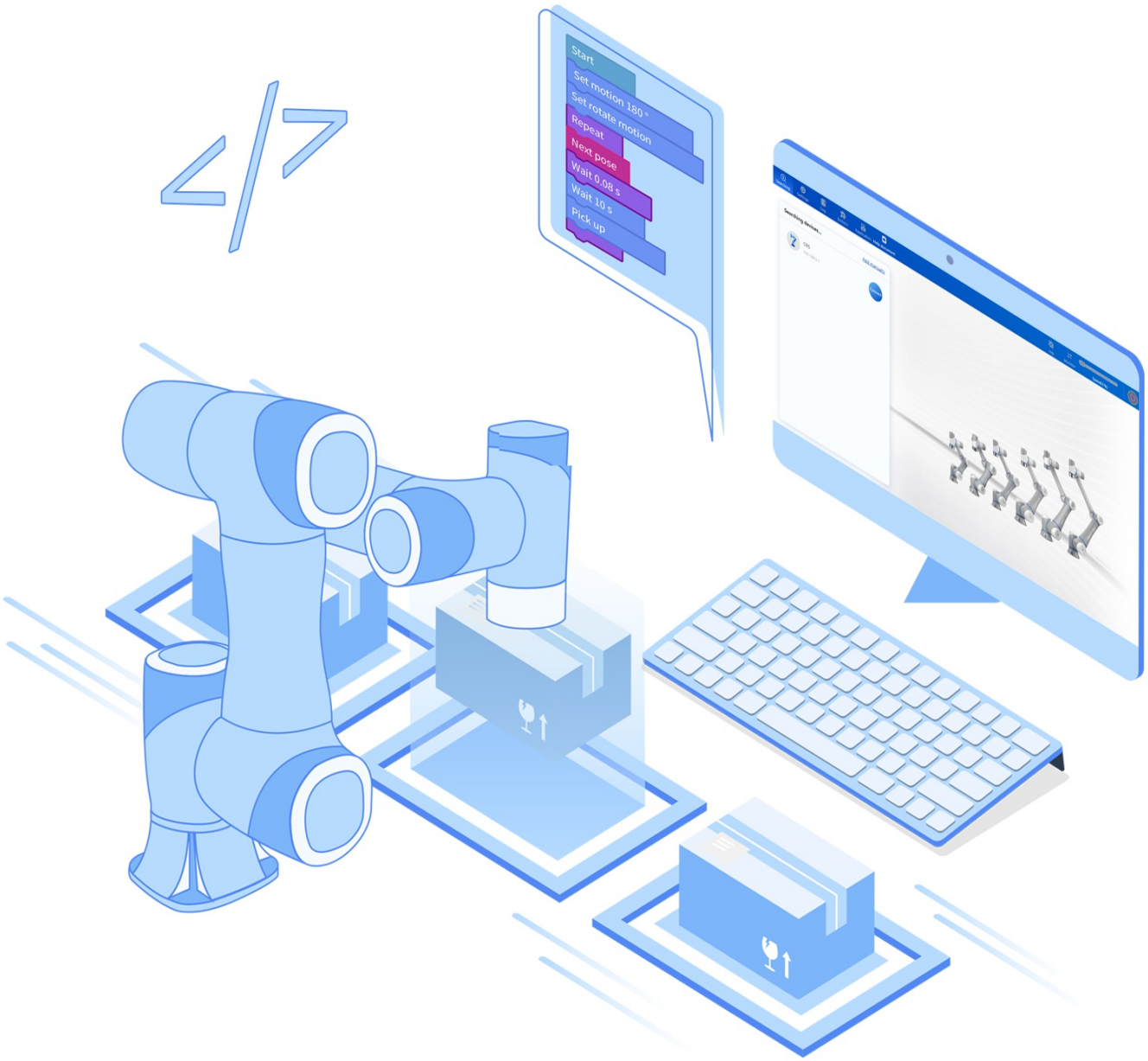
DOBOT

Dobot TCP/IP Remote Control Interface Guide



Shenzhen Yuejiang Technology Co.,Ltd.|China

Table of Contents

Preface

Purpose

This document introduces the TCP/IP secondary development interfaces and their usage of Dobot industrial robot controller (V3), 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	Revised content
2024/01/05	The second release
2023/05/25	The first release

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.

To support TCP/IP protocol, the controller version should be V3.5.1.19 or above for six-axis robots.

> **NOTE**
> For CR series, the end button functions cannot be used in TCP/IP control mode.

Port description

According to the design, Dobot robots will open 29999, 30003, 30004, 30005 and 30006 server ports.

- Server port 29999: The upper computer can send some **setting-related commands** directly to the robot via port 29999, or **acquire** certain status of the robot. These functions are called Dashboard.
- Server port 30003: The upper computer can send some **motion-related commands** directly to the robot via port 30003 to control the robot to move.
- Server port 30004, 30005 and 30006: Port 30004 (real-time feedback port) receives robot information every **8ms**. Port 30005 feeds back robot information every **200ms**. Port 30006 is a **configurable** port to feed back robot information (feed back every **50ms** by default. If you need to modify, please contact technical support). There are 1440 bytes per packet received via the real-time feedback port, and these bytes are arranged in a standard format.

> **NOTE**
> - Controller V3.5.2 and above support ports 30004, 30005 and 30006.
> - Controller V3.5.1 provides real-time feedback on robot status via port 30003.
> - Controller V3.5.5 and above can configure the feedback period of port 30005. If you need to modify it, please contact technical support.

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.

Immediate command and Queue command

TCP/IP remote control commands are categorized as immediate command and queued command.

- The immediate commands will be executed immediately after being delivered, and the execution results will be returned.
- The queue commands return immediately after being delivered, but instead of being executed, they enter the background algorithmic queue and wait to be executed.

Most of the Dashboard commands (delivered by port 29999) are immediate commands, and some of the motion and IO related commands are queued commands.

Motion-related commands (delivered by port 30003) are queued commands.

If an immediate command is called after a queue command, the immediate command may be executed before the queue command is completed, as shown in the following example.

```
MovJ(-500,100,200,150,0,90)  // Queue command
RobotMode()  // Immediate command
```

In this example, RobotMode() will be executed before the robot completes its motion, and return 7 (in motion).

If you want to make sure that all previous commands are completed when the immediate command is executed, you can call the Sync() command before calling the immediate command. The Sync() command will block program execution until all previous commands are completed, as shown in the following example.

```
MovJ(-500,100,200,150,0,90)  // Queue command
Sync()
RobotMode()  // Immediate command
```

In this example, RobotMode() will be executed after the robot completes its motion, and return 5 (idle).

Get DEMO

Dobot provides DEMOs for secondary development in various programming languages, stored in Github. Please obtain the DEMOs you need and refer to them for secondary development.

2 Dashboard Command

The dashboard commands need to be delivered through **port 29999**.

2.1 Control command

PowerOn (Immediate command)

Command

```
PowerOn()
```

Description

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

Return

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

Example

```
PowerOn()
```

Power on the robot.

EnableRobot (Immediate command)

Command

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

Description

Enable the robot, which is necessary before executing the queue commands (robot motion, queue IO, etc.).

> **NOTE**
>
> - Each time you enter TCP mode, call this command once even if the robot arm is enabled, otherwise TCP command execution exceptions may occur.
> - Calling this command while the robot arm is moving will cause the robot arm to stop its current motion.

Parameter

Parameter	Type	Description
load	double	Load weight. The value range should not exceed the load range of corresponding robot models, unit: kg.
centerX	double	Eccentric distance in X-axis direction. Range: -500 – 500, unit: mm.
centerY	double	Eccentric distance in Y-axis direction. Range: -500 – 500, unit: mm.
centerZ	double	Eccentric distance in Z-axis direction. Range: -500 – 500, unit: mm.

