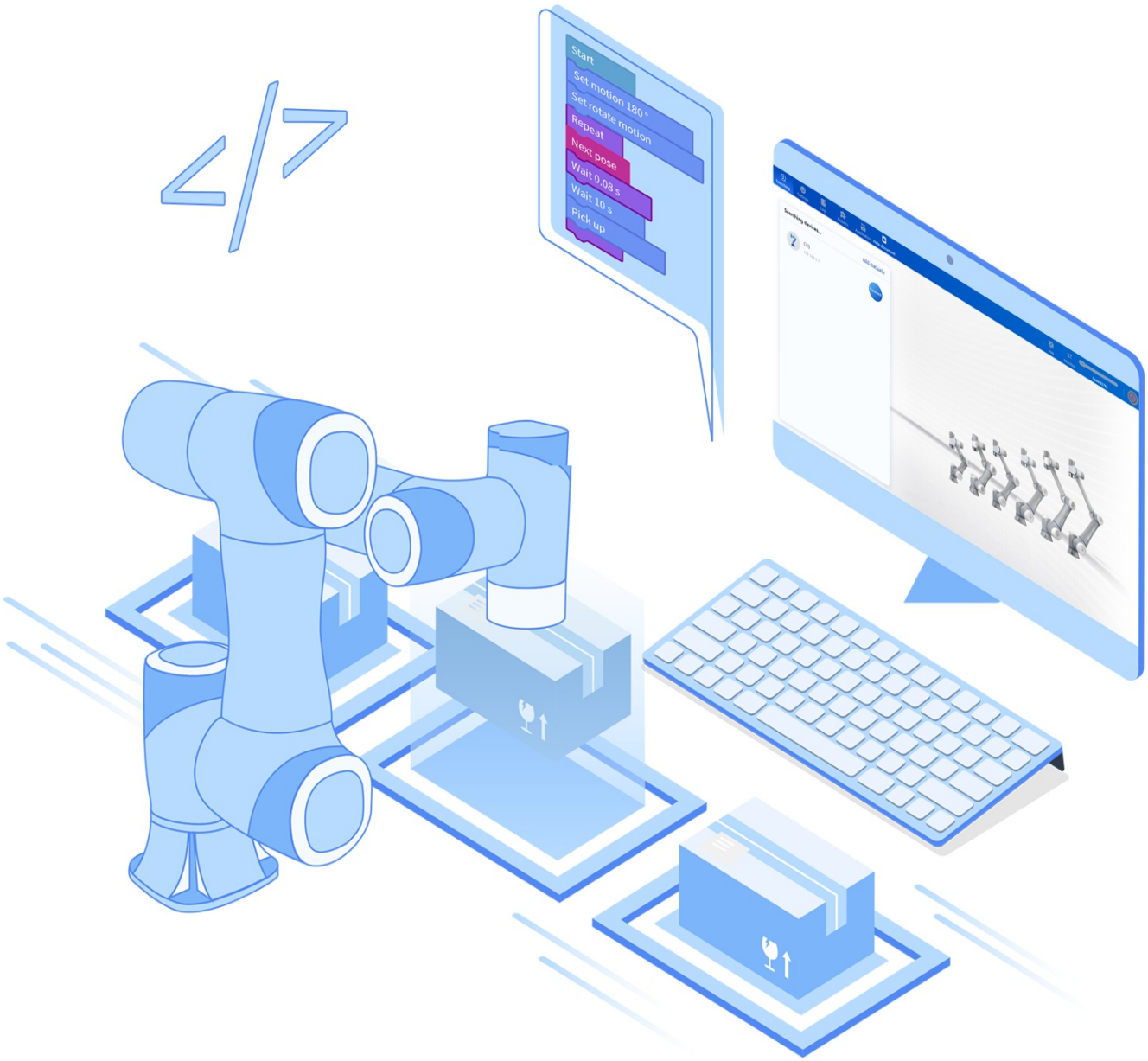
DOBOT

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 four-axis industrial robot controller, 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
2024/04/19	The Third release, corresponding to controller V1.6.0.0. Explanation of the new value 4 for RobotType in the real-time feedback message.
2024/03/07	The Second release, corresponding to controller V1.6.0.0. Add SetUser, CalcUser, SetTool, CalcTool, PalletCreate, and GetPalletPose commands.
2024/01/12	The first release, corresponding to controller V1.5.9.0

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.

