafeWallEnable(index,value)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
SetSafeWallEnable(1,1)
```

Switch on the safety wall 1.

SetWorkZoneEnable

Command

```
SetWorkZoneEnable(index,value)
```

Description

Switch on/off the specified safety area.

Required parameter:

Parameter	Type	Description
index	int	Safety area index, which needs to be added in the software first. Range: [1.6].
value	int	ON/OFF status of safety area. 0: OFF, 1: ON

Return

```
ErrorID,{ResultID},SetWorkZoneEnable(index,value)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
SetWorkZoneEnable(1,1)
```

Switch on the safety area 1.

2.3 Calculating and obtaining command

Command list

Command	Function
RobotMode	Get current status of robot
PositiveKin	Forward kinematics
InverseKin	Inverse kinematics
GetAngle	Get joint coordinates of current posture
GetPose	Get Cartesian coordinates of current posture under the specific coordinate system
GetErrorID	Get current error code of robot
CreateTray	Create pallet
GetTrayPoint	Get pallet point

RobotMode

Command

```
RobotMode()
```

Description

Get the current status of the robot.

Return

```
ErrorID,{Value},RobotMode();
```

Value range:

Value	Definition	Description
1	ROBOT_MODE_INIT	Initialized status
2	ROBOT_MODE_BRAKE_OPEN	Brake switched on
3	ROBOT_MODE_POWEROFF	Power-off status
4	ROBOT_MODE_DISABLED	Disabled (no brake switched on)
5	ROBOT_MODE_ENABLE	Enabled and idle

6	ROBOT_MODE_BACKDRIVE	Drag mode (Joint drag or Force-control drag)
7	ROBOT_MODE_RUNNING	Running status (project, TCP queue motion, etc.)
8	ROBOT_MODE_SINGLE_MOVE	Single motion status (jog, RunTo, etc.)
9	ROBOT_MODE_ERROR	There are uncleared alarms. This status has the highest priority. It returns 9 when there is an alarm, regardless of the status of the robot arm.
10	ROBOT_MODE_PAUSE	Pause status
11	ROBOT_MODE_COLLISION	Collision detection triggered status

Example

```
RobotMode()
```

Get the current status of the robot.

PositiveKin

Command

```
PositiveKin(J1,J2,J3,J4,J5,J6,User,Tool)
```

Description

Forward kinematics. Calculate the coordinates of the end of the robot in the specified Cartesian coordinate system, based on the given angle of each joint.

Required parameter

Parameter	Type	Description
J1	double	J1-axis position, unit: °
J2	double	J2-axis position, unit: °
J3	double	J3-axis position, unit: °
J4	double	J4-axis position, unit: °
J5	double	J5-axis position, unit: °
J6	double	J6-axis position, unit: °

Optional parameter

Parameter	Type	Description
User	string	Format: user=index (index: index of the calibrated user coordinate system). The global user coordinate system will be used when it is not specified.

Tool	string	Format: tool=index (index: index of the calibrated tool coordinate system). The global tool coordinate system will be used when it is not specified.

Return

```
ErrorID,{x,y,z,a,b,c},PositiveKin(J1,J2,J3,J4,J5,J6,User,Tool);
```

{x,y,z,a,b,c} refers to the Cartesian coordinates of the point.

Example

```
PositiveKin(0,0,-90,0,90,0,user=1,tool=1)
```

Calculate the coordinates of the end of the robot in the User coordinate system 1 and Joint coordinate system 1, based on the joint coordinates {0,0,-90,0,90,0}.

InverseKin

Command

```
InverseKin(X,Y,Z,Rx,Ry,Rz,User,Tool,useJointNear,JointNear)
```

Description

inverse kinematics. Calculate the joint angles of the robot, based on the given coordinates in the specified Cartesian coordinate system.

As Cartesian coordinates only define the spatial coordinates and tilt angle of the TCP, the robot arm can reach the same posture through different gestures, which means that one posture variable can correspond to multiple joint variables. To get a unique solution, the system requires a specified joint coordinate, and the solution closest to this joint coordinate is selected as the inverse solution.

Required parameter

Parameter	Type	Description
X	double	X-axis position, unit: mm
Y	double	Y-axis position, unit: mm
Z	double	Z-axis position, unit: mm
Rx	double	Rx-axis position, unit: °
Ry	double	Ry-axis position, unit: °
Rz	double	Rz-axis position, unit: °

Optional parameter

Parameter	Type	Description
User	string	Format: user=index (index: index of the calibrated user coordinate system). The global user coordinate system will be used when it is not specified.
Tool	string	Format: tool=index (index: index of the calibrated tool coordinate system). The global tool coordinate system will be used when it is not specified.
useJointNear	string	It is used to set whether JointNear is effective. If the value is 0 or null, JointNear data is ineffective. The algorithm selects the joint angles according to the current angle. If the value is 1, the algorithm selects the joint angles according to JointNear data. This parameter is ineffective when carrying only this parameter without jointNear.
jointNear	string	Format: jointNear={j1,j2,j3,j4,j5,j6}, joint coordinates for selecting joint angles.

Return

```
ErrorID,{J1,J2,J3,J4,J5,J6},InverseKin(X,Y,Z,Rx,Ry,Rz,User,Tool,useJointNear,JointNear);
```

{j1,j2,j3,j4,j5,j6} refers to the joint coordinates of the point.

Example

```
InverseKin(473.000000,-141.000000,469.000000,-180.000000,0.000,-90.000)
```

The Cartesian coordinates of the end of the robot arm in global user coordinate system and global joint coordinate system are {473,-141,469,-180,0,-90}. Calculate the joint coordinates and select the nearest solution to the current joint angle of the robot arm.

GetAngle

Command

```
GetAngle()
```

Description

Get the joint coordinates of current posture.

Return

```
ErrorID,{J1,J2,J3,J4,J5,J6},GetAngle();
```

{J1,J2,J3,J4,J5,J6} refers to the joint coordinates of the current posture.

Example

```
GetAngle()
```

Get the joint coordinates of current posture.

GetPose

Command

```
GetPose(User,Tool)
```

Description

Get the Cartesian coordinates of the current posture under the specific coordinate system.

Optional parameter

Parameter	Type	Description
User	string	Format: user=index (index: index of the calibrated user coordinate system).
Tool	string	Format: tool=index (index: index of the calibrated tool coordinate system).

They should be set or not set simultaneously, which default to global user and tool coordinate system when not set.

Return

```
ErrorID,{X,Y,Z,Rx,Ry,Rz},GetPose(User,Tool);
```

{X,Y,Z,Rx,Ry,Rz} refers to the Cartesian coordinates of the current posture.

Example

```
GetPose(user=1,tool=1)
```

Get the Cartesian coordinates of the current posture under User coordinate system 1 and Tool coordinate system 1.

GetErrorID

Command

```
GetErrorID()
```

Description

Get the current error code of the robot.

Return

```
ErrorID,{[[id,...,id], [id], [id], [id], [id], [id], [id]]},GetErrorID();
```

- [id,..., id] is the alarm information of the controller and algorithm, and [] refers to no alarm. If there are multiple alarms, they are separated by a comma “,”. The collision detection value is 117. For other alarm definition, see the controller alarm document “alarm_controller.json”.
- The last six [id] represent the alarm information of six servos respectively, and [] refers to no alarm. For alarm definitions, see the servo alarm document “File alarm_servo.json”.

Example

```
GetErrorID()
```

Get the current error code of the robot.

CreateTray

Command:

```
CreateTray(Trayname, {Count}, {P1,P2})  -- 1D pallet
CreateTray(Trayname, {row,col}, {P1,P2,P3,P4})  -- 2D pallet
CreateTray(Trayname, {row,col,layer}, {P1,P2,P3,P4,P5,P6,P7,P8})  -- 3D pallet
```

Description:

Create pallet, e.g. 1D, 2D and 3D pallets. Up to 20 pallets can be created, and the existing pallets will be overwritten when creating pallets with the same name.

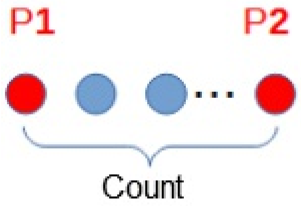
Required parameter:

Parameter	Type	Description
Trayname	string	Pallet name, a string of up to 32 bytes. Pure numbers or spaces are not allowed.

The last two parameters are lists. The number of values in the list varies depending on the dimension of the pallet to be created, as described below.

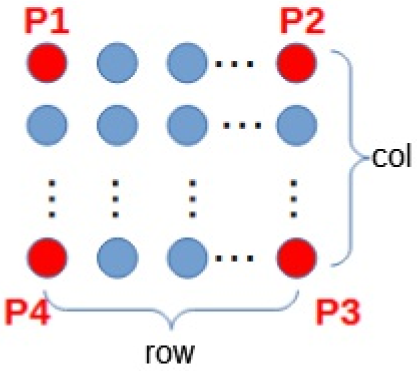
- Create 1D pallet: 1D pallet is a set of points equidistantly spaced on a straight line.

Parameter	Type	Description
{Count}	table	Count: the number of points, range: [2, 50]. If you enter a non-integer, it will be automatically rounded down.
{P1, P2}	table	P1 and P2 are the two endpoints of the 1D pallet respectively, and the format of each point is pose = {x,y,z,rx,ry,rz}.



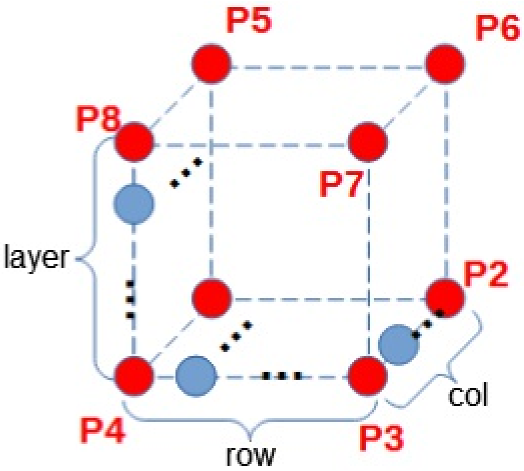
- Create 2D pallet: 2D pallet is a set of points distributed in an array on a plane.

Parameter	Type	Description
{row,col}	table	row: the number of points in the row direction (P1 to P2), col: the number of points in the column direction (P1 to P4). The value range is the same as the Count of 1D pallet.
{P1, P2, P1, P2}	table	P1, P2, P3 and P4 are the four vertices of the 2D pallet respectively, and the format of each point is pose = {x,y,z,rx,ry,rz}.



- Create 3D pallet: 3D pallet is a set of points distributed three-dimensionally in space and can be considered as multiple 2D pallets arranged

Parameter	Type	Description
{row,col,layer}	table	row: the number of points in the row direction (P1 to P2), col: the number of points in the column direction (P2 to P4), layer: the number of layers (P1 to P5).
{P1,P2,P3,P4,P5,P6,P7,P8}	table	P1 to P8 are the eight vertices of the 3D pallet respectively, and the format of each point is pose = {x,y,z,rx,ry,rz}.



Return

```
ErrorID,{},CreateTray( ... );
```

Example:

