# Dobot Magician API Description

Issue: V1.2.2

Date: 2018-03-26

Shenzhen Yuejiang Technology Co., Ltd

## Disclaimer

To the maximum extent permitted by applicable law, the products described (including its hardware, software and firmware, etc.) in this document are provided **AS IS**, which may have flaws, errors or faults. Yuejiang makes no warranties of any kind, express or implied, including but not limited to, merchantability, satisfaction of quality, fitness for a particular purpose and non-infringement of third party rights. In no event will Yuejiang be liable for any special, incidental, consequential or indirect damages resulting from the use of our products and documents.

Before using our product, please thoroughly read and understand the contents of this document and related technical documents that are published online, to ensure that the robotic arm is used on the premise of fully understanding the robotic arm and related knowledge. Please use this document with technical guidance from professionals. Even if follow this document or any other related instructions, Damages or losses will be happen in the using process, Dobot shall not be considered as a guarantee regarding to all security information contained in this document.

The user has the responsibility to make sure following the relevant practical laws and regulations of the country, in order that there is no significant danger in the use of the robotic arm.

Shenzhen Yuejiang Technology Co., Ltd

Address: 4F, A8, Tanglang Industrial Area, Taoyuan Street, Nanshan District, Shenzhen, PRC

Website: www.dobot.cc

# Preface

**Purpose**

The document is aiming to have a detailed description of Dobot API and general process of Dobot API development program.

**Intended Audience**

This document is intended for:

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

**Change History**

| Date | Change Description |
| --- | --- |
| 2018/03/26 | The first release |

**Symbol Conventions**

The symbols that may be founded in this document are defined as follows.

| Symbol | Description |
| --- | --- |
| ⚠DANGER | Indicates a hazard with a high level of risk which, if not avoided, could result in death or serious injury |
| ⚠WARNING | Indicates a hazard with a medium level or low level of risk which, if not avoided, could result in minor or moderate injury, robotic arm damage |
| ⚠NOTICE | Indicates a potentially hazardous situation which, if not avoided, can result in robotic arm damage, data loss, or unanticipated result |
| 🕮NOTE | Provides additional information to emphasize or supplement important points in the main text |

 

# Contents



 

 

# 1. API Interface Description

## 1.1 Dobot Commands

Dobot controller supports two kind of commands: Immidiate command and queue command:

- Immidiate command: Dobot controller will process the command once received regardless of whether there is the rest commands processing or not in the current controller;
- Queue command: When Dobot controller receives a command, this command will be pressed into the controller internal command queue. Dobot controller will execute commands in the order in which the commands were pressed into the queue.

For more detailed information about Dobot commands, please refer to *Dobot protocol*.

## 1.2 Command Timeout

### 1.2.1 Setting Command Timeout

As described in *1.1 Dobot Commands*, all commands sent to Dobot controller have returns. When a command error occurs due to a communication link interference or any other factors, this command cannot be recognized by the controller and will have no return. Therefore, each command issued to the controller has a timeout period. The timeout period can be set by the following API.

Table 1.1 Set timeout

| | |
|---|---|
| Prototype | void SetCmdTimeout(uint32_t cmdTimeout) |
| Description | Set command timeout. If a command is required to return data within a given time after issuing it, please call this API to set timeout to check whether the return of this command is overtime |
| Parameter | cmdTimeout: Command timeout. Unit: ms |
| Return | DobotCommunicate_NoError:There is no error |

## 1.3 Connect/Disconnect

### 1.3.1 Searching for the Dobot

Table 1.2 Search for the Dobot

| | |
|---|---|
| Prototype | int SearchDobot(char *dobotList, uint32_t maxLen) |
| Description | Search for Dobot, DLL will store the information of Dobot that has been searched for and use ConnectDobot to connect the searched Dobot |
| Parameter | dobotList: String pointer, DLL will write serial port/UDP searched into dobotList. For example, a specific dobotList is "**COM1 COM3 COM6 192.168.0.5**", different serial port or IP address should be separated by the space |

 

| | |
|---|---|
| | maxLen: Maximum String length, to avoid memory overflow |
| Return | The number of Dobot |

### 1.3.2 Connecting to the Dobot

Table 1.3 Connect to the Dobot

| | |
|---|---|
| Prototype | int ConnectDobot(const char *portName, uint32_t baudrate, char *fwType, char *version) |
| Description | Connecing to the Dobot. In this process, portName can be obtained from **dobotList** in the **SearchDobot(char *dobotList, uint32_t maxLen)** API.<br>If portName is empty, and **ConnectDobot** is called directly, DLL will connect the random searched Dobot automatically |
| Parameter | portName: Dobot port. As for the serial port, portName is **COM3**; While for UDP, portName is **192.168.0.5**<br>baudrate: Baud rates<br>fwType: Firmware type. Dobot or Marlin<br>version: Version |
| Return | DobotConnect_NoError： The connection is successful<br>DobotConnect_NotFound：Dobot interface was not found<br>DobotConnect_Occupied：Dobot interface is occupied or unavailable |

⚠NOTICE

In order to make the API recognize the Dobot controller interface, please install the required driver in advance. For more details, please refer to *Dobot User Guide*.

### 1.3.3 Disconnecting the Dobot

Table 1.4 Disconnect the Dobot

| | |
|---|---|
| Prototype | void DisconnectDobot(void) |
| Description | Disconnect the Dobot |
| Parameter | None |
| Return | DobotConnect_NoError :There is no error |

### 1.3.4 Demo: Connection Example

Program 1.1 Connection Example

```
#include "DobotDll.h"
```

```
int split(char **dst, char* str, const char* spl)
{
    int n = 0;
    char *result = NULL;
    result = strtok(str, spl);
    while( result != NULL )
    {
        strcpy(dst[n++], result);
        result = strtok(NULL, spl);
    }
    return n;
}

int main(void)
{
    int maxDevCount = 100;
    int maxDevLen = 20;

    char *devsChr = new char[maxDevCount * maxDevLen]();
    char **devsList = new char*[maxDevCount]();
    for(int i=0; i<maxDevCount; i++)
        devsList[i] = new char[maxDevLen]();

    SearchDobot(devsChr, 1024);
    split(devsList, devsChr, " ");
    ConnectDobot(devsList[0], 115200, NULL, NULL, NULL);

    // Control Dobot

    DisconnectDobot();

    delete[] devsChr;
    for(int i=0; i<maxDevCount; i++)
        delete[] devsList[i];
    delete[] devsList;
}
```

## 1.4 Command queue controlling

There is a queue in Dobot controller to store and execute commands in order. You can also start and stop a command in the command queue to realize asynchronous operations.

⚠NOTICE

Only the API where the **isQueued** parameter is set to **1** can be added to the command queue.

### 1.4.1 Starting Command in Command queue

Table 1.5 Start command in command queue

| | |
|---|---|
| Prototype | int SetQueuedCmdStartExec(void) |
| Description | The Dobot controller starts to query command queue periodically. If there are commands in queue, Dobot controller will take them out and execute the commands in order, indicating that Dobot executes commands one after another |
| Parameter | None |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.4.2 Stopping Command in Command queue

Table 1.6 Stop command in command queue

| | |
|---|---|
| Prototype | int SetQueuedCmdStopExec(void) |
| Description | The Dobot controller stops to query command queue and execute command. However, if one command is being executed when this API is called, this command will continue to be executed. |
| Parameter | None |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.4.3 Stopping Command in Command queue Forcedly

Table 1.7 Stop command in command queue forcedly

| | |
|---|---|
| Prototype | int SetQueuedCmdForceStopExec(void) |

| | |
|---|---|
| Description | Dobot controller stops to query command queue and execute command. If one command is being executed when this API is called, this command will be stopped forcedly. |
| Parameter | None |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout (aka error) |

### 1.4.4 Demo: Processing PTP Command and Control Queue Synchronously

For details about PTP, please refer to *1.12 PTP*.

Program 1.2 Process PTP command and control queue synchronously

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd      cmd;

    cmd.ptpMode = 0;
    cmd.x           = 200;
    cmd.y           = 0;
    cmd.z           = 0;
    cmd.r           = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetQueuedCmdStopExec();

    DisconnectDobot();
}
```

### 1.4.5 Demo: Processing PTP Command and Controlling Queue Asynchronously

Program 1.3 Process PTP command and control queue asynchronously

```
#include "DobotDll.h"


// Main thread
int main(void)
{
    ConnectDobot(NULL, 115200, NULL, NULL, NULL);
}


int onButtonClick()
{
    static bool flag = True;
    if (flag)
        SetQueuedCmdStartExec();
    else
        SetQueuedCmdStopExec();
}


// Child thread
int thread(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd   cmd;


    cmd.ptpMode = 0;
    cmd.x       = 200;
    cmd.y       = 0;
    cmd.z       = 0;
    cmd.r       = 0;


    while(true)
        SetPTPCmd(&cmd, true, &queuedCmdIndex);
}
```

### 1.4.6 Downloading Commands

The Dobot controller supports downloading commands to the controller's external Flash, and

the commands can be triggered by pressing the keys on the controller. That is, the operation is in offline mode.

Table 1.8 Download commands

| | |
|---|---|
| Prototype | int SetQueuedCmdStartDownload(uint32_t totalLoop, uint32_t linePerLoop) |
| Description | Download commands. If the operation of Dobot need to be in offline mode, please call this API |
| Parameter | totalLoop: Loops of commands in offline mode<br>linePerLoop: loops of per command in offline mode. The number of the issued commands must be the same as **linePerLoop**. The issued commands should be added to the command queue. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.4.7 Stopping Downloading Commands.

Table 1.9 Stop to download commands

| | |
|---|---|
| Prototype | int SetQueuedCmdStopDownload(void) |
| Description | Stop downloading commands. If the Dobot is in offline mode, please call this API |
| Parameter | None |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.4.8 Demo: Downloading PTP Command

Program 1.4 Download PTP command

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd    cmd;
```

```
    cmd.ptpMode = 0;
    cmd.x          = 200;
    cmd.y          = 0;
    cmd.z          = 0;
    cmd.r          = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    // Issue only one PTP command, so linePerLoop is set to 1
    // totalLoop is set to 2, so Dobot controller executes the PTP command twice.
    SetQueuedCmdStartDownload(2, 1);
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetQueuedCmdStopDownload();
    DisconnectDobot();
}
```

The general flow of commands to download is:

(1) Call the **SetQueuedCmdStartDownload** API.

