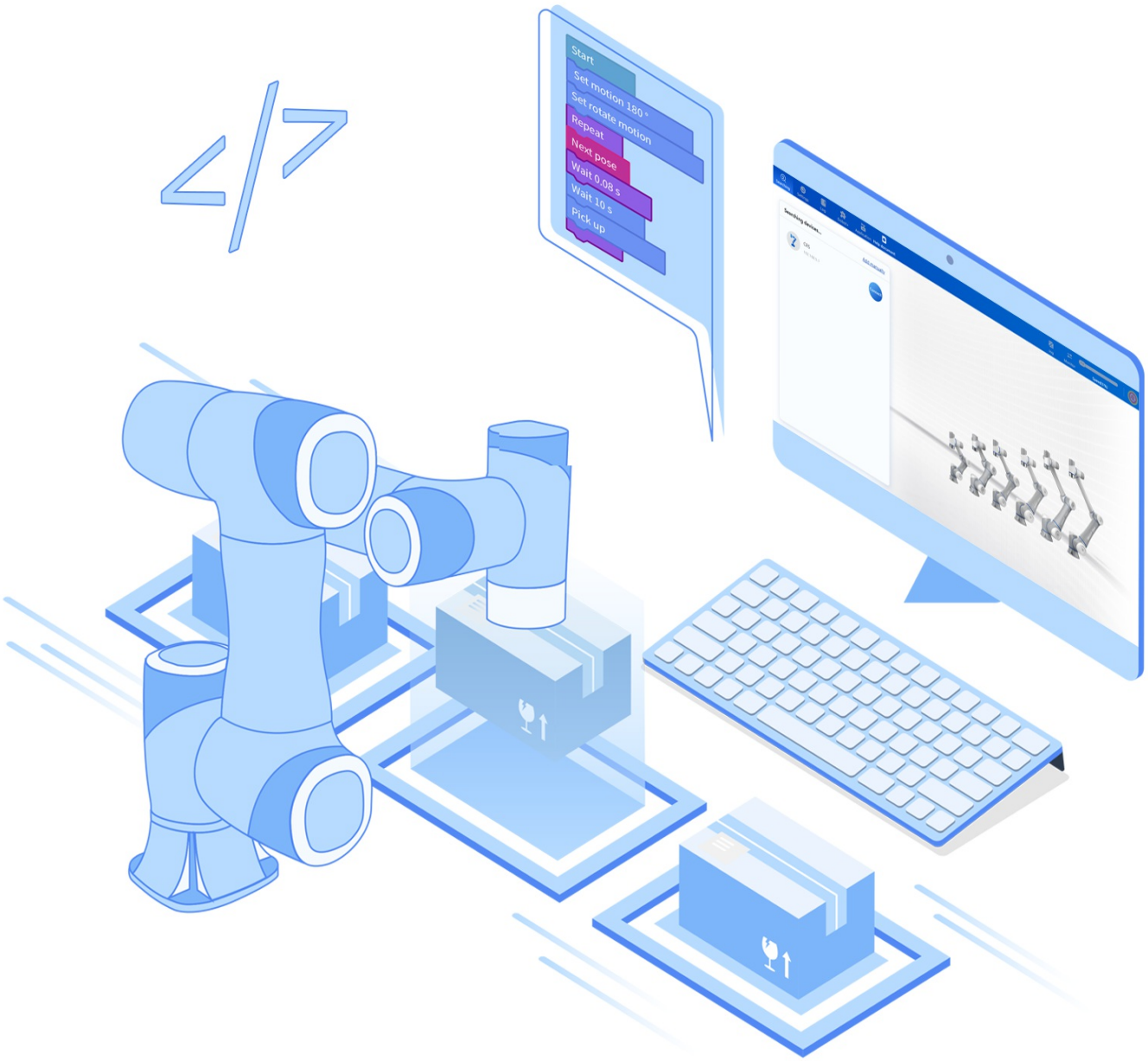
DOBOT

Dobot TCP/IP Remote Control Interface Guide



SHENZHEN DOBOT CORP 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
2026/05/27	V4.6.6	1. Added SetSingleCoil and SetSingleHoldReg commands in Modbus command 2. Updated port 30005 definition 3. Added error code "-9" in Error Code 4. Updated CreateTray command
2025/11/26	V4.6.5	1. Added GetScrName command in Calculating and obtaining command 2. Added DOGroupDEC, DIGroupDEC and GetDOGroupDEC commands in IO command 3. Added MovS, ArcIO, StartRTOffset, EndRTOffset and OffsetPara commands in Motion command 4. Added Conveyor command 5. Optimized Runto, Arc, Circle and GetStartPose commands in Motion command
2025/05/08	V4.6.2	1. Corresponds to six-axis robot controller V4.6.2 2. Added the SetFCCollision and FCCollisionSwitch commands
2025/03/20	V4.6.0	Updated the return value of the GetErrorID command

Date	Version	Revised content
2025/01/15	V4.6.0	1. Corresponds to six-axis robot controller V4.6.0 2. Added the RequestControl, trajectory recovery, log export, force control commands, and motion-related commands RelPointTool, RelPointUser, RelJoint 3. Added error code -8, and added allowed TCP commands for each status 4. Corrected the StartPath command format 5. Corrected the index range for DO/DI/ToolDO/ToolDI/ToolAI commands 6. Corrected the execution time range for ServoP and ServoJ commands 7. Optimized real-time feedback information
2024/08/15	V4.5.1	Corrected the example for ServoJ command Added return values for ServoJ and ServoP commands
2024/03/25	V4.5.1	Corresponds to six-axis robot controller V4.5.1
2023/11/28	V4.5.0	Corresponds to six-axis robot controller V4.5.0 Added the CreateTray, GetTrayPoint, ServoJ, ServoP commands, and optimized some descriptions
2023/07/28	V4.4.0	Corresponds to six-axis robot controller V4.4.0
2023/06/07	V4.3.0	Corresponds to six-axis robot 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**. Ports 30005 and 30006 are **configurable** to feed back robot information (Port 30005 feeds back every **200ms** by default, and port 30006 feeds back every **1000ms** by default. If you need to modify the feed back time, 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.

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 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.

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 Common Command

2.1 Control command

Command list

Command	Function	Command type
RequestControl	Request to change the device control mode to TCP mode	Immediate command
PowerOn	Power on the robot	Immediate command
EnableRobot	Enable the robot	Immediate command
DisableRobot	Disable the robot	Immediate command
ClearError	Clear alarms of robot	Immediate command
RunScript	Run the project	Immediate command
Stop	Stop moving (or stop running the project)	Immediate command
Pause	Pause moving (or pause running the project)	Immediate command
Continue	Continue moving (or continue running the paused project)	Immediate command
EmergencyStop	Stop the robot in an emergency	Immediate command
BrakeControl	Control the brake of specified joint	Immediate command
StartDrag	Robot enters the drag mode	Immediate command
StopDrag	Robot exits the drag mode	Immediate command

RequestControl

Command

```
RequestControl()
```

Description

Request to change the device control mode to TCP mode. Only in TCP mode can other TCP commands be executed.

TCP mode can only be entered when the robot is in a non-powered or disabled status (and not in a paused or brake-on status).

Return

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

Example

```
RequestControl()
```

Request to change to TCP mode.

Scenarios allowing TCP mode switching

Controller status	TCP mode switching allowed
Not powered on	Allowed
Disabled (non-paused, brake not on)	Allowed
Idle (enabled)	Not allowed
Drag mode	Not allowed
Single motion in progress	Not allowed
Running	Not allowed
Pause	Not allowed
Error (enabled)	Not allowed
Brake on	Not allowed
Switching between manual and automatic modes	Not allowed

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 fully powered on and initialized, as this may lead to abnormal robot behavior.

Return

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

Example

```
PowerOn()
```

Power on the robot.

EnableRobot

Command

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

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	X-axis eccentric distance, unit: mm.
centerY	double	Y-axis eccentric distance, unit: mm.
centerZ	double	Z-axis eccentric distance, 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 valu
e 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")
```

When saving a Blockly programming project in DobotStudio Pro, the system automatically adds the prefix "blockly_" to the project name. For example, if a Blockly programming project is saved in DobotStudio Pro with the name "test", the project name must be specified 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 jogging, script running, joint motion, etc.

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 a series of motions such as movj(), making the robot in a paused status. Jogging operations cannot be paused.

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 the motion. Commands in the algorithm queue during the paused status can continue, and the robot transitions to the running status.

EmergencyStop

Command

```
EmergencyStop(mode)
```

Description

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

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. Range: [1,6].
value	int	Set the brake status. 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

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

Return

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

Example

```
StartDrag()
```

Robot enters the drag mode.

StopDrag

Command

```
StopDrag()
```

Description

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

Return

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

Example

```
StopDrag()
```

Robot exits the drag mode.

2.2 Settings command

Command list

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

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	Select the index of the calibrated user coordinate system. It needs to be calibrated by software before it can be selected here. Range: [0,50].

Return

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

If ErrorID returns -1, the setting failed. 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,value,type)
```

Description:

Modify the specified user coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Select the index of the calibrated user coordinate system. It needs to be calibrated by software before it can be selected here. Range: [1,50].
value	string	The modified user coordinate system, formatted as {x, y, z, rx, ry, rz}. It is recommended to use the CalcUser command to obtain them.

Optional parameter:

Parameter	Type	Description
type	int	Whether changes to the coordinate system take effect globally. 0: The coordinate system modified through this command only takes effect when the project is running, and it will be restored to its original value upon exiting TCP mode. 1: The coordinate system modified through this command is saved by the controller, and the modified value is still remained even 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,offset)
```

Description:

Calculate the user coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Select the index of the calibrated user coordinate system. It needs to be calibrated by software before it can be selected here. Range: [0,50].
matrix	int	Calculation method. **1:** Left-multiply, meaning the coordinate system specified by the **index** is rotated relative to the base coordinate system using the values provided in **offset**. **0:** Right-multiply, meaning the coordinate system specified by the **index** is rotated relative to itself using the values provided in **offset**.
offset	string	Format: {x, y, z, rx, ry, rz}, represents the offset of the user coordinate system.

Return:

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

{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	Select the index of the calibrated tool coordinate system. It needs to be calibrated by software before it can be selected here. Range: [0,50].

Return

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

If ErrorID returns -1, it 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,value,type)
```

Description:

Modify the specified tool coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Select the index of the calibrated tool coordinate system. It needs to be calibrated by software before it can be selected here. Range: [1,50].
value	string	The modified tool coordinate system, formatted as {x, y, z, rx, ry, rz}. It represents the offset of this coordinate system relative to the default tool coordinate system.

Optional parameter:

Parameter	Type	Description
type	int	Whether changes to the coordinate system take effect globally. 0: The coordinate system modified through this command only takes effect when the project is running, and it will be restored to its original value upon exiting TCP mode. 1: The coordinate system modified through this command is saved by the controller, and the modified value is still remained even 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,offset)
```

Description:

Calculate the tool coordinate system.

Required parameter:

Parameter	Type	Description
index	int	Select the index of the calibrated tool coordinate system. It needs to be calibrated by software before it can be selected here. Range: [0,50].
matrix	int	Calculation method. **1:** Left-multiply, meaning the tool coordinate system specified by the **index** is rotated relative to the flange coordinate system using the values provided in **offset**. **0:** Right-multiply, meaning the tool coordinate system specified by the **index** is rotated relative to itself using the values provided in **offset**.
offset	string	Format: {x, y, z, rx, ry, rz}, represents the offset of the tool coordinate system.

Return:

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

{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 is equivalent to: A coordinate system with an initial posture the same as Tool coordinate system 1 is translated by {x=10, y=10, z=10} and rotated by {rx=10, ry=10, rz=10} along Tool coordinate system 1 to get the newTool.

SetPayload

Command

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

```
SetPayload(name)
```

Description

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

Method 1: Set the load parameters directly.

Required parameter 1

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

Optional parameter 1

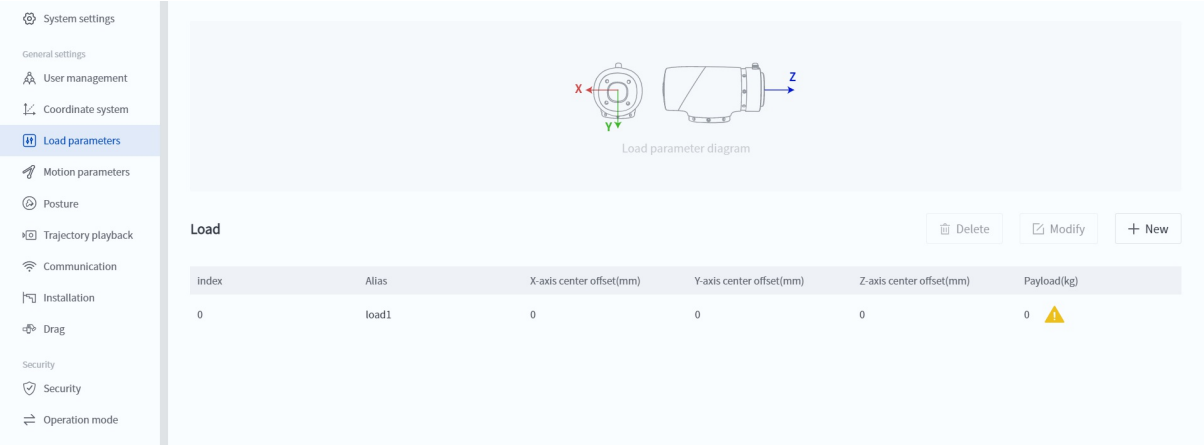
Parameter	Type	Description
x	double	X-axis eccentric coordinates of payload. Unit: mm.
y	double	Y-axis eccentric coordinates of payload. Unit: mm.
z	double	Z-axis eccentric coordinates of payload. Unit: mm.

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.



Return

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

```
ErrorID,{ResultID},SetPayload(name);
```

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 payload 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	Joint 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

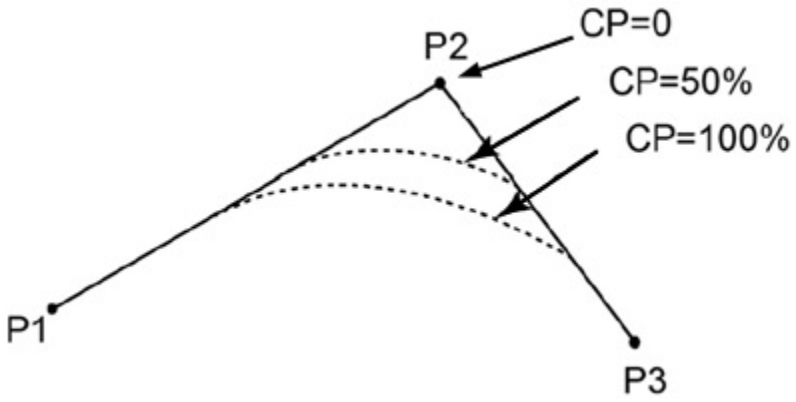
Command

```
CP(R)
```

Description

Set the CP (continuous path) ratio, which determines whether the robot transitions between multiple points with right angles or curved paths when moving continuously.

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, range: [0, 5]. **0:** Disable the 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 distance the robot will retract along its original path after a collision is detected.

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 status that the robot enters after detecting a 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 a collision. **1:** Enter pause status after detecting a 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 index, range: [0, 6]. 0: All axes are set to the same sensitivity. 1 – 6: Set the sensitivity of J1 to J6 individually.
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. If ErrorID returns -1, there may be no SafeSkin.

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 (the forearm with SafeSkin). 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:

Enable or disable 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:

Enable or disable the specified safety zone.

Required parameter:

Parameter	Type	Description
index	int	Safety zone index, which needs to be added in the software first. Range: [1,6].
value	int	ON/OFF status of safety zone. 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)
```

Enable the safety zone 1.

2.3 Calculating and obtaining command

Command list

Command	Function	Command type
RobotMode	Get current status of robot	Immediate command
PositiveKin	Forward kinematics	Immediate command
InverseKin	Inverse kinematics	Immediate command
GetAngle	Get joint coordinates of current posture	Immediate command
GetPose	Get Cartesian coordinates of current posture under the specific coordinate system	Immediate command
GetErrorID	Get current error code of robot	Immediate command
CreateTray	Create tray	Immediate command
GetTrayPoint	Get tray point	Immediate command
GetScrName	Get the name of the script currently running by the robot	Immediate command

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.
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

Perform 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. Range: [0,50].
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. Range: [0,50].