```
-- Create a 1D pallet of 5 points named t1.
CreateTray(t1, {5}, {pose = {x1,y1,z1,rx1,ry1,rz1},pose = {x2,y2,z2,rx2,ry2,rz2}})
-- Create a 4x5 2D pallet named t2. In the example below, P1 to P4 are all points in the forma
t of pose = {x,y,z,rx,ry,rz}.
CreateTray(t2, {4,5}, {P1,P2,P3,P4})
-- Create a 4x5x6 3D pallet named t3. In the example, P1 to P8 are all points in the format of
 pose = {x,y,z,rx,ry,rz}.
CreateTray(t2, {4,5,6}, {P1,P2,P3,P4,P5,P6,P7,P8})
```

GetTrayPoint

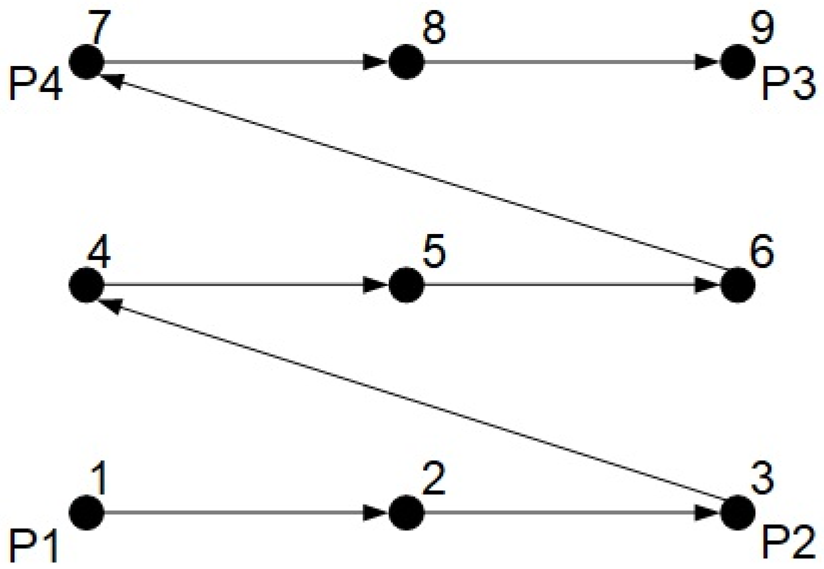
Command:

```
GetTrayPoint(Trayname, index)
```

Description:

Get the point of the specified index of the specified pallet. The point index is related to the order of points passed in when creating the pallet.

- 1D pallet: The index of P1 is 1, the index of P2 is the same as the number of points, and so on.
- 2D pallet: The following figure takes a 3x3 pallet as an example to illustrate the relationship between teaching point and point index.



- 3D pallet: Referring to 2D pallet, the index of the first point on the second layer is the index of the last point on the first layer plus one, and so on.

Required parameter:

Parameter	Type	Description
Trayname	string	Created pallet name, a string of up to 32 bytes.
index	int	The number of the point to be obtained.

Return:

```
ErrorID,{isErr,x,y,z,rx,ry,rz},GetTrayPoint(trayName,index);
```

isErr: the result of obtaining point, 0: obtain point successfully, -1: failed to obtain point.

x,y,z,rx,ry,rz are coordinates of the gotten point.

Example:

```
-- Get the point numbered 3 of the pallet named t1.
GetTrayPoint(t1,3)
```

2.4 IO command

Command list

Command	Function
DO	Set status of DO port (queue command)
DOInstant	Set status of DO port (immediate command)
GetDO	Get status of DO port
DOGroup	Set status of multiple DO ports (queue command)
GetDOGroup	Get status of multiple DO ports
ToolDO	Set status of tool DO port (queue command)
ToolDOInstant	Set status of tool DO port (immediate command)
GetToolDO	Get status of tool DO port
AO	Set value of AO port (queue command)
AOInstant	Set value of AO port (immediate command)
GetAO	Get value of AO port
DI	Get status of DI port
DIGroup	Get status of multiple DI ports
ToolDI	Get status of tool DI port
AI	Get value of AI port
ToolAI	Get value of tool AI port
SetTool485	Set data type corresponding to RS485 interface of end tool
SetToolPower	Set power status of end tool
SetToolMode	Set mode of tool multiplexed terminal

Queue command and Immediate command

- Queue command: The system will execute this command after the previous commands have been executed.
  For example, if a DO command is preceded by a series of motion commands, the system will wait for the robot arm to finish moving before setting the DO.

- Immediate command: The system ignores the command queue and executes this command as soon as the command is read.
  For example, if a DOInstant command is preceded by a series of motion commands, the system will not wait for the robot arm to finish moving but will set DO as soon as it reads the command.

If not otherwise specified, the commands for getting input are all immediate commands.

DO

Command

```
DO(index,status,time)
```

Description

Set the status of digital output port (queue command).

Required parameter

Parameter	Type	Description
index	int	DO index
status	int	DO status. 1: ON, 0: OFF

Optional parameter

Parameter	Type	Description
time	int	Continuous output time. Value range: [25, 60000]. Unit: ms. If this parameter is set, the system will automatically invert the DO after the specified time. The inversion is an asynchronous action, which will not block the command queue. After the DO output is executed, the system will execute the next command.

Return

```
ErrorID,{ResultID},DO(index,status,time);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
DO(1,1,2000)
```

Set DO_1 to 1, and automatically reverse it after 2 seconds.

DOInstant

Command

```
DOInstant(index,status)
```

Description

Set the status of digital output port (immediate command).

Required parameter

Parameter	Type	Description
index	int	DO index
status	int	DO status. 1: ON, 0: OFF

Return

```
ErrorID,{},DOInstant(index,status);
```

Example

```
DOInstant(1,1)
```

Set the status of DO_1 to 1 immediately regardless of the current command queue.

GetDO

Command

```
GetDO(index)
```

Description

Get the status of digital output port.

Required parameter

Parameter	Type	Description
index	int	DO index

Return

```
ErrorID,{value},GetDO(index);
```

value: status of DO port. 0: OFF, 1: ON.

Example

```
GetDO(1)
```

Get the status of DO_1.

DOGroup

Command

```
DOGroup(index1,value1,index2,value2,...,indexN,valueN)
```

Description

Set the status of multiple digital output ports (queue command).

Required parameter

Parameter	Type	Description
index1	int	Index of the first DO
value1	int	Status of the first DO. 1: ON, 0: OFF
...	...	...
indexN	int	Index of the last DO
valueN	int	Status of the last DO. 1: ON, 0: OFF

Return

```
ErrorID,{ResultID},DOGroup(index1,value1,index2,value2,...,indexN,valueN);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
DOGroup(4,1,6,0,2,1,7,0)
```

Set DO_4 to 1, DO_6 to 0, DO_2 to 1, and DO_7 to 0.

GetDOGroup

Command

```
GetDOGroup(index1,index2,...,indexN)
```

Description

Get the status of multiple digital output ports.

Required parameter

Parameter	Type	Description
index	int	Index of the first DO
...	...	...
indexN	int	Index of the last DO

Return

```
ErrorID,{value1,value2,...,valueN},GetDOGroup(index1,index2,...,indexN);
```

{value1,value2,...,valueN}: status of DO_1 to DO_N. 0: OFF, 1: ON.

Example

```
GetDOGroup(1,2)
```

Get the status of DO_1 and DO_2.

ToolDO

Command

```
ToolDO(index,status)
```

Description

Set the status of tool digital output port (queue command).

Required parameter

Parameter	Type	Description
index	int	Tool DO index
status	int	Tool DO status. 1: ON, 0: OFF

Return

```
ErrorID,{ResultID},ToolDO(index,status);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
ToolDO(1,1)
```

Set the tool DO_1 to 1.

ToolDOInstant

Command

```
ToolDOInstant(index,status)
```

Description

Set the status of tool digital output port (immediate command).

Required parameter

Parameter	Type	Description
index	int	Tool DO index
status	int	Tool DO status. 1: ON, 0: OFF

Return

```
ErrorID,{},ToolDOInstant(index,status);
```

Example

```
ToolDOInstant(1,1)
```

Set the status of tool DO_1 to 1 immediately regardless of the current command queue.

GetToolDO

Command

```
GetToolDO(index)
```

Description

Get the status of tool digital output port.

Required parameter

Parameter	Type	Description
index	int	Tool DO index

Return

```
ErrorID,{value},GetToolDO(index);
```

value: status of tool DO port. 0: OFF, 1: ON.

Example

```
GetToolDO(1)
```

Get the status of tool DO_1.

AO

Command

```
AO(index,value)
```

Description

Set the value of analog output port (queue command).

Required parameter

Parameter	Type	Description
index	int	AO index
value	double	AO output. Voltage range: [0,10], unit: V; Current range: [4,20], unit: mA

Return

```
ErrorID,{ResultID},AO(index,value);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
AO(1,2)
```

Set AO_1 output to 2.

AOInstant

Command

```
AOInstant(index,value)
```

Description

Set the value of analog output port (immediate command).

Required parameter

Parameter	Type	Description
index	int	AO index
value	double	AO output. Voltage range: [0,10], unit: V; Current range: [4,20], unit: mA

Return

```
ErrorID,{},AOInstant(index,value);
```

Example

```
AOInstant(1,2)
```

Set the AO_1 output to 2 immediately regardless of the current command queue.

GetAO

Command

```
GetAO(index)
```

Description

Get the value of analog output port.

Required parameter

Parameter	Type	Description
index	int	AO index

Return

```
ErrorID,{value},GetAO(index);
```

value: value of AO port.

Example

```
GetAO(1)
```

Get the value of AO_1.

DI

Command

```
DI(index)
```

Description

Get the status of digital input port.

Required parameter

Parameter	Type	Description
index	int	DI index

Return

```
ErrorID,{value},DI(index);
```

value: status of DI port. 0: OFF, 1: ON.

Example

```
DI(1)
```

Get the status of DI_1.

DIGroup

Command

```
DIGroup(index1,index2,...,indexN)
```

Description

Get the status of a group of digital input ports.

Required parameter

Parameter	Type	Description
index1	int	Index of the first DI
...	...	...
indexN	int	Index of the last DI

Return

```
ErrorID,{value1,value2,...,valueN},DIGroup(index1,index2,...,indexN);
```

{value1,value2,...,valueN}: status of DI_1 to DI_N. 0: OFF, 1: ON.

Example

```
DIGroup(4,6,2,7)
```

Get the status of DI_4, DI_6, DI_2 and DI_7.

ToolDI

Command

```
ToolDI(index)
```

Description

Get the status of tool digital input port.

Required parameter

Parameter	Type	Description
index	int	Tool DI index

Return

```
ErrorID,{value},ToolDI(index);
```

value: status of tool DI port. 0: OFF, 1: ON.

Example

```
ToolDI(1)
```

Get the status of tool DI_1.

AI

Command

```
AI(index)
```

Description

Get the value of analog input port.

Required parameter

Parameter	Type	Description
index	int	AI index

Return

```
ErrorID,{value},AI(index);
```

value: AI input.

Example

```
AI(1)
```

Get the AI_1 input.

ToolAI

Command

```
ToolAI(index)
```

Description

Get the value of tool analog input port. You need to set the port to analog input mode through SetToolMode before using it.

Required parameter

Parameter	Type	Description
index	int	Tool AI index

Return

```
ErrorID,{value},ToolAI(index);
```

value: tool AI input.

Example

```
ToolAI(1)
```

Get the value of tool AI_1.

SetTool485

Command:

```
SetTool485(baud,parity,stopbit,identify)
```

Description

Set the data type corresponding to the RS485 interface of the end tool.

Required parameter

Parameter	Type	Description
baud	int	Baud rate of RS485 interface

Optional parameter

Parameter	Type	Description
parity	string	Whether there are parity bits. “O”: odd, “E”: even, “N”: no parity bit (“N” by default)
stopbit	int	Stop bit length. Range: 1, 2 (1 by default)
identify	int	When the robot arm has multiple aviation sockets, it is used to specify the socket. 1: aviation socket 1; 2: aviation socket 2

Return

```
ErrorID,{},SetTool485(baud,parity,stopbit);
```

Example

```
SetTool485(115200,"N",1)
```

Set the baud rate corresponding to the tool RS485 interface to 115200Hz, parity bit to N, and stop bit length to 1.

SetToolPower

Command

```
SetToolPower(status,identify)
```

Description

Set the power status of the end tool, generally used for restarting the end power, such as re-powering and re-initializing the gripper. If you need to call the interface continuously, it is recommended to keep an interval of at least 4 ms.

> **NOTE**
> As Magician E6 does not support this command, there is no effect to call this command.

Required parameter

Parameter	Type	Description
status	int	Power status of end tool. 0: power off; 1: power on

Optional parameter

Parameter	Type	Description
identify	int	When the robot arm has multiple aviation sockets, it is used to specify the socket. 1: aviation socket 1; 2: aviation socket 2

Return