(2) Send commands and add to the command queue.

(3) Call the **SetQueuedCmdStopDownload** API.

### 1.4.9 Clearing Command queue

This API can clear the command queue buffered in the Dobot controller.

Table 1.10 Clear command queue

| | |
|---|---|
| Prototype | int SetQueuedCmdClear(void) |
| Description | Clear command queue |
| Parameter | None |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.4.10 Getting Command Index

In the Dobot controller, there is a 64-bit internal counter. When the controller executes a command, the counter will automatically increment. With this internal index, you can get how many commands the controller has executed.

Table 1.11 Get command index

| | |
|---|---|
| Prototype | int GetQueuedCmdCurrentIndex(uint64_t *queuedCmdCurrentIndex) |
| Description | Get the index of the command the controller has executed currently |
| Parameter | queuedCmdCurrentIndex: Command index |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.4.11 Demo: Checking Whether the Commands Have Been Executed

Program 1.5 Check whether the commands have been executed by comparing the indexes

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex    = 0;
    uint64_t executedCmdIndex = 0;
    PTPCmd    cmd;

    cmd.ptpMode = 0;
    cmd.x          = 200;
    cmd.y          = 0;
    cmd.z          = 0;
    cmd.r          = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);

    // Check whether the commands have been executed by comparing the indexes
    While(executedCmdIndex < queuedCmdIndex)
        GetQueuedCmdCurrentIndex(&executedCmdIndex);

    SetQueuedCmdStopExec();
    DisconnectDobot();
```

```
}
```

## 1.5 Device Information

### 1.5.1 Setting the Device Serial Number

Table 1.12 Set the device serial number

| | |
|---|---|
| Prototype | int SetDeviceSN(const char *deviceSN) |
| Description | Set the device serial number. This API is valid only when shipped out (The special password is required) |
| Parameter | deviceSN: String pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.2 Getting the Device Serial Number

Table 1.13 Get the device serial number

| | |
|---|---|
| Prototype | int GetDeviceSN(char *deviceSN, uint32_t maxLen) |
| Description | Get the device serial number |
| Parameter | deviceSN: Strings of device serial number<br>maxLen: Maximum string length, to avoid overflow |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.3 Setting the Device Name

Table 1.14 Set the device name

| | |
|---|---|
| Prototype | int SetDeviceName(const char *deviceName) |
| Description | Set the device name. When there are multiple machines, you can use this API to set the device name for distinction |
| Parameter | deviceName: String pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.4 Getting the Device Name

Table 1.15 Get the device name

| | |
|---|---|
| Prototype | int GetDeviceName(char *deviceName, uint32_t maxLen) |
| Description | Get the device name. When there are multiple machines, you can use this API to get the device name for distinction. |
| Parameter | deviceName: String pointer<br>maxLen: Maximum string length, to avoid overflow |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.5 Getting the Device Version

Table 1.16 Get the device version

| | |
|---|---|
| Prototype | int GetDeviceVersion(uint8_t *majorVersion, uint8_t *minorVersion, uint8_t *revision) |
| Description | Get the device version |
| Parameter | majorVersion: Main version<br>minorVersion: Secondary version<br>revision: Revised version |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.6 Setting the Sliding Rail Status

Table 1.17 Set the sliding rail status

| | |
|---|---|
| Prototype | int SetDeviceWithL(bool isEnable) |
| Description | Set the sliding rail status. When the sliding rail kit is used, please call this API. Only the status of the sliding rail is enabled, the commands related to the sliding rail can be effected. |
| Parameter | isEnable: **0**, Disabled. **1**, Enabled<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** |

| | |
|---|---|
| | indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.7 Getting the Sliding Rail Status

Table 1.18 Get the sliding rail status

| | |
|---|---|
| Prototype | int GetDeviceWithL(bool *isEnable) |
| Description | Get the sliding rail status. When the sliding rail kit is used, please call this API |
| Parameter | isEnable: **0**, Disabled. **1**, Enabled |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.5.8 Getting the Device Clock

Table 1.19 Get the device clock

| | |
|---|---|
| Prototype | int GetDeviceTime(unit32_t *deviceTime) |
| Description | Get the device clock |
| Parameter | deviceTime: Device clock |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.6 Real-time pose

In DobotV2.0, the Dobot controller calculates the reference value of the real-time pose based on the following information.

- Encoder value on the base (can be obtained by Homing).
- Rear Arm angle sensor value (power on or press UNLOCK button on Forearm);
- Forearm angle sensor value (power on or press UNLOCK button on Forearm).

When controlling the Dobot, the Dobot controller will update the real-time pose based on the reference value and the real-time motion status.

### 1.6.1 Getting the Real-time Pose of the Dobot

Table 1.20 Get the real-time pose of Dobot

| | |
|---|---|
| Prototype | int GetPose(Pose *pose) |
| Description | Get the real-time pose of the Dobot |
| Parameter | Pose:<br>typedef struct tagPose {<br>float x; //Cartesian coordinate system X-axis<br>float y; //Cartesian coordinate system Y-axis<br>float z; // Cartesian coordinate system Z-axis<br>float r; //Cartesian coordinate system R-axis<br>float jointAngle[4]; //Joints (including base, Rear Arm, Forearm, and End-effector) angles<br>}Pose;<br>Pose: Pose pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.6.2 Getting the Real-time Pose of the Sliding Rail

Table 1.21 Get the real-time pose of sliding rail

| | |
|---|---|
| Prototype | int GetPose(Pose *pose) |
| Description | Get the real-time pose of the sliding rail |
| Parameter | Pose: The current position of sliding rail. Unit: mm |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.6.3 Resetting the Reference Value of the Real-time Pose

The reference value of the real-time pose can be reset in the following cases.

- Angle sensor is damaged.
- Angle sensor accuracy is too poor.

Table 1.22 Reset the reference value of the real-time pose

| | |
|---|---|
| Prototype | int ResetPose(bool manual, float rearArmAngle, float frontArmAngle) |
| Description | Reset the reference value of the real-time pose |

| | |
|---|---|
| Parameter | manual: Indicate whether to reset reference value of real-time pose automatically. **0**, reset the reference value automatically and **rearArmAngle** and **frontArmAngle** are not to set. **1**, **rearArmAngle** and **frontArmAngle** need to be set<br>rearArmAngle: Rear Arm angle<br>frontArmAngle: Forearm angle |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.7 ALARM

### 1.7.1 Getting the Alarm Status

Table 1.23 Get the alarm status

| | |
|---|---|
| Prototype | int GetAlarmsState(uint8_t *alarmsState, uint32_t *len, uint32_t maxLen) |
| Description | Get the alarm status |
| Parameter | alarmsState: The first address of the array. Each byte in the array **alarmsState** identifies the alarms status of the eight alarm items, with the MSB (Most Significant Bit) at the top and LSB (Least Significant Bit) at the bottom.<br>len: The byte occupied by the alarm.<br>maxLen: Maximum array length, to avoid overflow |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.7.2 Clearing the Statuses of All Alarms

Table 1.24 Clear the statuses of all alarms

| | |
|---|---|
| Prototype | int ClearAllAlarmsState(void) |
| Description | Clear the statuses of all alarms |
| Parameter | None |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.8 Homing Function

If your Dobot is running too fast or the load is too large for the dobot, the position precision can be reduced. You can execute the homing function to improve the precision.

### 1.8.1 Setting the Homing Position

Table 1.25 Set the homing position

| | |
|---|---|
| Prototype | int SetHOMEParams(HOMEParams *homeParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the homing position |
| Parameter | HOMEParams:<br>typedef struct tagHOMEParams {<br>float x; //Cartesian coordinate system X-axis<br>float y; //Cartesian coordinate system Y-axis<br>float z; // Cartesian coordinate system Z-axis<br>float r; //Cartesian coordinate system R-axis<br>}HOMEParams;<br>homeParams: HOMEParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.8.2 Getting the Homing Position

Table 1.26 Get the homing position

| | |
|---|---|
| Prototype | int GetHOMEParams(HOMEParams *homeParams) |
| Description | Get the homing position |
| Parameter | HOMEParams:<br>typedef struct tagHOMEParams {<br>float x; //Cartesian coordinate system X-axis<br>float y; //Cartesian coordinate system Y-axis<br>float z; // Cartesian coordinate system Z-axis<br>float r; //Cartesian coordinate system R-axis<br>}HOMEParams; |

| | |
|---|---|
| | homeParams: HOMEParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.8.3 Executing the Homing Function

Table 1.27 Execute the homing function

| | |
|---|---|
| Prototype | int SetHOMECmd(HOMECmd *homeCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the homing function. If you call the **SetHOMEParams** API before calling this API, Dobot will move to the user-defined position. If not, Dobot will move to the default position directly. |
| Parameter | HOMECmd:<br>typedef struct tagHOMECmd {<br>uint32_t reserved; // Reserved for future use<br>}HOMECmd;<br>homeCmd: HOMECmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.8.4 Executing the Automatic Leveling Function

If the value of the Rear Arm angle sensor or the Forearm angle sensor is error, it means that the position precision is reduced. You can call this API to improve the precision. If the high position accuracy is required, you need to perform leveling manually. For more details, please see *Dobot Magician User Guide*.

Table 1.28 Execute the Automatic leveling function

| | |
|---|---|
| Prototype | int SetAutoLevelingCmd(AutoLevelingCmd *autoLevelingCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the automatic leveling function |

| | |
|---|---|
| Parameter | AutoLevelingCmd :<br>typedef struct tagAutoLevelingCmd{<br>uint8_t controlFlag; //Enabe Flag<br>float precision; //Leveling precision, the minimum is 0.02<br>}AutoLevelingCmd;<br>autoLevelingCmd : AutoLevelingCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.8.5 Getting the Automatic Leveling Results

Table 1.29 Get the automatic leveling results

| | |
|---|---|
| Prototype | int GetAutoLevelingResult(float *precision) |
| Description | Get the automatic leveling results |
| Parameter | precision: Leveling precision |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.9 HHT Function

HHT indicates Hand-Hold Teaching. In general, you can press and hold down **Unlock** key on Forearm and drag Dobot to any position. And then save point after releasing **Unlock** key.

### 1.9.1 Setting the Hand-Hold Teaching Trigger Mode

Table 1.30 Set the hand-hold teaching mode

| | |
|---|---|
| Prototype | int SetHHTTrigMode (HHTTrigMode hhtTrigMode) |
| Description | Set the hand-hold teaching triggering mode. If this API is not called, Dobot will save points when releasing the **UNLOCK** key on Forearm |
| Parameter | HHTTrigMode:<br>typedef enum tagHHTTrigMode {<br>TriggedOnKeyReleased, //Trigger when releasing the |

| | |
|---|---|
| | **UNLOCK** key<br>TriggeredOnPeriodicInterval //Trigger when pressing the **UNLOCK** key<br>}HHTTrigMode;<br>hhtTrigMode: HHTTrigMode enum |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.9.2 Getting the Hand-Hold Teaching Trigger Mode

Table 1.31 Get the hand-hold teaching trigger mode

| | |
|---|---|
| Prototype | int GetHHTTrigMode (HHTTrigMode hhtTrigMode) |
| Description | Get the handhold teaching trigger mode. |
| Parameter | HHTTrigMode:<br>typedef enum tagHHTTrigMode {<br>TriggedOnKeyReleased, //Trigger when releasing the **UNLOCK** key<br>TriggeredOnPeriodicInterval //Trigger when pressing the **UNLOCK** key<br>}HHTTrigMode;<br>hhtTrigMode: HHTTrigMode enum |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.9.3 Setting the Status of the Hand-Hold Teaching Function

Table 1.32 Set the status of the hand-hold teaching function

| | |
|---|---|
| Prototype | int SetHHTTrigOutputEnabled (bool isEnabled) |
| Description | Set the status of the hand-hold teaching function |
| Parameter | isEnabled: **0** : Disabled. **1** : Enabled |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

 

### 1.9.4 Getting the Status of the Hand-Hold Teaching Function

Table 1.33 Get the status of the hand-hold teaching function

| | |
|---|---|
| Prototype | int GetHHTTrigOutputEnabled (bool *isEnabled) |
| Description | Get the status of the hand-hold teaching function |
| Parameter | isEnabled: **0** : Disabled. **1** : Enabled |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.9.5 Getting the Hand-Hold Teaching Trigger Single

Table 1.34 Get the hand-hold teaching trigger single

| | |
|---|---|
| Prototype | int GetHHTTrigOutput(bool *isTriggered) |
| Description | Get the hand-hold teaching trigger single<br>Please call the **SetHHTTrigOutputEnabled** API before calling this API |
| Parameter | isTriggered: **0**: Not triggered. **1**: Triggered |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.9.6 Demo: Hand-Hold Teaching

Program 1.6 Hand-hold Teaching