To support TCP/IP protocol, the controller version should be V1.5.5.0 or above.

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 V1.5.9 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)
```

Return:

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

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

Send:

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

Return:

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

-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(-100,100,200,150)  // 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)  // Queue command
Sync()
RobotMode()  // Immediate command
```

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

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

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.).

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 contained 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(0.5)
```

Enable the robot and set the load weight to 0.5kg.

```
EnableRobot(0.5,0,0,5.5)
```

Enable the robot. Set the load weight to 1.5kg and Z-axis eccentric distance to 5.5mm.

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 restart the motion queue via the Continue command.

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.

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.

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.

Return

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

Example

```
StopDrag()
```

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

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.

Return

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

Example

```
EmergencyStop()
```

Stop the robot arm in an emergency.

Wait (Queue command)

Command

```
wait(time)
```

Description

The command queue delays for a specified time.

This command is supported by controller V1.5.9 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 global speed ratio set by this command only takes effect in the current TCP/IP control mode. When it is not set, the value set by the software before entering TCP/IP control mode will be adopted.

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 global user coordinate system set by this command only takes effect in the current TCP/IP control mode. When it is not set, the default global user coordinate system is the coordinate system set by the software before entering TCP/IP control mode.

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 only takes effect in the current TCP/IP control mode. When it is not set, the default global tool coordinate system is the coordinate system set by the software before entering TCP/IP control mode.

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.

SetPayLoad (Queue command)

Command

```
SetPayLoad(weight,inertia)
```

Description

Set the load of the robot arm.

Parameter

Parameter	Type	Description
weight	float	Load weight. The value range should not exceed the load range of corresponding robot models. Unit: kg.
interia	float	**Optional parameter.** Load inertia, unit: kgm².

Return

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

Example

```
SetPayLoad(0.3)
```

Set the load weight to 0.3kg.

AccJ (Queue command)

Command

```
AccJ(R)
```

Description

Set acceleration ratio of joint motion.

The acceleration ratio set in this command is only valid in the current TCP/IP control mode, and it is 100 by default when not set.

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 acceleration ratio set in this command is only valid in the current TCP/IP control mode, and it is 100 by default when not set.

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 speed ratio set in this command is only valid in the current TCP/IP control mode, and it is 100 by default when not set.

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 speed ratio set in this command is only valid in the current TCP/IP control mode, and it is 100 by default when not set.

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%.

Arch (Queue command)

Command

```
Arch(Index)
```

Description

Set the index of global jump parameter for the Jump motion. You can specify the jump parameters when calling the Jump motion command. If not specified, the global jump parameter index is used.

The jump parameter index set in this command is only valid in the current TCP/IP control mode, and it is 0 by default when not set.

Parameter

Parameter	Type	Description

Index	int	Jump parameter index, which needs to be set by software before it can be selected here.

Return

```
ErrorID,{},Arch(Index);
```

Example

```
Arch(1)
```

Set the index of global jump parameter to 1.

CP (Queue command)

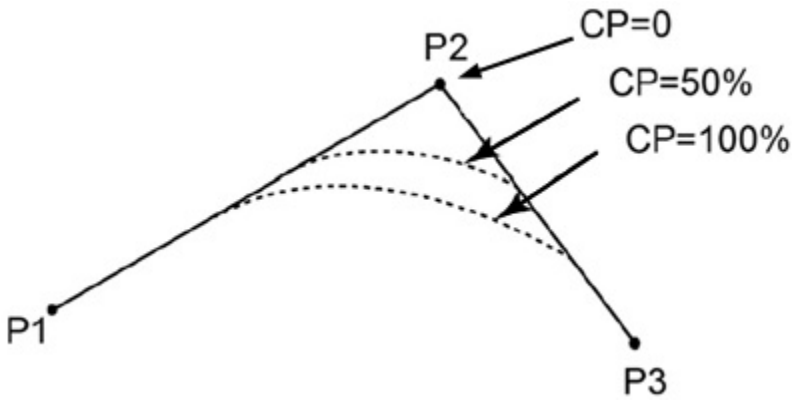
Command

```
CP(R)
```

Description

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

The CP ratio set in this command is only valid in the current TCP/IP control mode, and it is 0 by default when not set.



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)
```

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 hand system set in this command is only valid in the current TCP/IP control mode. If it is not set, it means that no hand system is specified.