```
ErrorID,{},SetToolPower(status);
```

Example

```
SetToolPower(0)
```

Power off the tool.

SetToolMode

Command

```
SetToolMode(mode,type,identify)
```

Description

When AI_1 interface multiplexes with 485 interface at the end of the robot, the mode of the multiplexed terminals can be set through this command. The default mode is 485 mode.

> ℹ NOTE
> For robot arms that do not support tool mode switching, calling this command will have no effect.

Required parameter

Parameter	Type	Description
mode	int	Mode of the multiplexed terminal. 1: 485 mode, 2: Analog input (AI) mode
type	int	When mode is 1, this parameter is invalid. When mode is 2, you can set the AI mode. The units digit indicates the mode of AI1, the tens digit indicates the mode of AI2. When the tens digit is 0, you can just input the units digit.

- Mode:
  - 0: 0–10V voltage input mode
  - 1: Current collection mode
  - 2: 0–5V voltage input mode
- Example:
  - 0: AI1 and AI2 are in 0–10V voltage input mode
  - 1: AI2 is in 0–10V voltage input mode, AI1 is in current collection mode
  - 11: AI2 and AI1 are in current collection mode
  - 12: AI2 is in current collection mode, AI1 is in 0–5V voltage input mode
  - 20: AI2 is in 0–5V voltage input mode, AI1 is in 0–10V voltage input mode

Optional parameter

Parameter	Type	Description
identify	int	When the robot arm has multiple aviation sockets, it is used to specify the socket. 1: aviation socket 1; 2: aviation socket 2

Return

```
ErrorID,{},SetToolMode(mode,type);
```

Example

```
SetToolMode(2,0)
```

Set the mode of the tool multiplexed terminal to AI, with 0–10V voltage input mode for both inputs.

2.5 Modbus command

Command list

Command	Function
ModbusCreate	Create Modbus master station
ModbusRTUCreate	Create Modbus master station based on RS485
ModbusClose	Disconnect with Modbus slave station
GetInBits	Get contact register
GetInRegs	Get input register
GetCoils	Get coil register
SetCoils	Set coil register
GetHoldRegs	Get holding register
SetHoldRegs	Set holding register

The Modbus commands are used to establish the communication between Modbus master and slave. For the range and definition of the register address, please refer to the instructions of the corresponding slave.

The Modbus function codes corresponding to various registers follow the standard Modbus protocol.

Register type	Read register	Write single register	Write multiple registers
Coil register	01	05	0F
Contact register	02	-	-
Input register	04	-	-
Holding register	03	06	10

ModbusCreate

Command

```
ModbusCreate(ip,port,slave_id,isRTU)
```

Description

Create Modbus master, and establish connection to the slave (support connecting to at most 5 devices).

Required parameter

Parameter	Type	Description
ip	string	Slave IP address
port	int	Slave port
slave_id	int	Slave ID

Optional parameter

Parameter	Type	Description
isRTU	int	null or 0: establish ModbusTCP communication; 1: establish ModbusRTU communication.

⚠NOTICE

This parameter determines the protocol format used to transmit data once the connection has been established, and it does not affect the connection result. Therefore, if you set the parameter incorrectly when creating a master, the master can still be created successfully, but an exception may occur in the subsequent communication.

Return

```
ErrorID,{index},ModbusCreate(ip,port,slave_id,isRTU);
```

- ErrorID: 0 indicates that the Modbus master is created successfully. -1 indicates that the Modbus master fails to be created. For other error codes, refer to the error code description.
- index: Master index, which is used when other Modbus commands are called.

Example

```
ModbusCreate("127.0.0.1",60000,1,0)
```

Establish RTU communication master and connect to the local Modbus slave (port: 60000, slave ID: 1).

ModbusRTUCreate

Command:

```
ModbusRTUCreate(slave_id, baud, parity, data_bit, stop_bit)
```

Description:

Create Modbus master based on RS485, and establish connection to the slave (support connecting to at most 5 devices).

Required parameter

Parameter	Type	Description
slave_id	int	Slave ID
baud	int	Baud rate of RS485 interface

Optional parameter

Parameter	Type	Description
parity	string	Whether there are parity bits. “O”: odd, “E”: even, “N”: no parity bit (“E” by default)
data_bit	int	Data bit length. Value: 8 (8 by default)
stop_bit	int	Stop bit length. Value range: 1, 2 (1 by default)

Return:

```
ErrorID,{index},ModbusRTUCreate(slave_id, baud, parity, data_bit, stop_bit)
```

- ErrorID: 0 indicates that the Modbus master is created successfully. -1 indicates that the Modbus master fails to be created. For other error codes, refer to the error code description.
- index: Master index, which is used when other Modbus commands are called.

Example:

```
ModbusRTUCreate(1, 115200)
```

Create Modbus master, and establish connection with the slave through RS485. The slave ID is 1 and baud rate is 115200.

ModbusClose

Command

```
ModbusClose(index)
```

Description

Disconnect with Modbus slave and release the master.

Required parameter

Parameter	Type	Description
index	int	Master index

Return

```
ErrorID,{},ModbusClose(index);
```

Example

```
ModbusClose(0)
```

Release the Modbus master 0.

GetInBits

Command

```
GetInBits(index,addr,count)
```

Description

Read the contact register (discrete input) data from the Modbus slave.

Required parameter

Parameter	Type	Description
index	int	Master index
addr	int	The start address of contact register
count	int	Number of contact registers. Range: [1, 16].

Return

```
ErrorID,{value1,value2,...,valuen},GetInBits(index,addr,count);
```

{value1,value2,...,valuen}: values read from the contact register (number of values equals to **count**).

Example

```
GetInBits(0,3000,5)
```

Read five values from the contact register (starting from address 3000).

GetInRegs

Command

```
GetInRegs(index,addr,count,valType)
```

Description

Read the input register value with the specified data type from the Modbus slave.

Required parameter

Parameter	Type	Description
index	int	Master index
addr	int	The start address of input register
count	int	Number of values read from input register. Range: [1, 4].

Optional parameter

Parameter	Type	Description
valType	string	Data type: U16: 16-bit unsigned integer (two bytes, occupy one register); U32: 32-bit unsigned integer (four bytes, occupy two registers); F32: 32-bit single-precision floating-point number (four bytes, occupy two registers); F64: 64-bit double-precision floating-point number (eight bytes, occupy four registers). U16 by default.

Return

```
ErrorID,{value1,value2,...,valuen},GetInRegs(index,addr,count,valType);
```

{value1,value2,...,valuen}: values read from the input register (number of values equals to **count**).

Example

```
GetInRegs(0,4000,3)
```

Read three U16 values from the input register (starting from address 4000).

GetCoils

Command

```
GetCoils(index,addr,count)
```

Description

Read the coil register value from the Modbus slave.

Required parameter

Parameter	Type	Description
index	int	Master index
addr	int	The start address of coil register
count	int	Number of values read from coil register. Range: [1, 16].

Return

```
ErrorID,{value1,value2,...,valuen},GetCoils(index,addr,count);
```

{value1,value2,...,valuen}: values read from the coil register (number of values equals to **count**).

Example

```
GetCoils(0,1000,3)
```

Read three values from the coil register (starting from address 1000).

SetCoils

Command

```
SetCoils(index,addr,count,valTab)
```

Description

Write the specified value to the specified address of coil register.

Required parameter

Parameter	Type	Description
index	int	Master index
addr	int	The start address of coil register
count	int	Number of values to be written to the coil register. Range: [1, 16].
valTab	string	Values to be written to the coil register (number of values equals to **count**)

Return

```
ErrorID,{},SetCoils(index,addr,count,valTab);
```

Example

```
SetCoils(0,1000,3,{1,0,1})
```

Write three values (1 , 0, 1) in succession to the coil register starting from address 1000.

GetHoldRegs

Command

```
GetHoldRegs(index,addr, count,valType)
```

Description

Read the holding register value with the specified data type from the Modbus slave.

Required parameter

Parameter	Type	Description
index	int	Master index
addr	int	The start address of holding register
count	int	Number of values read from holding register. Range: [1, 4].

Optional parameter

Parameter	Type	Description
valType	string	Data type: U16: 16-bit unsigned integer (two bytes, occupy one register); U32: 32-bit unsigned integer (four bytes, occupy two registers); F32: 32-bit single-precision floating-point number (four bytes, occupy two registers); F64: 64-bit double-precision floating-point number (eight bytes, occupy four registers). U16 by default.

Return

```
ErrorID,{value1,value2,...,valuen},GetHoldRegs(index,addr, count,valType);
```

{value1,value2,...,valuen}: values read from the holding register (number of values equals to **count**).

Example

```
GetHoldRegs(0,3095,1)
```

Read a U16 value from the holding register (starting from address 3095).

SetHoldRegs

Command

```
SetHoldRegs(index,addr, count,valTab,valType)
```

Description

Write the specified value with specified data type to the specified address of holding register.

Required parameter

Parameter	Type	Description
index	int	Master index
addr	int	The start address of holding register
count	int	Number of values to be written to the holding register. Range: [1, 4].
valTab	string	Values to be written to the holding register (number of values equals to **count**)

Optional parameter

Parameter	Type	Description
valType	string	Data type: U16: 16-bit unsigned integer (two bytes, occupy one register); U32: 32-bit unsigned integer (four bytes, occupy two registers); F32: 32-bit single-precision floating-point number (four bytes, occupy two registers); F64: 64-bit double-precision floating-point number (eight bytes, occupy four registers). U16 by default.

Return

```
ErrorID,{},SetHoldRegs(index,addr, count,valTab,valType);
```

Example

```
SetHoldRegs(0,3095,2,{6000,300}, U16)
```

Write two U16 values (6000, 300) to the holding register starting from address 3095.

2.6 Bus register command

Command list

The bus register commands are used to read and write Profinet or Ethernet/IP bus registers.

Command	Function
GetInputBool	Get boolean value of specified input register address
GetInputInt	Get int value of specified input register address
GetInputFloat	Get float value of specified input register address
GetOutputBool	Get boolean value of specified output register address
GetOutputInt	Get int value of specified output register address
GetOutputFloat	Get float value of specified output register address
SetOutputBool	Set boolean value of specified output register address
SetOutputInt	Set int value of specified output register address
SetOutputFloat	Set float value of specified output register address

GetInputBool

Command

```
GetInputBool(address)
```

Description

Get the boolean value of the specified input register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 63]

Return

```
ErrorID,{value},GetInputBool(address);
```

value: value of the specified register address, 0 or 1.

Example

```
GetInputBool(0)
```

Get the boolean value of input register 0.

GetInputInt

Command

```
GetInputInt(address)
```

Description

Get the int value of the specified input register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23]

Return

```
ErrorID,{value},GetInputInt(address);
```

value: value of the specified register address, which is an integer (data type: int32).

Example

```
GetInputInt(1)
```

Get the int value of input register 1.

GetInputFloat

Command

```
GetInputFloat(address)
```

Description

Get the float value of the specified input register address.

Required parameter

Parameter	Type	Description

address	int	Register address, range: [0, 23]

Return

```
ErrorID,{value},GetInputInt(address);
```

value: value of the specified register address, which is a single-precision floating-point number (data type: float).

Example

```
GetInputFloat(2)
```

Get the float value of input register 2.

GetOutputBool

Command

```
GetOutputBool(address)
```

Description

Get the boolean value of the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 63]

Return

```
ErrorID,{value},GetOutputBool(address);
```

value: value of the specified register address, 0 or 1.

Example

```
GetOutputBool(0)
```

Get the boolean value of output register 0.

GetOutputInt

Command

```
GetOutputInt(address)
```

Description

Get the int value of the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23]

Return

```
ErrorID,{value},GetOutputInt(address);
```

value: value of the specified register address, which is an integer (data type: int32).

Example

```
local regInt = GetOutputInt(1)
```

Get the value of output register 1 and assign it to the variable “regInt”.

GetOutputFloat

Command

```
GetOutputFloat(address)
```

Description

Get the float value of the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23]

Return

```
ErrorID,{value},GetOutputInt(address);
```

value: value of the specified register address, which is a single-precision floating-point number (data type: float).

Example