All are optional parameters. The number of parameters that can be carried is as follows:

- 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).

Return

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

Example

```
EnableRobot()
```

Enable the robot without setting load weight and eccentric parameters.

```
EnableRobot(1.5)
```

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

```
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.

DisableRobot (Immediate command)

Command

```
DisableRobot()
```

Description

Disable the robot.

Return

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

Example

```
DisableRobot()
```

Disable the robot.

ClearError (Immediate command)

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.

> **NOTE**
> After clearing the alarm, you need to call EnableRobot command again before delivering motion commands.

Return

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

Example

```
ClearError()
```

Clear the alarms of the robot.

ResetRobot (Immediate command)

Command

```
ResetRobot()
```

Description

Stop the robot and clear the planned command queue.

Return

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

Example

```
ResetRobot()
```

Stop the robot and clear the planned command queue.

RunScript (Immediate command)

Command

```
RunScript(projectName)
```

Description

Run the project.

Parameter

Parameter	Type	Description
projectName	string	Project name

Return

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

Example

```
RunScript("demo")
```

Run the project named "demo".

StopScript (Immediate command)

Command

```
StopScript()
```

Description

Stop running the project.

Return

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

Example

```
StopScript()
```

Stop running the project.

PauseScript (Immediate command)

Command

```
PauseScript()
```

Description

Pause running the project.

Return

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

Example

```
PauseScript()
```

Pause running the project.

ContinueScript (Immediate command)

Command

```
ContinueScript()
```

Description

Continue running the paused project.

Return

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

Example

```
ContinueScript()
```

Continue running the paused project.

Pause (Immediate command)

Command

```
Pause()
```

Description

Pause the motion commands that are not delivered by project (generally, the motion commands delivered by TCP), without clearing the motion queue.

Return

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

Example

```
Pause()
```

Pause the motion commands that are not delivered by project.

Continue (Immediate command)

Command

```
Continue()
```

Description

Corresponding to the Pause command, continue to run the motion command paused by Pause command.

Return

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

Example

```
Continue()
```

Continue to run the motion command paused by Pause command.

EmergencyStop (Immediate command)

Command

```
EmergencyStop()
```

Description

Stop the robot arm in an emergency. After the emergency stop, the robot arm will power off and alarm. The alarm needs to be cleared before it can be powered on and enabled again. When the software triggers the emergency stop status, delivering the PowerOn() command can directly reset and power on the robot arm.

Return

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

Example

```
EmergencyStop()
```

Stop the robot arm in an emergency.

BrakeControl (Immediate command)

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.

This command is supported by controller V3.5.2 and above.

Parameter

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

Return

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

Example

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

Switch on the brake of Joint 1.

StartDrag (Immediate command)

Command

```
StartDrag()
```

Description

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

This command is supported by controller V3.5.2 and above.

Return

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

Example

```
StartDrag()
```

The robot arm enters the drag mode when there is no alarm.

StopDrag (Immediate command)

Command

```
StopDrag()
```

Description

The robot arm exits the drag mode. The robot arm cannot exit the drag mode through this command in error status.

This command is supported by controller V3.5.2 and above.

Return

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

Example

```
StopDrag()
```

The robot arm exits the drag mode when there is no alarm.

SetCollideDrag (Immediate command)

Command

```
SetCollideDrag(status)
```

Description

The robot arm is forced to enter or exit the drag mode. The robot arm can enter or exit the drag mode through this command in error status.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
status	int	Forced drag switch. 0: forced to exit the drag mode. 1: forced to enter the drag mode.

Return

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

Example

```
SetCollideDrag(1)
```

The robot arm is forced to enter the drag mode.

SetSafeSkin (Queue command)

Command

```
SetSafeSkin(status)
```

Description

Switch on or off the SafeSkin. You can use this command to control only after installing and switching on the SafeSkin plugin in the software. The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

Parameter

Parameter	Type	Description
status	int	SafeSkin switch, 0: off, 1: on.

Return

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

Example

```
SetSafeSkin(1)
```

Switch on the SafeSkin.

Wait (Queue command)

Command

```
wait(time)
```

Description

The command queue delays for a specified time.

This command is supported by controller V3.5.5 and above.

Parameter

Parameter	Type	Description
time	int	Delayed time, unit: ms, range: (0,3600*1000)

Return

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

Example

```
wait(1000)
```

The command queue delays for 1000ms.

2.2 Settings command

SpeedFactor (Immediate command)

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.

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

> **NOTE**
> After modifying the global speed by this command, if the EnableRobot or RunScript command is used again, the global speed will change back to the value set in the software before entering TCP/IP control mode.

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 (Queue command)

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.

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting. After exiting the TCP mode, it will be restored to the coordinate system set by the jog panel.

> **NOTE**
> The global user coordinate system may be reset to 0 after using the RunScript command.

Parameter

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

Return

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

-1 indicates that the index of the set user coordinate system does not exist.

Example

```
User(1)
```

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

Tool (Queue command)

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.

The global tool coordinate system set by this command takes effect only in the TCP/IP mode. If it is not set, the default global tool coordinate system is the one set in the software before entering TCP/IP mode.

> **NOTE**
> The global tool coordinate system may be reset to 0 after using the RunScript command.

Parameter

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

Return

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

-1 indicates that the index of the set tool coordinate system does not exist.

Example

```
Tool(1)
```

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

PayLoad (Queue command)

Command

```
PayLoad(weight,inertia)
```

Description

Set the load of the robot arm.

This command can also be written as LoadSet. Calling LoadSet (weight, inertia) has the same effect as PayLoad (weight, inertia).

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

Parameter

Parameter	Type	Description
weight	double	Load weight. The value range should not exceed the load range of corresponding robot models. Unit: kg
interia	double	Load inertia. Unit: kgm²

Return

```
ErrorID,{},PayLoad(weight,inertia);
```

Example

```
PayLoad(3,0.4)
```

Set the load weight to 3kg, and inertia to 0.4kgm².

LoadSwitch (Queue command)

Command

```
LoadSwitch(status)
```

Description

Switch on/off the load setting. The load setting is off by default. Switching on the load setting can improve the sensitivity of collision detection.

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

Parameter

Parameter	Type	Description
status	int	Load setting switch. 0: off, 1: on

Return

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

Example

```
LoadSwitch(1)
```

Switch on the load setting.

AccJ (Queue command)

Command

```
AccJ(R)
```

Description

Set acceleration ratio of joint motion.

The default value is 50 when this command is not called, and the value set by this command will continue to be maintained after exiting TCP mode.

> **NOTE**
> The parameter may be reset to 50 after using the RunScript command.

Parameter

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

Return

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

Example

```
AccJ(50)
```

Set the acceleration ratio of joint motion to 50%.

AccL (Queue command)

Command

```
AccL(R)
```

Description

Set acceleration ratio of linear and arc motion.

The default value is 50 when this command is not called, and the value set by this command will continue to be maintained after exiting TCP mode.

> **NOTE**
> The parameter may be reset to 50 after using the RunScript command.

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%.

SpeedJ (Queue command)

Command

```
SpeedJ(R)
```

Description

Set the speed ratio of joint motion.

The default value is 50 when this command is not called, and the value set by this command will continue to be maintained after exiting TCP mode.

> **NOTE**

The parameter may be reset to 50 after using the RunScript command.

Parameter

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

Return

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

Example

```
SpeedJ(50)
```

Set the speed ratio of joint motion to 50%.

SpeedL (Queue command)

Command

```
SpeedL(R)
```

Description

Set the speed ratio of linear and arc motion.

The default value is 50 when this command is not called, and the value set by this command will continue to be maintained after exiting TCP mode.

NOTE
The parameter may be reset to 50 after using the RunScript command.

Parameter

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

Return

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

Example