```
#include "DobotDll.h"
#include <queue>
#include <windows.h>


int main(void)
{
    ConnectDobot(NULL, 115200, NULL, NULL, NULL);


    SetHHTTrigMode(TriggeredOnPeriodicInterval);
    SetHHTTrigOutputEnabled(true);
```

```
    bool isTriggered = false;
    queue<Pose> poseQueue;
    Pose pose;
    while(true) {
        GetHHTTrigOutput(&isTriggered);
        if(isTriggered) {
            GetPose(&pose);
            poseQueue.push(pose);
        }
    }

    DisconnectDobot();
}
```

## 1.10 End-effector

### 1.10.1 Setting the Offset of the End-effector

Table 1.35 Set the offset of the end-effector

| | |
|---|---|
| Prototype | int SetEndEffectorParams(EndEffectorParams *endEffectorParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the offset of the end-effector. If the end-effector is installed, this API is required<br>If a standard end-effector is used, please refer to *Dobot Magician User Guide* to obtain the X-axis offset and Y-axis offset and call this API. Otherwise, please confirm the structural parameters. |
| Parameter | EndEffectorParams:<br>typedef struct tagEndEffectorParams {<br>float xBias; //X-axis offset of end-effector<br>float yBias; //Y-axis offset of end-effector<br>float zBias; //Z-axis offset of end-effector<br>}EndEffectorParams;<br>endEffectorParams: EndEffectorParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full |

| | |
|---|---|
| | DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.2 Getting the Offset of the End-effector

Table 1.36 Get offset of end-effector

| | |
|---|---|
| Prototype | int GetEndEffectorParams(EndEffectorParams *endEffectorParams) |
| Description | Get the offset of the end-effector |
| Parameter | EndEffectorParams:<br>typedef struct tagEndEffectorParams {<br>float xBias; //X-axis offset of end-effector<br>float yBias; //Y-axis offset of end-effector<br>float zBias; //Z-axis offset of end-effector<br>}EndEffectorParams;<br>endEffectorParams: EndEffectorParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.3 Setting the Status of the Laser

Table 1.37 Set the status of the laser

| | |
|---|---|
| Prototype | int SetEndEffectorLaser(bool enableCtrl, bool on, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the status of the laser |
| Parameter | enableCtrl: Control end-effector. **0**: Disabled. **1**: Enabled<br>on: Start or stop laser. **0**, Off. **1**, On<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.4 Getting the Status of the Laser

Table 1.38 Get the status of the laser

| | |
|---|---|
| Prototype | int GetEndEffectorLaser(bool *isCtrlEnabled, bool *isOn) |
| Description | Get the status of the laser |
| Parameter | isCtrlEnabled: If the status of the end-effector is enabled. **0**: Disabled. **1**: Enabled<br>isOn: If the status of the laser is on. **0**, Off. **1**, On |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.5 Setting the Status of the Air Pump

Table 1.39 Set the status of the air pump

| | |
|---|---|
| Prototype | int SetEndEffectorSuctionCup(bool enableCtrl, bool suck, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the status of the air pump |
| Parameter | enableCtrl: Control end-effector. **0**: Disabled. **1**: Enabled<br>suck: Control the intake and outtake of the air pump. **0**: Outtake. **1:** Intake<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.6 Getting the Status of the Air Pump

Table 1.40 Get the status of the air pump

| | |
|---|---|
| Prototype | int GetEndEffectorSuctionCup(bool *isCtrlEnabled, bool *isSucked) |
| Description | Get the status of the air pump |
| Parameter | isCtrlEnabled: If the status of the end-effector is enabled. **0**: Disabled. **1**: Enabled<br>isSucked: If the status of the air pump is intake or outtake. **0**: Outtake. 1: Intake |
| Return | DobotCommunicate_NoError: The command returns with no error |

| | |
|---|---|
| | DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.7 Setting the Status of the Gripper

Table 1.41 Set the status of the gripper

| | |
|---|---|
| Prototype | int SetEndEffectorGripper(bool enableCtrl, bool grip, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the status of the gripper |
| Parameter | enableCtrl: Control end-effector. **0**: Disabled. **1**: Enabled<br>grip: Control the gripper to grip or release. **0**: Released, **1**: Grabbed<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.10.8 Getting the Status of the Gripper

Table 1.42 Get the status of the gripper

| | |
|---|---|
| Prototype | int GetEndEffectorGripper(bool *isCtrlEnabled, bool *isGripped) |
| Description | Get the status of the gripper |
| Parameter | isCtrlEnabled: If the status of the end-effector is enabled. **0**: Disabled. **1**: Enabled<br>isGripped: If the status of the gripper is gripped or released. **0**: Released. **1**: Grabbed |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.11 JOG

### 1.11.1 Setting the Velocity and Acceleration of the Joint Coordinate Axis when Jogging

Table 1.43 Set the velocity and acceleration of the joints coordinate axis when jogging

| | |
|---|---|
| Prototype | int SetJOGJointParams(JOGJointParams *jogJointParams, bool isQueued, |

| | |
|---|---|
| | uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration of the joint coordinate axis when jogging |
| Parameter | JOGJointParams:<br>typedef struct tagJOGJointParams {<br>float velocity[4]; //Joint velocity<br>float acceleration[4]; //Joint acceleration<br>}JOGJointParams;<br>jogJointParams: JOGJointParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.2 Getting the Velocity and Acceleration of the Joint Coordinate Axis when Jogging

Table 1.44 Get the velocity and acceleration of joint coordinate axis when jogging

| | |
|---|---|
| Prototype | int GetJOGJointParams(JOGJointParams *jogJointParams) |
| Description | Get the velocity and acceleration of the joint coordinate axis when jogging |
| Parameter | JOGJointParams:<br>typedef struct tagJOGJointParams {<br>float velocity[4]; //Joint velocity<br>float acceleration[4]; //Joint acceleration<br>}JOGJointParams;<br>jogJointParams: JOGJointParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.3 Setting the velocity and acceleration of the Cartesian Coordinate Axis when Jogging

Table 1.45 Set the velocity and acceleration of the Cartesian coordinate axis when jogging

| | |
|---|---|
| Prototype | int SetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams, |

| | |
|---|---|
| | bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration of the Cartesian coordinate axis when jogging |
| Parameter | JOGCoordinateParams:<br>typedef struct tagJOGCoordinateParams {<br>float velocity[4]; //Cartesian coordinate axis (X,Y,Z,R)velocity<br>float acceleration[4]; //Cartesian coordinate axis (X,Y,Z,R) acceleration<br>}JOGCoordinateParams;<br>jogCoordinateParams: JOGCoordinateParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.4 Getting the velocity and acceleration of the Cartesian Coordinate Axis when Jogging

Table 1.46 Get the velocity and acceleration of the Cartesian coordinate axis when jogging

| | |
|---|---|
| Prototype | int GetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams) |
| Description | Get the velocity and acceleration of the Cartesian coordinate axis when jogging |
| Parameter | typedef struct tagJOGCoordinateParams {<br>float velocity[4]; //Cartesian coordinate axis (X,Y,Z,R)velocity<br>float acceleration[4]; //Cartesian coordinate axis (X,Y,Z,R) acceleration<br>}JOGCoordinateParams;<br>jogCoordinateParams: JOGCoordinateParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.5 Setting the velocity and acceleration of the Sliding Rail when Jogging

Table 1.47 Set the velocity and acceleration of the sliding rail when jogging

| | |
|---|---|
| Prototype | int SetJOGLParams(JOGLParams *jogLParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration of the sliding rail when jogging |
| Parameter | JOGLParams:<br>typedef struct tagJOGLParams {<br>float velocity; //Sliding rail velocity<br>float acceleration; //Sliding rail acceleration<br>}JOGLParams;<br>jogLParams: JOGLParams<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid. |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.6 Getting the velocity and acceleration of the Sliding Rail when Jogging

Table 1.48 Get the velocity and acceleration of the sliding rail when jogging

| | |
|---|---|
| Prototype | int GetJOGLParams(JOGLParams * jogLParams ) |