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 Tool coordinate system 1, based on the joint coordinates {0,0,-90,0,90,0}.

InverseKin

Command

```
InverseKin(X,Y,Z,Rx,Ry,Rz,useJointNear,jointNear,user,tool)
```

Description

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

Since Cartesian coordinates only define the spatial coordinates and tilt angle of the TCP, the robot can reach the same pose with different configurations. This means that one pose can have multiple joint configurations. To get a unique solution, the system requires a specified joint coordinate, and the solution closest to this joint coordinate is selected as the result.

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
useJointNear	string	Format: useJointNear=value. It is used to specify whether JointNear is used. 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.
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. Range: [0,50].
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. Range: [0,50].

Return

```
ErrorID,{J1,J2,J3,J4,J5,J6},InverseKin(X,Y,Z,Rx,Ry,Rz,useJointNear,jointNear,user,tool);
```

{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 robot end 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.

GetAngle

Command

```
GetAngle()
```

Description

Get 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 joint coordinates of current posture.

GetPose

Command

```
GetPose(user,tool)
```

Description

Get Cartesian coordinates of 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). Range: [0,50].
tool	string	Format: tool=index (index: index of the calibrated tool coordinate system). Range: [0,50].

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]},GetErrorID();
```

- 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.
- [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 ",".

Example

```
GetErrorID()
```

Get the current error code of the robot.

CreateTray

Command:

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

Description:

Create tray, e.g. 1D, 2D and 3D trays. You can create up to 20 trays. Creating a tray with an existing name will overwrite the current tray without increasing the tray count.

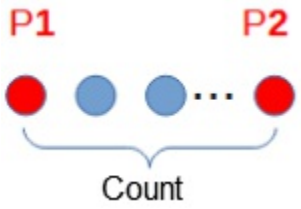
Required parameter:

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

The last two parameters are table variable. The number of values in the table varies depending on the dimension of the tray to be created, as described below.

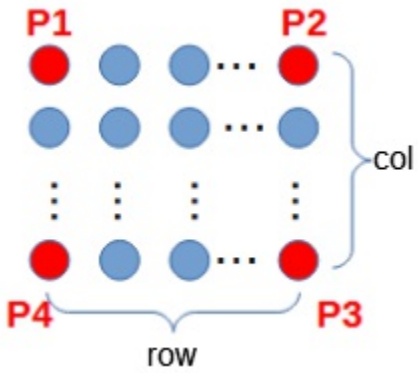
- Create 1D tray: 1D tray is a set of points equidistantly distributed on a straight line.

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 tray respectively, and the format of each point is pose = {x,y,z,rx,ry,rz}.



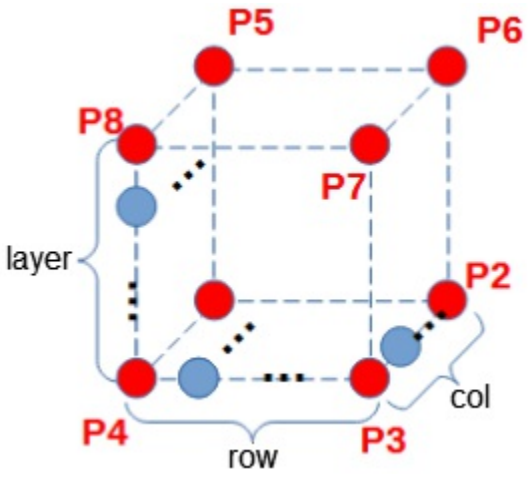
- Create 2D tray: 2D tray is a set of points distributed in an array on a plane.

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 tray.
{P1, P2, P1, P2}	table	P1, P2, P3 and P4 are the four vertices of the 2D tray respectively, and the format of each point is pose = {x,y,z,rx,ry,rz}.



- Create 3D tray: 3D tray is a set of points distributed three-dimensionally in space and can be considered as multiple 2D trays 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 (P1 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 tray respectively, and the format of each point is pose = {x,y,z,rx,ry,rz}.



Return

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

Example:

```
-- Create a 1D tray 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 tray named t2. In the example below, P1 to P4 are all points in the format
of pose = {x,y,z,rx,ry,rz}.
CreateTray(t2, {4,5}, {P1,P2,P3,P4})
-- Create a 4x5x6 3D tray named t3. In the example, P1 to P8 are all points in the format of p
ose = {x,y,z,rx,ry,rz}.
CreateTray(t2, {4,5,6}, {P1,P2,P3,P4,P5,P6,P7,P8})
```

GetTrayPoint

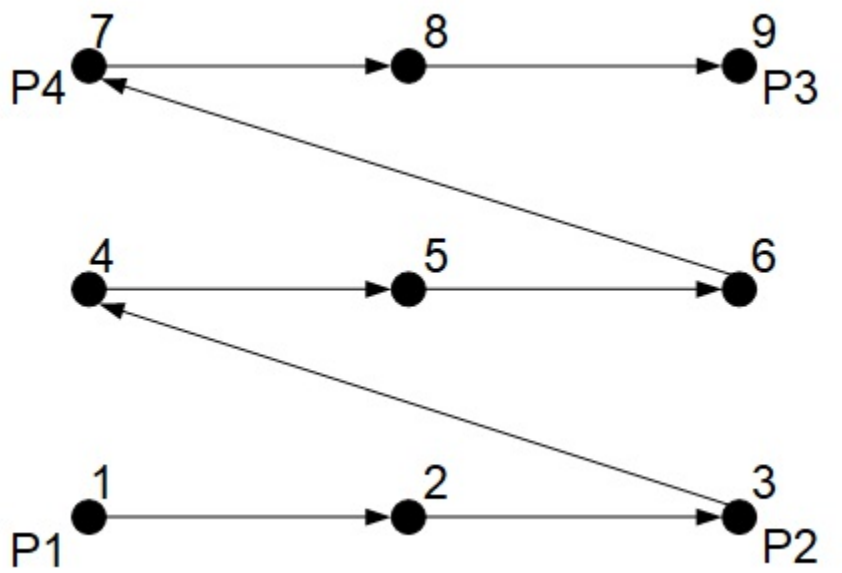
Command:

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

Description:

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

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



- 3D tray: Similar to 2D tray, 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	The created tray 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 tray named t1.
GetTrayPoint(t1,3)
```

GetScrName

Command

```
GetScrName()
```

Description

Get the name of the script currently running by the robot.

Return

```
ErrorID,{"test"};
```

- `ErrorID` return value of 0 indicates the command was successfully received.
- `ErrorID` return value of -1 indicates the system is not in normal operation, with no current running script name.
- `ErrorID` returning any other value indicates an operation failure or other abnormal condition. For details, refer to Error Code.
- `test` represents the file name of the script project.

2.4 IO command

Command list

Command	Function	Command type
DO	Set status of DO port	Queue command
DOInstant	Set status of DO port	Immediate command
GetDO	Get status of DO port	Immediate command
DOGroup	Set status of multiple DO ports	Queue command
DOGroupDEC	Set the status of multiple DO ports by assigning a decimal value	Queue command
GetDOGroup	Get status of multiple DO ports	Immediate command
GetDOGroupDEC	Get the current status of multiple DO ports, with the return value being a decimal number	Immediate command
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	Immediate command
AO	Set value of AO port	Queue command
AOInstant	Set value of AO port	Immediate command
GetAO	Get value of AO port	Immediate command
DI	Get status of DI port	Immediate command
DIGroup	Get status of multiple DI ports	Immediate command
DIGroupDEC	Get the status of multiple DI ports, with the return value being a decimal number	Queue command
ToolDI	Get status of tool DI port	Immediate command
AI	Get value of AI port	Immediate command
ToolAI	Get value of tool AI port	Immediate command
SetTool485	Set data type corresponding to RS485 interface of end tool	Immediate command
SetToolPower	Set power status of end tool	Immediate command
SetToolMode	Set the mode of tool multiple terminal	Immediate command

DO

Command

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

Description

Set the status of DO port.

Required parameter

Parameter	Type	Description
index	int	DO index. Range: [1, MAX] or [100, 1000]. MAX represents the range of DO terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.
status	int	DO status. 1: ON, 0: OFF.

Optional parameter

Parameter	Type	Description
time	int	Continuous output time. 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 (set to 0) after 2 seconds.

DOInstant

Command

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

Description

Set the status of DO port.

Required parameter

Parameter	Type	Description
index	int	DO index. Range: [1, MAX] or [100, 1000]. MAX represents the range of DO terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.
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 DO port.

Required parameter

Parameter	Type	Description
index	int	DO index. Range: [1, MAX] or [100, 1000]. MAX represents the range of DO terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.

Return

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

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

Example

```
GetDO(1)
```

Get the ON/OFF status of DO_1.

DOGroup

Command

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

Description

Set the status of multiple DO ports, supporting a maximum of 64.

Required parameter

Parameter	Type	Description
index1	int	Index of the first DO. Range: [1, MAX] or [100, 1000]. MAX represents the range of DO terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.
value1	int	Status of the first DO. 1: ON, 0: OFF.
...	...	...
indexN	int	Index of the Nth DO. Range: [1, MAX] or [100, 1000]. MAX represents the range of DO terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.
valueN	int	Status of the Nth 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.

DOGroupDEC

Command

```
DOGroupDEC({index1,index2,...,indeN},value)
```

Description

Convert the given decimal value to binary, and then set the DO port status according to the corresponding bits (low-order bit first).

Required parameter

Parameter	Type	Description
indexN	int	The number of the Nth DO terminal. Valid range: [1, 24].
value	int	Decimal value.

Return

```
ErrorID,{ResultID},DOGroupDEC({index1,index2,...,indeN},value);
```

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

Example 1

```
DOGroupDEC({1,2,3,4,5},18)
------------------
1. First, convert the decimal number 18 to binary 10010
2. Assign values according to the bits with low-order bit priority, as shown below:
  index: 1  2  3  4  5
  value: 0  1  0  0  1
3. After executing the above commands, DO2 and DO5 will be switched to the ON state.
```

Example 2

```
DOGroupDEC({5,4,3,2,1},18)
------------------
1. First, convert the decimal number 18 to binary: 10010
2. Assign values according to low-order bit priority as shown below:
  index: 5  4  3  2  1
  value: 0  1  0  0  1
3. After executing the above commands, DO4 and DO1 will be switched to the ON state.
```

GetDOGroup

Command

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

Description

Get the status of multiple DO ports.

Required parameter

Parameter	Type	Description
index	int	DO index. Range: [1, MAX] or [100, 1000]. MAX represents the range of DO terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.

Return

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

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

Example

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

Get the status of DO_1 and DO_2.

GetDOGroupDEC

Command

```
GetDOGroupDEC({index1,...,indexN})
```

Description

Get the status of multiple DO ports, form a binary number using DO levels (0/1), and then convert it to a decimal number for output.

Required parameter

Parameter	Type	Description
indexN	int	The serial number of the Nth DO terminal. Valid value range: [1, 24].

Return

```
ErrorID,{value},GetDOGroupDEC({index1,...,indexN});
```

value: Decimal number corresponding to the status of DO terminals.

Example

```
GetDOGroupDEC({1,2,3})
```

Read the level values of DO1, DO2, and DO3. If DO1 is high level, DO2 is low level, and DO3 is low level, the binary number formed is 001, which converts to the decimal number 1.

If DO1 is low level, DO2 is low level, and DO3 is high level, the binary number formed is 100, which converts to the decimal number 4.

ToolDO

Command

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

Description

Set the status of tool DO port.

Required parameter

Parameter	Type	Description
index	int	Tool DO index. Range: [1, MAX]. MAX represents the range of DO terminals supported by the current end effector, which varies depending on the number of DO resources available for different end effectors.
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 DO port.

Required parameter

Parameter	Type	Description
index	int	Tool DO index. Range: [1, MAX]. MAX represents the range of DO terminals supported by the current end effector, which varies depending on the number of DO resources available for different end effectors.
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 DO port.

Required parameter

Parameter	Type	Description
index	int	Tool DO index. Range: [1, MAX]. MAX represents the range of DO terminals supported by the current end effector, which varies depending on the number of DO resources available for different end effectors.

Return

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

value: The 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 AO port.

Required parameter

Parameter	Type	Description
index	int	AO index. Range: 1/2.
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 AO port.

Required parameter

Parameter	Type	Description
index	int	AO index. Range: 1/2.
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 AO port.

Required parameter

Parameter	Type	Description
index	int	AO index. Range: 1/2.

Return

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

value: The value of AO port.

Example

```
GetAO(1)
```

Get the value of AO_1.

DI

Command

```
DI(index)
```

Description

Get the status of DI port.

Required parameter

Parameter	Type	Description
index	int	DI index. Range: [1, MAX] or [100, 1000]. MAX represents the range of DI terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.

Return

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

value: The 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 multiple DI ports, supporting a maximum of 64.

Required parameter

Parameter	Type	Description
index	int	DI index. Range: [1, MAX] or [100, 1000]. MAX represents the range of DI terminals supported by the current controller, which varies depending on the controller model. A value in the range [100, 1000] requires hardware support for an extended IO module.

Return

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

{value1,value2,...,valueN}: The status of index1 to indexN. 0: OFF, 1: ON.

Example

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

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

DIGroupDEC

Command

```
DIGroupDEC({index1,index2,...,indexN})
```

Description

Read the status of multiple DI ports, form a binary number using DI levels (0/1), and then convert it to a decimal number for output.

Required parameter

Parameter	Type	Description
indexN	int	The serial number of the Nth DI terminal. Valid range: [1, 24].

Return

```
ErrorID,{value},DIGroupDEC({index1,index2,...,indexN});
```

value: Decimal number corresponding to the status of DI terminals.

NOTE

- The return value is calculated using low-order bit priority. For example, if a DI group is defined in the order {1, 3, 5}, the binary number is formed by saving the level states of DI5, DI3, and DI1 in sequence.
- The order of DI terminal numbers matters. The results of DIGroupDEC({1,2,3}) and DIGroupDEC({1,3,2}) are different. Assuming DI1 is high level, DI2 is high level, and DI3 is low level, then DIGroupDEC({1,2,3}) forms a binary number of 011, which converts to a decimal number of 3. Meanwhile, DIGroupDEC({1,3,2}) forms a binary number of 101, converting to a decimal number of 5.

Assuming the signal status of DI1 to DI6 is as shown in the table below:

Index (DI number)	1	2	3	4	5	6
DI signal state	ON	OFF	ON	OFF	ON	OFF
Corresponding binary	1	0	1	0	1	0

Example 1

```
DIGroupDEC({1,3,6})
----------------------
1. First, read the DI signal states for {1,3,6}, which are ON, ON, OFF.
2. When returning the decimal value, save the binary number following a low-bit priority order
; this means saving the binary number according to the signal states of DI6, DI3, DI1. In this
 case, the actually saved binary number is 011 (where DI6=0, DI3=1, DI1=1).
3. Convert the binary 011 to decimal, resulting in a return value of 3 once the command above
is executed
```

Example 2

```
DIGroupDEC({1,6,3})
----------------------
1. First, read the signal states for {1,6,3} as ON, OFF, ON.
2. When returning the decimal value, save the binary number following a low-bit priority order
; this means saving the binary according to the signal states of DI3, DI6, DI1. In this case,
```

```
the actually saved binary number is 101 (where DI3=1, DI6=0, DI1=1).
3. Convert the binary 101 to decimal, resulting in a return value of 5 once the command above
is executed
```

ToolDI

Command

```
ToolDI(index)
```

Description

Get the status of tool DI port.

Required parameter

Parameter	Type	Description
index	int	Tool DI index. Range: [1, MAX]. MAX represents the range of DI terminals supported by the current controller, which varies depending on the controller model.

Return

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

value: The 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 AI port.

Required parameter

Parameter	Type	Description
index	int	AI index. Range: 1/2.

Return

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

value: The input value of AI port.

Example

```
AI(1)
```

Get the AI_1 input.

ToolAI

Command

```
ToolAI(index)
```

Description

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

Required parameter

Parameter	Type	Description
index	int	Tool AI index. Range: [1, MAX]. MAX represents the range of AI terminals supported by the current controller, which varies depending on the controller model.