```
SpeedL(50)
```

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

CP (Queue command)

Command

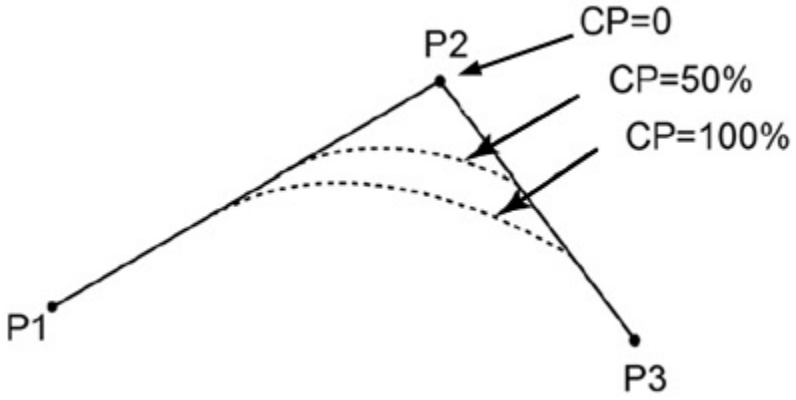
```
CP(R)
```

Description

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

The default value is 50 when this command is not called, and the value set by this command will continue to be maintained after exiting TCP mode.

> **NOTE**
> The parameter may be reset to 50 after using the RunScript command.



Parameter

Parameter	Type	Description
R	unsigned int	Continuous path ratio, range: [0, 100]

Return

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

Example

```
CP(50)
```

Set the continuous path ratio to 50.

SetArmOrientation (Queue command)

Command

```
SetArmOrientation(LorR,UorD,ForN,Config6)
```

Description

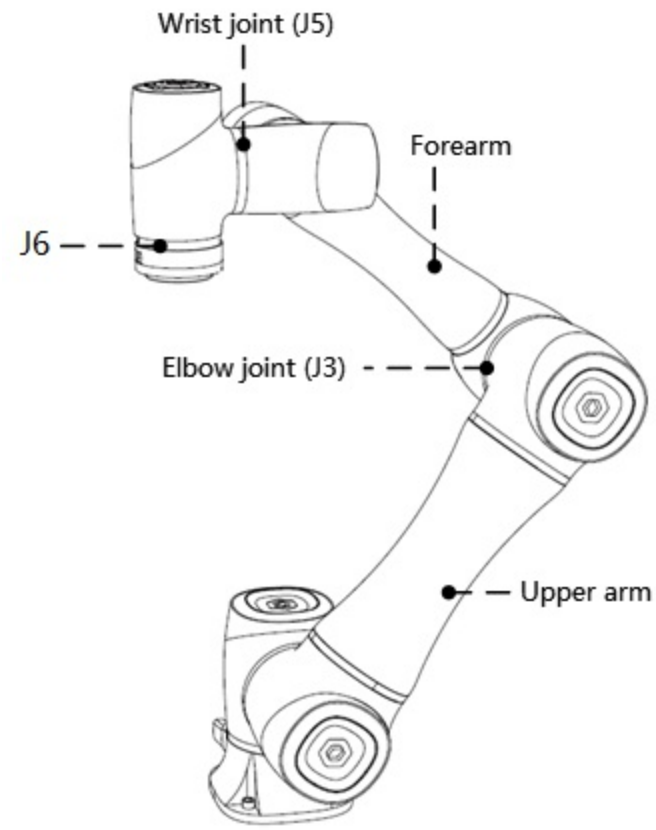
Set the hand system of the target point. When the target point is a Cartesian coordinate point, you can

determine the unique posture of the robot arm by the hand system. Once the hand system is set, the subsequent motion commands with a Cartesian coordinate point as the target will plan the motion path based on the hand system.

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

Parameter

Parameter	Type	Description
LorR	int	Orientation of the upper arm. 1: forward (joint angle of J2: positive), -1: backward (joint angle of J2: negative).
UorD	int	Orientation of the elbow joint. 1: upward (joint angle of J3: positive), -1: downward (joint angle of J3: negative).
ForN	int	Indicates whether the wrist joint is flipped or not. 1: wrist joint is not flipped (joint angle of J5: positive), -1: wrist joint is flipped (joint angle of J5: negative).
Config6	int	J6 angle range. -1: [0, -90], -2: [-90, -180], 1: [0, 90], 2: [90, 180] ...



Return

```
ErrorID,{},SetArmOrientation(LorR,UorD,ForN,Config6);
```

Example

```
SetArmOrientation(1,1,-1,1)
```

Set the hand system of the six-axis robot to upper arm forward, elbow joint upward, wrist joint flipped, and J6 angle within [0, 90].

SetCollisionLevel (Queue command)

Command

```
SetCollisionLevel(level)
```

Description

Set the collision detection level.

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

Parameter

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

Return

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

Example

```
SetCollisionLevel(1)
```

Set the collision detection level to 1.

TCPSpeed (Queue command)

Command

```
TCPSpeed(vt)
```

Description

Set the absolute speed. The motion commands of Cartesian coordinate system after this command will run at the set absolute speed, and the joint coordinate motion commands will not be affected. After setting TCPSpeed, SpeedL no longer takes effect, but the maximum speed is still limited by the global speed (including reduced mode).

When using the welding process package, if the command conflicts with the related commands of welding, the welding commands will take precedence.

This command is supported by controller V3.5.5 and above.

Parameter

Parameter	Type	Description
vt	unsigned int	Absolute speed, unit: mm/s, range: [0,100000)

Return

```
ErrorID,{},TCPSpeed(vt);
```

Example

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

The robot arm moves to {-500,100,200,150,0,90} at an absolute speed of 100mm/s through linear motion.

TCPSpeedEnd (Queue command)

Command

```
TCPSpeedEnd()
```

Description

The command is used with TCPSpeed command for switching off the absolute speed settings.

This command is supported by controller V3.5.5 and above.

Return

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

Example

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

The robot arm moves to {-500,100,200,150,0,90} at an absolute speed of 100mm/s through linear motion, and then moves to {500,100,200,150,0,90} at the global speed through linear motion.

SetUser (Immediate command)

Command:

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

Description:

Modify the specified user coordinate system.

This command is supported by controller V3.5.7 and above.

Required parameter:

Parameter	Type	Description
index	int	Index of the user coordinate system, range: [0,9]. The initial value of coordinate system 0 is the base coordinate system.
table	string	User coordinate system after modification, format: {x, y, z, rx, ry, rz}), which is recommended to obtain through CalcUser command.

Return

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

Example:

```
SetUser(1,{10,10,10,0,0,0})
// 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 (Immediate command)

Command:

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

Description:

Calculate the user coordinate system.

This command is supported by controller V3.5.7 and above.

Required parameter:

Parameter	Type	Description
index	int	Index of the user coordinate system, range: [0,9]. The initial value of coordinate system 0 is the base coordinate system.
matrix_direction	int	Calculation method. 1: left multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along the base coordinate system. 0: right multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along itself.
table	string	User coordinate system offset, format: {x, y, z, rx, ry, rz}.

Return:

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

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

Example 1:

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

Calculate: User coordinate system 1 left-multiplies {10,10,10,10,10,10}.
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})
// newUser = CalcUser(1,0,10,10,10,10,10,10)
```

Calculate: User coordinate system 1 right-multiplies {10,10,10,10,10,10}.
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.

SetTool (Immediate command)

Command:

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

Description:

Modify the specified tool coordinate system.

This command is supported by controller V3.5.7 and above.

Required parameter:

Parameter	Type	Description
index	int	Index of the tool coordinate system, range: [0,9]. The initial value of coordinate system 0 is the flange coordinate system.
table	string	Tool coordinate system after modification, format: {x, y, z, rx, ry, rz}, which is recommended to obtain through CalcTool command.

Return

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

Example:

```
SetTool(1,{10,10,10,0,0,0})
// 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 (Immediate command)

Command:

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

Description:

Calculate the tool coordinate system.