```
local regFloat = GetOutputFloat(2)
```

Get the value of output register 2 and assign it to the variable "regFloat".

SetOutputBool

Command

```
SetOutputBool(address, value)
```

Description

Set the boolean value of the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 63]
value	int	The value to be set, 0 or 1

Return

```
ErrorID,{},SetOutputBool(address, value);
```

Example

```
SetOutputBool(0,0)
```

Set the value of output register 0 to 0.

SetOutputInt

Command

```
SetOutputInt(address, value)
```

Description

Set the int value of the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23]
value	int	The value to be set (support “integer”)

Return

```
ErrorID,{},SetOutputInt(address, value);
```

Example

```
SetOutputInt(1,123)
```

Set the value of output register 1 to 123.

SetOutputFloat

Command

```
SetOutputFloat(address, value)
```

Description

Set the float value of the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23]
value	float	The value to be set (support “single-precision floating-point number”)

Return

```
ErrorID,{},SetOutputFloat(address, value);
```

Example

```
SetOutputFloat(2,12.3)
```

Set the value of output register 2 to 12.3.

2.7 Motion command

Parameter format

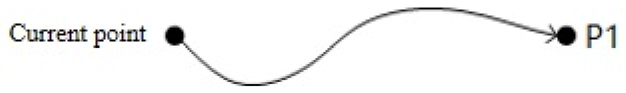
The **point parameter** and **optional parameter** in motion command are of “string” type, formatted as “key=value”, such as “joint = {10, 10, 10, 0, 0, 0}”, “user=1”. To make it easier for the user to understand the parameters, the “Type“ column of the parameters in the tables below refers to the type of the value.

Motion mode

The motion modes of the robot include the following types.

Joint motion

The robot arm plans the motion of each joint according to the difference between the current joint angle and the joint angle of the target point, so that each joint completes the motion at the same time. The joint motion does not constrain the trajectory of TCP (Tool Center Point), which is generally not a straight line.



As the joint motion is not limited by the singularity position (see the corresponding hardware guide for details), if there is no requirement for the motion trajectory, or the target point is near the singularity position, it is recommended to use joint motion.

Linear motion

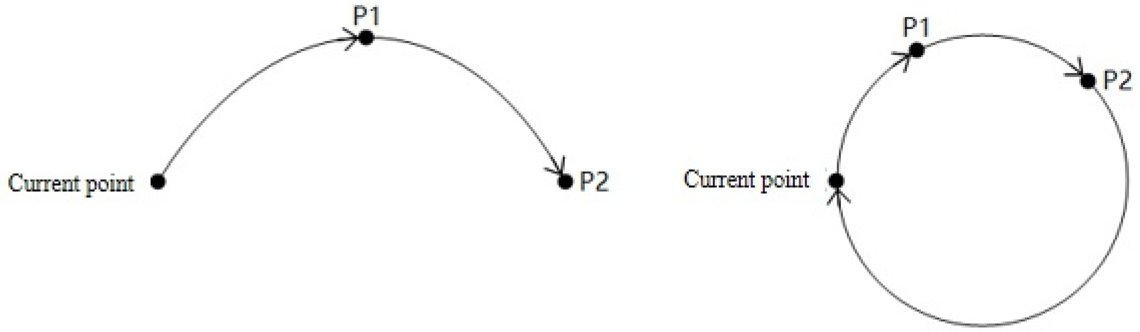
The robot arm plans the motion trajectory according to the current posture and the posture of the target point, so that the TCP motion trajectory is a straight line, and the posture of the end changes uniformly during the movement.

Current point
P1

When the trajectory may pass through the singularity position, an error will be reported when the linear motion command is delivered to the robot arm. It is recommended to re-plan the point or use joint motion near the singularity position.

Arc motion

The robot arm determines an arc or a circle through three non-collinear points: the current position, P1 and P2. The posture of the end of the robot arm during the movement is calculated by the posture interpolation of the current point and P2, and the posture of P1 is not included in the operation (i.e, the posture of the robot arm when it reaches P1 during the movement may be different from the taught posture).



When the trajectory may pass through the singularity position, an error will be reported when the arc motion command is delivered to the robot arm. It is recommended to re-plan the point or use joint motion near the singularity position.

Point parameters

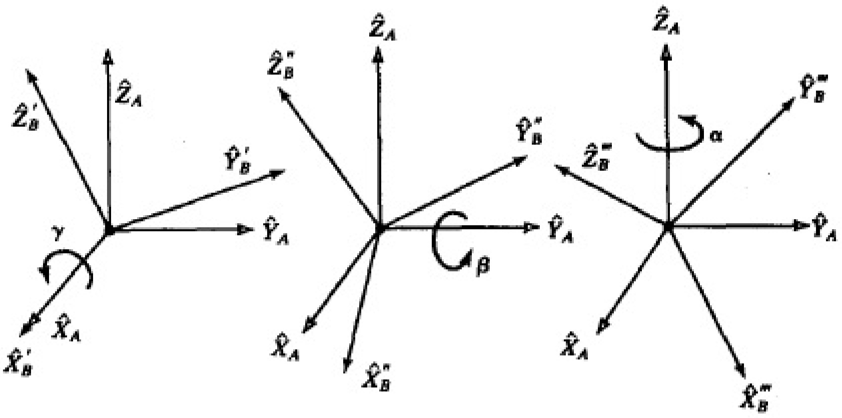
All point parameters (P) in motion command support two expressions unless otherwise specified.

- Joint variable: describe the target point using the angle of each joint (j1 – j6) of the robot arm. When it serves as a target point, the system will convert it into a posture variable through forward kinematics.

```
joint = {j1, j2, j3, j4, j5, j6}
```

- Posture variable: describe the spatial position of the target point in the user coordinate system using Cartesian coordinates (x, y, z), and describe the rotation angle of the tool coordinate system relative to the user coordinate system when the TCP (Tool Center Point) reaches a specified point using Euler angles (rx， ry， rz).

  The rotation order when calculating the Euler angle of Dobot robot is X->Y->Z, and each axis rotates around a fixed axis (user coordinate system), as shown below (rx=γ, ry=β, rz=α).



Once the rotation order is determined, the rotation matrix (where cα stands for cosα, sα stands for sinα, and so on)

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha)=R_{Z}(\alpha)R_{Y}(\beta)R_{X}(\gamma)
$$

$$
=\begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}
$$

can be derived as the equation

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha)=\begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

The posture of the end of robot arm can be calculated through the equation.

```
pose = {x, y, z, rx, ry, rz}
```

Coordinate system parameters

For motion commands related to the Cartesian coordinate system, The optional parameters “user” and “tool” are used to specify the user and tool coordinate systems of the target point.

Now the coordinate system can be specified only through the index, and the corresponding coordinate system needs to be added in the software first.

If parameters “user” and “tool” are not carried, the global user and tool coordinate system are used. See the description on User and Tool in Settings command for details (the default coordinate system is 0 when not set through commands).

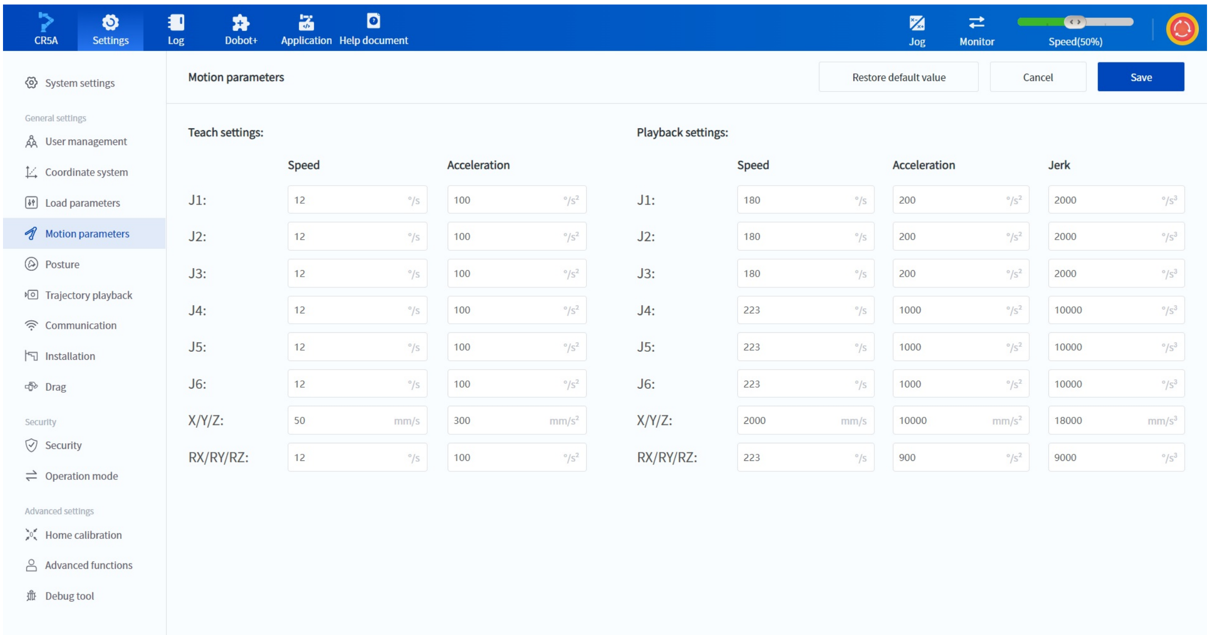
Motion parameters

Relative speed rate

The “a” and “v” in the optional parameters are used to specify the acceleration and speed rate when the robot arm executes motion commands.

```
Robot actual motion speed = maximum speed x global speed x command rate
Robot actual motion acceleration = maximum acceleration x command rate
```

The maximum speed/acceleration is limited by **Playback settings**, which can be viewed and modified in “Motion parameters” page in the software.



Motion parameters

Teach settings:

	Speed	Acceleration
J1:	12 $^\circ/s$	100 $^\circ/s^2$
J2:	12 $^\circ/s$	100 $^\circ/s^2$
J3:	12 $^\circ/s$	100 $^\circ/s^2$
J4:	12 $^\circ/s$	100 $^\circ/s^2$
J5:	12 $^\circ/s$	100 $^\circ/s^2$
J6:	12 $^\circ/s$	100 $^\circ/s^2$
X/Y/Z:	50 mm/s	300 mm/s^2
RX/RY/RZ:	12 $^\circ/s$	100 $^\circ/s^2$

Playback settings:

	Speed	Acceleration	Jerk
J1:	180 $^\circ/s$	200 $^\circ/s^2$	2000 $^\circ/s^3$
J2:	180 $^\circ/s$	200 $^\circ/s^2$	2000 $^\circ/s^3$
J3:	180 $^\circ/s$	200 $^\circ/s^2$	2000 $^\circ/s^3$
J4:	223 $^\circ/s$	1000 $^\circ/s^2$	10000 $^\circ/s^3$
J5:	223 $^\circ/s$	1000 $^\circ/s^2$	10000 $^\circ/s^3$
J6:	223 $^\circ/s$	1000 $^\circ/s^2$	10000 $^\circ/s^3$
X/Y/Z:	2000 mm/s	10000 mm/s^2	18000 mm/s^3
RX/RY/RZ:	223 $^\circ/s$	900 $^\circ/s^2$	9000 $^\circ/s^3$

The global speed can be set through the control software (upper right corner of the figure above) or the SpeedFactor command.

The command rate is carried by the optional parameters of the motion commands. When the acceleration/speed rate is not specified through the optional parameters, the value set in the motion parameters is used by default (see VelJ, AccJ, VelL, AccL commands for details, and the default value is 100 when the command setting is not called).

Example:

```
AccJ(50) -- Set the default acceleration of joint motion to 50%
VelJ(60) -- Set the default speed of joint motion to 60%
AccL(70) -- Set the default acceleration of linear motion to 70%
VelL(80) -- Set the default speed of linear motion to 80%

-- Global speed rate: 20%;

MovJ(P1) -- Move to P1 at the acceleration of (maximum joint acceleration x 50%) and speed of
(maximum joint speed x 20% x 60%) through the joint motion
MovJ(P2,{a = 30, v = 80}) -- Move to P1 at the acceleration of (maximum joint acceleration x 3
0%) and speed of (maximum joint speed x 20% x 80%) through the joint motion