| Description | Get the velocity and acceleration of the sliding rail when jogging |
| Parameter | JOGLParams:<br>typedef struct tagJOGLParams {<br>float velocity; //Sliding rail velocity<br>float acceleration; //Sliding rail acceleration<br>}JOGLParams;<br>jogLParams: JOGLParams |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.7 Setting the Velocity Ratio and Acceleration Ratio when Jogging

Table 1.49 Set the velocity ratio and acceleration ratio when jogging

| | |
|---|---|
| Prototype | int SetJOGCommonParams(JOGCommonParams *jogCommonParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity ratio and acceleration ratio for each axis (in both Joint and Cartesian coordinate system) when jogging |
| Parameter | JOGCommonParams:<br>typedef struct tagJOGCommonParams {<br>float velocityRatio; //Velocity ratio<br>float accelerationRatio; //Acceleration ratio<br>}JOGCommonParams;<br>jogCommonParams: JOGCommonParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.8 Getting the Velocity Ratio and Acceleration Ratio when Jogging

Table 1.50 Get the velocity ratio and acceleration ratio when jogging

| | |
|---|---|
| Prototype | int GetJOGCommonParams(JOGCommonParams *jogCommonParams) |
| Description | Get the velocity ratio and acceleration ratio for each axis (in Joint and Cartesian coordinate system) when jogging |
| Parameter | JOGCommonParams:<br>typedef struct tagJOGCommonParams {<br>float velocityRatio; //Velocity ratio<br>float accelerationRatio; //Acceleration ratio<br>}JOGCommonParams;<br>jogCommonParams: JOGCommonParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.11.9 Executing the Jogging Command

Table 1.51 Execute the Jogging command

| | |
|---|---|
| Prototype | int SetJOGCmd(JOGCmd *jogCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the Jogging command. Please call this API after setting the related parameters of jogging |
| Parameter | JOGCmd:<br>typedef struct tagJOGCmd {<br>uint8_t isJoint; //Jogging mode: 0, Jog in Cartesian coordinate system. 1, Jog in Joint coordinate system<br>uint8_t cmd; //Jogging command: 0-10<br>}JOGCmd;<br>//Details for jogging commands<br>enum {<br>IDLE, // Idle<br>AP_DOWN, // X+/Joint1+<br>AN_DOWN, // X-/Joint1-<br>BP_DOWN, // Y+/Joint2+<br>BN_DOWN, // Y-/Joint2-<br>CP_DOWN, // Z+/Joint3+<br>CN_DOWN, // Z-/Joint3-<br>DP_DOWN, // R+/Joint4+<br>DN_DOWN, // R-/Joint4-<br>LP_DOWN, // L+<br>LN_DOWN // L-<br>};<br>jogCmd: JOGCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.12 PTP

PTP mode supports MOVJ, MOVL, and JUMP, which is point-to-point movement. The

trajectory of playback depends on the motion mode.

- MOVJ: Joint movement. From point A to point B, each joint will run from initial angle to its target angle, regardless of the trajectory, as shown in Figure 1.1.

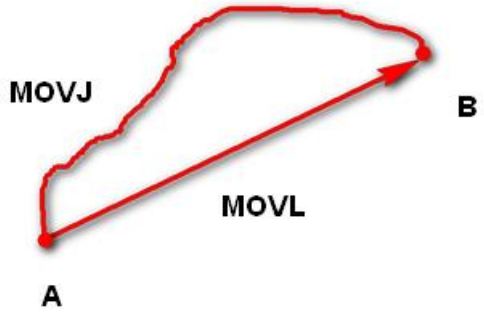


Figure 1.1 MOVL/MOVJ mode

- MOVL: Rectilinear movement. The joints will perform a straight line trajectory from point A to point B, as shown in Figure 1.1.
- JUMP: From point A to point B, the trajectory is shown in Figure 1.2., the end effector will lift upwards by amount of Height (in mm) and move horizontally to a point that is above B by Height and then move down to Point B.


Figure 1.2 JUMP mode

### 1.12.1 Setting the Velocity and Acceleration of the Joint Coordinate Axis in PTP Mode

Table 1.52 Set the velocity and acceleration of the joint coordinate axis in PTP mode

| | |
|---|---|
| Prototype | int SetPTPJointParams(PTPJointParams *ptpJointParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration of the joint coordinate axis in PTP mode |
| Parameter | PTPJointParams:<br>typedef struct tagPTPJointParams { |

| | |
|---|---|
| | float velocity[4]; // Joint velocity in PTP mode<br>float acceleration[4]; //Joint acceleration in PTP mode<br>}PTPJointParams;<br>ptpJointParams: PTPJointParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.2 Getting the Velocity and Acceleration of the Joint Coordinate Axis in PTP Mode

Table 1.53 Get the velocity and acceleration of the joint coordinate axis in PTP mode

| | |
|---|---|
| Prototype | int GetPTPJointParams(PTPJointParams *ptpJointParams) |
| Description | Get the velocity and acceleration of the joint coordinate axis in PTP mode |
| Parameter | PTPJointParams<br>typedef struct tagPTPJointParams {<br>float velocity[4]; //Joint velocity in PTP mode<br>float acceleration[4]; //Joint acceleration in PTP mode<br>}PTPJointParams;<br>ptpJointParams: PTPJointParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.3 Setting the Velocity and Acceleration of the Cartesian Coordinate Axis in PTP Mode

Table 1.54 Set the velocity and acceleration of the Cartesian coordinate axis in PTP mode

| | |
|---|---|
| Prototype | int SetPTPCoordinateParams(PTPCoordinateParams *ptpCoordinateParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration of the Cartesian coordinate axis in PTP mode |
| Parameter | PTPCoordinateParams: |

 

| | |
|---|---|
| | typedef struct tagPTPCoordinateParams {<br>float xyzVelocity; //Cartesian coordinate axis (X,Y,Z) velocity<br>float rVelocity; // Cartesian coordinate axis (R) velocity<br>float xyzAcceleration; //Cartesian coordinate axis (X,Y,Z) acceleration<br>float rAcceleration; //Cartesian coordinate axis (R) acceleration<br>}PTPCoordinateParams;<br>ptpCoordinateParams: PTPCoordinateParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.4 Getting the Velocity and Acceleration of the Cartesian Coordinate Axis in PTP Mode

Table 1.55 Get the velocity and acceleration of the Cartesian coordinate axis in PTP mode

| | |
|---|---|
| Prototype | int GetPTPCoordinateParams(PTPCoordinateParams *ptpCoordinateParams) |
| Description | Get the velocity and acceleration of the Cartesian coordinate axis in PTP mode |
| Parameter | PTPCoordinateParams:<br>typedef struct tagPTPCoordinateParams {<br>float xyzVelocity; //Cartesian coordinate axis (X,Y,Z) velocity<br>float rVelocity; // Cartesian coordinate axis (R) velocity<br>float xyzAcceleration; //Cartesian coordinate axis (X,Y,Z) acceleration<br>float rAcceleration; //Cartesian coordinate axis (R) acceleration<br>}PTPCoordinateParams;<br>ptpCoordinateParams: PTPCoordinateParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.5 Setting the Lifting Height and the Maximum Lifting Height in JUMP mode

Table 1.56 Set the lifting height and the maximum lifting height in JUMP mode

| | |
|---|---|
| Prototype | int SetPTPJumpParams(PTPJumpParams *ptpJumpParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the lifting height and the maximum height in JUMP mode |
| Parameter | PTPJumpParams:<br>typedef struct tagPTPJumpParams {<br>float jumpHeight; //Lifting height<br>float zLimit; //Maximum lifting height<br>}PTPJumpParams;<br>ptpJumpParams: PTPJumpParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.6 Getting the Lifting Height and the Maximum Lifting Height in JUMP mode

Table 1.57 Get the lifting height and the maximum lifting height in JUMP mode

| | |
|---|---|
| Prototype | int GetPTPJumpParams(PTPJumpParams *ptpJumpParams) |
| Description | Get the lifting height and the maximum lifting height in JUMP mode |
| Parameter | PTPJumpParams:<br>typedef struct tagPTPJumpParams {<br>float jumpHeight; //Lifting height<br>float zLimit; //Maximum lifting height<br>}PTPJumpParams;<br>ptpJumpParams: PTPJumpParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.7 Setting the Extended Parameters in JUMP mode

Table 1.58 Set the extended parameters in JUMP mode

| | |
|---|---|
| Prototype | int SetPTPJump2Params(PTPJumpParams *ptpJump2Params, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the extended parameters in JUMP mode |
| Parameter | PTPJump2Params:<br>typedef struct tagPTPJump2Params {<br>float startJumpHeight; //Lifting height of starting point<br>float endJumpHeight; //Lifting height of end point<br>float zLimit; //Maximum lifting height<br>}PTPJump2Params;<br>ptpJump2Params: PTPJump2Params pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.8 Getting the Extended Parameters in JUMP mode

Table 1.59 Get extended parameters in JUMP mode

| | |
|---|---|
| Prototype | int GetPTPJump2Params(PTPJumpParams *ptpJump2Params) |
| Description | Get the extended parameters in JUMP mode |
| Parameter | PTPJump2Params:<br>typedef struct tagPTPJump2Params {<br>float startJumpHeight; //Lifting height of starting point<br>float endJumpHeight; //Lifting height of end point<br>float zLimit; //Maximum lifting height<br>}PTPJump2Params; |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.9 Setting the Velocity and Acceleration of the Sliding Rail in PTP Mode