This command is for M1 Pro.

Parameter

Parameter	Type	Description
LorR	int	0: left-hand system, 1: right-hand system

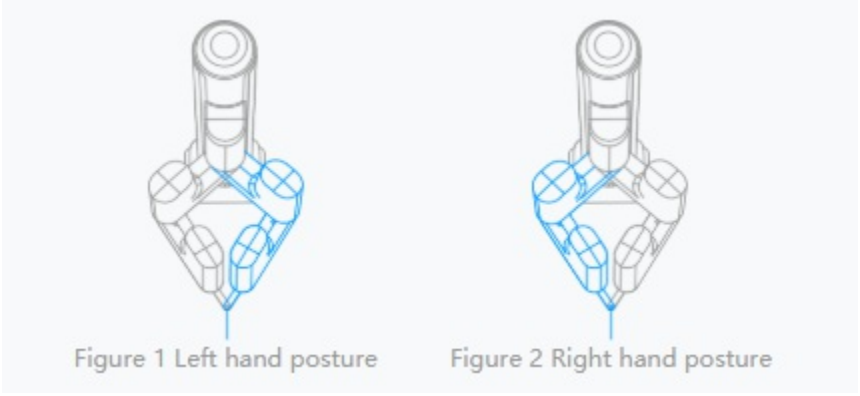


Figure 1 Left hand posture

Figure 2 Right hand posture

Return

```
ErrorID,{},SetArmOrientation(LorR);
```

Example

```
SetArmOrientation(1)
```

Set M1 Pro to 0.

SetCollisionLevel (Queue command)

Command

```
SetCollisionLevel(level)
```

Description

Set the collision detection level.

The collision detection level set by this command only takes effect in the current TCP/IP control mode. When it is not set, the value set by the software before entering TCP/IP control mode will be adopted.

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.

SetUser (Immediate command)

Command:

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

Description:

Modify the specified user coordinate system.

This command is supported by controller V1.6.0 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	-	User coordinate system after modification, format: {x, y, z, r}), which is recommended to obtain through CalcUser command.

Return

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

Example:

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

Modify user coordinate system 1 to X=10, Y=10, Z=10, R=0.

CalcUser (Immediate command)

Command:

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

Description:

Calculate the user coordinate system.

This command is supported by controller V1.6.0 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	-	User coordinate system offset, format: {x, y, z, r}.

Return:

```
ErrorID,{x,y,z,r},CalcUser(index,matrix_direction,table);
```

{x, y, z, r} is the user coordinate system after calculation.

Example 1:

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

Calculate: User coordinate system 1 left-multiplies {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 r=10, and the new coordinate system is newUser.

Example 2:

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

Calculate: User coordinate system 1 right-multiplies {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 user coordinate system and rotates r=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 V1.6.0 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	-	Tool coordinate system after modification, format: {x, y, z, r}, which is recommended to obtain through CalcTool command.

Return

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

Example:

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

Modify tool coordinate system 1 to X=10, Y=10, Z=10, R=0.

CalcTool (Immediate command)

Command:

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

Description:

Calculate the tool coordinate system.

This command is supported by controller V1.6.0 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	-	Tool coordinate system offset, format: {x, y, z, r}.

Return:

```
ErrorID,{x,y,z,r},CalcTool(index,matrix_direction,table);
```

{x, y, z, r} is the tool coordinate system after calculation.

Example 1:

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

Calculate: Tool coordinate system 1 left-multiplies {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 r=10, and the new coordinate system is newTool.

Example 2:

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

Calculate: Tool coordinate system 1 right-multiplies {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 tool coordinate system and rotates r=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 project running, no alarm)
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.

GetAngle (Immediate command)

Command

```
GetAngle()
```

Description

Get the joint coordinates of current posture.