Return

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

value: The input value of tool AI port.

Example

```
ToolAI(1)
```

Get the value of tool AI_1.

SetTool485

Command:

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

Description:

Set the data type for 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	Select the length of the stop bit. Range: 1, 2. 1 by default.
identify	int	When the robot has multiple aviation sockets, this parameter specifies the socket to be configured. 1: Aviation socket 1. 2: Aviation socket 2.

Return

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

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)
```

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. It is recommended to wait at least **4 ms** between consecutive calls to this interface.

> **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-of-arm tool. 0: power off; 1: power on.

Return

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

Example:

```
SetToolPower(0)
```

Power off the tool.

SetToolMode

Command:

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

Description:

For robots with end AI interfaces that share a multiplex terminal with RS485, you can use this command to set the mode of the multiplex terminal. The default mode is **485 mode**.

> **NOTE**
> For robots 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 mode.

Optional parameter

Parameter	Type	Description
type	int	When mode is 1, this parameter is invalid. When mode is 2, the analog input mode can be set (see type parameter description for details). The units digit indicates the mode of AI1, the tens digit indicates the mode of AI2. If the tens digit is 0, only the unit digit can be specified.
identify	int	When the robot has multiple aviation sockets, this parameter specifies the socket to be configured. 1: Aviation socket 1. 2: Aviation socket 2. Default is Aviation socket 1 if not specified.

- Meaning of **type** parameter:
  - 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

Return

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

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	Command type
ModbusCreate	Create Modbus master	Immediate command
ModbusRTUCreate	Create Modbus master based on RS485	Immediate command
ModbusClose	Disconnect with Modbus slave	Immediate command
GetInBits	Read contact registers	Immediate command
GetInRegs	Read input registers	Immediate command
GetCoils	Read coil registers	Immediate command
SetCoils	Write to coil registers	Immediate command
SetSingleCoil	Write to single coil register	Immediate command
GetHoldRegs	Read holding registers	Immediate command
SetHoldRegs	Write to holding registers	Immediate command
SetSingleHoldReg	Write to single holding register	Immediate command

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 for different types of registers follow the standard Modbus protocol:

Register type	Read register	Write single register	Write multiple registers
Coil register	01	05	0F
Contact (discrete input) 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 with the slave. A maximum of 5 devices can be connected at the same time.

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 the RS485 interface and establish connection with the slave. A maximum of 5 devices can be connected at the same time.

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. 8 by default.
stop_bit	int	Select the length of the stop bit. 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 from the Modbus slave and release the master.

Required parameter

Parameter	Type	Description
index	int	The master index returned when a master is created.

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	The master index returned when a master is created.
addr	int	The starting address of the 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 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	The master index returned when a master is created.
addr	int	The starting address of the input register.
count	int	Number of values read from input register. Range: [1, 4].

Optional parameter

Parameter	Type	Description
valType	string	Data format: 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 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	The master index returned when a master is created.
addr	int	The starting address of the 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 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 specified values to the coil register at the designated address.

Required parameter

Parameter	Type	Description
index	int	The master index returned when a master is created.
addr	int	The starting address of the 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 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.

SetSingleCoil

Command

```
SetSingleCoil(index,addr,val)
```

Description

Write specified value to the coil register at the designated address.

Required parameter

Parameter	Type	Description
index	int	The master index returned when a master is created.
addr	int	The starting address of the coil register, depending on the slave's configuration.
val	int	Value to be written to the coil register, allowed range: 0 or 1.

Return

```
ErrorID,{},SetSingleCoil(index,addr,val);
```

Example

```
SetSingleCoil(0,1000,1})
```

Write 1 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	The master index returned when a master is created, supporting up to 5 devices. Range: [0,4].
addr	int	The starting address of the holding register.
count	int	Number of values read from holding register.

Optional parameter

Parameter	Type	Description
valType	string	Data format: 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 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	The master index returned when a master is created, supporting up to 5 devices. Range: [0,4].
addr	int	The starting address of the 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 register (number of values equals to **count**)

Optional parameter

Parameter	Type	Description
valType	string	Data format: 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.

SetSingleHoldReg

Command

```
SetSingleHoldReg(index,addr,val)
```

Description

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

Required parameter

Parameter	Type	Description
index	int	The master index returned when a master is created, supporting up to 5 devices. Range: [0, 4].
addr	int	The starting address of the holding register, depending on the slave's configuration.
val	int	Value to be written to the holding register. The data will be automatically converted to U16; values exceeding the range will be clipped.

Return

```
ErrorID,{},SetSingleHoldReg(index,addr,val);
```

Example

```
SetSingleHoldReg(0,3095,6000)
```

Write 6000 (U16 value) 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	Command type
GetInputBool	Get boolean value from the specified input register address	Immediate command
GetInputInt	Get int value from the specified input register address	Immediate command
GetInputFloat	Get float value from the specified input register address	Immediate command
GetOutputBool	Get boolean value from the specified output register address	Immediate command
GetOutputInt	Get int value from the specified output register address	Immediate command
GetOutputFloat	Get float value from the specified output register address	Immediate command
SetOutputBool	Set boolean value at the specified output register address	Immediate command
SetOutputInt	Set int value at the specified output register address	Immediate command
SetOutputFloat	Set float value at the specified output register address	Immediate command

GetInputBool

Command:

```
GetInputBool(address)
```

Description:

Get the boolean value from the specified input register address.

Required parameter

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

Return

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

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

Example:

```
GetInputBool(0)
```

Get the boolean value of input register address 0.

GetInputInt

Command:

```
GetInputInt(address)
```

Description:

Get the int value from the specified input register address.

Required parameter

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

Return

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

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

Example:

```
GetInputInt(1)
```

Get the int value of input register address 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},GetInputFloat(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 address 2.

GetOutputBool

Command:

```
GetOutputBool(address)
```

Description:

Get the boolean value from the specified output register address.

Required parameter

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

Return

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

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

Example:

```
GetOutputBool(0)
```

Get the boolean value of output register address 0.

GetOutputInt

Command:

```
GetOutputInt(address)
```

Description:

Get the int value from the specified output register address.

Required parameter

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

Return

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

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

Example:

```
GetOutputInt(1)
```

Get the int value of output register address 1.

GetOutputFloat

Command:

```
GetOutputFloat(address)
```

Description:

Get the float value from the specified output register address.

Required parameter

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

Return

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

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

Example:

```
GetOutputFloat(2)
```

Get the float value of output register address 2.

SetOutputBool

Command:

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

Description:

Set the boolean value at 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 at the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23].
value	int	The value to be set, supporting signed 32-bit integers.

Return

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

Example:

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

Set the value of output register address 1 to 123.

SetOutputFloat

Command:

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

Description:

Set the float value at the specified output register address.

Required parameter

Parameter	Type	Description
address	int	Register address, range: [0, 23].
value	float	The value to be set, supporting single-precision floating-point numbers.

Return

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

Example:

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

Set the float value of output register address 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 robot supports the following motion modes:

Joint motion

The robot plans the motion of each joint based on the difference between the current and target joint angles, ensuring that all joints complete their motion simultaneously. Joint motion does not constrain the TCP (Tool Center Point) trajectory, and the path is usually not a straight line.

Joint motion is not restricted by singularities (refer to the corresponding hardware guide for details). If there are no specific trajectory requirements or the target point is near a singularity, joint motion is recommended.

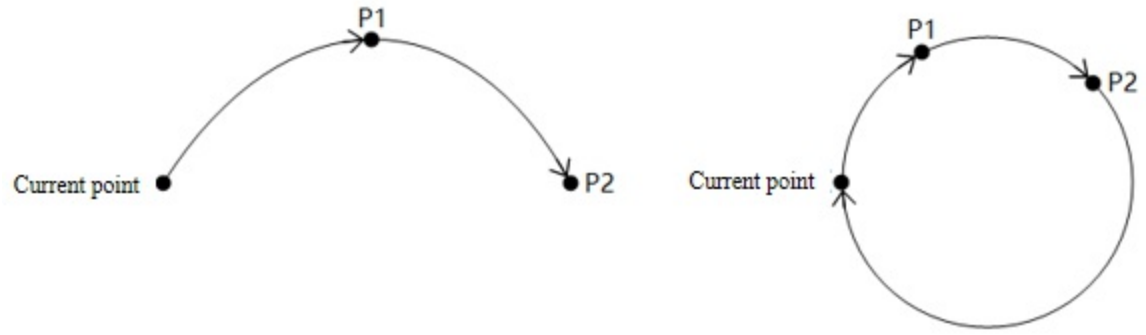
Linear motion

The robot plans the motion trajectory based on the current posture and the target point's posture, ensuring that the TCP moves in a straight line and the end posture changes uniformly during the motion.

When the trajectory passes through a singularity, issuing a linear motion command will result in an error. It is recommended to re-plan the point or use joint motion near the singularity.

Arc motion

The robot determines an arc or a circle based on three non-collinear points: the current position, P1, and P2. During the motion, the end posture of the robot is interpolated between the current point and the posture at P2, while the posture at P1 is not considered (i.e., when the robot reaches P1 during the motion, its posture may differ from the taught posture).



When the trajectory passes through a singularity, issuing an arc motion command will result in an error. It is recommended to re-plan the point or use joint motion near the singularity.

Point parameters

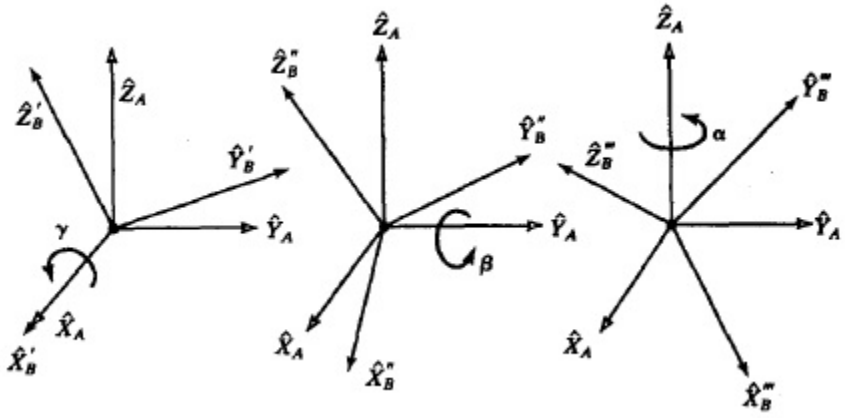
All point parameters (P) in motion command support two expressions unless otherwise specified.

- Joint variable: describe the target point using the angle of each joint (j1 – j6) of the robot. 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 specified through commands).

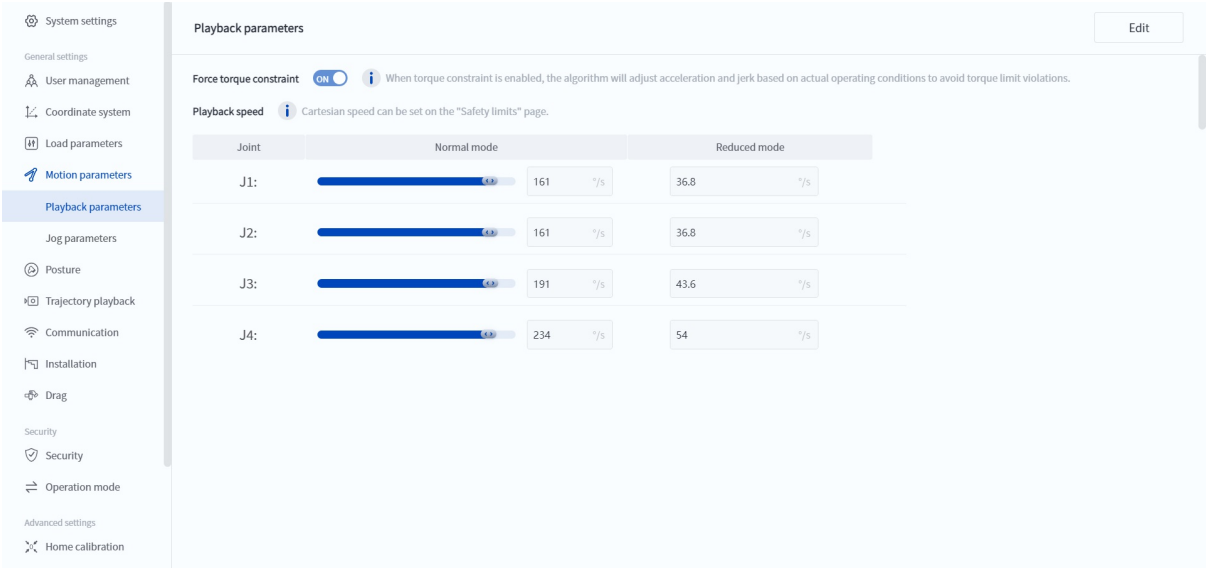
Motion parameters

Relative speed

The optional parameters "a" and "v" are used to specify the acceleration and speed ratio for the robot when executing the motion command.