Table 1.60 Set the velocity and acceleration of the sliding rail in PTP mode

| | |
|---|---|
| Prototype | int SetPTPLParams(PTPLParams * ptpLParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Sets the velocity and acceleration of the sliding rail in PTP mode |
| Parameter | PTPLParams:<br>typedef struct tagPTPJointParams {<br>float velocity; //Sliding rail velocity in PTP mode<br>float acceleration; //Sliding rail acceleration in PTP mode<br>}PTPLParams;<br>ptpLParams: PTPLParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.10 Getting the Velocity and Acceleration of the Sliding rail in PTP Mode

Table 1.61 Get the velocity and acceleration of the Sliding rail s in PTP mode

| | |
|---|---|
| Prototype | int GetPTPLParams(PTPLParams *ptpLParams) |
| Description | Get the velocity and acceleration of the sliding rail in PTP mode |
| Parameter | PTPLParams:<br>typedef struct tagPTPJointParams {<br>float velocity; //Sliding rail velocity in PTP mode<br>float acceleration; //Sliding rail acceleration in PTP mode<br>}PTPLParams;<br>ptpLParams: PTPLParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.11 Setting the Velocity Ratio and Acceleration Ratio in PTP Mode

Table 1.62 Set the velocity ratio and the acceleration ratio in PTP mode

| | |
|---|---|
| Prototype | int SetPTPCommonParams(PTPCommonParams *ptpCommonParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity ratio and acceleration ratio in PTP mode |
| Parameter | PTPCommonParams:<br>typedef struct tagPTPCommonParams {<br>float velocityRatio; //Velocity ratio<br>float accelerationRatio; //Acceleration ratio<br>}PTPCommonParams;<br>ptpCommonParams: PTPCommonParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.12 Getting the Velocity Ratio and Acceleration Ratio in PTP Mode

Table 1.63 Get the velocity ratio and acceleration ratio in PTP mode

| | |
|---|---|
| Prototype | int GetPTPCommonParams(PTPCommonParams *ptpCommonParams) |
| Description | Get the velocity ratio and acceleration ratio in PTP mode |
| Parameter | PTPCommonParams:<br>typedef struct tagPTPCommonParams {<br>float velocityRatio; //Velocity ratio<br>float accelerationRatio; //Acceleration ratio<br>}PTPCommonParams;<br>ptpCommonParams: PTPCommonParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.13 Executing a PTP Command

Table 1.64 Execute a PTP command

| | |
|---|---|
| Prototype | int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute a PTP command. Please call this API after setting the related parameters in PTP mode to make the Dobot move to the target point |
| Parameter | PTPCmd:<br>typedef struct tagPTPCmd {<br>uint8_t ptpMode; //PTP mode (0-9)<br>float x; //Coordinate parameters in PTP mode. (x,y,z,r) can be set to Cartesian coordinate, joints angle, or increment of them<br>float y;<br>float z;<br>float r;<br>}PTPCmd;<br>Details for ptpMode:<br>enum {<br>JUMP_XYZ, //JUMP mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVJ_XYZ, //MOVJ mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVL_XYZ, //MOVL mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>JUMP_ANGLE, //JUMP mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVJ_ANGLE, //MOVJ mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVJ_INC, //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system<br>MOVL_INC, //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system<br>MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system<br>JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian |

| | |
|---|---|
| | coordinate system<br>};<br>ptpCmd: PTPCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.14 Executing a PTP Command with the I/O Control

Table 1.65 Execute a PTP command with the I/O control

| | |
|---|---|
| Prototype | int SetPTPPOCmd(PTPCmd *ptpCmd, ParallelOutputCmd *parallelCmd, int parallelCmdCount, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute a PTP command with the I/O control. You can control the suction cup or gripper by I/O control. For more details on the I/O description, please see *Dobot Magician User Guide* |
| Parameter | PTPCmd:<br>typedef struct tagPTPCmd {<br>uint8_t ptpMode; //PTP mode (0-9)<br>float x; //Coordinate parameters in PTP mode. (x,y,z,r) can be set to Cartesian coordinate, joints angle, or increment of them<br>float y;<br>float z;<br>float r;<br>}PTPCmd;<br>Details for ptpMode:<br>enum {<br>JUMP_XYZ, //JUMP mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVJ_XYZ, //MOVJ mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVL_XYZ, //MOVL mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>JUMP_ANGLE, //JUMP mode, (x,y,z,r) is the target point in Joint |

| | |
|---|---|
| | coordinate system<br>MOVJ_ANGLE, //MOVJ mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVJ_INC, //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system<br>MOVL_INC, //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system<br>MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system<br>JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system<br>};<br>ParallelOutputCmd:<br>typedef struct tagParallelOutputCmd {<br>uint8_t ratio; //The distance ratio between the two points in PTP mode, namely, the position where I/O is triggered<br>uint16_t address; //I/O address (0-20)<br>uint8_t level; //Output value<br>}ParallelOutputCmd;<br>ptpCmd: PTPCmd pointer<br>parallelCmd: ParallelOutputCmd pointer<br>parallelCmdCount::I/O number<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.15 Executing a PTP Command with the Sliding Rail

Table 1.66 Execute a PTP command with the sliding rail

| | |
|---|---|
| Prototype | int SetPTPWithLCmd(PTPWithLCmd *ptpWithLCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute a PTP command with the sliding rail |
| Parameter | PTPWithLCmd<br>typedef struct tagPTPWithL {<br>uint8_t ptpMode; //PTP mode (0-9)<br>float x; //Coordinate parameters in PTP mode. (x,y,z,r) can be set to Cartesian coordinate, joints angle, or increment of them<br>float y;<br>float z;<br>float r;<br>float l; //The distance that sliding rail moves<br>}PTPWithLCmd ;<br>Details for ptpMode:<br>enum {<br>JUMP_XYZ, //JUMP mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVJ_XYZ, //MOVJ mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVL_XYZ, //MOVL mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>JUMP_ANGLE, //JUMP mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVJ_ANGLE, //MOVJ mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVJ_INC, //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system<br>MOVL_INC, //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system<br>MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system<br>JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian |

| | |
|---|---|
| | coordinate system<br>};<br>ptpWithLCmd : PTPWithLCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.12.16 Executing a PTP Command with the Sliding Rail and I/O Control

Table 1.67 Execute a PTP command with the sliding rail and I/O control

| | |
|---|---|
| Prototype | int SetPTPOWithLCmd(PTPWithLCmd *ptpWithLCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute a PTP command with the sliding rail and I/O control |
| Parameter | PTPWithLCmd<br>typedef struct tagPTPWithL {<br>uint8_t ptpMode; //PTP mode (0-9)<br>float x; //Coordinate parameters in PTP mode. (x,y,z,r) can be set to Cartesian coordinate, joints angle, or increment of them<br>float y;<br>float z;<br>float r;<br>float l; //The distance that sliding rail moves<br>}PTPWithLCmd;<br>Details for ptpMode:<br>enum {<br>JUMP_XYZ, //JUMP mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVJ_XYZ, //MOVJ mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>MOVL_XYZ, //MOVL mode, (x,y,z,r) is the target point in Cartesian coordinate system<br>JUMP_ANGLE, //JUMP mode, (x,y,z,r) is the target point in Joint coordinate system |

| | |
|---|---|
| | MOVJ_ANGLE, //MOVJ mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system<br>MOVJ_INC, //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system<br>MOVL_INC, //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system<br>MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system<br>JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system<br>};<br>ParallelOutputCmd:<br>typedef struct tagParallelOutputCmd {<br>uint8_t ratio; //The distance ratio between the two points in PTP mode, namely, the position where I/O is triggered<br>uint16_t address; //I/O address (0-20)<br>uint8_t level; //Output value<br>}ParallelOutputCmd;<br>ptpWithLCmd : PTPWithLCmd pointer<br>parallelCmd: ParallelOutputCmd pointer<br>parallelCmdCount: I/O number<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.13 CP