Return

```
ErrorID,{J1,J2,J3,J4},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,R},GetPose();
```

{X,Y,Z,R} 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.

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. For other alarm definition, see the controller alarm document "alarm_controller.json".
- The last four [id] represent the alarm information of four 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.

PositiveSolution (Immediate command)

Command

```
PositiveSolution(J1,J2,J3,J4,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: °
User	int	Index of the calibrated user coordinate system
Tool	int	Index of the calibrated tool coordinate system

Return

```
ErrorID,{x,y,z,r},PositiveSolution(J1,J2,J3,J4,User,Tool);
```

{x,y,z,r} refers to the Cartesian coordinates of the point.

Example

```
PositiveSolution(0,0,-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}.

InverseSolution (Immediate command)

Command

```
InverseSolution(X,Y,Z,R,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 rotation 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
R	double	R-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},InverseSolution(X,Y,Z,R,User,Tool,isJointNear,JointNear);
```

{J1,J2,J3,J4} refers to the joint coordinates of the point.

Example

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

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}. 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,-180.000000,0,0,1,{0,0,-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}. Calculate the joint coordinates and select the nearest solution of {0,0,-90,0}.

PalletCreate (Immediate command)

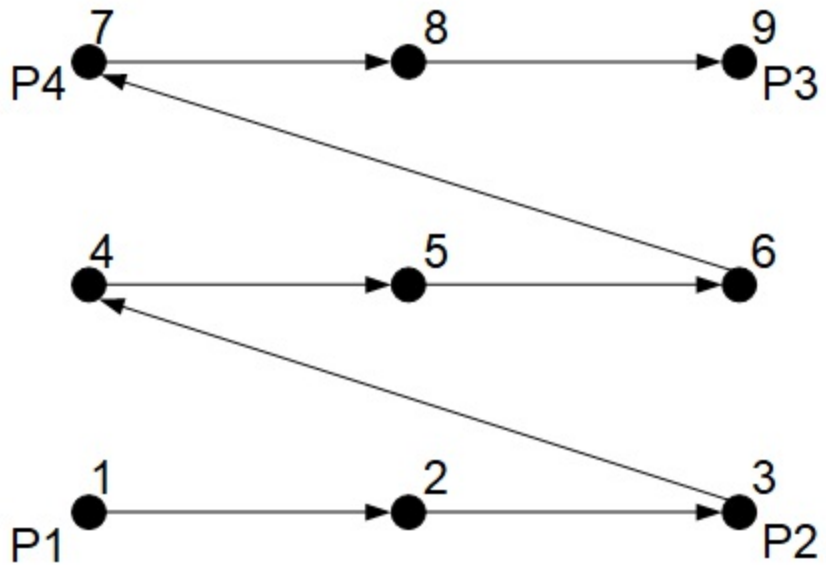
Command

```
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 V1.6.0 and above.

Parameter

Parameter	Type	Description
P1	array[double]	Cartesian coordinates of P1, in the format {X,Y,Z,R}
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},{156,-568,337,175.5755},{156,-468,337,175.5755},{56,-468,337,175.5755},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 V1.6.0 and above.

Parameter

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

Return

```
ErrorID,{X,Y,Z,R},GetPalletPose(Palletname,index);
```

{X,Y,Z,R} 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.

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.

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.

2.5 Modbus command

ModbusCreate (Immediate command)

Command

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

Description

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

Parameter

Parameter	Type	Description
ip	string	IP address of slave station.
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.

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.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the contact register
count	int	Number of values read from the contact register, range: [1, 16]

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

```
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.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the input register
count	int	Number of values read from the 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.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the coil register
count	int	Number of values read from the 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.

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 the coil register, range: [1, 16]
valTab	string	Values to be written to the holding 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.