```
Robot actual speed = Maximum speed x Global speed x Command speed
Robot actual acceleration = Maximum acceleration x Command speed
```

Where, the maximum speed/acceleration is controlled by the **playback parameters** and can be viewed and modified on the Motion parameters page of DobotStudio Pro.



The global speed can be set through the DobotStudio Pro (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 parameters** (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 parameters, 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 parameters), 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 (with a reduction rate of 10%), the maximum speed becomes 200 mm/s, which is lower than 1000 mm/s, so 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.

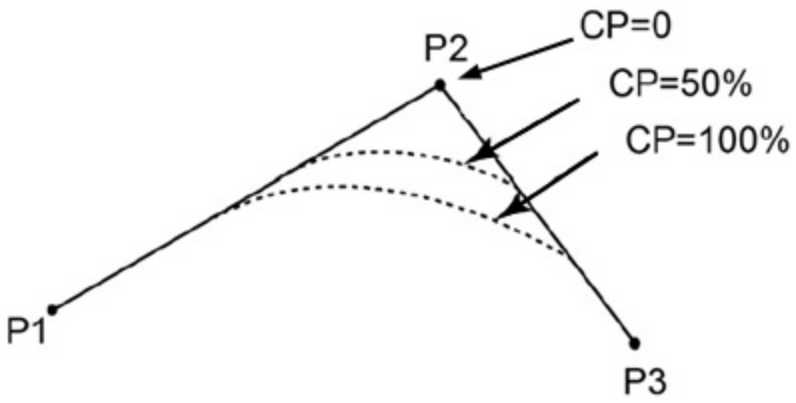
Continuous path parameters

When the robot moves through multiple points continuously, it can pass through the intermediate point through a smooth transition so the robot will not turn too bluntly. Smooth transitions cannot be applied if the user-specified path points are based on different tool coordinate systems.

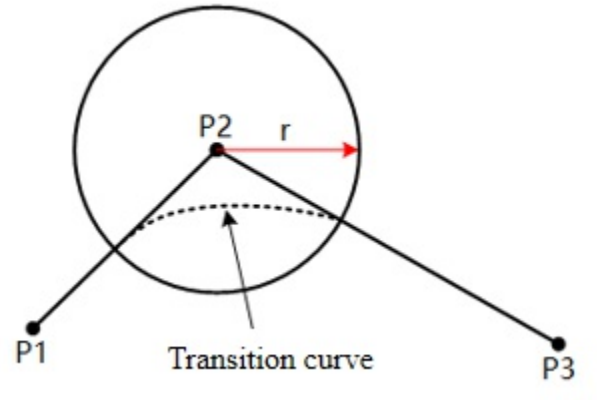
The “cp” or “r” in the optional parameters are used to specify the continuous path rate (cp) or continuous path radius (r) between the current and the next motion commands. The two parameters are mutually exclusive. If both exist, “r” takes precedence.

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

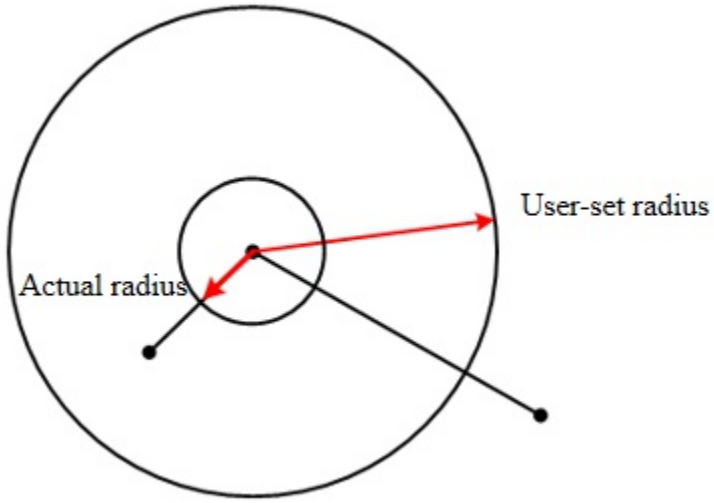
When setting the CP, the system automatically calculates the curvature of the transition curve. The larger the CP value, the smoother the curve, as shown in the figure below. The CP transition curve is affected by speed and acceleration. Even if the path points and CP values are the same, different speeds/accelerations will result in different curvatures.



When setting the R, the system calculates the transition curve using the transition point as the center and the specified radius. The R transition curve is not affected by speed or acceleration, but is determined by the path points and the specified 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 ratio and radius are not specified in the optional parameters, the continuous path ratio 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 CP causes the robot to bypass intermediate points, if CP is set, any IO signal outputs or function settings (e.g., enabling/disabling SafeSkin) between two motion commands will be executed during the transition.

If you want the robot to execute commands precisely when it reaches the intermediate point, set the CP parameters of the previous command to 0.

Command list

Command	Function	Command type
MovJ	Joint motion	Queue command
MovL	Linear motion	Queue command
MovLIO	Move in linear mode and output DO	Queue command
MovJIO	Move in joint mode and output DO	Queue command
Arc	Arc motion	Queue command
ArcIO	Arc motion with digital output	Queue command
Circle	Circle motion	Queue command
ServoJ	Dynamic following command based on joint space	Queue command
ServoP	Dynamic following command based on Cartesian space	Queue command
MoveJog	Jog the robot	Immediate command
RunTo	Move to the specified point	Immediate command
GetStartPose	Get start point of specified trajectory	Immediate command
MovS	Fit the imported trajectory	Queue command
StartPath	Playback recorded motion trajectory	Queue command
RelMovJTool	Perform relative joint motion along the tool coordinate system	Queue command
RelMovLTool	Perform relative linear motion along the tool coordinate system	Queue command
RelMovJUser	Perform relative joint motion along the user coordinate system	Queue command
RelMovLUser	Perform relative linear motion along the user coordinate system	Queue command
RelJointMovJ	Perform relative joint motion along the joint coordinate system	Queue command

RelPointTool	Perform Cartesian point offset along the tool coordinate system	Immediate command
RelPointUser	Perform Cartesian point offset along the user coordinate system	Immediate command
RelJoint	Perform relative joint position offset	Immediate command
GetCurrentCommandID	Get algorithm queue ID of current command	Immediate command
StartRTOffset	Start coordinate system offset	Queue command
EndRTOffset	End coordinate system offset	Queue command
OffsetPara	Set coordinate system offset value	Immediate 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. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

Optional parameter

Parameter	Type	Description
user	string	Format: "user=index" (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1,100].
cp	string	Format: "cp=value". **value:** Continuous path ratio. 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 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|speed,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. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

Optional parameter

Parameter	Type	Description
user	string	Format: "user=index" (index: index of the calibrated user coordinate system). Range: [0, 50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0, 50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1, 100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command, incompatible with "speed". Range: [1, 100].
speed	string	Format: "speed=value". **value:** The target speed for the robot when executing this command, incompatible with "v". If both "speed" and "v" exist, "speed" takes precedence. Range: [1, maximum motion speed]. Unit: mm/s.
cp	string	Format: "cp=value". **value:** Continuous path ratio, incompatible with

cp	string	"r". Range: [0, 100].
r	string	Format: "r=value". **value:** 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|speed,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 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|speed,cp|r)
```

Description

Move from the current position to the target position in a linear mode, while simultaneously setting the status of digital output port.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

{Mode,Distance,Index,Status}: Parallel digital output parameters. It is used to set the specified DO to be triggered when the robot moves to a specified distance or percentage. Multiple groups (at least 1 group) of parameters can be set, with the following meanings:

Parameter	Type	Description
Mode	int	Trigger mode. 0: distance percentage. 1: distance value.
		Specified distance. If Distance is positive, it refers to the distance away from the starting point;

Distance	int	point; If Mode is 0, Distance refers to the percentage of the distance between the starting point and the target point. The value range is [0, 100]; If Mode is 1, Distance refers to the distance value. Unit: mm.
Index	int	DO index. Range: [1, 24] or [100, 1000]. A value in the range [100, 1000] requires hardware support for an extended IO module. The value range may vary depending on the device model.
Status	int	DO status. 1: ON. 0: OFF.

Optional parameter

Parameter	Type	Description
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command, incompatible with "speed". Range: [1,100].
speed	string	Format: "speed=value". **value:** The target speed for the robot when executing this command, incompatible with "v". If both "speed" and "v" exist, "speed" takes precedence. Range: [1, maximum motion speed]. Unit: mm/s.
cp	string	Format: "cp=value". **value:** Continuous path ratio, incompatible with "r". Range: [0,100].
r	string	Format: "r=value". **value:** Continuous path radius, incompatible with "cp". If both "r" and "cp" exist, "r" takes precedence. Unit: mm.

Return

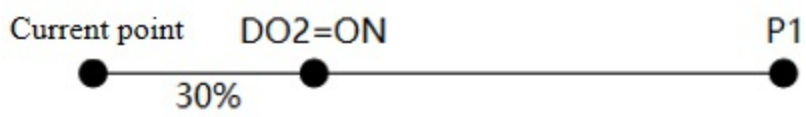
```
ErrorID,{ResultID},MovLIO(P,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user,tool,a,v|speed,cp|r);
```

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, while simultaneously setting the status of digital output port.

Required parameter

Parameter	Type	Description
P	string	Target point, supporting joint variables or posture variables. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

{Mode,Distance,Index,Status}: Parallel digital output parameters. It is used to set the specified DO to be triggered when the robot 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. The system will synthesize the joint angles into an angular vector and calculate the angle 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 the distance between the starting point and the target point. The value range is [0, 100]; If Mode is 1, Distance refers to the distance in degrees. Unit: °.
Index	int	DO index. Range: [1, 24] or [100, 1000]. A value in the range [100, 1000] requires hardware support for an extended IO module. The value range may vary depending on the device model.
Status	int	DO status. 1: ON. 0: OFF.

Optional parameter

Parameter	Type	Description
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	int	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	int	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1,100].
cp	int	Format: "cp=value". **value:** Continuous path ratio. Range: [0,100].

Return

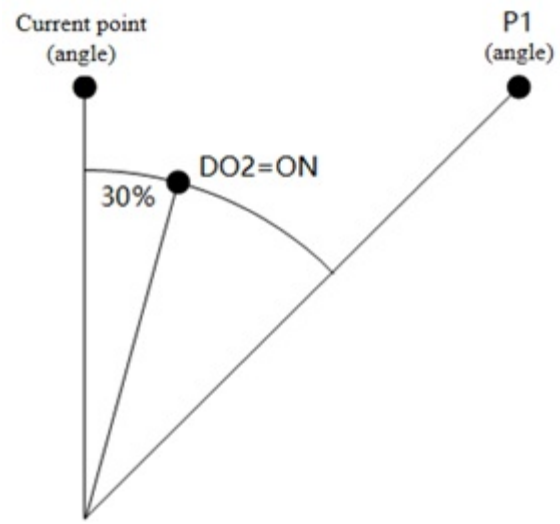
```
ErrorID,{ResultID},MovJIO(P,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user
,tool,a,v,cp);
```

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})
```

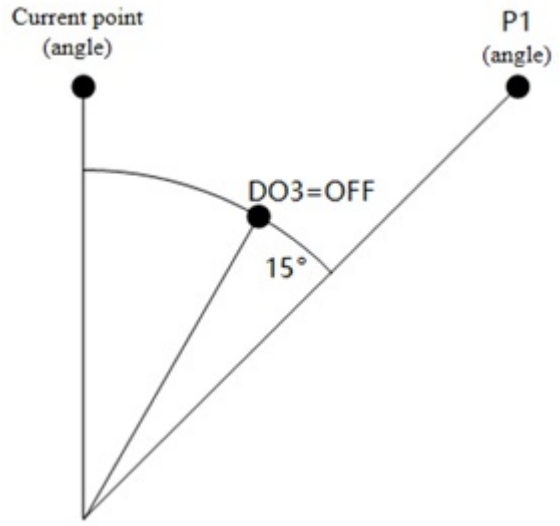
The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} through joint motion. When it moves 30% distance away from the starting point, set DO2 to 1.



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

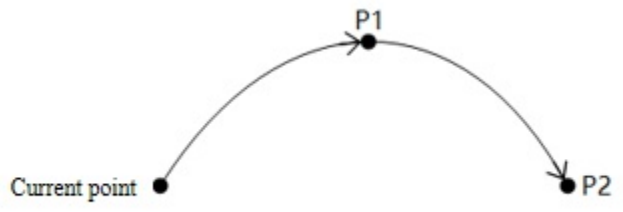
Command

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

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.



During the motion, the end posture of the robot arm is calculated through interpolation between the current point and the posture of point P2. The posture of point P1 does not participate in the calculation (meaning the posture of the robot arm when it reaches point P1 during the motion may differ from the teaching posture).

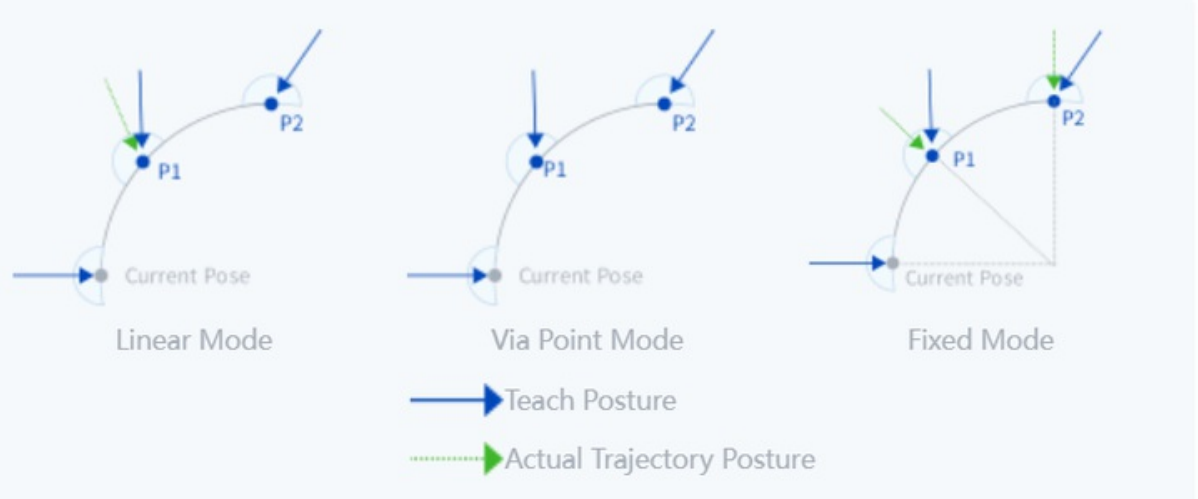
Required parameter

Parameter	Type	Description
P1	string	Intermediate point of the arc, supporting joint variables or posture variables. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".
P2	string	Target point, supporting joint variables or posture variables. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

Optional parameter

Parameter	Type	Description
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1,100].
speed	string	Format: "speed=value". **value:** The target speed for the robot when executing this command, incompatible with "v". If both "speed" and "v" exist, "speed" takes precedence. Range: [1, maximum motion speed]. Unit: mm/s.
cp	string	Format: "cp=value". **value:** Continuous path ratio, incompatible with "r". Range: [0,100].
r	string	Format: "r=value". **value:** Continuous path radius, incompatible with "cp". If both "r" and "cp" exist, "r" takes precedence. Unit: mm.
		The format is "mode=value", which sets the posture control parameter to enable adaptive control of the robot's posture relative to the arc during

mode	int	interpolation. This allows the robot to meet the needs of different scenarios. The valid range for value is [0, 2]. mode=0: Linear mode. Interpolates from the current posture to the target posture P2, ignoring the posture at P1. In this mode, posture changes are limited to less than 180°. Suitable for applications with no specific requirements on robot posture. mode=1: Via-point mode. Interpolates from the current posture, passing through the intermediate posture at P1, to the target posture P2. Primarily used in welding applications. mode=2: Fixed mode. Starting from the current posture, the TCP maintains a constant orientation relative to the arc tangent, ignoring postures at P1 and P2. In this mode, the posture rotation angle matches the arc angle, enabling posture changes exceeding 180°. Mainly used in applications such as gluing and grinding.



NOTE

- When set to **mode=1** (Via-point mode), to ensure uniform motion speed along the arc trajectory, it is recommended to position the intermediate point as close as possible to the midpoint of the actual arc during teaching.
- When set to **mode=1** (Via-point mode), the posture at each point should be properly adjusted to ensure that the posture change from the start point to the intermediate point is approximately equal to the posture change from the intermediate point to the target point. Otherwise, the constructed posture curve may exceed the robot's reachable range, resulting in a runtime error.

Return

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

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.

ArcIO

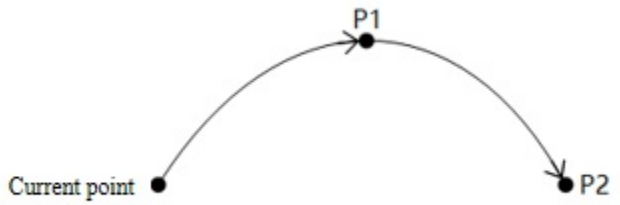
Command