CP: Continuous Path.

### 1.13.1 Setting the Velocity and Acceleration in CP Mode

Table 1.68 Set the velocity and acceleration in CP mode

| | |
|---|---|
| Prototype | int SetCPParams(CPParams *cpParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration in CP mode |
| Parameter | CPParams<br>typedef struct tagCPParams {<br>float planAcc; //The maximum planning acceleration<br>float junctionVel; //The maximum junction velocity<br>union {<br>float acc; //The maximum actual acceleration. It is valid only when **realTimeTrack** is set to **0**<br>float period; //Interpolation period. It is valid only when **realTimeTrack** is set to **1**<br>};<br>uint8_t realTimeTrack; //**0**: Non-real-time mode, all commands will be executed after they are issued. **1**: Real-time mode, the command is executed while being issued.<br>}CPParams;<br>cpParams: CPParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.13.2 Getting the Velocity and Acceleration in CP Mode

Table 1.69 Get the velocity and acceleration in CP mode

| | |
|---|---|
| Prototype | int GetCPParams(CPParams *cpParams) |
| Description | Get the velocity and acceleration in CP mode |
| Parameter | CPParams<br>typedef struct tagCPParams { |

 

| | |
|---|---|
| | float planAcc; //The maximum planning acceleration<br>float junctionVel; //The maximum junction velocity<br>union {<br>float acc; //The maximum actual acceleration. It is valid only when **realTimeTrack** is set to **0**<br>float period; //Interpolation period. It is valid only when **realTimeTrack** is set to **1**<br>};<br>uint8_t realTimeTrack; //**0**: Non-real-time mode, all commands will be executed after they are issued. **1**: Real-time mode, the command is executed while being issued.<br>}CPParams;<br>cpParams: CPParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.13.3 Executing the CP Command

Table 1.70 Execute the CP command

| | |
|---|---|
| Prototype | int SetCPCmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the CP commands |
| Parameter | CPCmd<br>typedef struct tagCPCmd {<br>uint8_t cpMode; //CP mode. **0**: indicate that (x,y,z) is the Cartesian coordinate increment. **1**:indicate (x,y,z) is the target point in Cartesian coordinate system<br>float x; //(x,y,z )can be set to Cartesian coordinate, or Cartesian coordinate increment<br>float y;<br>float z;<br>union {<br>float velocity; //Reserved<br>float power; //Reserved<br>}CPCmd;<br>cpCmd: CPCmd pointer |

 

| | |
|---|---|
| | isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

⚠NOTICE

When there are multiple CP commands in the command queue, the Dobot controller will look ahead automatically. The look-ahead condition is that there are no JOG, PTP, ARC, WAIT, and TRIG commands between the CP commands.

### 1.13.4 Executing the CP Command with the Laser Engraving

Table 1.71 Execute the CP command with laser engraving

| | |
|---|---|
| Prototype | int SetCPCmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the CP command with the laser engraving. |
| Parameter | typedef struct tagCPCmd {<br>uint8_t cpMode; //CP mode. **0**: indicate that (x,y,z) is the Cartesian coordinate increment. **1**:indicate (x,y,z) is the target point in Cartesian coordinate system<br>float x; //(x,y,z )can be set to Cartesian coordinate, or Cartesian coordinate increment<br>float y;<br>float z;<br>union {<br>float velocity; // Reserved<br>float power; //Laser power 0-100<br>}CPCmd;<br><br>cpCmd: CPCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |

| | |
|---|---|
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.14 ARC

The trajectory of the Dobot in ARC mode is an arc, which is determined by three points (the current point, any point and the end point on the arc), as shown in Figure 1.3.

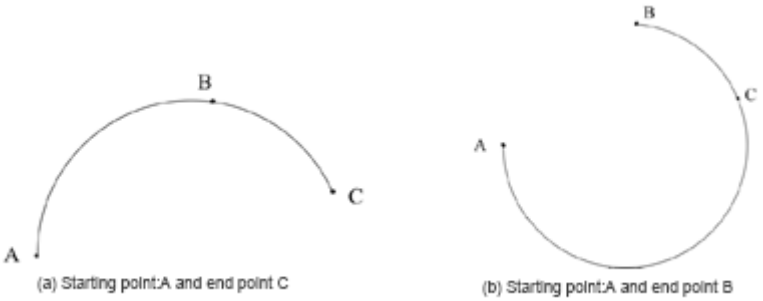


Figure 1.3 ARC mode

### 1.14.1 Setting the Velocity and Acceleration in ARC Mode

Table 1.72 Set the velocity and acceleration in ARC mode

| | |
|---|---|
| Prototype | int SetARCParams(ARCParams *arcParams, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity and acceleration in PTP mode |
| Parameter | ARCParams<br>typedef struct tagARCParams {<br>float xyzVelocity; //Cartesian coordinate axis (X,Y,Z) velocity<br>float rVelocity; //Cartesian coordinate axis (R) velocity<br>float xyzAcceleration; //Cartesian coordinate axis (X,Y,Z) acceleration<br>float rAcceleration; //Cartesian coordinate axis (R) acceleration<br>}ARCParams;<br>arcParams: ARCParams pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full |

| | |
|---|---|
| | DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.14.2 Getting the Velocity and Acceleration in ARC Mode

Table 1.73 Get the velocity and acceleration in ARC mode

| | |
|---|---|
| Prototype | int GetARCParams(ARCParams *arcParams) |
| Description | Get the velocity and acceleration in ARC mode |
| Parameter | ARCParams<br>typedef struct tagARCParams {<br>float xyzVelocity; //Cartesian coordinate axis (X,Y,Z) velocity<br>float rVelocity; //Cartesian coordinate axis (R) velocity<br>float xyzAcceleration; //Cartesian coordinate axis (X,Y,Z) acceleration<br>float rAcceleration; //Cartesian coordinate axis (R) acceleration<br>}ARCParams;<br>arcParams: ARCParams pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.14.3 Executing the ARC Command

Table 1.74 Execute the ARC command

| | |
|---|---|
| Prototype | int SetARCCmd(ARCCmd *arcCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the ARC command. Please call this API after setting the related parameters in ARC mode to make Dobot move to the target point.<br>In ARC mode, it is necessary to confirm the three points with other motion modes. |
| Parameter | ARCCmd:<br>typedef struct tagARCCmd {<br>struct {<br>float x;<br>float y;<br>float z;<br>float r; |

| | |
|---|---|
| | }cirPoint ; //Middle point. (x,y,z,r) can be set to Cartesian coordinate<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}toPoint; //End point. (x,y,z,r) can be set to Cartesian coordinate<br>}ARCCmd;<br>arcCmd: ARCCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.14.4 Executing the CIRCLE Command

The CIRCLE mode is similar to the ARC mode, where the trajectory is a circle.

Table 1.75 Execute the CIRCLE command

| | |
|---|---|
| Prototype | int SetCircleCmd(CircleCmd *circleCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the CIRCLE command. Please call this API after setting the related parameters of playback in CIRCLE mode to make Dobot move to the target point.<br>In CIRCLE mode, it is necessary to confirm the three points with other motion modes. |
| Parameter | CircleCmd<br>typedef struct tagCircleCmd {<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}cirPoint ; //Middle point.(x,y,z,r) can be set to Cartesian |

| | |
|---|---|
| | coordinate<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}toPoint; //End point. (x,y,z,r) can be set to Cartesian coordinate<br>uint32_t count //Circle number<br>}CircleCmd;<br>circleCmd: CircleCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.15 Losing-Step Detection

### 1.15.1 Setting the losing-step threshold

Table 1.76 Set the losing-step threshold

| | |
|---|---|
| Prototype | int SetLostStepParams(float threshold) |
| Description | Set the losing-step threshold, checking for whether the position error exceeds this threshold. If this threshold is exceeded, the motor loses step<br>If you do not call this API, the default threshold is 5 |
| Parameter | threshold: Losing-step threshold |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.15.2 Executing the Losing-Step Command

Table 1.77 Execute the losing-step command

| | |
|---|---|
| Prototype | int SetLostStepCmd(bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the losing-step command. If the motor loses step, the Dobot |

| | |
|---|---|
| | controller will stop to query the command queue and stop executing commands.<br>This command must be added to the command queue, namely, **isQueued** must be set to **1**. |
| Parameter | isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.15.3 Demo: Executing the Losing-Step Command

Program 1.7 Execute the losing-step command