Parameter

Parameter	Type	Description
index	int	Master index
addr	int	Starting address of the holding register
count	int	Number of values read from the holding 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},GetHoldRegs(index,addr, count,valType);
```

{value1,value2,...,valuen}: values read from the coil 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.

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.

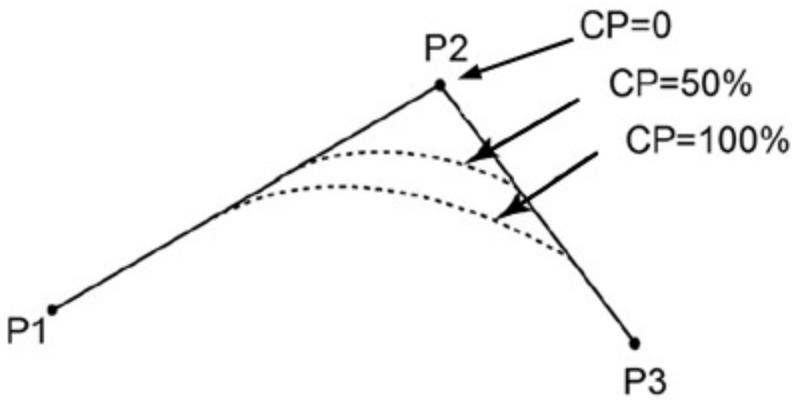
Actual robot acceleration/speed ratio = ratio set in motion command × value in Playback settings × global speed ratio.

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.

Continuous path parameters

CP in the optional parameter is used to specify the continuous path (CP) ratio between the current motion command and the next motion command, 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.



Limitations

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,R,User=index,Tool=index,SpeedJ=R,AccJ=R,CP=R)
```

Description

Move from the current position to the target Cartesian position through joint motion. The trajectory of joint motion is not linear, and all joints complete the motion at the same time.

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
R	double	R-axis position of the target point, unit: °

Return

```
ErrorID,{},MovJ(X,Y,Z,R);
```

Example

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

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

MovL

Command

```
MovL(X,Y,Z,R,User=index,Tool=index,SpeedL=R,AccL=R,CP=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
R	double	R-axis position of the target point, unit: °

Return

```
ErrorID,{},MovL(X,Y,Z,R);
```

Example

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

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

JointMovJ

Command

```
JointMovJ(J1,J2,J3,J4,SpeedJ=R,AccJ=R,CP=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: °

Return

```
ErrorID,{},JointMovJ(J1,J2,J3,J4);
```

Example

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

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

MovLIO

Command

```
MovLIO(X,Y,Z,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},User=index,Tool=i
ndex,SpeedL=R,AccL=R,CP=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
R	double	R-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,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},SpeedL=R,AccL=R);
```

Example

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

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

MovJIO

Command

```
MovJIO(X,Y,Z,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},User=index,Tool=index,SpeedJ=R,AccJ=R,CP=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
R	double	R-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.

Distance	int	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,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status});
```

Example

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

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

Arc

Command

```
Arc(X1,Y1,Z1,R1,X2,Y2,Z2,R2,User=index,Tool=index,SpeedL=R,AccL=R,CP=R)
```

Description

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

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

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
R1	double	R-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
R2	double	R-axis position of the target point, unit: °

Return

```
ErrorID,{},Arc(X1,Y1,Z1,R1,X2,Y2,Z2,R2);
```

Example

```
Arc(-350,-200,200,150,-300,-250,200,150)
```

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

Circle

```
Circle(count,{X1,Y1,Z1,R1},{X2,Y2,Z2,R2})
```

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.

Parameter

Parameter	Type	Description
count	int	The circles of Circle motion, an integer ≥ 1.
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
R1	double	R-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
R2	double	R-axis position of P2, unit: °

Return

```
ErrorID,{},Circle(count,{X1,Y1,Z1,R1},{X2,Y2,Z2,R2});
```

Example

```
Circle(1,{-350,-200,200,150},{-300,-250,200,150})
```

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

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 V1.5.6 and above.

Parameter

Parameter	Type	Description
axisID	string	Jog the axis. 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; X+ means axis X is moving in the positive direction and X- means axis X is moving in the negative direction; Y+ means axis Y is moving in the positive direction and Y- means axis Y is moving in the negative direction; Z+ means axis Z is moving in the positive direction and Z- means axis Z is moving in the negative direction; R+ means axis R is moving in the positive direction and R- means axis R 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.