MovL(P1) -- Move to P1 at the acceleration of (maximum Cartesian acceleration x 70%) and speed
 of (maximum Cartesian speed x 20% x 80%) in the linear mode
MovL(P1,{a = 40, v = 90})  -- Move to P1 at the acceleration of (maximum Cartesian acceleratio
n x 40%) and speed of (maximum Cartesian speed x 20% x 90%) in the linear mode
```

Absolute speed

The “speed” in the optional parameter of linear and arc motion commands is used to specify the absolute speed when the robot executes the command.

The absolute speed is not affected by the global speed, but limited by the maximum speed in **Playback settings** (or the maximum speed after reduction if the robot is in reduced mode), i.e. if the target speed set by the speed parameter is greater than the maximum speed in Playback settings, then the maximum speed takes precedence.

Example:

```
MovL(P1,{speed = 1000})  -- Move to P1 in the linear mode at a absolute speed of 1000
```

If the speed set in MovL is 1000 (less than the maximum speed of 2000 in Playback settings), the robot will move at a target speed of 1000 mm/s, which is independent of the global speed at this point. However, if the robot is in reduced mode (assuming a reduction rate of 10%), the maximum speed turns to 200 (less than 1000), and the robot will move at a target speed of 200 mm/s.

The “speed” and “v” cannot be set at the same time. If both exist, “speed” takes precedence.

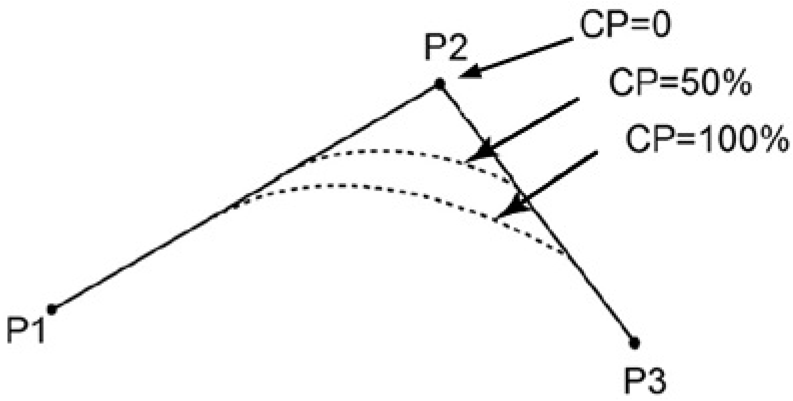
Continuous path parameters

When the robot arm moves through multiple points continuously, it can pass through the intermediate point through a smooth transition so the robot arm will not turn too bluntly. If several path points specified by the user are based on different tool coordinate systems, a smooth transition cannot be achieved.

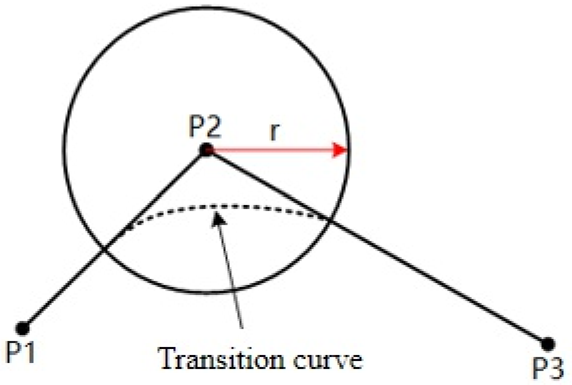
The “cp” or “r” in the optional parameters are used to specify the continuous path rate (cp) or continuous path radius (r) between the current and the next motion commands. The two parameters are mutually exclusive. If both exist, “r” takes precedence.

> ℹ NOTE
> Joint motion related commands do not support setting the continuous path radius (r). See the optional parameters of each command for details.

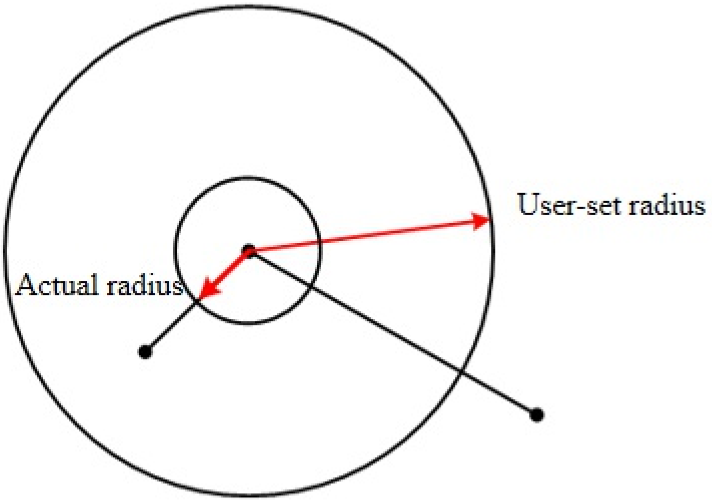
When setting the continuous path rate, the system will automatically calculate the curvature of the transition curve. The larger the CP value, the smoother the curve, as shown in the figure below. The CP transition curve will be affected by the motion speed/acceleration. Even if the point and CP values are the same, the curvature of the transition curve will vary due to different motion speed/acceleration.



When setting the continuous path radius, the system will calculate the transition curve according to the specified radius with the transition point as the center of the circle. The R transition curve is not affected by the motion speed/acceleration, but determined by the point and continuous path radius.



If the continuous path radius is set too large (more than the distance between the start/end point and the intermediate point), the system will automatically calculate the transition curve using half of the shorter distance between the start/end point and the transition point as the continuous path radius.



When the continuous path rate and radius are not specified in the optional parameters, the continuous path rate set in the motion parameter is used by default (See CP command for details. The default value is 0 when no command is called).

NOTE

As the continuous path causes the robot to move without passing the intermediate point, if the continuous path is set, the IO signal output or function settings (such as switching on/off SafeSkin) commands between two motion commands will be executed in the transition process.
If you want to execute the command when the robot arm reaches exactly the intermediate point, please set the continuous parameter of last command to 0.

Command list

Command	Function
MovJ	Joint motion
MovL	Linear motion
MovLIO	Move in linear mode and output DO
MovJIO	Move in joint mode and output DO
Arc	Arc motion
Circle	Circle motion
ServoJ	Dynamic following command based on joint space
ServoP	Dynamic following command based on Cartesian space
MoveJog	Jog the robot
RunTo	Move to the specified point
GetStartPose	Get start point of specified trajectory
StartPath	Play back recorded trajectory
RelMovJTool	Move relatively through joint motion along tool coordinate system
RelMovLTool	Move relatively in linear mode along tool coordinate system
RelMovJUser	Move relatively through joint motion along user coordinate system
RelMovLUser	Move relatively in linear mode along user coordinate system
RelJointMovJ	Move relatively through joint motion along joint coordinate system
GetCurrentCommandID	Get algorithm queue ID of current command

MovJ

Command

```
MovJ(P,user,tool,a,v,cp)
```

Description

Move from the current position to the target position through joint motion.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
cp	int	Continuous path rate. Range: [0,100].

Return

```
ErrorID,{ResultID},MovJ(P,user,tool,a,v,cp);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
MovJ(pose={-500,100,200,150,0,90},user=1, tool=0, a=20, v=50, cp=100)
```

The robot arm moves from the current position to the target Cartesian position {-500,100,200,150,0,90} through joint motion with 50% speed, 20% acceleration and 100% CP in User coordinate system 1 and Tool coordinate system 0.

MovL

Command

```
MovL(P,user,tool,a,v,cp|r)
```

Description

Move from the current position to the target position in a linear mode.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate, incompatible with “speed”. Range: (0,100].
speed	int	Target speed, incompatible with “v” (If both "speed" and "v” exist, “speed” takes precedence). Range: [1, maximum motion speed], unit: mm/s.
cp	int	Continuous path rate, incompatible with “r”. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp” (If both “r” and “cp” exist, “r” takes precedence). Unit: mm.

Return

```
ErrorID,{ResultID},MovL(P,user,tool,a,v,cp|r);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
MovL(pose={-500,100,200,150,0,90},v=60)
```

The robot arm moves from the current position to the target Cartesian position {-500,100,200,150,0,90} in the linear mode with 60% speed.

MovLIO

Command

```
MovLIO(P,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user,tool,a,v,cp|r)
```

Description

Move from the current position to the target position in a linear mode, and set the status of digital output port when the robot is moving.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables

{Mode,Distance,Index,Status}: parallel digital output parameters, used to set the specified DO to be triggered when the robot arm moves to a specified distance or percentage. Multiple groups of parameters can be set.

Parameter	Type	Description
Mode	int	Trigger mode. 0: distance percentage, 1: distance value.
Distance	int	Specified distance. If Distance is positive, it refers to the distance away from the starting point; If Distance is negative, it refers to the distance away from the target point; If Mode is 0, Distance refers to the percentage of total distance. Range: (0,100]; If Mode is 1, Distance refers to the distance value. Unit: mm.
Index	int	DO index
Status	int	DO status. 0: no signal; 1: with signal.

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate, incompatible with “speed”. Range: (0,100].
speed	int	Target speed, incompatible with “v” (If both "speed" and "v” exist, “speed” takes precedence). Range: [0, maximum motion speed], unit: mm/s.
cp	int	Continuous path rate, incompatible with “r”. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp” (If both “r” and “cp” exist, “r” takes precedence). Unit: mm.

Return

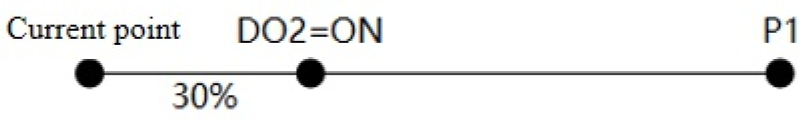
```
ErrorID,{ResultID},MovLIO(P,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user
,tool,a,v,cp|r);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example 1

```
MovLIO(pose={-500,100,200,150,0,90},{0, 30, 2, 1})
```

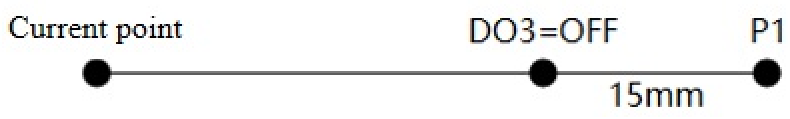
The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} in the linear mode. When it moves 30% distance away from the starting point, set DO2 to 1.



Example 2

```
MovLIO(pose={-500,100,200,150,0,90},{1, -15, 3, 0})
```

The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} in the linear mode. When it moves 15mm away from the end point, set DO3 to 0.



MovJIO

Command

```
MovJIO(P,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user,tool,a,v,cp)
```

Description

Move from the current position to the target position through joint motion, and set the status of digital output port when the robot is moving.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables

{Mode,Distance,Index,Status}: parallel digital output parameters, used to set the specified DO to be triggered when the robot arm moves to a specified distance or percentage. Multiple groups of parameters can be set.

Parameter	Type	Description
		Trigger mode. 0: distance percentage. 1: distance value.

Mode	int	The system will synthesise the joint angles into an angular vector and calculate the angular difference between the end point and the start point as the total distance of the motion.
Distance	int	Specified distance. If Distance is positive, it refers to the distance away from the starting point; If Distance is negative, it refers to the distance away from the target point; If Mode is 0, Distance refers to the percentage of total distance. Range: (0,100]; If Mode is 1, Distance refers to the angle. Unit: °.
Index	int	DO index
Status	int	DO status. 0: no signal; 1: with signal.

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
cp	int	Continuous path rate. Range: [0,100].