```
ArcIO(P1,P2,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},user,tool,a,v|speed,cp|r,mode)
```

Description

Parallel output of specified DO signals during the arc interpolation process. Suitable for gluing applications, controlling the early dispensing and early retraction of the glue head (e.g., speaker gluing, where the trajectory is primarily arcs).

An arc is defined by three points: the current point, P1, and P2. Therefore, the current position must not be collinear with P1 and P2.



Required parameter

Parameter	Type	Description
P1	string	The intermediate point of the arc supports either joint variables or pose variables. The format is "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".
P2	string	The motion target point supports either joint variables or pose variables. The format is "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

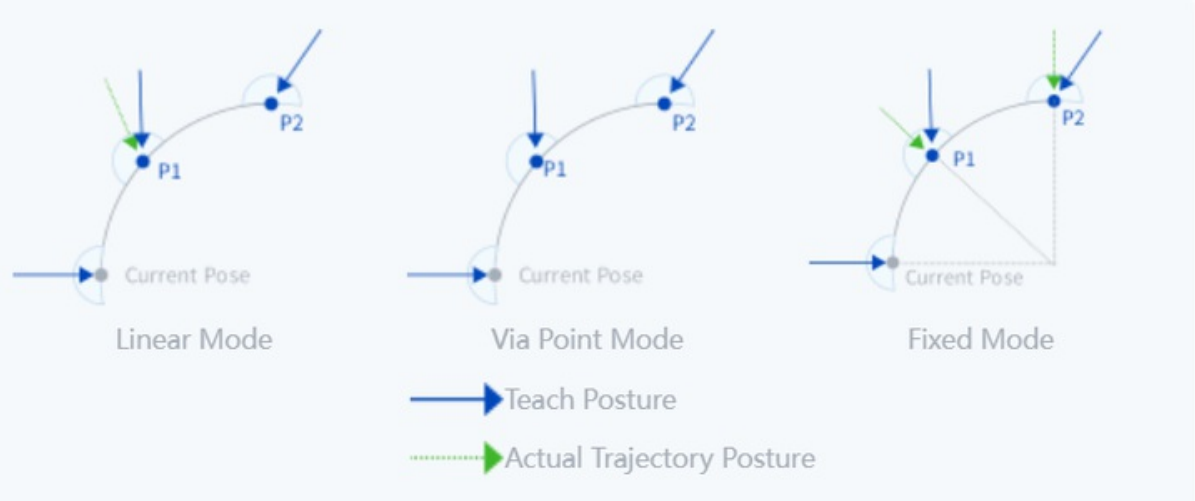
{Mode, Distance, Index, Status} are parallel digital output parameters used to set triggers for specific digital outputs (DO) when the robot moves to a specified distance or percentage. Multiple groups can be configured, with at least one group required. The specific meaning of each parameter is as follows:

Parameter	Type	Description
Mode	int	Set the trigger mode 0: Represents percentage trigger 1: Represents distance trigger
		Run the specified distance.

Distance	int	When Mode is 0, Distance represents the percentage of the distance between the starting point and the target point; value range: (0,100]. When Mode is 1, Distance represents the distance from the starting point or the target point; unit: mm. When Distance is 0, it indicates triggering at the starting point. When Distance is positive, it represents the percentage/distance from the starting point. When Distance is negative, it represents the percentage/distance from the target point.
Index	int	The numbering of DO terminals. Value range: [1, 24] or [100, 1000]. When the value range is [100, 1000], it requires hardware support from an extended IO module. The value range may vary depending on the device model.
Status	int	The DO status to set. 0 indicates no signal (DO off), and 1 indicates a signal is present (DO on).

Optional parameter

Parameter	Type	Description
user	string	The format is "user=index", where the index is the calibrated user coordinate system index. The valid range for index is [0,50].
tool	string	The format is "tool=index", where the index is the calibrated tool coordinate system index. The valid range for index is [0,50].
a	string	The format is "a=value", where value represents the motion acceleration ratio of the robot arm when executing this command. The valid range for value is [1,100].
v	string	The format is "v=value", where value represents the motion speed ratio of the robot arm when executing this command. The valid range for value is [1,100].
speed	string	The format is "speed=value". The value represents the motion target speed of the robot arm when executing this command, and is mutually exclusive with the parameter v. If both are specified, "speed" takes precedence. Value range: [1, maximum motion speed], unit: mm/s.
cp	string	The format is “cp=value”, where value represents the smooth transition ratio and is mutually exclusive with the parameter r. The valid range for value is [0,100].
r	string	The format is "r=value", where value represents the smooth transition radius and is mutually exclusive with the parameter cp. If both parameters are specified, r takes precedence. The unit is millimeters (mm). The smooth transition alters the robot's motion trajectory and may affect the timing of DO outputs, so please use it with caution.
mode	int	The format is "mode=value", which sets the posture control parameter to enable adaptive control of the robot's posture relative to the arc during interpolation. This allows the robot to meet the needs of different scenarios. The valid range for value is [0, 2]. • mode=0: Linear mode. Interpolates from the current posture to the target posture P2, ignoring the posture at P1. In this mode, posture changes are limited to less than 180°. Suitable for applications with no specific requirements on robot posture. • mode=1: Via-point mode. Interpolates from the current posture, passing through the intermediate posture at P1, to the target posture P2. Primarily used in welding applications. • mode=2: Fixed mode. Starting from the current posture, the TCP maintains a constant orientation relative to the arc tangent, ignoring postures at P1 and P2. In this mode, the posture rotation angle matches the arc angle, enabling posture changes exceeding 180°. Mainly used in applications such as gluing and grinding.



NOTE

- When set to **mode=1** (Via-point mode), to ensure uniform motion speed along the arc trajectory, it is recommended to position the intermediate point as close as possible to the midpoint of the actual arc during teaching.
- When set to **mode=1** (Via-point mode), the posture at each point should be properly adjusted to ensure that the posture change from the start point to the intermediate point is approximately equal to the posture change from the intermediate point to the target point. Otherwise, the constructed posture curve may exceed the robot's reachable range, resulting in a runtime error.

NOTICE

If Mode is 0 and Distance is not within the range [0, 100], a parameter out-of-limits error will be reported.

Example

```
ArcIO(pose={-1140.580322,-31.398853,93.642189,10.629999,21.659998,-86.040001},pose={-1220.207031,-281.265533,93.642189,10.629999,21.659998,-86.040001},{0,25,1,1},{0,50,2,1},{0,75,3,1},{0,100,4,1},user=1,tool=2,a=20,v=50,cp=100)
```

The robot arm performs an arc motion from the intermediate point P1 to the target point P2. When the motion reaches 25% of the distance from the starting point, set DO1 to ON. When it reaches 50% of the distance, set DO2 to ON. At 75% of the distance from the starting point, set DO3 to ON. Finally, when it reaches the end point, set DO4 to ON.

Circle

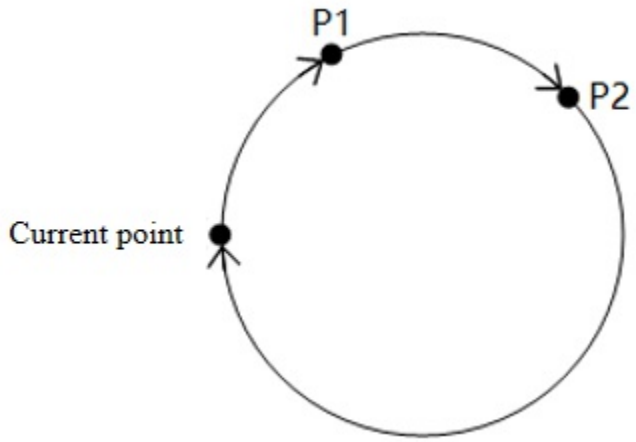
Command

```
Circle(P1,P2,count,user,tool,a,v|speed,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.



During the motion, the end posture of the robot arm is calculated through interpolation between the current point and the posture of point P2. The posture of point P1 does not participate in the calculation (meaning the posture of the robot arm when it reaches point P1 during the motion may differ from the teaching posture).

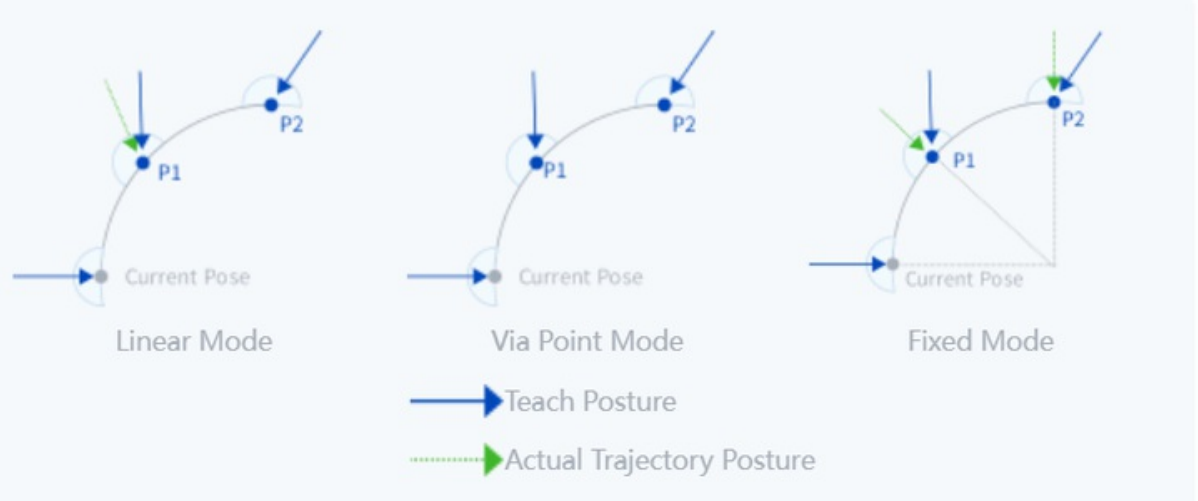
Required parameter

Parameter	Type	Description
P1	string	Intermediate point of the circle, supporting joint variables or posture variables. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".
P2	string	End point of the circle, supporting joint variables or posture variables. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".
count	int	Circles of motion. Range: [1,999].

Optional parameter

Parameter	Type	Description
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].

tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command, incompatible with "speed". Range: [1,100].
speed	string	Format: "speed=value". **value:** The target speed for the robot when executing this command, incompatible with "v". If both "speed" and "v" exist, "speed" takes precedence. Range: [1, maximum motion speed]. Unit: mm/s.
cp	string	Format: "cp=value". **value:** Continuous path ratio, incompatible with "r". Range: [0,100].
r	string	Format: "r=value". **value:** Continuous path radius, incompatible with "cp". If both "r" and "cp" exist, "r" takes precedence. Unit: mm.
mode	int	The format is "mode=value", which sets the posture control parameter to enable adaptive control of the robot's posture relative to the arc during interpolation. This allows the robot to meet the needs of different scenarios. The valid range for value is [0, 2]. • mode=0: Linear mode. Interpolates from the current posture to the target posture P2, ignoring the posture at P1. In this mode, posture changes are limited to less than 180°. Suitable for applications with no specific requirements on robot posture. • mode=1: Via-point mode. Interpolates from the current posture, passing through the intermediate posture at P1, to the target posture P2. Primarily used in welding applications. • mode=2: Fixed mode. Starting from the current posture, the TCP maintains a constant orientation relative to the arc tangent, ignoring postures at P1 and P2. In this mode, the posture rotation angle matches the arc angle, enabling posture changes exceeding 180°. Mainly used in applications such as gluing and grinding.



NOTE

- When set to **mode=1** (Via-point mode), to ensure uniform motion speed along the arc trajectory, it is recommended to position the intermediate point as close as possible to the midpoint of the

actual arc during teaching.

- When set to **mode=1** (Via-point mode), the posture at each point should be properly adjusted to ensure that the posture change from the start point to the intermediate point is approximately equal to the posture change from the intermediate point to the target point. Otherwise, the constructed posture curve may exceed the robot's reachable range, resulting in a runtime error.

Return

```
ErrorID,{ResultID},Circle(P1,P2,count,user,tool,a,v|speed,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(J1,J2,J3,J4,J5,J6,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
J1,J2,J3,J4,J5,J6	double	J1,J2,J3,J4,J5,J6 axis position, unit: °.

Optional parameter

Parameter	Type	Description
t	float	Format: "t=value". **value:** The runtime for the point. Unit: s. Range: [0.004, 3600.0], default: 0.1.
aheadtime	float	Format: "aheadtime=value". **value:** Advanced time, similar to the D term in PID control. Scalar, no unit. Range: [20.0, 100.0], default: 50.
gain	float	Format: "gain=value". **value:** Proportional gain for the target position, similar to the P term in PID control. Scalar, no unit. Range: [200.0, 1000.0], default: 500.

The parameters "aheadtime" and "gain" together determine the response time and trajectory smoothness of the robot motion. Smaller aheadtime or larger gain values enable quicker response, but may cause instability or jitter.

Return

```
ErrorID,{ResultID},ServoJ(J1,J2,J3,J4,J5,J6,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(X,Y,Z,Rx,Ry,Rz,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.

Required parameter

Parameter	Type	Description
X,Y,Z,Rx,Ry,Rz	double	Target point posture variables. X,Y,Z in millimeters (mm), Rx,Ry,Rz in degrees (°). 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	Format: "t=value". **value:** The runtime for the point. Unit: s. Range: [0.004, 3600.0], default: 0.1.
aheadtime	float	Format: "aheadtime=value". **value:** Advanced time, similar to the D term in PID control. Scalar, no unit. Range: [20.0, 100.0], default: 50.
gain	float	Format: "gain=value". **value:** Proportional gain for the target position, similar to the P term in PID control. Scalar, no unit. Range: [200.0, 1000.0], default: 500.

The parameters "aheadtime" and "gain" together determine the response time and trajectory smoothness of the robot motion. Smaller aheadtime or larger gain values enable quicker response, but may cause instability or jitter.

Return

```
ErrorID,{ResultID},ServoP(X,Y,Z,Rx,Ry,Rz,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. After the command is delivered, the robot will continuously jog along the specified axis, and it will stop once MoveJog() is delivered. In addition, when the robot is jogging, the delivery of MoveJog(string) with any non-specified string will also stop the motion of the robot.

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

Required parameter

Parameter	Type	Description
axisID	string	Jog the motion axis (**case sensitive**). If no parameter is specified or if it is specified with incorrect parameters, the robot will stop jogging. 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	string	Format: "coordtype=value". **value:** Specify the coordinate system for the motion axis. 0: Jog along joint, 1: User coordinate system, 2: Tool coordinate system. Default: The value from the last successful call. If axisID corresponds to a joint axis, coordtype must be 0. Any provided coordtype value will be ignored. If axisID corresponds to a Cartesian axis, coordtype must be either 1 or 2. Using 0 will return error code -6.
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].

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 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. Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

Optional parameter

Parameter	Type	Description

moveType	string	Set the motion type using the parameter format "moveType=value". The value range is [0, 4], with a default value of 1 (linear motion).
		moveType=0: Joint motion;
		moveType=1: Linear motion;
		moveType=2: Joint motion to a specified offset angle;
		moveType=3: Relative linear motion along the tool coordinate system (must use pose variables, not joint variables);
		moveType=4: Relative linear motion along the user coordinate system (must use pose variables, not joint variables).

user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1,100].

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 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 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.

MovS

Command

```
MovS(P1,P2,P3,... ,user,tool,a,v|speed,freq)
```

```
MovS(file,user,tool,a,v|speed,freq)
```

Description

Fit the specified trajectory. Before calling this command, ensure the robot has been manually moved to the starting point of the trajectory.

Optional parameter

Parameter	Type	Description
P1,P2,P3...	string	The points to be fitted can be either joint point or pose point. The format can be "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}". The number of points should range from [4, 50].
file	string	The trajectory file to be fitted, in the format "file=x.csv", represents the name of a trajectory file (including the extension).
user	string	The user coordinate system index corresponding to the specified trajectory points. If not specified, the user coordinate system index recorded in the trajectory file will be used. This is an optional parameter with the highest priority. The format is "user=index", where index is the calibrated user coordinate system index. Value range: [0, 50].
tool	string	The tool coordinate system index corresponding to the specified trajectory points. If not specified, the tool coordinate system index recorded in the trajectory file will be used. This is an optional parameter with the highest priority. The format is "tool=index", where index is the calibrated tool coordinate system index. Value range: [0, 50].

a	string	The format is "a=value". The value represents the robot's motion acceleration ratio when executing this command. Value range: [1, 100].
v	string	The format is "v=value". The value represents the robot's motion speed ratio when executing this command, and it is mutually exclusive with "speed". Value range: [1, 100].
speed	string	The format is "speed=value". The value represents the target speed of the robot's motion when executing this command. It is mutually exclusive with "v", and if both are specified, "speed" takes precedence. Value range: [1, maximum motion speed], unit: mm/s.
freq	string	The filtering coefficient, formatted as "freq=value". A smaller value results in a smoother fitted trajectory curve but causes more significant deformation relative to the original trajectory. Please set an appropriate filtering coefficient based on the smoothness of the original trajectory. Value range: (0, 1], default is 1 (indicating filtering is disabled). For trajectories output by CAD, set to 1 to ensure precision; for curves from sources like 3D cameras, it is recommended to enable filtering to ensure smoothness of the curve.

NOTICE

This command must include either a point list (P1, P2, P3, ...) or a trajectory file parameter (file).

Return

```
ErrorID,{},MovS(P1,P2,P3,... ,user,tool,a,v|speed,freq);
```

```
ErrorID,{},MovS(file,user,tool,a,v|speed,freq);
```

Example

```
MovS(pose={100,0,100,0,0,0},pose={100,20,100,0,0,0},pose={100,30,100,0,0,0}, pose={100,40,100,0,0,0})
```

GetStartPose

Command

```
GetStartPose(traceName)
```

Description

Get the start point of specified trajectory.

Required parameter

Parameter	Type	Description
traceName	string	Trajectory file name (including the .csv extension). The trajectory file is stored in either `/dobot/userdata/project/process/trajectory/` or `/dobot/userdata/project/process/track/` . 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
pathType	int	The type of trajectory, which can be left blank or set to 1 or 2. 1: Default value, used for playback trajectories. The trajectory is stored in /dobot/userdata/project/process/trajectory/. 2: Used for fitting trajectories. The trajectory file is stored in /dobot/userdata/project/process/track/.

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
ErrorID,{2,{x,y,z,rx,ry,rz}},GetStartPose(traceNamel,2);// posture variable
```

Example

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

Get the first point recorded in recv_string.csv.