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,r)
Sync()
RobotMode()
```

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

RelMovJUser

Command

```
RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetR, User,SpeedJ=R, AccJ=R,Tool=Index,CP=R)
```

Description

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

This command is supported by controller V1.5.6 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
offsetR	double	R-axis offset, unit: °
User	int	Select the index of the calibrated user coordinate system

Return

```
ErrorID,{},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetR,User,SpeedJ=R,AccJ=R,Tool=Index);
```

Example

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

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,OffsetR,User,SpeedL=R,AccL=R,Tool=Index,CP=R)
```

Description

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

This command is supported by controller V1.5.6 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
offsetR	double	R-axis offset, unit: °
User	int	Select the index of the calibrated user coordinate system

Return

```
ErrorID,{},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetR,User);
```

Example

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

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,SpeedJ=R,AccJ=R,CP=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 V1.5.6 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: °

Return

```
ErrorID,{},RelJointMovJ(Offset1,Offset2,Offset3,Offset4);
```

Example

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

Displace 10° in J1, J2 and J3 respectively.

MovJExt

Command

```
MovJExt(Angle|Distance,SpeedE=50,AccE=50,Sync=1)
```

Description

Control the sliding rail (extended axis) to move to the target angle or distance.

Parameter

Parameter	Type	Description
Angle or Distance	float	Target angle or distance of motion. The meaning of this parameter depends on the type of motion (joint/linear) set in **Advanced Settings** of the **Aux Joint** process. Unit: ° (type: joint) or mm (type: linear).
SpeedE	int	**Optional parameter.** Speed ratio, range: 1 – 100. 100 by default.
AccE	int	**Optional parameter.** Acceleration ratio, range: 1 – 100. 100 by default.
Sync	int	**Optional parameter.** Synchronization flag, range: 0 or 1 (0 by default). SYNC=0: asynchronous execution, which means returning immediately after being called, regardless of the execution process. SYNC=1: synchronous execution. which means not returning after being called until the command is executed completely.

Return

```
ErrorID,{},MovJExt(Angle|Distance,SpeedE=50,AccE=50,Sync=1);
```

Example

```
MovJExt(300)
```

If the motion type of the extended axis is mm, it indicates that the extended axis moves to 300mm.

SyncAll

Command

```
SyncAll()
```

Description

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

This command is mainly used in scenarios with extended axes: the

extended axis and the robot arm move independently, and Sync command will be returned after the last command in the queue (assuming the robot arm motion command) is executed. At this time, the extended axis motion command in front of the queue may not have been executed (and vice versa). If you want to ensure that all commands in the queue are executed, you can use SyncAll command.

Return

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

Example

```
MovJ(x1,y1,z1,r1)
MovJExt(distance)
MovJ(x2,y2,z2,r2)
SyncAll()
RobotMode()
```

Get the current status of the robot after the robot and the extended axis have completed the motion.

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	Description
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
				0072 –	

				0079	
VMain	double	1	8	0080 – 0087	Control board voltage
VRobot	double	1	8	0088 – 0095	Robot voltage
IRobot	double	1	8	0096 – 0103	Robot current
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	Reserved
ToolVectorActual	double	6	48	0624 – 0671	TCP actual Cartesian coordinates
TCPSpeedActual	double	6	48	0672 – 0719	TCP actual speed in Cartesian coordinate system
				0720 –	TCP force value

TCPForce	double	6	48	0720 – 0767	(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	Reserved
EnableButtonSignal	char	1	1	1033	Reserved
RecordButtonSignal	char	1	1	1034	Reserved
ReappearButtonSignal	char	1	1	1035	Reserved
JawButtonSignal	char	1	1	1036	Reserved
SixForceOnline	char	1	1	1037	Reserved
N/A	char	1	82	1038 – 1119	Reserved
MActual[6]	double	6	48	1120 – 1167	Actual torque of four 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	Reserved
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 0x00 (00000000), and the remaining bytes are all 0, which means DI1 is 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	Reserved	Reserved

Example:

0x03 (00000100): J4 brake is switched on

RobotType description

Value	Model
1	MG400
2	M1 Pro
4	M1 Pro(with RS485 function)

5 Error Code

Error code	Notes	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
...	...	...