```
#include "DobotDll.h"
int main(void)
{
    PTPCmd    cmd;
    cmd.ptpMode = 0;
    cmd.x          = 200;
    cmd.y          = 0;
    cmd.z          = 0;
    cmd.r          = 0;
    ConnectDobot(NULL, 115200, NULL, NULL, NULL);
    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetLostStepCmd(true, &queuedCmdIndex)
    SetQueuedCmdStopExec();
    DisconnectDobot();
}
```

## 1.16 WAITING

### 1.16.1 Executing the Waiting Command

Table 1.78 Execute the Waiting command

| | |
|---|---|
| Prototype | int SetWAITCmd(WAITCmd *waitCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the Waiting command. If you need to set the pause time between the two commands, please call this API<br>This command must be added to the command queue, namely, **isQueued** must be set to **1**. If not, the parameter **timeout** of **Waiting** command in the command queue being executed may be changed because the **WAITCmd** memory is shared |
| Parameter | WAITCmd:<br>typedef struct tagWAITCmd {<br>uint32_t timeout; //Unit:ms<br>}WAITCmd;<br>waitCmd: WAITCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.17 TRIGGERING

### 1.17.1 Executing the Triggering Command

Table 1.79 Execute the Triggering command

| | |
|---|---|
| Prototype | int SetTRIGCmd(TRIGCmd *trigCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Execute the triggering command.<br>This command must be added to the command queue, namely, **isQueued** must be set to **1**. If not, the parameter **condition** of the **Triggering** command in the queue command being executed may be changed because the **TRIGCmd** memory is shared |
| Parameter | TRIGCmd:<br>typedef struct tagTRIGCmd {<br>uint8_t address; // EIO address:**1-20**<br>uint8_t mode; //Triggering mode. **0**: Level trigger.**1**:A/D |

| | |
|---|---|
| | trigger<br>uint8_t condition; //Triggering condition<br>Level: **0**, equal. **1**, unequal<br>A/D: **0**, less than. **1**,less than or equal **2**, greater than or equal. **3**, greater than<br>uint16_t threshold; //Triggering threshold. Level : **0**,**1** .A/D: **0-4095**<br>}TRIGCmd;<br>trigCmd: TRIGCmd pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.18 EIO

In the Dobot controller, the addresses of the I/O interfaces are unified. Here, you can see as follows:

- High-low level output;
- PWM output;
- Read High-low level output;
- Read analog-digital conversion value output.

Some of the I/Os may have all the functions listed above. You need configure I/O multiplexing when using different functions. For more details, please see *Dobot Magician User Guide.*

### 1.18.1 Setting the I/O Multiplexing

Table 1.80 Set the I/O multiplexing

| | |
|---|---|
| Prototype | int SetIOMultiplexing(IOMultiplexing ioMultiplexing, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Sets the I/O multiplexing. When using any I/O interface, you need to call this API to set the I/O multiplexing |
| Parameter | IOMultiplexing:<br>typedef struct tagIOMultiplexing {<br>uint8_t address; //I/O address:**1-20**<br>uint8_t multiplex; //I/O multiplexing function: **0-6** |

| | |
|---|---|
| | }IOMultiplexing;<br>The values supported by **multiplex** are shown as follows:<br>typedef enum tagIOFunction {<br>IOFunctionDummy; //Invalid<br>IOFunctionDO; // I/O output<br>IOFunctionPWM; // PWM output<br>IOFunctionDI; //I/O input<br>IOFunctionADC; //A/D input<br>IOFunctionDIPU; //Pull-up input<br>IOFunctionDIPD //Pull-down input<br>}IOFunction;<br>ioMultiplexing: IOMultiplexing pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.2 Getting the I/O multiplexing

Table 1.81 Getting the I/O multiplexing

| | |
|---|---|
| Prototype | int GetIOMultiplexing(IOMultiplexing *ioMultiplexing) |
| Description | Get the I/O multiplexing |
| Parameter | IOMultiplexing:<br>typedef struct tagIOMultiplexing {<br>uint8_t address; //I/O address:**1-20**<br>uint8_t multiplex; //I/O multiplexing function: **0-6**<br>}IOMultiplexing;<br>The values supported by **multiplex** are as follows.<br>typedef enum tagIOFunction {<br>IOFunctionDummy; //Invalid<br>IOFunctionDO; // I/O output<br>IOFunctionPWM; // PWM output<br>IOFunctionDI; //I/O input |

| | |
|---|---|
| | IOFunctionADC; //A/D input<br>IOFunctionDIPU; //Pull-up input<br>IOFunctionDIPD //Pull-down input<br>}IOFunction;<br>ioMultiplexing: IOMultiplexing pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.3 Setting the I/O Output

Table 1.82 Set the I/O output

| | |
|---|---|
| Prototype | int SetIODO(IODO *ioDO, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the I/O output |
| Parameter | IODO:<br>typedef struct tagIODO {<br>uint8_t address; //I/O addres:**1-20**<br>uint8_t level; //**0**: Low level.**1**: High level<br>}IODO;<br>ioDO: IODO pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.4 Getting the I/O Output

Table 1.83 Get the I/O output

| | |
|---|---|
| Prototype | int GetIODO(IODO *ioDO) |
| Description | Get the I/O output |
| Parameter | IODO:<br>typedef struct tagIODO {<br>uint8_t address; //I/O addres:1-20 |

 

| | |
|---|---|
| | uint8_t level; //**0**: Low level.**1**: High level<br>}IODO;<br>ioDO: IODO pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.5 Setting the PWM Output

Table 1.84 Set PWM output

| | |
|---|---|
| Prototype | int SetIOPWM(IOPWM *ioPWM, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the PWM output |
| Parameter | IOPWM:<br>typedef struct tagIOPWM {<br>uint8_t address; //I/O address:**1-20**<br>float frequency; // PWM frequency: **10**Hz-**1**MHz<br>float dutyCycle; // PWM duty cycle: **0-100**<br>}IOPWM;<br>ioPWM: IOPWM pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.6 Getting the PWM Output

Table 1.85 Get the PWM output

| | |
|---|---|
| Prototype | int GetIOPWM(IOPWM *ioPWM) |
| Description | Get the PWM output |
| Parameter | IOPWM:<br>typedef struct tagIOPWM {<br>uint8_t address; //I/O address:**1-20** |

| | |
|---|---|
| | float frequency; // PWM frequency: **10**Hz-**1**MHz<br>float dutyCycle; // PWM duty cycle: **0-100**<br>}IOPWM;<br>ioPWM: IOPWM pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.7 Getting the I/O Input

Table 1.86 Get the I/O input

| | |
|---|---|
| Prototype | int GetIODI(IODI *ioDI) |
| Description | Get the I/O input |
| Parameter | IODI:<br>typedef struct tagIODI {<br>uint8_t address; //I/O address: **1-20**<br>uint8_t level; //**0**: Low level. **1**: High-level<br>}IODI;<br>ioDI: IODO pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.8 Getting the A/D Input

Table 1.87 Get the A/D Input

| | |
|---|---|
| Prototype | int GetIOADC(IOADC *ioADC) |
| Description | Get the A/D input |
| Parameter | IOADC:<br>typedef struct tagIOADC {<br>uint8_t address; //I/O address: **1-20**<br>uint16_t value; //Input value:**0-4095**<br>}IOADC;<br>ioADC: IOADC pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a |

| | |
|---|---|
| | timeout |

### 1.18.9 Setting the Velocity of the Extended Motor

Table 1.88 Set the velocity of the extended motor

| | |
|---|---|
| Prototype | int SetEMotor(EMotor *eMotor, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity of the extended motor. The motor will always be operated at a constant velocity after calling this API |
| Parameter | EMotor:<br>typedef struct tagEMotor {<br>uint8_t index; //Motor index. **0**: Stepper1. **1**:Stepper2<br>uint8_t isEnabled; //Control motor. **0**: Disabled. **1**: Enabled<br>uint32_t speed; //Motor velocity (Pulse number per second)<br>}EMotor;<br>eMotor: EMotor pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.10 Setting the Velocity of the Extended Motor and the Movement Distance

Table 1.89 Set the velocity of extended motor and the movement distance

| | |
|---|---|
| Prototype | int SetEMotorS(EMotorS *eMotorS, bool isQueued, uint64_t *queuedCmdIndex) |
| Description | Set the velocity of the extended motor and the movement distance. The Dobot will move for some distance at a constant velocity after calling this API |
| Parameter | EMotorS:<br>typedef struct tagEMotorS{<br>uint8_t index; //Motor index. **0**: Stepper1. **1**:Stepper2<br>uint8_t isEnabled; //Control motor. **0**: Disabled. **1**: Enabled<br>uint32_t speed; //Motor velocity (Pulse number per second)<br>uint32_t distance //Movement distance (Pulse number) |

 

| | |
|---|---|
| | }EMotorS;<br>eMotorS: EMotorS pointer<br>isQueued: Whether to add this command to the queue<br>queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_BufferFull: The command queue is full<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.11 Enabling the Photoelectric Sensor

Table 1.90 Enable the photoelectric sensor

| | |
|---|---|
| Prototype | int SetInfraredSensor(bool enable,InfraredPort infraredPort) |
| Description | Enable the photoelectric sensor |
| Parameter | InfraredPort:<br>enum InfraredPort {<br>IF_PORT_GP1;<br>IF_PORT_GP2;<br>IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>enable: **0**, Disabled. **1**, Enabled<br>infraredPort: The Dobot interface that the photoelectric sensor is connected to. Please select the corresponding interface |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.12 Getting the Photoelectric Sensor Value

Table 1.91 Get the photoelectric sensor value

| | |
|---|---|
| Prototype | int GetInfraredSensor (InfraredPort infraredPort, uint8_t *value) |
| Description | Get the photoelectric sensor value |
| Parameter | InfraredPort:<br>enum InfraredPort { |

| | |
|---|---|
| | IF_PORT_GP1;<br>IF_PORT_GP2;<br>IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>infraredPort: The Dobot interface that the photoelectric sensor is connected to. Please select the corresponding interface<br>value: The value of the photoelectric sensor |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.13 Enabling the Color Sensor

Table 1.92 Enable the color sensor

| | |
|---|---|
| Prototype | int SetColorSensor(bool enable,ColorPort colorPort) |
| Description | Enable the color sensor |
| Parameter | ColorPort:<br>enum ColorPort {<br>IF_PORT_GP1;<br>IF_PORT_GP2;<br>IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>enable: **0**, Disabled. **1**, Enabled<br>colorPort: The Dobot interface that the color sensor is connected to. Please select the corresponding interface |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.18.14 Getting the Color Sensor Value

Table 1.93 Get the color sensor value

| | |
|---|---|