This command is supported by controller V3.5.7 and above.

Required parameter:

Parameter	Type	Description
index	int	Index of the tool coordinate system, range: [0,9]. The initial value of coordinate system 0 is the flange coordinate system.
matrix_direction	int	Calculation method. 1: left multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along the flange coordinate system. 0: right multiplication, indicating that the coordinate system specified by "index" deflects the value specified by "table" along itself.
table	string	Tool coordinate system offset, format: {x, y, z, rx, ry, rz}.

Return:

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

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

Example 1:

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

Calculate: Tool coordinate system 1 left-multiplies {10,10,10,10,10,10}.
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,{10,10,10,10,10,10})
// CalcTool(1,0,10,10,10,10,10,10)
```

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

2.3 Calculating and obtaining command

RobotMode (Immediate command)

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_POWER_STATUS	Power-off status
4	ROBOT_MODE_DISABLED	Disabled (no brake switched on)
5	ROBOT_MODE_ENABLE	Enabled and idle (no alarm, no project running)
6	ROBOT_MODE_BACKDRIVE	Drag mode
7	ROBOT_MODE_RUNNING	Running status (including trajectory playback/fitting, executing motion commands, running project)
8	ROBOT_MODE_RECORDING	Trajectory recording mode
9	ROBOT_MODE_ERROR	There are uncleared alarms. This status has the highest priority. It returns 9 when there is an alarm, regardless of the status of the robot arm.
10	ROBOT_MODE_PAUSE	Pause status
11	ROBOT_MODE_JOG	Jogging status

Example

```
RobotMode()
```

Get the current status of the robot.

HandleTrajPoints (Immediate command)

Command

```
HandleTrajPoints(traceName)
```

Description

Preprocess the trajectory files. As the trajectory preprocessing results vary according to the size of the file,

and the processing time of algorithms will be different. **If you send this command without parameters, it refers to querying the result of the current command.**

- -3 indicates that the file content is incorrect.
- -2 indicates that the file does not exist.
- -1 indicates that the preprocessing is not completed.
- 0 indicates that preprocessing is completed with no errors.
- A value greater than 0 indicates that the point corresponding to the current result is fault.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
traceName	string	trajectory file name (with suffix) The trajectory file is stored in /dobot/userdata/project/process/trajectory/.

Return

```
ErrorID,{},HandleTrajPoints(traceName);
ErrorID,{},HandleTrajPoints();
```

Example

```
HandleTrajPoints(recv_string)
// Poll preprocessing result at certain intervals
HandleTrajPoints()
```

Deliver the “recv_string” trajectory for preprocessing, and poll the preprocessing result at certain intervals.

GetTraceStartPose (Immediate command)

Command

```
GetTraceStartPose(traceName)
```

Description

Get the first point (Cartesian point) of trajectory fitting, which is used with trajectory fitting motion commands.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
traceName	string	trajectory file name (with suffix) The trajectory file is stored in /dobot/userdata/project/process/trajectory/.

Return

```
ErrorID,{x,y,z,a,b,c},GetTraceStartPose(traceName);
```

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

Example

```
GetTraceStartPose(recv_string)
```

Get the first Cartesian point of the trajectory file named “recv_string” after trajectory fitting.

GetPathStartPose (Immediate command)

Command

```
GetPathStartPose(traceName)
```

Description

Get the first point (joint point) in trajectory playback, which is used with trajectory playback motion commands.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
traceName	string	trajectory file name (with suffix) The trajectory file is stored in /dobot/userdata/project/process/trajectory/.

Return

```
ErrorID,{j1,j2,j3,j4,j5,j6},GetTraceStartPose(traceName);
```

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

Example

```
GetTraceStartPose(recv_string)
```

Get the first joint point of the trajectory file named “recv_string”.

PositiveSolution (Immediate command)

Command

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

Description

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

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: °
User	int	Index of the calibrated user coordinate system
Tool	int	Index of the calibrated tool coordinate system

Return

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

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

Example

```
PositiveSolution(0,0,-90,0,90,0,1,1)
```

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

InverseSolution (Immediate command)

Command

```
InverseSolution(X,Y,Z,Rx,Ry,Rz,User,Tool,isJointNear,JointNear)
```

Description

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

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

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: °
User	int	Index of the calibrated user coordinate system
Tool	int	Index of the calibrated tool coordinate system
isJointNear	int	**Optional parameter.** It is used to set whether JointNear is effective. If the value is 0 or null, JointNear data is ineffective. The algorithm selects the joint angles according to the current angle. If the value is 1, the algorithm selects the joint angles according to JointNear data.
JointNear	string	**Optional parameter.** Joint coordinates for selecting joint angles.

Return

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

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

Example

```
InverseSolution(473.000000,-141.000000,469.000000,-180.000000,0.000,-90.000,0,0)
```

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

```
InverseSolution(473.000000,-141.000000,469.000000,-180.000000,0.000,-90.000,0,0,1,{0,0,-90,0,90,0})
```

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

GetSixForceData (Immediate command)

Command

```
GetSixForceData()
```

Description

Get six-dimensional force data. This command needs to be used with the six-dimensional force sensor.

Return

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

{Fx,Fy,Fz,Mx,My,Mz} refers to the original value of the current six-dimensional force.

Example

```
GetSixForceData()
```

Get the current six-dimensional force data.

GetAngle (Immediate command)

Command

```
GetAngle()
```

Description

Get the joint coordinates of current posture.

Return

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

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

Example

```
GetAngle()
```

Get the joint coordinates of current posture.

GetPose (Immediate command)

Command

```
GetPose(User=0,Tool=0)
```

Description

Get the Cartesian coordinates of current posture.

Parameter

Parameter	Type	Description
User	int	Index of the calibrated user coordinate system
Tool	int	Index of the calibrated tool coordinate system

Optional parameters. They are global user coordinate system and global tool coordinate system when not set.

Return

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

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

Example 1

```
GetPose()
```

Get the Cartesian coordinates of the current posture under global user coordinate system and global tool coordinate system.

Example 2

```
GetPose(User=1,Tool=0)
```

Get the Cartesian coordinates of the current posture under User coordinate system 1 and Tool coordinate system 0.

GetErrorID (Immediate command)

Command

```
GetErrorID()
```

Description

Get the current error code of the robot.

This command is supported by controller V3.5.2 and above.

Return

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

- [id,..., id] is the alarm information of the controller and algorithm, and [] refers to no alarm. If there are multiple alarms, they are separated by a comma ",". The collision detection value is -2, and SafeSkin collision detection value is -3. For other alarm definition, see the controller alarm document "alarm_controller.json".
- The last six [id] represent the alarm information of six servos respectively, and [] refers to no alarm. For alarm definitions, see the servo alarm document "alarm_servo.json".

Example

```
GetErrorID()
```

Get the current error code of the robot.

PalletCreate (Immediate command)

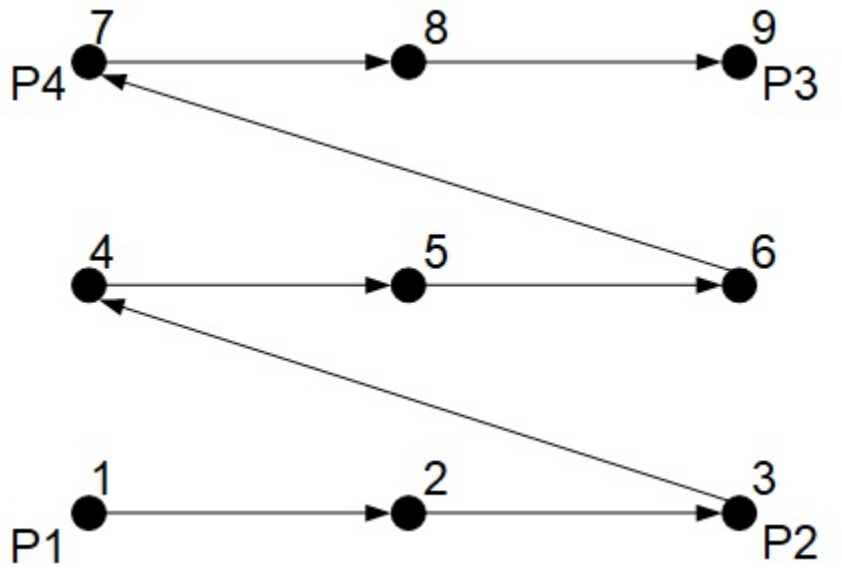
Command