Return

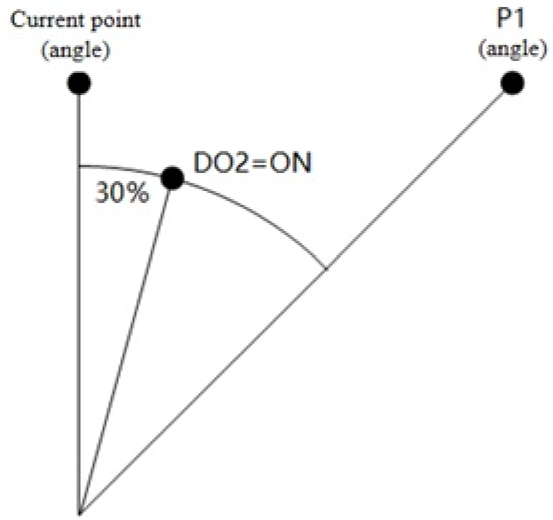
```
ErrorID,{ResultID},MovJIO(P,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user
,tool,a,v,cp);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example 1

```
MovJIO(pose={-500,100,200,150,0,90},{0, 30, 2, 1})
```

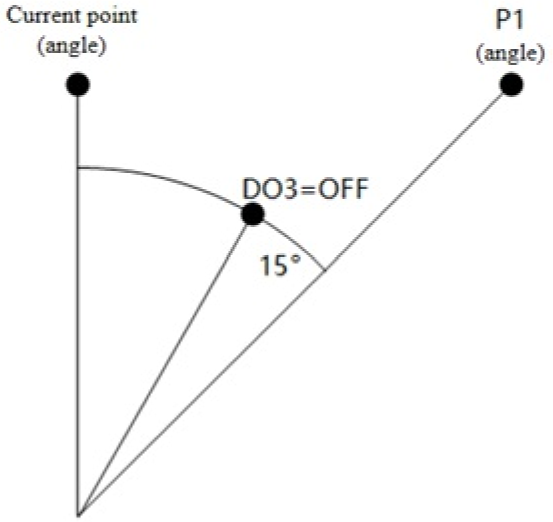
The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} through joint motion. When it moves 30% distance away from the starting point, set DO2 to 1.



Example 2

```
MovJIO(pose={-500,100,200,150,0,90},{1, -15, 3, 0})
```

The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} through joint motion. When it moves 15mm away from the end point, set DO3 to 0.



Arc

Command

```
Arc(P1,P2,user,tool,a,v,cp|r)
```

Description

Move from the current position to the target position in an arc interpolated mode.

As the arc needs to be determined through the current position, through point and target point, the current position should not be in a straight line determined by P1 and P2.

Required parameter

Parameter	Type	Description
P1	string	Through point of the arc, supporting joint variables or posture variables
P2	string	Target point, supporting joint variables or posture variables

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate, incompatible with “speed”. Range: (0,100].
speed	int	Target speed, incompatible with “v” (If both "speed" and "v” exist, “speed” takes precedence). Range: [0, maximum motion speed], unit: mm/s.
cp	int	Continuous path rate, incompatible with “r”. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp” (If both “r” and “cp” exist, “r” takes precedence). Unit: mm.

Return

```
ErrorID,{ResultID},Arc(P1,P2,user,tool,a,v,cp|r);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
Arc(pose={-350,-200,200,150,0,90},pose={-300,-250,200,150,0,90})
```

The robot moves from the current position to {-300,-250,200,150,0,90} via {-350,-200,200,150,0,90} in an arc interpolated mode.

Circle

Command

```
Circle(P1,P2,count,user,tool,a,v,cp|r)
```

Description

Move from the current position in a circle interpolated mode, and return to the current position after moving specified circles.

As the circle needs to be determined through the current position, P1 and P2, the current position should not be in a straight line determined by P1 and P2, and the circle determined by the three points cannot exceed the motion range of the robot arm.

Required parameter

Parameter	Type	Description
P1	string	Through point of the circle, supporting joint variables or posture variables
P2	string	End point of the circle, supporting joint variables or posture variables
count	int	Circles of motion. Range: [1,999].

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate, incompatible with “speed”. Range: (0,100].
speed	int	Target speed, incompatible with “v” (If both "speed" and "v” exist, “speed” takes precedence). Range: [0, maximum motion speed], unit: mm/s.
cp	int	Continuous path rate, incompatible with “r”. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp” (If both “r” and “cp” exist, “r” takes precedence). Unit: mm.

Return

```
ErrorID,{ResultID},Circle(P1,P2,count,user,tool,a,v,cp|r)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
Circle(pose={-350,-200,200,150,0,90},pose={-300,-250,200,150,0,90},1)
```

The robot moves a full circle from the current position via the Cartesian coordinate points {-350,-200,200,150,0,90} and {-300,-250,200,150,0,90}.

ServoJ

Command

```
ServoJ(Joint,t,aheadtime,gain)
```

Description

The dynamic following command based on joint space. It is generally used for the stepping function of online control to realize dynamic following by cyclic calling. The calling frequency is recommended to be set to 33Hz, that is, the interval of cyclic calling is 30ms.

⚠NOTICE

- This command is not affected by the global rate, but is constrained by the speed limit.
- If the t value is set too small, the robot will not be able to meet the specified t due to the speed limit when executing commands.
- Before calling this command, it is recommended to carry out speed planning for the running point, and issue the speed-planned points at a fixed interval t to ensure that the robot can smoothly track the target point.

Required parameter

Parameter	Type	Description
Joint	string	Target point joint variables

Optional parameter

Parameter	Type	Description
t	float	**Optional parameter.** Running time of the point, unit: s, value range: [0.02,3600.0], default value: 0.1
aheadtime	float	**Optional parameter.** Advanced time, similar to the D in PID control. Scalar, no unit, value range: [20.0,100.0], default value: 50.
gain	float	**Optional parameter.** Proportional gain of the target position, similar to the P in PID control. Scalar, no unit, value range: [200.0,1000.0], default value: 500.

The aheadtime and gain parameters together determine the response time and trajectory smoothness of the robot motion. A smaller aheadtime value or a larger gain value enables the robot to respond quickly, but may cause instability and jitter.

Return

```
ErrorID,{ResultID},ServoJ(Joint,t,aheadtime,gain)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
ServoJ(0,0,-90,0,90,0,t=0.1,aheadtime=50,gain=500)
// Called cyclically every 30ms, adding 1 to the third parameter each time
ServoJ(0,0,-89,0,90,0,t=0.1,aheadtime=50,gain=500)
```

The J3 axis moves in steps of 1 degree.

ServoP

Command

```
ServoP(Pose,t,aheadtime,gain)
```

Description

The dynamic following command based on Cartesian space. It is generally used for the stepping function of online control to realize dynamic following by cyclic calling. The calling frequency is recommended to be set to 33Hz, that is, the interval of cyclic calling is 30ms.

> ⚠NOTICE
>
> - This command is not affected by the global rate, but is constrained by the speed limit.
> - If the t value is set too small, the robot will not be able to meet the specified t due to the speed limit when executing commands.
> - Before calling this command, it is recommended to carry out speed planning for the running point, and issue the speed-planned points at a fixed interval t to ensure that the robot can smoothly track the target point.

Parameter

Parameter	Type	Description
Pose	string	Target point posture variables. The reference coordinate system is the global user and tool coordinate system, see the User and Tool command descriptions in Settings command (the default values are both 0).

Optional parameter

Parameter	Type	Description
t	float	**Optional parameter.** Running time of the point, unit: s, value range: [0.02,3600.0], default value: 0.1
aheadtime	float	**Optional parameter.** Advanced time, similar to the D in PID control. Scalar, no unit, value range: [20.0,100.0], default value: 50.
gain	float	**Optional parameter.** Proportional gain of the target position, similar to the P in PID control. Scalar, no unit, value range: [200.0,1000.0], default value: 500.

The aheadtime and gain parameters together determine the response time and trajectory smoothness of the robot motion. A smaller aheadtime value or a larger gain value enables the robot to respond quickly, but may cause instability and jitter.

Return

```
ErrorID,{ResultID},ServoP(Pose,t,aheadtime,gain)
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
ServoP(-500,100,200,150,0,90）
// Called cyclically every 30ms, adding 1 to the first parameter each time
ServoP(-499,100,200,150,0,90）
```

Move in steps of 1mm along the X-axis.

MoveJog

Command

```
MoveJog(axisID,coordtype,user,tool)
```

Description

Jog or stop jogging the robot arm. After the command is delivered, the robot arm will continuously jog along the specified axis, and it will stop once MoveJog() is delivered. In addition, when the robot arm is jogging, the delivery of MoveJog(string) with any non-specified string will also stop the motion of the robot arm.

This command is an immediate command for the welding version and is supported to be called when the project is paused.

Required parameter

Parameter	Type	Description
axisID	string	Jog the motion axis (**case sensitive**). No parameter or incorrect parameter means stopping jogging the robot. J1+ means joint 1 is moving in the positive direction and J1- means joint 1 is moving in the negative direction; J2+ means joint 2 is moving in the positive direction and J2- means joint 2 is moving in the negative direction; J3+ means joint 3 is moving in the positive direction and J3- means joint 3 is moving in the negative direction; J4+ means joint 4 is moving in the positive direction and J4- means joint 4 is moving in the negative direction; J5+ means joint 5 is moving in the positive direction and J5- means joint 5 is moving in the negative direction; J6+ means joint 6 is moving in the positive direction and J6- means joint 6 is moving in the negative direction; X+ means joint X is moving in the positive direction and X- means joint X is moving in the negative direction; Y+ means joint Y is moving in the positive direction and Y- means joint Y is moving in the negative direction; Z+ means joint Z is moving in the positive direction and Z- means joint Z is moving in the negative direction; Rx+ means joint Rx is moving in the positive direction and Rx- means joint Rx is moving in the negative direction; Ry+ means joint Ry is moving in the positive direction and Ry- means joint Ry is moving in the negative direction; Rz+ means joint Rz is moving in the positive direction and Rz- means joint Rz is moving in the negative direction.

Optional parameter

Parameter	Type	Description
coordType	int	Specify the coordinate system of axis. 0: jog along joint, 1: user coordinate system, 2: tool coordinate system. The default value is the value set at the last successful call. When axisID is a joint axis, the default value is 0. When axisID is a Cartesian axis, this parameter can only be set to 1 or 2, and an error code “-6” will be returned if it is set to 0.
user	int	User coordinate system
tool	int	Tool coordinate system

Return

```
ErrorID,{},MoveJog(axisID,coordtype,user,tool);
```

Example 1

```
MoveJog(J2-)
// Stop jogging
MoveJog()
```

Jog in the J2 negative direction, and then stop jogging.

Example 2

```
MoveJog(X+,coordtype=1,user=1)
// Stop jogging
MoveJog()
```

Jog in the X-axis positive direction in User coordinate system 1, and then stop jogging.

Example 3

```
MoveJog(J2-,coordtype=1,user=1)
// Stop jogging
MoveJog()
```

Jog in the J2 negative direction, and then stop jogging. The optional parameter is invalid when axisID specifies the the joint.

RunTo

Command

```
RunTo(P,moveType,user,tool,a,v)
```

Description

Move from the current position to the target position.

This command is an immediate command available only in the welding version, and is supported to be called when the project is paused.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables

Optional parameter

Parameter	Type	Description
moveType	int	Motion mode. 0: Joint motion, 1: Linear motion (0 by default)
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].

v	int	Velocity rate. Range: (0,100].

Return

```
ErrorID,{},RunTo(P,moveType,user,tool,a,v);
```

Example 1

```
RunTo(joint = {0, 0, 90, 0, 90, 90}, moveType = 0, a = 20, v = 50)
```

The robot arm moves from the current position to the target joint position {0, 0, 90, 0, 90, 90} through joint motion with 50% speed and 20% acceleration.

Example 2

```
RunTo( pose= {-500,100,200,150,0,90}, moveType = 1, user = 1, tool = 0, a = 20, v = 50)
```

The robot arm moves from the current position to the target Cartesian position {-500,100,200,150,0,90} through linear motion with 50% speed and 20% acceleration in User coordinate system 1 and Tool coordinate system 0.

GetStartPose

Command