StartPath

Command

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

Description

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

After the trajectory playback command is successfully delivered, you can check the robot status via 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	string	Format: "isConst=value". **value:** Whether to play back at a constant speed. Default: 0. isConst=1 means constant speed playback, where the robot will play back the trajectory at a constant speed. isConst=0 means playback at the original recorded speed, with the option to scale the speed proportionally using the "multi" parameter. In this case, the robot's speed is not affected by the global speed setting.
multi	string	Format: "multi=value". **value:** Speed multiplier for the playback, valid only when isConst=0. Range: [0.25, 2]. Default: 1.
sample	string	Format: "sample=value". **value:** The sampling interval of the trajectory points, i.e. the sampling time differences between two adjacent points when generating the trajectory file. Range: [8, 1000], unit: ms. Default: 50ms (the sampling interval when the controller records the trajectory file).
freq	string	Format: "freq=value". **value:** Filter coefficient. The smaller the value, the smoother the trajectory curve during playback, but the greater the distortion relative to the original trajectory. Set an appropriate filter coefficient based on the smoothness of the original trajectory. Range: (0, 1], 1 means filtering is OFF. Default: 0.2.
user	string	Format: "user=index". **index:** The user coordinate system index for the trajectory points. If it is not specified, the user coordinate system index recorded in the trajectory file is used. Range: [0, 50].
tool	string	Format: "tool=index". **index:** The tool coordinate system index for the trajectory points. If it is not specified, the tool coordinate system index recorded in the trajectory file is used. Range: [0, 50].

Return

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

Example

```
StartPath(recv_string.csv,isConst=0,multi=1,sample=20,freq=1,user=0,tool=0)
```

Play back the trajectory recorded in the “recv_string.csv” file at the original speed. Trajectory sampling interval: 20ms. Filter coefficient: 1 (perfectly restores the recorded trajectory). User and Tool coordinate systems: 0.

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: °.
offseRrz	double	Rz-axis offset, unit: °.

Optional parameter

Parameter	Type	Description
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0,50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1,100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1,100].
cp	string	Format: "cp=value". **value:** Continuous path ratio. 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 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|speed,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	string	Format: "user=index" (index: index of the calibrated user coordinate system). Range: [0, 50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0, 50].

a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1, 100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command, incompatible with "speed". Range: [1,100].
speed	string	Format: "speed=value". **value:** The target speed for the robot when executing this command, incompatible with "v". If both "speed" and "v" exist, "speed" takes precedence. Range: [1, maximum motion speed]. Unit: mm/s.
cp	string	Format: "cp=value". **value:** Continuous path ratio, incompatible with "r". Range: [0,100].
r	string	Format: "r=value". **value:** 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|speed,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 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	string	Format: "user=index" (index: index of the calibrated user coordinate system). Range: [0, 50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0, 50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1, 100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1, 100].
cp	string	Format: "cp=value". **value:** Continuous path ratio. 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 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|speed,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	string	Format: "user=index" (index: index of the calibrated user coordinate system). Range: [0, 50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0, 50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1, 100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command, incompatible with "speed". Range: [1,100].
speed	string	Format: "speed=value". **value:** The target speed for the robot when executing this command, incompatible with "v". If both "speed" and "v" exist, "speed" takes precedence. Range: [1, maximum motion speed]. Unit: mm/s.
cp	string	Format: "cp=value". **value:** Continuous path ratio, incompatible with "r". Range: [0,100].
r	string	Format: "r=value". **value:** 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|speed,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 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,user,tool,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
user	string	Format: "user=index" (index: index of the calibrated user coordinate system). Range: [0, 50].
tool	string	Format: "tool=index" (index: index of the calibrated tool coordinate system). Range: [0, 50].
a	string	Format: "a=value". **value:** The acceleration ratio for the robot when executing this command. Range: [1, 100].
v	string	Format: "v=value". **value:** The velocity ratio for the robot when executing this command. Range: [1, 100].
cp	string	Format: "cp=value". **value:** Continuous path ratio. Range: [0,100].

Return

```
ErrorID,{ResultID},RelJointMovJ(offset1,offset2,offset3,offset4,offset5,offset6,user,tool,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.

RelPointTool

Command

```
RelPointTool(p, {offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz})
```

Description

Perform Cartesian point offset along the tool coordinate system.

Required parameter

Parameter	Type	Description
p	string	Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}". It refers to the starting point for the offset.
{offsetX,offsetY,offsetZ, offsetRx,offsetRy,offsetRz}	double	It refers to the offset values along the Cartesian coordinate axes: X-axis, Y-axis, Z-axis, Rx-axis, Ry-axis, and Ry-axis.

Return

```
ErrorID,{X,Y,Z,Rx,Ry,Rz},RelPointTool(p, {offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz})
;
```

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

RelPointUser

Command

```
RelPointUser(p, {offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz})
```

Description

Perform Cartesian point offset along the user coordinate system.

Required parameter

Parameter	Type	Description
p	string	Format: "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}". It refers to the starting point for the offset.
{offsetX,offsetY,offsetZ, offsetRx,offsetRy,offsetRz}	double	It refers to the offset values along the Cartesian coordinate axes: X-axis, Y-axis, Z-axis, Rx-axis, Ry-axis, and Ry-axis.

Return

```
ErrorID,{X,Y,Z,Rx,Ry,Rz},RelPointUser(p, {offsetX,offsetY,offsetZ,offsetRx,offsetRy,offsetRz})
;
```

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

RelJoint

Command

```
RelJoint(J1,J2,J3,J4,J5,J6,{offset1,offset2,offset3,offset4,offset5,offset6})
```

Description

Perform relative joint position offset.

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: °.
{offset1,offset2,offset3, offset4,offset5,offset6}	double	Offset values for joints 1/2/3/4/5/6, unit: °.

Return

```
ErrorID,{J1,J2,J3,J4,J5,J6},RelJoint(J1,J2,J3,J4,J5,J6,{offset1,offset2,offset3,offset4,offset5,offset6});
```

{J1,J2,J3,J4,J5,J6} refers to the joint values.

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(), SetSafeWal
lEnable(), SetBackDistance(), SetPostCollisionMode(), SetUser(), SetTool(), MovJ(), MovL(), Mo
vLIO(), MovJIO(), Arc(), Circle(), StartPath(), RelMovJTool(), RelMovLTool(), RelMovJUser(), R
elMovLUser(), 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 curren
t 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 exe
cuted
    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.

StartRTOffset

Command

```
StartRTOffset()
```

Description

End coordinate system offset.

Return

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

EndRTOffset

Command

```
EndRTOffset()
```

Description

Start coordinate system offset.

Return

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

OffsetPara

Command

```
OffsetPara(x, y, z, rx, ry, rz)
```

Description

Set coordinate system offset value.

Required parameter

Parameter	Type	Description
x	double	X-axis offset in mm
y	double	Y-axis offset in mm
z	double	Z-axis offset in mm
rx	double	Rx-axis offset in degrees
ry	double	Ry-axis offset in degrees
rz	double	Rz-axis offset in degrees

Return

```
ErrorID,{},OffsetPara(x, y, z, rx, ry, rz);
```

Example

```
OffsetPara(10, 10, 10, 0, 0, 0)
```

The robot is offset by 10 mm each in the X, Y, and Z axes relative to the original coordinate system.

2.8 Trajectory recovery command

Overview

When the **Jog while paused** function is enabled, users can issue MoveJog, RunTo, and enter/exit drag mode commands to change the robot's posture while the project is paused. Before continuing the project, users can use the trajectory recovery command to restore the robot to the point where it was paused, so as to avoid abnormal motion of the robot.

Command list

Command	Function	Command type
SetResumeOffset	Set backoff distance for trajectory recovery	Immediate command
PathRecovery	Start trajectory recovery	Immediate command
PathRecoveryStop	Stop the robot during trajectory recovery	Immediate command
PathRecoveryStatus	Query the trajectory recovery status	Immediate command

SetResumeOffset

Command

```
SetResumeOffset(distance)
```

Description

This command is intended for use in welding solutions only. Set backoff distance for trajectory recovery along the weld seam from the point where the project was paused.

> **NOTE**
> - This command only takes effect during welding (i.e., when WeldArcSpeed is in effect).
> - The command can only properly plan the backoff point after the backoff distance is set first.

Required parameter

Parameter	Type	Description
distance	double	Set backoff distance for trajectory recovery in the direction of movement after pause (unit: mm).

Return

```
ErrorID,{},SetResumeOffset(distance);
```

PathRecovery

Command

```
PathRecovery()
```

Description

Start trajectory recovery: After the project is paused, control the robot to return to the posture when it was paused.

> **NOTE**
> - This command only controls the robot to return to the posture when it was paused. To resume the project, deliver the Continue command.
> - This command is an asynchronous interface. It returns immediately after being delivered. To determine whether the robot has returned to the paused posture, use the PathRecoveryStatus command.

Return

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

PathRecoveryStop

Command

```
PathRecoveryStop()
```

Description

Stop the robot during trajectory recovery.

Return

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

PathRecoveryStatus

Command

```
PathRecoveryStatus()
```

Description

Query the trajectory recovery status.

Return

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

Where {status} indicates the trajectory recovery status:

- 0: Returned to the paused posture.
- 1: Not returned to the paused posture, with a small deviation from the paused posture.
- 2: Not returned to the paused posture, with a large deviation from the paused posture.

Command Example

```
SetResumeOffset(10); // Set the welding backoff distance to 10mm

MovL(P1); // Move to P1 in linear mode at global speed
WeldArcSpeed(10); // Set the welding speed to 10mm/s
WeldArcSpeedStart(); // Enable the welding speed switch
MovL(P2); // Move to P2 in linear mode at the configured welding speed
WeldArcSpeedEnd(); // Disable the welding speed switch

// After getting the project pause status through real-time feedback information or RobotMode
polling
RunTo(P); // Move the paused robot to a safe point for manual processing
PathRecovery(); // The robot returns to the pause point after offset (10mm back along the weld
 seam)
PathRecoveryStop(); // Stop the robot if an anomaly is detected during trajectory recovery.
RunTo(P); // Move the robot to a safe point for manual processing again
PathRecovery(); // The robot returns to the pause point after offset
if(PathRecoveryStatus()=0)
{
// The robot has returned to the pause point after offset
Continue(); // Continue running the project
}
```

2.9 Log export command

Overview

This set of commands is used to export robot logs and view the export status.

Command list

Command	Function	Command type
LogExportUSB	Export robot logs to USB	Immediate command
GetExportStatus	Get log export status	Immediate command

LogExportUSB

Command

```
LogExportUSB(range)
```

Description

Export the robot logs to the root directory of a USB drive plugged into the USB interface of the robot controller.

> **NOTE**
> - It is recommended to insert only one USB drive when exporting logs to avoid failure.
> - If the USB drive contains multiple partitions, the logs will be exported to the first partition. For some storage devices (e.g., USB drives used as boot disks), the first partition may be hidden, making the exported logs inaccessible in Windows.
> - Do not remove the USB drive during the export process, otherwise, the file may be corrupted, requiring you to format the USB drive before exporting again.

Required parameter

Parameter	Type	Description
range	int	Export range. 0: Export the contents of the "logs/all" and "logs/user" folders. 1: Export all contents of the "logs" folder.

Return

```
ErrorID,{},LogExportUSB(range);
```

This command will return immediately. Please use GetExportStatus to get the log export status. If this command is delivered during the export process, it will return -1, indicating the execution failed.

Example

```
LogExportUSB(0)
```

Export the contents of the logs/all and logs/user folders to the USB drive.

GetExportStatus

Command

```
GetExportStatus()
```

Description

Get log export status.

Return

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

{status} indicates the log export status.

- 0: Export not started
- 1: Exporting
- 2: Export completed
- 3: Export failed, USB drive not found
- 4: Export failed, insufficient USB drive space
- 5: Export failed, USB drive removed during the export process

The status of Export Completed and Export Failed will remain until the next time the export function is used.

2.10 Force control command

Overview

Dobot supports matching a six-axis force sensor. The force-control drag function can be enabled and configured through the ForceControl plug-in. Force-control drag is an advanced function relies on the force sensor and sophisticated control algorithm. It allows the user to manually guide the robot's end-effector (six-axis force sensor) by applying force, while the robot senses and responds to the applied force, following the user's manual guidance. In practical applications, users can also constrain the robot's motion direction, making it to move only along one or several specified directions.

Command list

Command	Function	Command type
EnableFTSensor	Switch on/off the force sensor	Immediate command
SixForceHome	Zero the force sensor	Immediate command
GetForce	Get force sensor data	Immediate command
ForceDriveMode	Enter the force-control drag mode	Immediate command
ForceDriveSpeed	Set the force-control drag speed	Immediate command
FCForceMode	Enable force control with user-specified parameters	Queue command
FCSetDeviation	Set the offset and posture deviation in force control mode	Immediate command
FCSetForceLimit	Set the maximum force limit	Immediate command
FCSetMass	Set the mass coefficients for each direction in force control mode	Immediate command
FCSetStiffness	Set the stiffness coefficients for each direction in force control mode	Immediate command
FCSetDamping	Set the damping coefficients for each direction in force control mode	Immediate command
FCOff	Exit the force control mode	Queue command
FCSetForceSpeedLimit	Set the speed limits for force control adjustment in each direction	Immediate command
FCSetForce	Adjust the constant force settings in real time	Immediate command
SetFCCollision	Set the threshold parameters for the force sensor collision detection (CRAF only)	Immediate command

FCCollisionSwitch	Collision detection switch for the force sensor (CRAF only)	Immediate command

EnableFTSensor

Command

```
EnableFTSensor(status)
```

Description

Switch on/off the force control sensor.

Required parameter

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

Return

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

Example

```
EnableFTSensor(1)
```

Switch on the force sensor.

SixForceHome

Command

```
SixForceHome()
```

Description

Set the current data of the force sensor to 0, using the current force status as the zero point.

Return

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

Example

```
SixForceHome()
```

Set the current data of the force sensor to 0.

GetForce

Command

```
GetForce(tool)
```

Description

Get the current data of the force sensor.

Optional parameter

Parameter	Type	Description
tool	int	Specify the tool coordinate system for reference when getting values. Range: [0,50]. If not specified, it defaults to the global tool coordinate system.

Return

```
ErrorID,{Fx,Fy,Fz,Mx,My,Mz},GetForce(tool);
```

Fx, Fy, Fz: force values in the X, Y, and Z directions. Mx, My, Mz: torque values in the RX, RY, and RZ directions.

Example

```
GetForce(1)
```

Get the force value of the force sensor in tool coordinate system 1.

ForceDriveMode

Command

```
ForceDriveMode({x,y,z,rx,ry,rz},user)
```

Description

Specify the directions for dragging and enter force-control drag mode.

Required parameter

Parameter	Type	Description
{x,y,z,rx,ry,rz}	string	Specify the directions for dragging. 0: The robot arm cannot be dragged in this direction. 1: The robot arm can be dragged in this direction. For example, • {1,1,1,1,1,1} means the robot arm can be freely dragged in all directions. • {1,1,1,0,0,0} means the robot arm can only be dragged in XYZ directions. • {0,0,0,1,1,1} means the robot arm can only rotate around the RxRyRz axes.

Optional parameter

Parameter	Type	Description
user	int	Specify the user coordinate system for reference during dragging. Range: [0,50]. If not specified, it defaults to the global tool coordinate system.

Return

```
ErrorID,{},ForceDriveMode({x,y,z,rx,ry,rz},user);
```

Example 1

```
ForceDriveMode({1,1,1,1,1,1},1)
```

Enter the force-control drag mode, and you can drag freely in all directions in user coordinate system 1.

Example 2

```
ForceDriveMode({1,1,1,0,0,0})
```

Enter the force-control drag mode, and you can drag in the XYZ directions in the global tool coordinate system.

ForceDriveSpeed

Command

```
ForceDriveSpeed(speed)
```

Description

Set the force-control drag speed ratio.

Required parameter

Parameter	Type	Description
speed	int	Force-control drag speed ratio. Range: [1,100].

Return

```
ErrorID,{},ForceDriveSpeed(speed);
```

Example

```
ForceDriveSpeed(10)
```

Set the force-control drag speed ratio to 10.

FCForceMode

Command