```
PalletCreate(P1,P2,P3,P4,row=0,col=0,Palletname)
```

Description

Create pallet. Given the Cartesian coordinate points (P1 – P4) at the four corners of the pallet and the number of rows and columns of the pallet, the system automatically generates all pallet points. Up to 20 pallets can be created, all pallets are deleted when exiting TCP mode.

Taking a 3x3 pallet as an example, the relationship between the points specified by the four parameters and the generated pallet points is as follows.



This command is supported by controller V3.5.7 and above.

Parameter

Parameter	Type	Description
P1	array[double]	Cartesian coordinates of P1, in the format {X,Y,Z,Rx,Ry,Rz}
P2	array[double]	Cartesian coordinates of P2, in the same format as P1
P3	array[double]	Cartesian coordinates of P3, in the same format as P1
P4	array[double]	Cartesian coordinates of P4, in the same format as P1
row	int	Rows of pallets
col	int	Columns of pallets
Palletname	string	Pallet name, non-repeatable

Return

```
ErrorID,{number},PalletCreate(P1,P2,P3,P4,row,col,Palletname);
```

{number} refers to the number of pallets that have been created.

Example

```
PalletCreate({56,-568,337,175.5755,1,14},{156,-568,337,175.5755,1,14},{156,-468,337,175.5755,1
,14},{56,-468,337,175.5755,1,14},row=10,col=10,pallet1)
```

Create a pallet named pallet1 with ten rows and ten columns.

GetPalletPose (Immediate command)

Command

```
GetPalletPose(Palletname,index)
```

Description

Get the specified point of the created pallet. For the corresponding relationship between index and point, see the description of PalletCreate.

This command is supported by controller V3.5.7 and above.

Parameter

Parameter	Type	Description
Palletname	string	Pallet name
index	int	Point index, starting from 1

Return

```
ErrorID,{X,Y,Z,Rx,Ry,Rz},GetPalletPose(Palletname,index);
```

{X,Y,Z,Rx,Ry,Rz} refers to the Cartesian coordinates of the point corresponding to the index.

Example

```
GetPalletPose(pallet1,5)
```

Get the Cartesian coordinates of the point with index 5 of the pallet named pallet1.

2.4 IO command

DO (Queue command)

Command

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

Description

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

Parameter

Parameter	Type	Description
index	int	DO index
status	int	DO status. 1: signal, 0: without signal

Return

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

Example

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

Set DO1 to 1.

DOExecute (Immediate command)

Command

```
DOExecute(index,status)
```

Description

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

Parameter

Parameter	Type	Description
index	int	DO index

status	int	DO status. 1: signal, 0: without signal

Return

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

Example

```
DOExecute(1,1)
```

Set the status of DO1 to 1 immediately regardless of the current command queue.

DOGroup (Immediate command)

Command

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

Description

Set the status of a group of digital output ports (immediate command).

Parameter

Parameter	Type	Description
index1	int	Index of the first DO
value1	int	Status of the first DO. 1: signal, 0: without signal
...	...	...
indexN	int	Index of the last DO
valueN	int	Status of the last DO. 1: signal, 0: without signal

Return

```
ErrorID,{},DOGroup(index1,value1,index2,value2,...,indexn,valuen);
```

Example

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

Set DO4 to 1, DO6 to 0, DO2 to 1 and DO7 to 0.

ToolDO (Queue command)

Command

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

Description

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

Parameter

Parameter	Type	Description
index	int	Tool DO index
status	int	Tool DO status. 1: signal, 0: without signal

Return

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

Example

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

Set the tool DO1 to 1.

ToolDOExecute (Immediate command)

Command

```
ToolDOExecute(index,status)
```

Description

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

Parameter

Parameter	Type	Description
index	int	Tool DO index
status	int	Tool DO status. 1: signal, 0: without signal

Return

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

Example

```
ToolDOExecute(1,1)
```

Set the status of tool DO1 to 1 immediately regardless of the current command queue.

AO (Queue command)

Command

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

Description

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

Parameter

Parameter	Type	Description
index	int	AO index
value	double	AO output

Return

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

Example

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

Set AO1 output to 2.

AOExecute (Immediate command)

Command

```
AOExecute(index,value)
```

Description

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

Parameter

Parameter	Type	Description
index	int	AO index
value	double	AO output

Return

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

Example

```
AOExecute(1,2)
```

Set the AO1 output to 2 immediately regardless of the current command queue.

DI (Immediate command)

Command

```
DI(index)
```

Description

Get the status of digital input port.

Parameter

Parameter	Type	Description
index	int	DI index

Return

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

value: DI status. 0: signal, 1: without signal

Example

```
DI(1)
```

Get the status of DI1.

DIGroup (Immediate command)

Command

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

Description

Get the status of a group of digital input ports.

Parameter

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

Return

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

{value1,value2,...,valueN}: status of DI1 – DIN. 0: without signal, 1: signal.

Example

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

Get the status of DI4, DI6, DI2 and DI7.

ToolDI (Immediate command)

Command

```
ToolDI(index)
```

Description

Get the status of tool digital input port.

Parameter

Parameter	Type	Description
index	int	Tool DI index

Return

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

value: status of tool DI. 0: without signal, 1: signal.

Example

```
ToolDI(1)
```

Get the status of tool DI1.

AI (Immediate command)

Command

```
AI(index)
```

Description

Get the value of analog input port.

Parameter

Parameter	Type	Description
index	int	AI index

Return

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

value: AI input.

Example

```
AI(1)
```

Get the AI1 input.

ToolAI (Immediate command)

Command

```
ToolAI(index)
```

Description

Get the value of tool analog input port.

Parameter

Parameter	Type	Description
index	int	Tool AI index

Return

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

value: tool AI input.

Example

```
ToolAI(1)
```

Get the value of tool AI1.

SetTerminal485 (Immediate command)

Command

```
SetTerminal485(baudRate, dataLen, parityBit, stopBit)
```

Description

Set the parameters of RS485 interface.

The value set by the software before entering TCP/IP control mode will be adopted when this command is not called for setting, and the value set by this command will continue to be maintained after exiting TCP mode.

This command is supported by controller V3.5.7 and above.

Parameter

Parameter	Type	Description
baudRate	int	Baud rate
dataLen	int	**Optional parameter.** Data bit length, currently fixed at 8
parityBit	string	**Optional parameter.** Parity bit, currently fixed at N, indicating no parity
stopBit	int	**Optional parameter.** Stop bit, currently fixed at 1

Return

```
ErrorID,{},SetTerminal485(baudRate, dataLen, parityBit, stopBit);
```

Example

```
SetTerminal485(115200)
```

Set the baud rate of RS485 interface to 115200.

GetTerminal485 (Immediate command)

Command

```
GetTerminal485()
```

Description

Get the parameters of RS485 interface.

This command is supported by controller V3.5.7 and above.

Return

```
ErrorID,{baudRate, dataLen, parityBit, stopBit},GetTerminal485();
```

{baudRate, dataLen, parityBit, stopBit}: baud rate, data bit, parity bit, and stop bit, respectively.

Example