```
GetStartPose(traceName)
```

Description

Get the start point of specified trajectory.

Required parameter

Parameter	Type	Description
traceName	string	Trajectory file name (with suffix). The trajectory file is stored in /dobot/userdata/project/process/trajectory/. If the name contains Chinese, the encoding method of the sending side must be set to UTF-8, otherwise it will cause an exception in receiving Chinese.

Return

```
ErrorID,{pointtype,{j1,j2,j3,j4,j5,j6},user,tool,{x,y,z,rx,ry,rz}},GetStartPose(traceName);
```

{pointtype}: the type of the return point. 0: teaching point. 1: joint variable. 2: posture variable.
The point data varies depending on the point type, examples are shown below:

```
ErrorID,{0,{j1,j2,j3,j4,j5,j6},user,tool,{x,y,z,rx,ry,rz}},GetStartPose(traceName); // teachin
g point
ErrorID,{1,{j1,j2,j3,j4,j5,j6}},GetStartPose(traceName); // joint variable
ErrorID,{2,{x,y,z,rx,ry,rz}},GetStartPose(traceName); // posture variable
```

Example

```
GetStartPose(recv_string.csv)
```

Get the first point recorded in recv_string.csv.

StartPath

Command

```
StartPath(traceName,isConst,multi,user,tool)
```

Description

Move according to the recorded points in the specified trajectory file to play back the recorded trajectory.

After delivering the trajectory playback command, you can query the running status of the robot by RobotMode command.
ROBOT_MODE_RUNNING means the robot is running the trajectory playback. ROBOT_MODE_IDLE means the trajectory playback is completed. ROBOT_MODE_ERROR means an alarm.

Required parameter

Parameter	Type	Description
traceName	string	Trajectory file name (with suffix). The trajectory file is stored in /dobot/userdata/project/process/trajectory/. If the name contains Chinese, the encoding method of the sending side must be set to UTF-8, otherwise it will cause an exception in receiving Chinese.

Optional parameter

Parameter	Type	Description
isConst	int	Whether to play back at a constant speed. 1: The trajectory will be played back at a constant speed according to the global speed rate; 0: the trajectory will be played back at the same speed as when it was recorded, and the motion speed can be scaled equivalently using the "multi" parameter, where the motion speed of the arm is not affected by

		the global speed rate.
multi	double	Speed multiplier in playback, valid only when isConst=0. Range: [0.25, 2], 1 by default.
user	int	User coordinate system index corresponding to the specified trajectory point (use the user coordinate system index recorded in the trajectory file if not specified)
tool	int	Tool coordinate system index corresponding to the specified trajectory point (use the tool coordinate system index recorded in the trajectory file if not specified)

Return

```
ErrorID,{},StartPath(traceName,isConst,multi,sample,freq,user,tool);
```

Example

```
StartPath(recv_string.csv,isConst=0,multi=1)
```

Play back the trajectory recorded in the "recv_string.csv" file at the original speed.

RelMovJTool

Command

```
RelMovJTool(offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz,user,tool,a,v,cp)
```

Description

Perform relative motion along the tool coordinate system, and the end motion is joint motion.

Required parameter

Parameter	Type	Description
offsetX	double	X-axis offset, unit: mm
offsetY	double	Y-axis offset, unit: mm
offsetZ	double	Z-axis offset, unit: mm
offsetRx	double	Rx-axis offset, unit: °
offsetRy	double	Ry-axis offset, unit: °
offsetRz	double	Rz-axis offset, unit: °

Optional parameter

Parameter	Type	Description

user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
cp	int	Continuous path rate. Range: [0,100].

Return

```
ErrorID,{ResultID},RelMovJTool(offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz,user,tool,a,
v,cp);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
RelMovJTool(10,10,10,0,0,0)
```

The robot arm moves relatively in the joint mode along the tool coordinate system, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelMovLTool

Command

```
RelMovLTool(offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz,user,tool,a,v,cp|r)
```

Description

Perform relative motion along the tool coordinate system, and the end motion is linear motion.

Required parameter

Parameter	Type	Description
offsetX	double	X-axis offset, unit: mm
offsetY	double	Y-axis offset, unit: mm
offsetZ	double	Z-axis offset, unit: mm
offsetRx	double	Rx-axis offset, unit: °
offsetRy	double	Ry-axis offset, unit: °
offsetRz	double	Rz-axis offset, unit: °

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
speed	int	Target speed, incompatible with “v” (If both "speed" and "v” exist, “speed” takes precedence). Range: [0, maximum motion speed], unit: mm/s.
cp	int	Continuous path rate, incompatible with “r”. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp” (If both “r” and “cp” exist, “r” takes precedence). Unit: mm.

Return

```
ErrorID,{ResultID},RelMovLTool(offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz, user,tool,a
,v,cp|r);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
RelMovLTool(10,10,10,0,0,0)
```

The robot arm moves relatively in the linear mode along the tool coordinate system, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelMovJUser

Command

```
RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz,user,tool,a,v,cp)
```

Description

Perform relative motion along the user coordinate system, and the end motion is joint motion.

Required parameter

Parameter	Type	Description
offsetX	double	X-axis offset, unit: mm
offsetY	double	Y-axis offset, unit: mm
offsetZ	double	Z-axis offset, unit: mm

offsetRx	double	Rx-axis offset, unit: °
offsetRy	double	Ry-axis offset, unit: °
offsetRz	double	Rz-axis offset, unit: °

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
cp	int	Continuous path rate. Range: [0,100].

Return

```
ErrorID,{ResultID},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz,user,tool,a,
v,cp);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
RelMovJUser(10,10,10,0,0.0)
```

The robot arm moves relatively in the joint mode along the user coordinate system, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelMovLUser

Command

```
RelMovLUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz,user,tool,a,v,cp|r)
```

Description

Perform relative motion along the user coordinate system, and the end motion is linear motion.

Required parameter

Parameter	Type	Description
offsetX	double	X-axis offset, unit: mm
offsetY	double	Y-axis offset, unit: mm

offsetZ	double	Z-axis offset, unit: mm
offsetRx	double	Rx-axis offset, unit: °
offsetRy	double	Ry-axis offset, unit: °
offsetRz	double	Rz-axis offset, unit: °

Optional parameter

Parameter	Type	Description
user	int	User coordinate system
tool	int	Tool coordinate system
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
speed	int	Target speed, incompatible with “v” (If both "speed" and "v” exist, “speed” takes precedence). Range: [0, maximum motion speed], unit: mm/s.
cp	int	Continuous path rate, incompatible with “r”. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp” (If both “r” and “cp” exist, “r” takes precedence). Unit: mm.

Return

```
ErrorID,{ResultID},RelMovLUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz,user,tool,a,
v,cp|r);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
RelMovLUser(10,10,10,0,0.0)
```

The robot arm moves relatively in the linear mode along the user coordinate system, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelJointMovJ

Command

```
RelJointMovJ(Offset1,Offset2,Offset3,Offset4,Offset5,Offset6,a,v,cp)
```

Description

Perform relative motion along the joint coordinate system of each axis, and the end motion mode is joint motion.

Required parameter

Parameter	Type	Description
offset1	double	J1-axis offset, unit: °
offset2	double	J2-axis offset, unit: °
offset3	double	J3-axis offset, unit: °
offset4	double	J4-axis offset, unit: °
offset5	double	J5-axis offset, unit: °
offset6	double	J6-axis offset, unit: °

Optional parameter

Parameter	Type	Description
a	int	Acceleration rate. Range: (0,100].
v	int	Velocity rate. Range: (0,100].
cp	int	Continuous path rate. Range: [0,100].

Return

```
ErrorID,{ResultID},RelJointMovJ(Offset1,Offset2,Offset3,Offset4,Offset5,Offset6,a,v,cp);
```

ResultID is the algorithm queue ID, which can be used to judge the execution order of commands.

Example

```
RelJointMovJ(10,10,10,0,0,0)
```

Displace 10° in J1, J2 and J3 respectively.

GetCurrentCommandID

Command

```
GetCurrentCommandID()
```

Description

Get the algorithm queue ID of the current command. It can be used to determine which command the robot is executing.

The following commands will be returned immediately after successful delivery, which means that the command has been accepted. Actually, the command will enter the algorithm queue and be queued for execution in sequence in the background. The ResultID returned when delivery is the ID of the command in the algorithm queue.

```
User(), Tool(), SetPayload(), DO(), ToolDO(), AO(), SetCollisionLevel(), DOGroup(), SetSafeWallEnable(), SetBackDistance(), SetPostCollisionMode(), SetUser(), SetTool(), MovJ(), MovL(), MovLIO(), MovJIO(), Arc(), Circle(), StartPath(), RelMovJTool(), RelMovLTool(), RelMovJUser(), RelMovLUser(), RelJointMovJ(), EnableSafeSkin(), SetSafeSkin()
```

Which command the robot is actually executing and whether the command has been executed need to be judged combined with the algorithm command ID and the robot status. Please refer to the example of this command.

Return

```
ErrorID,{ResultID},GetCurrentCommandID();
```

ResultID is the algorithm queue ID of the current command.

Example

```
MovJ(P1)
uint64_t p2Id = parseResultId(MovJ(P2)); // parseResultId is used to obtain ResultID returned by the command