```
FCForceMode({x,y,z,rx,ry,rz},{fx,fy,fz,frx,fry,frz},reference,user,tool)
```

Description

Enable force control with user-specified parameters.

Required parameter

Parameter	Type	Description
{x,y,z,rx,ry,rz}	string	Enable/disable force control adjustment in specific Cartesian directions. 0: Disable force control for this direction. 1: Enable force control for this direction.
{fx,fy,fz,frx,fry,frz}	string	Target force: The target contact force between the tool's end and the object being acted upon. It is a simulated force that users can configure. The directions correspond to {x,y,z,rx,ry,rz} in Cartesian space. Range for displacement directions: [-200, 200], unit: N. Range for posture directions: [-12, 12], unit: N·m. When the target force is set to 0, it enters compliance mode, which is similar to force-control drag. If force control is not enabled for a specific direction, the target force in that direction will not take effect.

Optional parameter

Parameter	Type	Description
reference	string	Format: "reference=value". value: The reference coordinate system, with the tool coordinate system being the default. reference=0: Refers to the tool coordinate system, meaning force control adjustment will follow the tool coordinate system. reference=1: Refers to the user coordinate system, meaning force control adjustment will follow the user coordinate system.
user	string	Format: user=index (index: index of the calibrated user coordinate system). Range: [0,50].
tool	string	Format: tool=index (index: index of the calibrated tool coordinate system). Range: [0,50].

Return

```
ErrorID,{ResultID},FCForceMode({x,y,z,rx,ry,rz},{fx,fy,fz,frx,fry,frz},reference,user,tool);
```

Example

```
FCForceMode({1,1,1,1,1,1},{100,100,100,10,10,10},reference=1,user=1)
```

This example performs force control adjustment in all directions based on the calibrated user coordinate system (reference 1). The target force in the displacement directions is set to 100 N, and the target force in the posture directions is set to 10 N·m.

FCSetDeviation

Command

```
FCSetDeviation({x,y,z,rx,ry,rz}, controltype)
```

Description

Set displacement and posture deviation in force control mode, which will trigger a response if a large deviation occurs.

Required parameter

Parameter	Type	Description
{x,y,z,rx,ry,rz}	string	x, y, z: Displacement deviation in force control mode. Unit: mm. Range: (0,1000], default is 100mm. rx, ry, rz: Posture deviation in force control mode. Unit: °. Range: (0,360], default is 36°.

Optional parameter

Parameter	Type	Description
controltype	int	The handling method when the threshold is exceeded during force control. 0: When the threshold is exceeded, the robot will trigger an alarm (default). 1: When the threshold is exceeded, the robot will stop searching and continue running along the original trajectory.

Return

```
ErrorID,{},FCSetDeviation({x,y,z,rx,ry,rz}, controltype);
```

Example

```
FCSetDeviation({200,200,200,36,36,36})
```

This example sets the displacement deviation of the force control mode to 200 mm in the x, y, and z directions, and the posture deviation to 36° in the rx, ry, and rz directions. After exiting TCP mode, the parameters revert to their default values.

FCSetForceLimit

Command

```
FCSetForceLimit(x,y,z,rx,ry,rz)
```

Description

Set the maximum force limit for each direction (affects all directions, including those not enabled for force control).

Required parameter

Parameter	Type	Description
x	double	Force limit in x-direction, range: (0, 500], default is 500.
y	double	Force limit in y-direction, range: (0, 500], default is 500.
z	double	Force limit in z-direction, range: (0, 500], default is 500.
rx	double	Force limit in rx-direction, range: (0, 50], default is 50.
ry	double	Force limit in ry-direction, range: (0, 50], default is 50.
rz	double	Force limit in rz-direction, range: (0, 50], default is 50.

Return

```
ErrorID,{},FCSetForceLimit(x,y,z,rx,ry,rz);
```

Example

```
FCSetForceLimit(500,500,500,50,50,50)
```

When FCSetForceLimit is not called, the default maximum force limit is 500 N for the x, y, and z directions, and 50 N·m for the rx, ry, and rz directions. After exiting TCP mode, the parameters revert to their default values.

FCSetMass

Command

```
FCSetMass(x,y,z,rx,ry,rz)
```

Description

Set the inertia coefficients for each direction in force control mode.

Required parameter

Parameter	Type	Description
x	double	Inertia coefficient in x-direction, range: (0,10000], default is 20.
y	double	Inertia coefficient in y-direction, range: (0,10000], default is 20.
z	double	Inertia coefficient in z-direction, range: (0,10000], default is 20.
rx	double	Inertia coefficient in rx-direction, range: (0,10000], default is 20.
ry	double	Inertia coefficient in ry-direction, range: (0,10000], default is 20.
rz	double	Inertia coefficient in rz-direction, range: (0,10000], default is 20.

Return

```
ErrorID,{},FCSetMass(x,y,z,rx,ry,rz);
```

Example

```
FCSetMass(20,20,20,20,20,20)
```

When FCSetMass is not called, the default inertia coefficient for all directions is set to 20. After exiting TCP mode, the parameters revert to their default values.

FCSetStiffness

Command

```
FCSetStiffness(x,y,z,rx,ry,rz)
```

Description

Set the stiffness coefficients for each direction in force control mode.

Required parameter

Parameter	Type	Description
x	double	Stiffness coefficient in x-direction, range: [0,10000], default is 30.
y	double	Stiffness coefficient in y-direction, range: [0,10000], default is 30.
z	double	Stiffness coefficient in z-direction, range: [0,10000], default is 30.
rx	double	Stiffness coefficient in rx-direction, range: [0,10000], default is 30.
ry	double	Stiffness coefficient in ry-direction, range: [0,10000], default is 30.
rz	double	Stiffness coefficient in rz-direction, range: [0,10000], default is 30.

Return

```
ErrorID,{},FCSetStiffness(x,y,z,rx,ry,rz);
```

Example

```
FCSetStiffness(30,30,30,30,30,30)
```

When FCSetStiffness is not called, the default stiffness coefficient for all directions is set to 30. After exiting TCP mode, the parameters revert to their default values.

FCSetDamping

Command

```
FCSetDamping(x,y,z,rx,ry,rz)
```

Description

Set the damping coefficients for each direction in force control mode.

Required parameter

Parameter	Type	Description
x	double	Damping coefficient in x-direction, range: [0,1000], default is 50.
y	double	Damping coefficient in y-direction, range: [0,1000], default is 50.
z	double	Damping coefficient in z-direction, range: [0,1000], default is 50.
rx	double	Damping coefficient in rx-direction, range: [0,1000], default is 50.
ry	double	Damping coefficient in ry-direction, range: [0,1000], default is 50.
rz	double	Damping coefficient in rz-direction, range: [0,1000], default is 50.

Return

```
ErrorID,{},FCSetDamping(x,y,z,rx,ry,rz);
```

Example

```
FCSetDamping(50,50,50,50,50,50)
```

When FCSetDamping is not called, the default damping coefficient for all directions is set to 50. After exiting TCP mode, the parameters revert to their default values.

FCOff

Command

```
FCOff()
```

Description

Exit the force control mode. It is used in conjunction with FCForceMode. Any motion commands between the two will be subject to force compliance control.

Return

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

Example

```
FCOff()
```

Disable force control.

FCSetForceSpeedLimit

Command

```
FCSetForceSpeedLimit(x,y,z,rx,ry,rz)
```

Description

Set the speed limits for force control adjustment in each direction. When the force control speed limit is set lower, the adjustment speed is slower, making it suitable for gentle, low-speed contact. When the force control speed limit is set higher, the adjustment speed is faster, making it suitable for high-speed force control applications. It should be adjusted according to the specific application requirements.

Required parameter

Parameter	Type	Description
x	double	Force control speed limit in x-direction, default value: 20mm/s. For CRA models, range: (0, Safety limit TCP speed]. For other models, range: (0, 300].
y	double	Force control speed limit in y-direction, default value: 20mm/s. For CRA models, range: (0, Safety limit TCP speed]. For other models, range: (0, 300].
z	double	Force control speed limit in z-direction, default value: 20mm/s. For CRA models, range: (0, Safety limit TCP speed]. For other models, range: (0, 300].
rx	double	Force control speed limit in rx-direction, default value: 20°/s. For CRA models, range: (0, (4 × Safety limit TCP speed × 0.001 / 3.14 × 180)]. For other models, range: (0, 90].
ry	double	Force control speed limit in ry-direction, default value: 20°/s. For CRA models, range: (0, (4 × Safety limit TCP speed × 0.001 / 3.14 × 180)]. For other models, range: (0, 90].
rz	double	Force control speed limit in rz-direction, default value: 20°/s. For CRA models, range: (0, (4 × Safety limit TCP speed × 0.001 / 3.14 × 180)]. For other models, range: (0, 90].

Return

```
ErrorID,{},FCSetForceSpeedLimit(x,y,z,rx,ry,rz);
```

Example

```
FCSetForceSpeedLimit(20,20,20,20,20,20)
```

When FCSetForceSpeedLimit is not called, the default force control speed for all directions is 20mm/s. After exiting TCP mode, the parameters revert to their default values.

FCSetForce

Command

```
FCSetForce(x,y,z,rx,ry,rz)
```

Description

Adjust the constant force settings in each direction in real time.

Required parameter

Parameter	Type	Description
x	double	Constant force value in the x direction. Range: [-200, 200], unit: N.
y	double	Constant force value in the y direction. Range: [-200, 200], unit: N.
z	double	Constant force value in the z direction. Range: [-200, 200], unit: N.
rx	double	Constant force value in the rx direction. Range: [-12, 12], unit: N·m.
ry	double	Constant force value in the ry direction. Range: [-12, 12], unit: N·m.
rz	double	Constant force value in the rz direction. Range: [-12, 12], unit: N·m.

Return

```
ErrorID,{},FCSetForce(x,y,z,rx,ry,rz);
```

Example

```
FCSetForce(50,50,50,10,10,10)
```

Set the constant force in the x, y, z directions to 50 N. Set the constant force in the rx, ry, rz directions to 10 N·m.

SetFCCollision

Command

```
SetFCCollision(force, torque)
```

Description

Set the threshold parameters for the force sensor collision detection. When the collision force or collision torque at the robot's end exceeds the set thresholds, the robot will pause or stop depending on the settings. This command is only applicable to **CRAF models**. Using this command on other models will result in an

error.

Required Parameter

Parameter	Type	Description
force	double	Force threshold for triggering collision detection. The normal mode and reduced mode share the same parameters. Unit: N. The valid range differs for specific models: **CR5AF**: [5, 150] **CR10AF**: [5, 300] **CR20AF**: [5, 500]
torque	double	Torque threshold for triggering collision detection. The normal mode and reduced mode share the same parameters. Unit: N·m. The valid range differs for specific models: **CR5AF**: [0.5, 15] **CR10AF**: [0.5, 30] **CR20AF**: [0.5, 50]

Return

```
ErrorID, {}, SetFCCollision(force, torque);
```

Example

```
SetFCCollision(50, 10)
```

When the collision force at the robot's end exceeds **50 N** or the collision torque exceeds **10 N·m**, the robot will pause or stop depending on the settings.

FCCollisionSwitch

Command

```
FCCollisionSwitch(switch)
```

Description

Collision detection switch for the force sensor. This command is only applicable to **CRAF models**. Using this command on other models will result in an error.

Required Parameter

Parameter	Type	Description
switch	int	Collision detection switch for the force sensor. Valid values are: **0**: Disable the collision detection function. **1**: Enable the collision detection function.

Return

```
ErrorID, {}, FCCollisionSwitch(switch);
```

Example

```
FCCollisionSwitch(1)
```

Enable collision detection for the force sensor.

2.11 Conveyor command

Overview

The Dobot conveyor tracking solution utilizes photoelectric sensors/industrial cameras to accurately capture the initial position of workpieces on the conveyor belt and employs high-precision encoders to track the displacement changes of the workpieces in real time. With the conveyor tracking plugin, the robot control system can dynamically calculate the motion trajectory of the workpieces, enabling the robot to perform stable picking, precise assembly, or continuous dispensing operations on the workpieces on the conveyor belt.

Command list

Command	Function	Command type
CnvInit	Start conveyor belt	Immediate command
GetCnvObject	Wait for specified workpiece to enter picking area of conveyor	Immediate command
StartSyncCnv	Enable conveyor tracking function	Immediate command
CnvMovL	Execute conveyor following with linear interpolation	Queue command
CnvMovC	Execute conveyor following with circular interpolation	Queue command
StopSyncCnv	Disable conveyor tracking function	Immediate command
SetCnvPointOffset	Set X and Y axis offset in conveyor user coordinate system	Immediate command
SetCnvTimeCompensation	Set compensation time	Immediate command

CnvInit

Command

```
CnvInit(index)
```

Description

Start the conveyor belt and issues conveyor configuration parameters. Delete all queue data and initiates the detection and storage of new queue information.

Required parameter

Parameter	Type	Description
index	int	Conveyor belt numbers 1/2/3, with the robot supporting up to three conveyor belts. Setting values outside this range will result in an error.

Return

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

Example

```
CnvInit(1)
```

Start conveyor belt No. 1 and issue conveyor configuration parameters.

GetCnvObject

Command

```
GetCnvObject(objId)
```

Description

Wait for the specified workpiece to enter the picking area of the conveyor (i.e., the area defined by the upper and lower picking boundaries).

Required parameter

Parameter	Type	Description
objId	int	Workpiece type, value range [0, 15]. 0: No specific workpiece type; retrieve information on the first workpiece in the queue. For sensor triggers, objId defaults to 0. 1–15: Retrieve information on the first specified workpiece in the queue.

Return

```
ErrorID,{flag, objId, objframe},GetCnvObject(objId);
```

- flag: Numeric value descriptions are as follows
  - 0: No workpiece
  - 1: Workpiece present

    - -1: Execution error, retry required
    - -2: Unprocessed error present
    - -3: Not in tracking initialization state; execute `CnvInit` or `StopSyncCnv` command required
- objId: Workpiece type ID. This return value is valid only when the required parameter **objId** is 0.
- objframe: Return the workpiece coordinate system (referenced to the robot base coordinate system) at the current moment. The workpiece coordinate frame is returned even if the workpiece is not within the picking boundary range (primarily used for real-time monitoring).

Example

```
GetCnvObject(0)
```

StartSyncCnv

Command

```
StartSyncCnv()
```

Description

Enable the conveyor tracking function.

Return

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

CnvMovL

Command

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

Description

Based on the workpiece coordinate system, the robot performs linear movement to the target position to execute conveyor belt tracking.

Required parameter

Parameter	Type	Description
P	string	Target point supports either joint variables or pose variables. The format is "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

Optional parameter

Parameter	Type	Description
user	string	The format is "user=index", where index refers to the calibrated user coordinate system index.The range of values is [0,50].
tool	string	The format is "tool=index", where index represents the calibrated tool coordinate system index.The range of values is [0,50].
a	string	The format is “a=value”, where value indicates the acceleration ratio of the robot's movement when executing the command.The range of values is [1,100].
v	string	The format is "v=value", where value signifies the speed ratio of the robot's movement when executing the command.The range of values is [1,100].
cp	string	The format is "cp=value", where value represents the smooth transition ratio, which is mutually exclusive with r.The range of values is [0,100].
r	string	The format is "r=value", where value indicates the smooth transition radius, mutually exclusive with cp. If both exist, r takes precedence.The unit is millimeters (mm).

Return

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

flag: Tracking result, with values defined as follows:

- 0: Execution successful
- 1: Tracking failed, workpiece type not detected
- 2: Tracking failed, workpiece type detected but not within picking boundary range
- 3: Tracking failed, workpiece has moved beyond leave boundary

Example