```
GetTerminal485()
```

Get the parameters of RS485 interface.

2.5 Modbus command

ModbusCreate (Immediate command)

Command

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

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
ip	string	IP address of slave station. When IP is not specified, or is specified to 127.0.0.1 or 0.0.0.1, it indicates connecting to the local Modbus slave.
port	int	Port of slave station
slave_id	int	ID of slave station
isRTU	int	**Optional parameter.** null or 0: establish ModbusTCP communication; 1: establish ModbusRTU 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, range: 0 – 4, used when other Modbus commands are called.

Example

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

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

ModbusClose (Immediate command)

Command

```
ModbusClose(index)
```

Description

Disconnect with Modbus slave and release the master.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index

Return

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

Example

```
ModbusClose(0)
```

Release the Modbus master 0.

GetInBits (Immediate command)

Command

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

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the contact register
count	int	Number of values read from contact register (discrete input), range: 1 – 16

Return

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

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

Example

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

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

GetInRegs (Immediate command)

Command

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

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the input register
count	int	Number of values read from input register, range: 1 – 4
valType	string	**Optional parameter.** Data type: null or 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)

Return

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

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

Example

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

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

GetCoils (Immediate command)

Command

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

Description

Read the coil register value from the Modbus slave.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	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 coil register (number of values equals to **count**).

Example

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

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

SetCoils (Immediate command)

Command

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

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the coil register
count	int	Number of values to be written to coil register, range: 1 – 16
valTab	string	Values to be written to coil register (number of values equals to **count**)

Return

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

Example

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

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

GetHoldRegs (Immediate command)

Command

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

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the holding register
count	int	Number of values read from holding register, range: 1 – 4
		Optional parameter. Data type: null or U16: 16-bit unsigned integer (two bytes, occupy one register); U32: 32-bit unsigned integer (four bytes, occupy two registers);

valType	string	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)

Return

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

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

Example

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

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

SetHoldRegs (Immediate command)

Command

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

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	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 holding register (number of values equals to **count**)
valType	string	**Optional parameter.** Data type: null or 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)\

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.

3 Motion Command

The motion-related commands need to be delivered through **port 30003**.

3.1 General description

Coordinate system parameters

The "User" and "Tool" in the optional parameters of motion commands related to the Cartesian coordinate system are used to specify the user and tool coordinate systems of the target point.

- If the "User" and "Tool" parameters are carried, "User" and "Tool" are used to specify the indexes of the calibrated user coordinate system and tool coordinate system, respectively.
- If the "User" and "Tool“ parameters are not carried, the global user and tool coordinate system are used. See the description on User and Tool in Settings command for details.

Speed parameters

The "SpeedJ/SpeedL/AccJ/AccL" in the optional parameters are used to specify the acceleration and speed ratio when the robot arm executes motion commands.

Actual robot acceleration/speed ratio = 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.

When the motion acceleration/speed ratio is not specified through the optional parameters, the global settings are used by default. See the description on SpeedJ/SpeedL/AccJ/AccL in Settings command for details.

Limitations

- TCP motion commands do not support the "CP" parameter in the optional parameter to realize the continuous path function, please use the CP(R) command of port 29999.
- TCP motion commands do not support the "SYNC" parameter in the optional parameter to realize the continuous path function, please use the Sycn() or SyncAll() command.

The meaning of the above parameters will not be described again in the following text.

3.2 Command list

> **NOTE**
> Motion-related commands are all queued commands.

MovJ

Command

```
MovJ(X,Y,Z,Rx,Ry,Rz,User=index,Tool=index,SpeedJ=R,AccJ=R)
```

Description

Move from the current position to the target Cartesian position through joint motion. For joint motion, the trajectory is not linear, and all joints complete the motion simultaneously.

Parameter

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

Return

```
ErrorID,{},MovJ(X,Y,Z,Rx,Ry,Rz);
```

Example

```
MovJ(-500,100,200,150,0,90,AccJ=50)
```

The robot arm moves from the current position to the target Cartesian position {-500,100,200,150,0,90} through joint motion with 50% acceleration.

MovL

Command

```
MovL(X,Y,Z,Rx,Ry,Rz,User=index,Tool=index,SpeedL=R,AccL=R)
```

Description

Move from the current position to the target Cartesian position through linear motion.

Parameter

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

Return

```
ErrorID,{},MovL(X,Y,Z,Rx,Ry,Rz);
```

Example

```
MovL(-500,100,200,150,0,90,SpeedL=60)
```

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

JointMovJ

Command

```
JointMovJ(J1,J2,J3,J4,J5,J6,SpeedJ=R,AccJ=R)
```

Description

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

Parameter

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

Return

```
ErrorID,{},JointMovJ(J1,J2,J3,J4,J5,J6,SpeedJ=R,AccJ=R);
```

Example

```
JointMovJ(0,0,-90,0,90,0,SpeedJ=60,AccJ=50)
```

The robot arm moves from the current position to the target joint position {0,0,-90,0,90,0} through joint motion with 60% speed and 50% acceleration.

MovLIO

Command

```
MovLIO(X,Y,Z,Rx,Ry,Rz,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},User=index
,Tool=index,SpeedL=R,AccL=R)
```

Description

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

Parameter

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

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

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

Return

```
ErrorID,{},MovLIO(X,Y,Z,Rx,Ry,Rz,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status}
,SpeedL=R,AccL=R);
```

Example

```
MovLIO(-500,100,200,150,0,90,{0,50,1,0})
```

The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} through linear motion. When it moves 50% distance away from the starting point, set DO1 to 0.

MovJIO

Command

```
MovJIO(X,Y,Z,Rx,Ry,Rz,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},User=index
,Tool=index,SpeedJ=R,AccJ=R)
```

Description

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

Parameter

Parameter	Type	Description
X	double	X-axis position of the target point, unit: mm
Y	double	Y-axis position of the target point, unit: mm

Z	double	Z-axis position of the target point, unit: mm
Rx	double	Rx-axis position of the target point, unit: °
Ry	double	Ry-axis position of the target point, unit: °
Rz	double	Rz-axis position of the target point, unit: °

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

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

Return

```
ErrorID,{},MovJIO(X,Y,Z,Rx,Ry,Rz,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status}
,SpeedJ=R,AccJ=R);
```

Example

```
MovJIO(-500,100,200,150,0,90,{0,50,1,0})
```

The robot moves from the current position to the Cartesian point {-500,100,200,150,0,90} through joint motion. When it moves 50% distance away from the starting point, set DO1 to 0.

Arc

Command

```
Arc(X1,Y1,Z1,Rx1,Ry1,Rz1,X2,Y2,Z2,Rx2,Ry2,Rz2,User=index,Tool=index,SpeedL=R,AccL=R)
```

Description

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

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

Parameter

Parameter	Type	Description
X1	double	X-axis position of arc through point, unit: mm
Y1	double	Y-axis position of arc through point, unit: mm
Z1	double	Z-axis position of arc through point, unit: mm
Rx1	double	Rx-axis position of arc through point, unit: °
Ry1	double	Ry-axis position of arc through point, unit: °
Rz1	double	Rz-axis position of arc through point, unit: °
X2	double	X-axis position of the target point, unit: mm
Y2	double	Y-axis position of the target point, unit: mm
Z2	double	Z-axis position of the target point, unit: mm
Rx2	double	Rx-axis position of the target point, unit: °
Ry2	double	Ry-axis position of the target point, unit: °
Rz2	double	Rz-axis position of the target point, unit: °

Return

```
ErrorID,{},Arc(X1,Y1,Z1,Rx1,Ry1,Rz1,X2,Y2,Z2,Rx2,Ry2,Rz2,User=index,Tool=index,SpeedL=R,AccL=R
);
```

Example

```
Arc(-350,-200,200,150,0,90,-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.