while(true) {
  uint64_t currentId = parseResultId (GetCurrentCommndID()); // Get the ResultID of the current command
  bool isStop = parseResultId (RobotMode()) == 5; // RobotMode of 5 means the robot is enabled and idle, that is, the motion command has been executed
  if (currentId == p2Id && isStop ) { // currentId is p2Id and the motion command has been executed
    break; // Exit loop
  }
  Sleep(1);
}
```

The example above combines the algorithm queue ID and robot status to determine that the robot has moved to P2, and then exits the loop.

Real-time Feedback

The controller feeds back robot status through **port 30004, 30005 and 30006**.

- Port 30004 (real-time feedback port) receives robot information every **8ms**.
- Port 30005 feeds back robot information every **200ms**.
- Port 30006 is a **configurable** port to feed back robot information (feed back every **1000ms** by default. If you need to modify, please contact technical support).

Each packet received through the real-time feedback port has 1440 bytes, which are arranged in a standard format, as shown below.

Real-time feedback data is stored in little-endian (lower-bit-first) format, i.e., when a value is stored in more than one byte, the lower bit of the data is stored in the front byte.

For example, "1234", converted to binary 0000 0100 1101 0010, is passed in two bytes: the first byte is 1101 0010 (the lower 8 bits of the binary value) and the second byte is 0000 0100 (the upper 8 bits of the binary value).

Meaning	Data type	Number of values	Size in bytes	Byte position value	Notes
MessageSize	unsigned short	1	2	0000 – 0001	Total message length in bytes
N/A	N/A	N/A	6	0002 – 0007	Reserved
DigitalInputs	uint64	1	8	0008 – 0015	Current status of digital inputs. See DI/DO description.
DigitalOutputs	uint64	1	8	0016 – 0023	Current status of digital outputs. See DI/DO description.
RobotMode	uint64	1	8	0024 – 0031	Robot mode. See RobotMode.
TimeStamp	uint64	1	8	0032 – 0039	Unix timestamp (unit: ms)
RunTime	uint64	1	8	0040 – 0047	Robot running time (unit: ms)
TestValue	uint64	1	8	0048 – 0055	Memory test standard value 0x0123 4567 89AB CDEF

N/A	N/A	N/A	8	0056 – 0063	Reserved
SpeedScaling	double	1	8	0064 – 0071	Speed rate
N/A	N/A	N/A	8	0072 – 0087	Reserved
VRobot	double	1	8	0088 – 0095	Robot voltage
IRobot	double	1	8	0096 – 00103	Robot current
ProgramState	double	1	8	0104 – 0111	Script running status
N/A	N/A	N/A	80	0112 – 0191	Reserved
QTarget	double	6	48	0192 – 0239	Target joint position
QDTarget	double	6	48	0240 – 0287	Target joint speed
QDDTarget	double	6	48	0288 – 0335	Target joint acceleration
ITarget	double	6	48	0336 – 0383	Target joint current
MTarget	double	6	48	0384 – 0431	Target joint torque
QActual	double	6	48	0432 – 0479	Actual joint position
QDActual	double	6	48	0480 – 0527	Actual joint speed
IActual	double	6	48	0528 – 0575	Actual joint current
ActualTCPForce	double	6	48	0576 – 0623	TCP axes force (calculated by six-axis force original value)
ToolVectorActual	double	6	48	0624 – 0671	TCP actual Cartesian coordinates
TCPSpeedActual	double	6	48	0672 – 0719	TCP actual speed in Cartesian coordinate system
TCPForce	double	6	48	0720 – 0767	TCP force value (calculated by joint current)
				0768 –	TCP target Cartesian

				0815	coordinates
TCPSpeedTarget	double	6	48	0816 – 0863	TCP target Cartesian speed
MotorTemperatures	double	6	48	0864 – 0911	Joint temperature
JointModes	double	6	48	0912 – 0959	Joint control mode. 8: Position mode; 10: Torque mode.
VActual	double	6	48	960 – 1007	Joint voltage
N/A	N/A	N/A	4	1008 – 1011	Reserved
User	char	1	1	1012	User coordinate system
Tool	char	1	1	1013	Tool coordinate system
RunQueuedCmd	char	1	1	1014	Algorithm queue running flag
PauseCmdFlag	char	1	1	1015	Algorithm queue pause flag
VelocityRatio	char	1	1	1016	Joint speed ratio (0 – 100)
AccelerationRatio	char	1	1	1017	Joint acceleration ratio (0 – 100)
N/A	N/A	N/A	1	1018	Reserved
XYZVelocityRatio	char	1	1	1019	Cartesian position speed ratio (0 – 100)
RVelocityRatio	char	1	1	1020	Cartesian posture speed ratio (0 – 100)
XYZAccelerationRatio	char	1	1	1021	Cartesian position acceleration ratio (0 – 100)
RAccelerationRatio	char	1	1	1022	Cartesian posture acceleration ratio (0 – 100)
N/A	N/A	N/A	2	1023 – 1024	Reserved
BrakeStatus	char	1	1	1025	Brake status. See BrakeStatus description
EnableStatus	char	1	1	1026	Enabling status

DragStatus	char	1	1	1027	Drag status. 0: Not in drag status; 1: Joint drag status; 2: Force-control drag status.
RunningStatus	char	1	1	1028	Running status
ErrorStatus	char	1	1	1029	Alarm status
JogStatusCR	char	1	1	1030	Jog status
RobotType	char	1	1	1031	Robot type. See RobotType description
DragButtonSignal	char	1	1	1032	Drag signal
EnableButtonSignal	char	1	1	1033	Enabling signal
RecordButtonSignal	char	1	1	1034	Recording signal
ReappearButtonSignal	char	1	1	1035	Playback signal
JawButtonSignal	char	1	1	1036	Gripper control signal

SixForceOnline	char	1	1	1037	Six-axis force sensor online status.
					0: offline;
					1: online;
					2: abnormal.

CollisionState	char	1	1	1038	Collision status
ArmApproachState	char	1	1	1039	Forearm SafeSkin-approach-pause
J4ApproachState	char	1	1	1040	J4 SafeSkin-approach-pause
J5ApproachState	char	1	1	1041	J5 SafeSkin-approach-pause
J6ApproachState	char	1	1	1041	J6 SafeSkin-approach-pause
N/A	N/A	N/A	61	1043 – 1103	Reserved
VibrationDisZ	double	1	8	1104 – 1111	Z-axis jitter displacement measured by accelerometer
CurrentCommandId	uint64	1	8	1112 – 1119	Current queue id
MActual[6]	double	6	48	1120 – 1167	Actual torque of six joints
				1168 –	

				1175	
CenterX	double	1	8	1176 – 1183	Eccentric distance in X- direction (mm)
CenterY	double	1	8	1184 – 1191	Eccentric distance in Y- direction (mm)
CenterZ	double	1	8	1192 – 1199	Eccentric distance in Z- direction (mm)
User[6]	double	6	48	1200 – 1247	User coordinates
Tool[6]	double	6	48	1248 – 1295	Tool coordinates
N/A	N/A	N/A	8	1296 – 1303	Reserved
SixForceValue[6]	double	6	48	1304 – 1351	Six-axis force original value
TargetQuaternion[4]	double	4	32	1352 – 1383	Target quaternion [qw,qx,qy,qz]
ActualQuaternion[4]	double	4	32	1384 – 1415	Actual quaternion [qw,qx,qy,qz]
AutoManualMode	char	1	2	1416 – 1417	Manual/Automatic mode
N/A	N/A	N/A	22	1418 – 1440	Reserved
TOTAL			1440		1440byte package

Motion parameter feedback value description

If the motion parameters (speed, acceleration, etc.) are set individually in the project, the relevance feedback values are not updated immediately, but only when the robot executes the next motion command.

DI/DO description

DI/DO each occupies 8 bytes. Each byte has 8 bits (binary) and can represent the status of up to 64 ports each. Each byte from low to high indicates the status of one terminal. 1 indicates the corresponding terminal is ON, and 0 indicates the corresponding terminal is OFF or no corresponding terminal.

For example, the first byte of DI is 0x01 (00000001 in binary). The bits from low to high represent the status of D1_1 – DI_8 respectively, that is, DI_1 is ON and the remaining 7 DIs are OFF.

The second byte is 0x02 (00000010 in binary). The bits from low to high represent the status of D1_9 – DI_16 respectively, that is, DI_10 is ON and the remaining 7 DIs are OFF.

Different controllers vary in the number of IO terminals. The binary bits exceeding the number of IO terminals will be filled with 0.

BrakeStatus description

This byte indicates the brake status of the each joints by bit. 1 means that the corresponding joint brake is switched on. The bits correspond to the joints in the following table:

Bit	7	6	5	4	3	2	1	0
Meaning	Reserved	Reserved	J1	J2	J3	J4	J5	J6

Example:

- 0x01 (00000001): J6 brake is switched on
- 0x02 (00000010): J5 brake is switched on
- 0x03 (00000011): J5 and J6 brakes are switched on
- 0x03 (00000100): J4 brake is switched on

RobotType description

Value	Model
3	CR3
5	CR5
7	CR7
10	CR10
12	CR12
16	CR16
101	Nova 2
103	Nova 5
113	CR3A
115	CR5A
117	CR7A
120	CR10A
122	CR12A
126	CR16A
130	CR20A
150	Magician E6

5 Error Code

Error code	Description	Note
0	No error	The command has been delivered successfully.
-1	Failed to execute	The command has been received but failed to be executed.
-2	In alarm status	The robot cannot execute commands in the alarm status. You need to clear the alarm and redeliver the command.
-3	In emergency stop status	The robot cannot execute commands in the emergency stop status. You need to release the emergency stop switch, clear the alarm and redeliver the command.
-4	In power-off status	The robot cannot execute commands in the power-off status. You need to power the robot on.
-5	In script running/paused status	The robot cannot execute some commands when it is in the script running/paused status, you need to stop the script. The list of commands that can be executed while the script is running/paused is detailed below.
...	...	...
-10000	Command error	The command does not exist.
-20000	Parameter number error	The number of parameters in the command is incorrect.
-30001	When there is a parameter with name among the required parameters, it indicates that the type of any required parameter with name is incorrect. Otherwise, it indicates that the type of the first parameter is incorrect.	-3000X indicates that the type of the required parameter is incorrect. When there is a required parameter with name, it indicates that the type of the required parameter with name is incorrect, such as joint="a". Otherwise, the last bit 1 indicates that the type of the first required parameter is incorrect.
-30002	The type of the second required parameter without name is incorrect	-3000X indicates that the parameter type is incorrect. The last bit 2 indicates that the type of the second required parameter is incorrect.
...	...	...
-40001	When there is a parameter with name among the required parameters, it indicates that the range of any required parameter	-4000X indicates that the range of the required parameter is incorrect. When there is a required parameter with name, it indicates that the range of the required parameter with name is incorrect, such as joint=

	The range of the first parameter is incorrect.	{999,999,999,999,999,999,999}. Otherwise, the last bit 1 indicates that the range of the first required parameter is incorrect.
-40002	The range of the second required parameter without name is incorrect	-4000X indicates that the range of the required parameter is incorrect. The last bit 2 indicates that the range of the second required parameter is incorrect.
...	...	...
-50001	When there is a parameter with name among the optional parameters, it indicates that the type of any optional parameter with name is incorrect. Otherwise, it indicates that the type of the first optional parameter is incorrect.	-5000X indicates that the type of the optional parameter is incorrect. When there is an optional parameter with name, it indicates that the type of the optional parameter with name is incorrect, such as user="ss". Otherwise, the last bit 1 indicates that the type of the first optional parameter is incorrect.
-50002	The type of the second optional parameter without name is incorrect	-5000X indicates that the type of the optional parameter is incorrect. The last bit 2 indicates that the type of the second parameter is incorrect.
...	...	...
-60001	When there is a parameter with name among the optional parameters, it indicates that the range of any optional parameter with name is incorrect. Otherwise, it indicates that the range of the first optional parameter without name is incorrect.	-6000X indicates that the range of the optional parameter is incorrect. When there is an optional parameter with name, it indicates that the optional parameter with name is incorrect, such as a=200. The last bit 1 indicates that the range of the first optional parameter is incorrect.
-60002	The range of the second optional parameter without name is incorrect	-60000 indicates that the range of the optional parameter is incorrect. The last bit 2 indicates that the range of the second optional parameter is incorrect.
...	...	...

Note: The parameter with name refers to the parameter in the format of "key=value". The system will check the parameters from front to back. If there are multiple parameter errors, the error code of the first error detected will be reported.

Error example 1

```
// Command: MovJ(P,user,tool,a,v,cp)
MovJ(joint="a",user=1, tool=0, a=20, v=50, cp=100)
```

In the above example, the data type of the required parameter "joint" with name is incorrect, and "-30001 error" is reported.

Error example 2

```
// Command: DO(index,status,time)
DO(1,"2")
```

In the above example, the data type of the second required parameter without name is incorrect, and "-30002 error" is reported.

Error example 3

```
// Command: MovJ(P,user,tool,a,v,cp)
MovJ(pose={-500,100,200,150,0,90},user="ss", tool=0, a=20, v=50, cp=100)
```

In the above example, the data type of the optional parameter "user" with name is incorrect, and "-50001 error" is reported.

Error example 4

```
// Command: EnableRobot(load,centerX,centerY,centerZ)
EnableRobot(1.5,“a”,0,30.5)
```

In the above example, the data type of the second optional parameter without name is incorrect, and "-50002 error" is reported.

Error example 5

```
// Command: SetUser(index,table,type)
SetUser(1,{0,0,100,0,0,0}123,1)
```

The system will check the number of parameters before checking the parameter type. If there are other characters between `}` and the next `,` in the command parameter, the parameter will be incorrectly decomposed. For example, `{0,0,100,0,0,0}123` will be decomposed into `{0,0,100,0,0,0}` and `123`, and the command will report -20000 (the number of the parameter is incorrect) instead of -30002 (the type of the required parameter 2 is incorrect).

Commands that can be executed in script running/paused status

Only the following commands are received and executed when the robot is in the script running/paused status.

```
SpeedFactor(), RobotMode(), DOInstant(), ToolDOInstant(), AOInstant(), Stop(), Pause(), Contin
ue(), GetStartPose(), PositiveKin(), InverseSolution(), GetAngle(), GetPose(), EmergencyStop()
, ModbusRTUCreate(), ModbusCreate(), ModbusClose(), GetInBits(), GetInRegs(), GetCoils(), SetC
oils(), GetHoldRegs(), SetHoldRegs(), GetErrorID(), DI(), ToolDI(), AI(), ToolAI(), DIGroup(),
 GetDO(), GetAO(), GetDOGroup(), SetTool485(), SetToolPower(), SetToolMode(), CalcUser(), Calc
Tool(), GetInputBool(), GetInputInt(), GetInputFloat(), GetOutputBool(), GetOutputInt(), GetOu
```

```
tputFloat(), SetOutputBool(), SetOutputInt(), SetOutputFloat(), GetCurrentCommandId()
```