```
CnvMovL(pose= {x,y,z,rx,ry,rz},user = 1, tool = 0, a = 20, v = 50, cp = 100)
```

CnvMovC

Command

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

Description

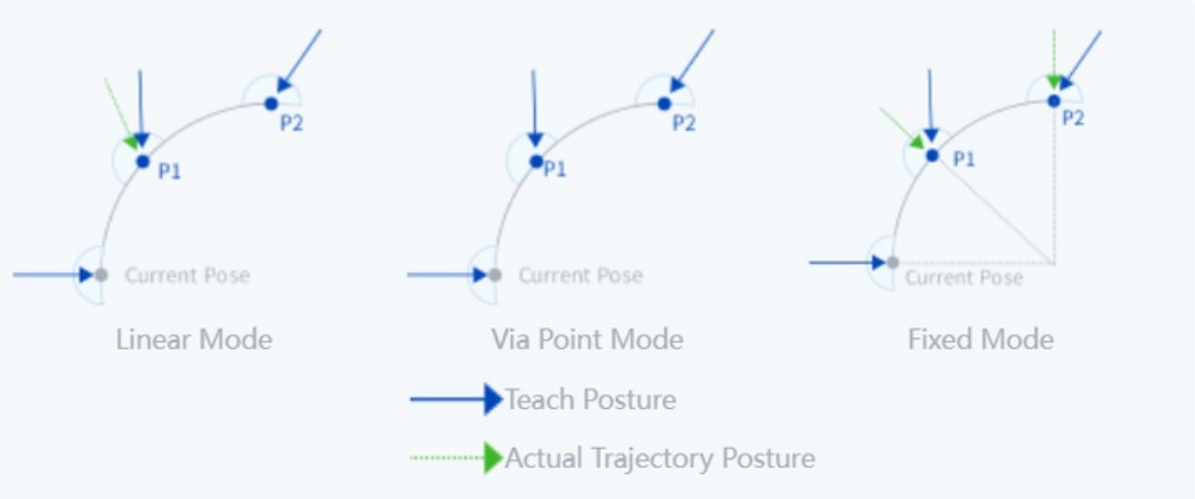
Based on the workpiece coordinate system, the robot moves from its current position to the target point P2 through an arc motion via the intermediate point P1. After reaching point P2, it performs conveyor tracking.

Required parameter

Parameter	Type	Description
P1	string	The intermediate point of the arc supports either joint variables or pose variables. The format is "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".
P2	string	The target point for motion supports either joint variables or pose variables. The format is "joint = {j1, j2, j3, j4, j5, j6}" or "pose = {x, y, z, rx, ry, rz}".

Optional parameter

Parameter	Type	Description
user	string	The format is "user=index", where the index is the calibrated user coordinate system index. The valid range for index is [0,50].
tool	string	The format is "tool=index", where the index is the calibrated tool coordinate system index. The valid range for index is [0,50].
a	string	The format is "a=value", where value represents the motion acceleration ratio of the robot arm when executing this command. The valid range for value is [1,100].
v	string	The format is "v=value", where value represents the motion speed ratio of the robot arm when executing this command. The valid range for value is [1,100].
cp	string	The format is “cp=value”, where value represents the smooth transition ratio and is mutually exclusive with the parameter r. The valid range for value is [0,100].
r	string	The format is "r=value", where value represents the smooth transition radius and is mutually exclusive with the parameter cp. If both parameters are specified, r takes precedence. The unit is millimeters (mm). The smooth transition alters the robot's motion trajectory and may affect the timing of DO outputs, so please use it with caution.
mode	int	The format is "mode=value", which sets the posture control parameter to enable adaptive control of the robot's posture relative to the arc during interpolation. This allows the robot to meet the needs of different scenarios. The valid range for value is [0, 2]. • mode=0: Linear mode. Interpolates from the current posture to the target posture P2, ignoring the posture at P1. In this mode, posture changes are limited to less than 180°. Suitable for applications with no specific requirements on robot posture. • mode=1: Via-point mode. Interpolates from the current posture, passing through the intermediate posture at P1, to the target posture P2. Primarily used in welding applications. • mode=2: Fixed mode. Starting from the current posture, the TCP maintains a constant orientation relative to the arc tangent, ignoring postures at P1 and P2. In this mode, the posture rotation angle matches the arc angle, enabling posture changes exceeding 180°. Mainly used in applications such as gluing and grinding.



NOTE

- When set to **mode=1** (Via-point mode), to ensure uniform motion speed along the arc trajectory, it is recommended to position the intermediate point as close as possible to the midpoint of the actual arc during teaching.
- When set to **mode=1** (Via-point mode), the posture at each point should be properly adjusted to ensure that the posture change from the start point to the intermediate point is approximately equal to the posture change from the intermediate point to the target point. Otherwise, the constructed posture curve may exceed the robot's reachable range, resulting in a runtime error.

Return

```
ErrorID,{flag},CnvMovC(P1,P2,user, tool, a, v, cp|r, mode);
```

flag: Tracking result, with values defined as follows:

- 0: Execution successful
- 1: Tracking failed, workpiece type not detected
- 2: Tracking failed, workpiece type detected but not within picking boundary range
- 3: Tracking failed, workpiece has moved beyond leave boundary

Example

```
CnvMovC(joint = {1, 2, 3, 4, 5, 6},joint = {7, 8, 9, 10, 11, 12},user = 1, tool = 0, a = 20, v
 = 50, cp = 100)
```

StopSyncCnv

Command

```
StopSyncCnv()
```

Description

Stop the conveyor tracking function. Once this command is executed, the subsequent commands will continue to be executed.

This command must be used in conjunction with the `StartSyncCnv()` command. Between `StartSyncCnv()` and `StopSyncCnv()`, the program must not call any motion commands other than `CnvMovL` and `CnvMovC`, otherwise an error will be reported.

Return

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

SetCnvPointOffset

Command

```
SetCnvPointOffset(xOffset, yOffset)
```

Description

Set the offset values in the X and Y directions within the conveyor user coordinate system.

Required parameter

Parameter	Type	Description
xOffset	double	Offset in the X-axis direction, unit: mm.
yOffset	double	Offset in the Y-axis direction, unit: mm.

Return

```
ErrorID,{},SetCnvPointOffset(xOffset, yOffset);
```

Example

```
SetCnvPointOffset(10, 10)
```

SetCnvTimeCompensation

Command

```
SetCnvTimeCompensation(time)
```

Description

Set the compensation time to offset the position deviation in workpiece picking caused by time delays from vision triggering.

Required parameter

Parameter	Type	Description
time	int	Compensation time, measured in milliseconds (ms).

Return

```
ErrorID,{},SetCnvTimeCompensation(time);
```

Example

```
SetCnvTimeCompensation(100)
```

2.12 Point reachability detection command

Overview

This set of commands is used to check whether each point in the specified motion trajectory is reachable.

Command list

Command	Function	Command type
CheckOddMovL	Check the point reachability for linear motion	Immediate command
CheckOddMovJ	Check the point reachability for joint motion	Immediate command
CheckOddMovC	Check the point reachability for arc motion	Immediate command

CheckOddMovL

Command

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

Description

Check the point reachability for linear motion. Point parameters only support joint variables `(joint = {j1, j2, j3, j4, j5, j6})` .

This command can only be called when the robot arm is stationary.

Required parameter

Parameter	Type	Description
P1	string	The start point of linear motion.
P2	string	The end point of linear motion.

Optional parameter

Parameter	Type	Description
user	int	User coordinate system, applicable to all points in the command.
tool	int	Tool coordinate system, applicable to all points in the command.
a	int	The acceleration ratio for the robot when executing this command. Range:

a	int	(0,100].
v	int	The velocity ratio for the robot when executing this command. Range: (0,100].
cp	int	Continuous path ratio. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp”. If both “r” and “cp” exist, “r” takes precedence. Unit: mm.

Return

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

result: The detection result.

- 0: All trajectory points are reachable.
- -1: The detection cannot be performed. Usually because the robot arm is moving when the command is called.
- For other return values, please refer to General Error Codes for Point Reachability Detection.

Example

```
CheckOddMovL(joint = {0, 0, 90, 0, 0, 0},joint = {90, 30, 0, 0, 0, 0})
```

Check the point reachability for linear motion from `{0, 0, 90, 0, 0, 0}` to `{90, 30, 0, 0, 0, 0}`.

CheckOddMovJ

Command

```
CheckOddMovJ(P1,P2,a,v,cp)
```

Description

Check the point reachability for joint motion. Point parameters only support joint variables `(joint = {j1, j2, j3, j4, j5, j6})`.

This command can only be called when the robot is stationary.

Required parameter

Parameter	Type	Description
P1	string	The start point of joint motion.
P2	string	The end point of joint motion.

Optional parameter

Parameter	Type	Description
a	int	The acceleration ratio for the robot when executing this command. Range: (0,100].
v	int	The velocity ratio for the robot when executing this command. Range: (0,100].
cp	int	Continuous path ratio. Range: [0,100].

Return

```
ErrorID,{result},CheckOddMovJ(P1,P2,a,v,cp);
```

result: The detection result.

- 0: All trajectory points are reachable.
- -1: The detection cannot be performed. Usually because the robot is moving when the command is called.
- For other return values, please refer to General Error Codes for Point Reachability Detection.

Example

```
CheckOddMovJ(joint = {0, 0, 90, 0, 0, 0},joint = {90, 30, 0, 0, 0, 0})
```

Check the point reachability for joint motion from `{0, 0, 90, 0, 0, 0}` to `{90, 30, 0, 0, 0, 0}` .

CheckOddMovC

Command

```
CheckOddMovC(P1,P2,P3,user,tool,a,v,cp|r)
```

Description

Check the point reachability for arc motion. Point parameters only support joint variables `(joint = {j1, j2, j3, j4, j5, j6})` .

This command can only be called when the robot is stationary.

Required parameter

Parameter	Type	Description
P1	string	The start point of arc motion.
P2	string	The through point of arc motion.

P3	string	The end point of arc motion.

Optional parameter

Parameter	Type	Description
user	int	User coordinate system, applicable to all points in the command.
tool	int	Tool coordinate system, applicable to all points in the command.
a	int	The acceleration ratio for the robot when executing this command. Range: (0,100].
v	int	The velocity ratio for the robot when executing this command. Range: (0,100].
cp	int	Continuous path ratio. Range: [0,100].
r	int	Continuous path radius, incompatible with “cp”. If both “r” and “cp” exist, “r” takes precedence. Unit: mm.

Return

```
ErrorID,{result},CheckOddMovC(P1,P2,P3,user,tool,a,v,cp|r);
```

result: The detection result.

- 0: All trajectory points are reachable.
- -1: The detection cannot be performed. Usually because the robot arm is moving when the command is called.
- For other return values, please refer to General Error Codes for Point Reachability Detection.

Example

```
CheckOddMovC(joint = {0, 0, 90, 0, 0, 0},joint = {60, 30, 0, 0, 0, 0},joint = {90, 30, 0, 0, 0
, 0})
```

Check the point reachability for arc motion `{0, 0, 90, 0, 0, 0}` => `{60, 30, 0, 0, 0, 0}` => `{90, 30, 0, 0, 0, 0}` .

General Error Codes for Point Reachability Detection

- 16: A point in the trajectory is near the shoulder singularity.
- 17: A point in the trajectory is unreachable.
- 18: A point in the trajectory will trigger the joint limit.
- 19: There are duplicate points in the arc motion.
- 26: A point in the trajectory is near the wrist singularity.
- 27: A point in the trajectory is near the elbow singularity.
- 29: Speed parameter error.

3 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 is a **configurable** port to feed back robot information (feed back every **200ms** by default. If you need to modify, please contact technical support).
- 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	Description
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 structure test standard value 0x0123 4567 89AB CDEF
N/A	N/A	N/A	8	0056 – 0063	Reserved
SpeedScaling	double	1	8	0064 – 0071	Speed ratio
N/A	N/A	N/A	16	0072 – 0087	Reserved

VRobot	double	1	8	0088 – 0095	Robot voltage
IRobot	double	1	8	0096 – 0103	Robot current
ProgramState	double	1	8	0104 – 0111	Script running status
SafetyIOIn	char	2	2	0112 – 0113	Safety IO input status
SafetyIOOut	char	2	2	0114 – 0115	Safety IO output status
N/A	N/A	N/A	76	0116 – 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 (calculate through joint current)
ToolVectorTarget	double	6	48	0768 – 0815	TCP target Cartesian 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	0960 – 1007	Joint voltage

HandType	char	4	4	1008 – 1011	Hand system (alternate parameter)
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
CRRobotType	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	1042	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
Load	double	1	8	1168 – 1175	Payload (kg)
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	[qw,qx,qy,qz] Target quaternion
ActualQuaternion[4]	double	4	32	1384 – 1415	[qw,qx,qy,qz] Actual quaternion
AutoManualMode	char	1	2	1416 – 1417	Manual/Automatic mode
ExportStatus	unsigned short	1	2	1418 – 1419	USB export status
SafetyState	char	1	1	1420	1420 Safety status 1420:0 Emergency stop (low active) 1420:1 Protective stop (low active) 1420:2 Reduced mode (low active) 1420:3 Non-stop status (low active) 1420:4 In motion (low active) 1420:5 System emergency stop (low active) 1420:6 User emergency stop (low active) 1420:7 Safety home output (low active, active when not at safe home position)
SafeState N/A	char	1	1	1421	Reserved for safety status
N/A	N/A	N/A	18	1422 – 1439	Reserved
TOTAL			1440		1440byte package

Motion parameter feedback values

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
- 0x04 (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
116	CR5AF
117	CR7A
120	CR10A
121	CR10AF
122	CR12A
126	CR16A
127	CR20AF
130	CR20A
150	Magician E6
160	New Nova 2
161	New Nova 5
162	New Nova 2S
203	CR3V
205	CR5V
207	CR7V
210	CR10V
212	CR12V
216	CR16V
220	CR20V

4 Error Code

Error code	Description	Remark
0	No error	The command has been delivered successfully.
-1	Command execution failed	The command has been received but failed to be executed.
-2	Robot in alarm status	The robot cannot execute commands in the alarm status. You need to clear the alarm and redeliver the command.
-3	Robot 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	Robot in power-off status	The robot cannot execute commands in the power-off status. You need to power the robot on.
-5	Robot in script running status	The robot cannot execute some commands when it is in the script running status, you need to pause/stop the script first.
-6	The axis and motion type of MoveJog command do not match	Adjust the coordtype parameter. See the MoveJog command description for details.
-7	Robot in script paused status	The robot cannot execute some commands when it is in the script paused status, you need to stop the script first.
-8	Robot certification expired	The robot is in an unavailable status. Please contact FAE for assistance.
-9	Corresponding command is disabled	The safety commands SetSafeWallEnable, SetFCCollision, FCCollisionSwitch, SetCollisionLevel, and SetBackDistance are disabled.
...	...	...
-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	-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".

	the first parameter is incorrect.	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 with name is incorrect. The range of the first parameter is incorrect	-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= {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).

5 TCP Commands Allowed in Various Statuses

The running status of RequestControl() is not described here. Please refer to the specific RequestControl() command for details.

Error status

TCP commands allowed when the robot is in an error status (including emergency stop):

- ClearError()
- GetErrorID()
- EmergencyStop()
- Stop()
- RobotMode()
- LogExportUSB()
- GetExportStatus()

Power-off status

The status in which the controller is powered on but the robot is not.

In addition to commands allowed in **Error status**, the following commands are also supported:

- PowerOn()

Script running status

Commands allowed when running a script project:

- SpeedFactor()
- RobotMode()
- DoInstant()
- ToolDoInstant()
- AOInstant()
- Stop()
- Pause()
- Continue()
- GetStartPose()
- PositiveKin()
- InverseSolution()
- GetAngle()
- GetPose()

- EmergencyStop()
- ModbusCreate()
- ModbusClose()
- GetInBits()
- GetInRegs()
- GetCoils()
- SetCoils()
- GetHoldRegs()
- SetHoldRegs()
- GetErrorID()
- DI()
- ToolDI()
- AI()
- ToolAI()
- DIGroup()
- GetDO()
- GetAO()
- GetDOGroup()
- SetTool485()
- SetToolPower()
- SetToolMode()
- CalcUser()
- CalcTool()
- GetInputBool()
- GetInputInt()
- GetInputFloat()
- GetOutputBool()
- GetOutputInt()
- GetOutputFloat()
- SetOutputBool()
- SetOutputInt()
- SetOutputFloat()
- GetCurrentCommandId()
- LogExportUSB()
- GetExportStatus()
- ClearError()
- GetForce()

Script paused status

The status where a script project is paused.

In addition to commands allowed in **Script running status**, the following commands are also supported:

- EnableRobot()
- DisableRobot()
- RunTo()
- PathRecovery()
- StartDrag()
- StopDrag()
- SixForceHome()
- EnableFTSensor()
- ForceDriveMode()
- ForceDriveSpeed()

Other status

No restrictions. All commands can be sent.