Circle3

```
Circle3({X1,Y1,Z1,Rx1,Ry1,Rz1},{X2,Y2,Z2,Rx2,Ry2,Rz2},count,User=index,Tool=index)
```

Description

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

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

This command is supported by controller V3.5.5 and above.

Parameter

Parameter	Type	Description
X1	double	X-axis position of P1, unit: mm
Y1	double	Y-axis position of P1, unit: mm
Z1	double	Z-axis position of P1, unit: mm
Rx1	double	Rx-axis position of P1, unit: °
Ry1	double	Ry-axis position of P1, unit: °
Rz1	double	Rz-axis position of P1, unit: °
X2	double	X-axis position of P2, unit: mm
Y2	double	Y-axis position of P2, unit: mm
Z2	double	Z-axis position of P2, unit: mm
Rx2	double	Rx-axis position of P2, unit: °
Ry2	double	Ry-axis position of P2, unit: °
Rz2	double	Rz-axis position of P2, unit: °
count	int	Circles of motion, range: 1 – 999.

Return

```
ErrorID,{},Circle3({X1,Y1,Z1,Rx1,Ry1,Rz1},{X2,Y2,Z2,Rx2,Ry2,Rz2},count,User=0,Tool=0);
```

Example

```
Circle3({-350,-200,200,150,0,90},{-300,-250,200,150,0,90},1,User=0,Tool=0)
```

The robot arm moves a full circle determined by the current point and two specified points, and then return to the current point.

ServoJ

Command

```
ServoJ(J1,J2,J3,J4,J5,J6,t,lookahead_time,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.

The optional parameters are only supported by controller V3.5.5 and above.

NOTICE

- Starting from controller V3.5.5, this command is no longer affected by global speed.
- When the joint angle difference between the target point and the current point is too large and the motion time **t** is too small, the joint motion speed will be too fast, which may cause the servo error or even cause the robot body to shut down.
- Before calling this command, it is recommended to carry out speed planning for the running point, and issue the speed-planned points at a fixed interval t to ensure that the robot can smoothly track the target point.

Parameter

Parameter	Type	Description
J1	double	J1-axis position of the target point, unit: °
J2	double	J2-axis position of the target point, unit: °
J3	double	J3-axis position of the target point, unit: °
J4	double	J4-axis position of the target point, unit: °
J5	double	J5-axis position of the target point, unit: °
J6	double	J6-axis position of the target point, unit: °
t	float	**Optional parameter.** Running time of the point, unit: s, value range: [0.02,3600.0], default value: 0.1
lookahead_time	float	**Optional parameter.** Advanced time, acting in a similar way to the D in PID control. Scalar, no unit, value range: [20.0,100.0], default value: 50.
gain	float	**Optional parameter.** Proportional gain of the target position, acting in a similar way to the P in PID control. Scalar, no unit, value range: [200.0,1000.0], default value: 500.

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

Return

None

Example

```
ServoJ(0,0,-90,0,90,0,t=0.1,lookahead_time=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,lookahead_time=50,gain=500)
```

The J3 axis moves in steps of 1 degree.

ServoP

Command

```
ServoP(X1,Y1,Z1,Rx1,Ry1,Rz1)
```

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.

Parameter

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

Return

None

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=typeValue,User=index,Tool=index)
```

Description

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
axisID	string	Jog motion axis. J1+ means joint 1 is moving in the positive direction, J1- means joint 1 is moving in the negative direction; J2+ means joint 2 is moving in the positive direction, J2- means joint 2 is moving in the negative direction; J3+ means joint 3 is moving in the positive direction, J3- means joint 3 is moving in the negative direction; J4+ means joint 4 is moving in the positive direction, J4- means joint 4 is moving in the negative direction; J5+ means joint 5 is moving in the positive direction, J5- means joint 5 is moving in the negative direction; J6+ means joint 6 is moving in the positive direction, J6- means joint 6 is moving in the negative direction; X+ means joint X is moving in the positive direction, X- means joint X is moving in the negative direction; Y+ means joint Y is moving in the positive direction, Y- means joint Y is moving in the negative direction; Z+ means joint Z is moving in the positive direction, Z- means joint Z is moving in the negative direction; Rx+ means joint Rx is moving in the positive direction, Rx- means joint Rx is moving in the negative direction; Ry+ means joint Ry is moving in the positive direction, Ry- means joint Ry is moving in the negative direction; Rz+ means joint Rz is moving in the positive direction, Rz- means joint Rz is moving in the negative direction.
CoordType	int	**Optional parameter.** Specify the coordinate system of axis (effective only when axisID specifies the axis in Cartesian coordinate system). 0: user coordinate system, 1: tool coordinate system.

Return

```
ErrorID,{},MoveJog(axisID,CoordType=typeValue,User=index,Tool=index);
```

Example

```
MoveJog(j2-)
// Stop jogging
MoveJog()
```

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

StartTrace

Command

```
StartTrace(traceName)
```

Description

Fit the recorded trajectory according to the recorded points (including at least 4 points) in the specified trajectory file. Before calling this command, you need to move the robot to the first point of the trajectory.

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

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
traceName	string	Trajectory file name (with suffix) The trajectory file is stored in /dobot/userdata/project/process/trajectory/.

Return

```
ErrorID,{},StartTrace(traceName);
```

Example

```
// Get the first trajectory fitting point {x,y,z,rx,ry,rz}
GetTraceStartPose(recv_string)
// Run to {x,y,z,rx,ry,rz}
MovJ(x,y,z,rx,ry,rz)
// Start trajectory fitting
StartTrace(recv_string)
```

Get the first point of the trajectory file named "recv_string". Move to this point and start fitting.

StartPath

Command

```
StartPath(traceName,const,cart)
```

Description

Move according to the recorded points (including at least 4 points) in the specified trajectory file to play back the recorded trajectory. Before calling this command, you need to move the robot to the first point of the trajectory.

After delivering the trajectory playback command, you can query the running status of the robot by RobotMode. 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.

This command is supported by controller V3.5.2 and above.

Parameter

Parameter	Type	Description
traceName	string	Trajectory file name (with suffix) The trajectory file is stored in /dobot/userdata/project/process/trajectory/.
const	int	Whether to play back at a constant speed. 1: play back at a constant speed, and pauses will be removed from the trajectory. 0: play back at the original speed.
cart	int	The type of playback path. 1: play back as Cartesian path. 0: play back as joint path.

Return

```
ErrorID,{},StartPath(traceName,const,cart);
```

Example

```
// Get the first trajectory joint point {j1,j2,j3,j4,j5,j6}
GetPathStartPose(recv_string)
// Run to {j1,j2,j3,j4,j5,j6}
JointMovJ(j1,j2,j3,j4,j5,j6)
// Start trajectory playback
StartPath(recv_string,0,1)
```

Get the first joint point of the trajectory file named "recv_string". Move to this point, and start playback as Cartesian path at the original speed.

Sync

Command

```
Sync()
```

Description

Block the program to execute queue commands and does not return until all queue commands have been executed.

Return

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

Example

```
MovJ(x,y,z,rx,ry,rz)
Sync()
RobotMode()
```

Get the current status of the robot after the robot moves to {x,y,z,rx,ry,rz}.

RelMovJTool

Command

```
RelMovJTool(offsetX, offsetY,offsetZ, offsetRx,offsetRy,offsetRz, Tool,SpeedJ=R, AccJ=R,Tool=I
ndex)
```

Description

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

This command is supported by controller V3.5.2 and above.

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: °
Tool	int	Select the index of the calibrated tool coordinate system

Return

```
ErrorID,{},RelMovJTool(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz,Tool,SpeedJ=R, AccJ=R,Tool=Index);
```

Example

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

The robot arm moves relatively in the joint mode along Tool coordinate system 0, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelMovLTool

Command

```
RelMovLTool(offsetX, offsetY,offsetZ, offsetRx,offsetRy,offsetRz, Tool,SpeedL=R, AccL=R,Tool=Index)
```

Description

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

This command is supported by controller V3.5.2 and above.