| Prototype | int GetColorSensor(ColorPort colorPort, uint8_t *value) |
| Description | Get the color sensor value |

| | |
|---|---|
| Parameter | ColorPort:<br>enum ColorPort {<br>IF_PORT_GP1;<br>IF_PORT_GP2;<br>IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>ColorPort: The Dobot interface that the color sensor is connected. Please select the corresponding interface<br>value: The value of the color sensor |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.19 CAL

The Angle sensors on the Forearm and Rear Arm may have static errors due to angle sensor welding, device status, etc. It is possible to get this static error through various means (such as leveling, compared with the standard source), and write into the device through this API.

Forearm/Rear Arm angle = angle sensor static error of Forearm/Rear Arm + angle senor value of Forearm/Rear Arm *Linearization parameter of Forearm/Rear Arm angle sensor

Base angle = Static error of Base Encoder + Base Encoder value

### 1.19.1 Setting the Angle Sensor Static Error

Table 1.94 Set the angle sensor static error

| | |
|---|---|
| Prototype | int SetAngleSensorStaticError(float rearArmAngleError, float frontArmAngleError) |
| Description | Set the angle sensor static errors of Forearm and Rear Arm |
| Parameter | rearArmAngleError: The angle sensor static error of the Rear Arm<br>frontArmAngleError: The angle sensor static error of the Forearm |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.19.2 Getting the Angle Sensor Static Error

Table 1.95 Get the angle sensor static error

| | |
|---|---|
| Prototype | int GetAngleSensorStaticError(float *rearArmAngleError, float *frontArmAngleError) |
| Description | Get the angle sensor static errors of the Forearm and Rear Arm |
| Parameter | rearArmAngleError: The angle sensor static error of the Rear Arm<br>frontArmAngleError: The angle sensor static error of the Forearm |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.19.3 Setting the Linearization Parameter of the Angle Sensor

Table 1.96 Set the linearization parameter of the angle sensor

| | |
|---|---|
| Prototype | int SetAngleSensorStaticCoef(float rearArmAngleCoef, float frontArmAngleCoef) |
| Description | Set the linearization parameter of the angle sensor |
| Parameter | rearArmAngleCoef : The linearization parameter of the Rear Arm angle sensor<br>frontArmAngleCoef : The linearization parameter of the Forearm angle sensor |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.19.4 Getting the Linearization Parameter of the Angle Sensor

Table 1.97 Get the linearization parameter of the angle sensor

| | |
|---|---|
| Prototype | int GetAngleSensorStaticCoef(float *rearArmAngleCoef, float *frontArmAngleCoef) |
| Description | Get the linearization parameter of the angle sensor |
| Parameter | rearArmAngleCoef : The linearization parameter of the Rear Arm angle sensor<br>frontArmAngleCoef : The linearization parameter of the Forearm angle sensor |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a |

| | |
|---|---|
| | timeout |

### 1.19.5 Setting the Static Error of the Base Encoder

Table 1.98 Set static error of the base Encoder

| | |
|---|---|
| Prototype | int SetBaseDecoderStaticError(float baseDecoderError) |
| Description | Set the static error of the base Encoder |
| Parameter | baseDecoderError: The static error of the base Encoder |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.19.6 Getting the Static Error of the Base Encoder

Table 1.99 Get the static error of the base Encoder

| | |
|---|---|
| Prototype | int GetBaseDecoderStaticError (float baseDecoderError) |
| Description | Get the static error of the base Encoder |
| Parameter | baseDecoderError: The static error of the base Encoder |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.20 WIFI

The Dobot can be connected to a Computer via a WIFI module. After the WIFI module is connected to the Dobot, you need to set the IP address, Sub netmask, Gateway and enable WIFI to make the Dobot access WLAN. After the access is successful, you can connect your Dobot to your Computer without using a USB cable.

### 1.20.1 Enabling WIFI

Table 1.100 Enable WIFI

| | |
|---|---|
| Prototype | int SetWIFIConfigMode(bool enable) |
| Description | Enable WIFI |
| Parameter | enable: **0**, Disabled. **1**,Enabled |
| Return | DobotCommunicate_NoError: The command returns with no error |

| | |
|---|---|
| | DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.2 Getting the WIFI Status

Table 1.101 Get the WIFI Status

| | |
|---|---|
| Prototype | int GetWIFIConfigMode(bool *isEnabled) |
| Description | Get the WIFI status |
| Parameter | isEnabled: **0**, Disabled. **1**,Enabled |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.3 Setting the SSID

SSID (Service Set Identifier): WIFI network name.

Table 1.102 Set the SSID

| | |
|---|---|
| Prototype | int SetWIFISSID(const char *ssid) |
| Description | Set the SSID |
| Parameter | ssid: String pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.4 Getting the SSID

Table 1.103 Get the SSID

| | |
|---|---|
| Prototype | int GetWIFISSID(char *ssid, uint32_t maxLen) |
| Description | Get the SSID |
| Parameter | ssid: String pointer<br>maxLen: Maximum String length, to avoid overflow |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.5 Setting the Network Password

Table 1.104 Set the Network password

| | |
|---|---|
| Prototype | int SetWIFIPassword(const char *password) |
| Description | Set the network password |
| Parameter | password: String pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.6 Getting the Network Password

Table 1.105 Get the Network password

| | |
|---|---|
| Prototype | int GetWIFIPassword(char *password, uint32_t maxLen) |
| Description | Get the network password |
| Parameter | password: String pointer<br>maxLen: Maximum String length, to avoid overflow |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.7 Setting the IP Address

Table 1.106 Set the IP Address

| | |
|---|---|
| Prototype | int SetWIFIIPAddress(WIFIIPAddress *wifiIPAddress) |
| Description | Set the IP address |
| Parameter | WIFIIPAddress<br>typedef struct tagWIFIIPAddress {<br>uint8_t dhcp; //Whether to enable DHCP. **0**: Disabled**1**:Enabled<br>uint8_t addr[4]; // The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFIIPAddress;<br>wifiIPAddr: WIFIIPAddress pointer |
| Return | DobotCommunicate_NoError: The command returns with no error |

| | |
|---|---|
| | DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.8 Getting the IP Address

Table 1.107 Get the IP Address

| | |
|---|---|
| Prototype | int GetWIFIIPAddress(WIFIIPAddress *wifiIPAddress) |
| Description | Get the IP address |
| Parameter | WIFIIPAddress<br>typedef struct tagWIFIIPAddress {<br>uint8_t dhcp; //Whether to enable DHCP. **0**: Disabled**1**:Enabled<br>uint8_t addr[4]; // The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFIIPAddress;<br>wifiIPAddr: WIFIIPAddress pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.9 Setting the Sub Netmask

Table 1.108 Set the sub netmask

| | |
|---|---|
| Prototype | int SetWIFINetmask(WIFINetmask *wifiNetmask) |
| Description | Set the sub netmask |
| Parameter | WIFINetmask<br>typedef struct tagWIFINetmask {<br>uint8_t addr[4]; //The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFINetmask;<br>wifiNetmask: WIFINetmask pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.10 Getting the Sub Netmask

Table 1.109 Get the sub netmask

| | |
|---|---|
| Prototype | int GetWIFINetmask(WIFINetmask *wifiNetmask) |
| Description | Get the sub netmask |
| Parameter | WIFINetmask<br>typedef struct tagWIFINetmask {<br>uint8_t addr[4]; //The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFINetmask;<br>wifiNetmask: WIFINetmask pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.11 Setting the Gateway

Table 1.110 Set the gateway

| | |
|---|---|
| Prototype | int SetWIFIGateway(WIFIGateway *wifiGateway) |
| Description | Set the gateway |
| Parameter | WIFIGateway<br>typedef struct tagWIFIGateway {<br>uint8_t addr[4]; //The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFIGateway;<br>wifiGateway: WIFIGateway pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.12 Getting the Gateway

Table 1.111 Get the gateway

| | |
|---|---|
| Prototype | int GetWIFIGateway(WIFIGateway *wifiGateway) |

| | |
|---|---|
| Description | Gets the gateway |
| Parameter | WIFIGateway<br>typedef struct tagWIFIGateway {<br>uint8_t addr[4]; //The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFIGateway;<br>wifiGateway: WIFIGateway pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.13 Setting the DNS

Table 1.112 Set the DNS

| | |
|---|---|
| Prototype | int SetWIFIDNS(WIFIDNS *wifiDNS) |
| Description | Set the DNS |
| Parameter | WIFIDNS<br>typedef struct tagWIFIDNS {<br>uint8_t addr[4]; //The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFIDNS;<br>wifiDNS: WIFIDNS pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.14 Getting the DNS

Table 1.113 Get the DNS

| | |
|---|---|
| Prototype | int GetWIFIDNS(WIFIDNS *wifiDNS) |
| Description | Get the DNS |
| Parameter | WIFIDNS<br>typedef struct tagWIFIDNS { |

 

| | |
|---|---|
| | uint8_t addr[4]; //The IP address is divided into 4 segments, the value range of each segment is 0-255<br>}WIFIDNS;<br>wifiDNS: WIFIDNS pointer |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

### 1.20.15 Getting the WIFI Connection Status

Table 1.114 Get the WIFI connection status

| | |
|---|---|
| Prototype | int GetWIFIConnectStatus(bool *isConnected) |
| Description | Get the WIFI connection status |
| Parameter | isConnected: **0,** Non-connected. **1**, Connected |
| Return | DobotCommunicate_NoError: The command returns with no error<br>DobotCommunicate_Timeout: The command does not return, resulting in a timeout |

## 1.21 Other functions

### 1.21.1 Event Loop

In some languages, the application exits directly after calling an API because there is no event loop, resulting in the command unable to be issued to the Dobot controller. To avoid this, we provide an event loop API, which is called before the application exits (currently known, Python need to follow this).

Table 1.115 Event loop

| | |
|---|---|
| Prototype | void DobotExec(void) |
| Description | Event loop |
| Parameter | None |
| Return | Void |