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: °
Tool	int	Select the index of the calibrated tool coordinate system

Return

```
ErrorID,{},RelMovLTool(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz,Tool,SpeedL=R, AccL=R,User=Index);
```

Example

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

The robot arm moves relatively in the linear mode along Tool coordinate system 0, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelMovJUser

Command

```
RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz, User,SpeedJ=R, AccJ=R,User=Index)
```

Description

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

This command is supported by controller V3.5.2 and above.

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: °
User	int	Select the index of the calibrated user coordinate system

Return

```
ErrorID,{},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz, User,SpeedJ=R, AccJ=R,User=Index);
```

Example

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

The robot arm moves relatively in the joint mode along User coordinate system 0, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelMovLUser

Command

```
RelMovLUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz, User,SpeedL=R, AccL=R,User=Index)
```

Description

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

This command is supported by controller V3.5.2 and above.

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: °
User	int	Select the index of the calibrated user coordinate system

Return

```
ErrorID,{},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetRx,OffsetRy,OffsetRz, User,SpeedL=R, AccL=R,User=Index);
```

Example

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

The robot arm moves relatively in the linear mode along User coordinate system 0, and displaces 10mm in X-axis, Y-axis and Z-axis respectively.

RelJointMovJ

Command

```
RelJointMovJ(Offset1,Offset2,Offset3,Offset4,Offset5,Offset6,SpeedJ=R, AccJ=R)
```

Description

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

This command is supported by controller V3.5.2 and above.

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: °

Return

```
ErrorID,{},RelJointMovJ(Offset1,Offset2,Offset3,Offset4,Offset5,Offset6,SpeedJ=R, AccJ=R);
```

Example

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

Displace 10° in J1, J2 and J3 respectively.

4 Real-time Feedback

The controller feeds back robot status through **port 30004, 30005 and 30006**. Each packet received through the real-time feedback port has 1440 bytes, which are arranged in a standard format, as shown below.

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

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

Meaning	Data type	Number of values	Size in bytes	Byte position value	Notes
MessageSize	unsigned short	1	2	0000 – 0001	Total message length in bytes
N/A	unsigned short	3	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 (in ms)
N/A	uint64	1	8	0040 – 0047	Reserved
TestValue	uint64	1	8	0048 – 0055	Memory structure test standard value 0x0123 4567 89AB CDEF
N/A	double	1	8	0056 – 0063	Reserved
SpeedScaling	double	1	8	0064 – 0071	Speed rate
N/A	double	1	8	0072 –	Reserved

N/A	double	1	8	0079	Reserved
VMain	double	1	8	0080 – 0087	Control board voltage (Not supported by CCBOX)
VRobot	double	1	8	0088 – 0095	Robot voltage (Not supported by CCBOX)
IRobot	double	1	8	0096 – 0103	Robot current (Not supported by CCBOX)
N/A	double	1	8	0104 – 0111	Reserved
N/A	double	1	8	0112 – 0119	Reserved
N/A	double	3	24	0120 – 0143	Reserved
N/A	double	3	24	0144 – 0167	Reserved
N/A	double	3	24	0168 – 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 sensor force (Six-dimensional force sensor needs to be installed)
ToolVectorActual	double	6	48	0624 – 0671	TCP actual Cartesian coordinates

TCPSpeedActual	double	6	48	0672 – 0719	TCP actual speed in Cartesian coordinate system
TCPForce	double	6	48	0720 – 0767	TCP force value (calculated by joint current)
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	960 – 1007	Joint voltage
HandType	char	4	4	1008 – 1011	Hand system. See SetArmOrientation.
User	char	1	1	1012	User coordinate system
Tool	char	1	1	1013	Tool coordinate system
RunQueuedCmd	char	1	1	1014	Algorithm queue running signal
PauseCmdFlag	char	1	1	1015	Algorithm queue pause signal
VelocityRatio	char	1	1	1016	Joint speed ratio (0 – 100)
AccelerationRatio	char	1	1	1017	Joint acceleration ratio (0 – 100)
JerkRatio	char	1	1	1018	Joint jerk ratio (0 – 100)
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)
XYZJerkRatio	char	1	1	1023	Cartesian position jerk ratio (0 – 100)
RJerkRatio	char	1	1	1024	Cartesian posture jerk ratio (0 – 100)
BrakeStatus	char	1	1	1025	Robot brake status. See BrakeStatus description
EnableStatus	char	1	1	1026	Robot enabling status
DragStatus	char	1	1	1027	Robot drag status
RunningStatus	char	1	1	1028	Robot drag status
ErrorStatus	char	1	1	1029	Robot alarm status
JogStatusCR	char	1	1	1030	Robot jog status
RobotType	char	1	1	1031	Robot type. See RobotType description
DragButtonSignal	char	1	1	1032	End button drag signal
EnableButtonSignal	char	1	1	1033	End button enabling signal (Not supported by Nova series)
RecordButtonSignal	char	1	1	1034	End button record signal (Not supported by Nova series)
ReappearButtonSignal	char	1	1	1035	End button playback signal (Not supported by Nova series)
JawButtonSignal	char	1	1	1036	End button gripper control signal (Not supported by Nova series)
SixForceOnline	char	1	1	1037	Six-dimensional force online status (Six-dimensional force sensor needs to be installed)
N/A	char	1	82	1038 – 1119	Reserved
MActual[6]	double	6	48	1120 – 1167	Actual torque of six joints

Load	double	1	8	1168 – 1175	End load (kg)
CenterX	double	1	8	1176 – 1183	End load X-directional eccentric distance (mm)
CenterY	double	1	8	1184 – 1191	End load Y-directional eccentric distance (mm)
CenterZ	double	1	8	1192 – 1199	End load Z-directional eccentric distance (mm)
User[6]	double	6	48	1200 – 1247	User coordinates
Tool[6]	double	6	48	1248 – 1295	Tool coordinates
TraceIndex	double	1	8	1296 – 1303	Trajectory playback running index
SixForceValue[6]	double	6	48	1304 – 1351	Original value of current six-dimentional force (Six-dimensional force sensor needs to be installed)
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
N/A	char	1	24	1416 – 1440	Reserved
TOTAL			1440		1440byte package

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 that the corresponding terminal has signal, and 0 indicates that the corresponding terminal has no signal or no corresponding terminal.

For example, the first byte is 0x01 (00000001), the second byte is 0x02 (00000010), and the remaining bytes are all 0, which means DI1 and DI10 are 1, and the other terminals are 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:

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

Example:

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

RobotType description

Value	Model
3	CR3
31	CR3L
5	CR5
7	CR7
10	CR10
12	CR12
16	CR16
1	MG400
2	M1 Pro
101	Nova 2
103	Nova 5
113	CR3V2
115	CR5V2
117	CR7V2
120	CR10V2
122	CR12V2
126	CR16V2

5 Error Code

Error code	Description	Note
0	No error	Deliver successfully
-1	Fail to get	Failed to receive or execute
...	...	...
-10000	Command error	The command does not exist
-20000	Parameter number error	The number of parameters in the command is incorrect
-30001	The type of the first parameter is incorrect	-30000 indicates that the parameter type is incorrect. The last bit 1 indicates that the type of the first parameter is incorrect
-30002	The type of the second parameter is incorrect	-30000 indicates that the parameter type is incorrect. The last bit 2 indicates that the type of the second parameter is incorrect
...	...	...
-40001	The range of the first parameter is incorrect	-40000 indicates that the parameter range is incorrect. The last bit 1 indicates that the range of the first parameter is incorrect
-40002	The range of the second parameter is incorrect	-40000 indicates that the parameter range is incorrect. The last bit 2 indicates that the range of the second parameter is incorrect
...	...	...