



# Dobot Magician API Description

---

Issue: V1.2.3

Date: 2019-07-19

**Copyright © ShenZhen Yuejiang Technology Co., Ltd 2018. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of 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: 3F, Building NO.3, Tongfuyu Industrial Town, Nanshan District, Shenzhen, China

Website: [www.dobot.cc](http://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
2019/07/19	<ul style="list-style-type: none"> <li>• Add a note for API return result</li> <li>• Add the description of the relationship between the speed and speed rate</li> <li>• Adjust the content structure</li> </ul>
2019/05/05	Add a note for API SetJOGCmd
2018/11/06	Modify some mistakes of API function
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

<b>1. Dobot Commands.....</b>	<b>1</b>
<b>2. Command Timeout .....</b>	<b>2</b>
2.1 Setting Command Timeout .....	2
<b>3. Connect/Disconnect.....</b>	<b>3</b>
3.1 Searching for the Dobot .....	3
3.2 Connecting to the Dobot .....	3
3.3 Disconnecting the Dobot.....	4
3.4 Demo: Connection Example .....	4
<b>4. Command queue controlling.....</b>	<b>6</b>
4.1 Starting Command in Command queue .....	6
4.2 Stopping Command in Command queue .....	6
4.3 Stopping Command in Command queue Forcedly .....	6
4.4 Demo: Processing PTP Command and Control Queue Synchronously .....	7
4.5 Demo: Processing PTP Command and Controlling Queue Asynchronously .....	8
4.6 Downloading Commands.....	9
4.7 Stopping Downloading Commands. ....	9
4.8 Demo: Downloading PTP Command.....	9
4.9 Clearing Command queue.....	10
4.10 Getting Command Index .....	10
4.11 Demo: Checking Whether the Commands Have Been Executed .....	11
<b>5. Device Information .....</b>	<b>13</b>
5.1 Setting the Device Serial Number.....	13
5.2 Getting the Device Serial Number .....	13
5.3 Setting the Device Name .....	13
5.4 Getting the Device Name .....	13
5.5 Getting the Device Version .....	14
5.6 Setting the Sliding Rail Status .....	14
5.7 Getting the Sliding Rail Status.....	15
5.8 Getting the Device Clock.....	15
<b>6. Real-time pose .....</b>	<b>16</b>
6.1 Getting the Real-time Pose of the Dobot .....	16
6.2 Getting the Real-time Pose of the Sliding Rail .....	16
6.3 Resetting the Reference Value of the Real-time Pose .....	17
<b>7. ALARM .....</b>	<b>18</b>
7.1 Getting the Alarm Status .....	18
7.2 Clearing the Statuses of All Alarms .....	18
<b>8. Homing Function .....</b>	<b>19</b>
8.1 Setting the Homing Position .....	19
8.2 Getting the Homing Position .....	19
8.3 Executing the Homing Function .....	20
8.4 Executing the Automatic Leveling Function.....	20
8.5 Getting the Automatic Leveling Results .....	21

<b>9. HHT Function .....</b>	<b>22</b>
9.1 Setting the Hand-Hold Teaching Trigger Mode .....	22
9.2 Getting the Hand-Hold Teaching Trigger Mode .....	22
9.3 Setting the Status of the Hand-Hold Teaching Function .....	23
9.4 Getting the Status of the Hand-Hold Teaching Function .....	23
9.5 Getting the Hand-Hold Teaching Trigger Single .....	23
9.6 Demo: Hand-Hold Teaching .....	23
<b>10. End-effector .....</b>	<b>25</b>
10.1 Setting the Offset of the End-effector .....	25
10.2 Getting the Offset of the End-effector .....	25
10.3 Setting the Status of the Laser.....	26
10.4 Getting the Status of the Laser.....	26
10.5 Setting the Status of the Air Pump .....	26
10.6 Getting the Status of the Air Pump .....	27
10.7 Setting the Status of the Gripper.....	27
10.8 Getting the Status of the Gripper .....	28
<b>11. JOG .....</b>	<b>29</b>
11.1 Setting the Velocity and Acceleration of the Joint Coordinate Axis when Jogging .....	29
11.2 Getting the Velocity and Acceleration of the Joint Coordinate Axis when Jogging .....	29
11.3 Setting the velocity and acceleration of the Cartesian Coordinate Axis when Jogging ..	30
11.4 Getting the velocity and acceleration of the Cartesian Coordinate Axis when Jogging ..	30
11.5 Setting the velocity and acceleration of the Sliding Rail when Jogging .....	31
11.6 Getting the velocity and acceleration of the Sliding Rail when Jogging .....	31
11.7 Setting the Velocity Ratio and Acceleration Ratio when Jogging.....	32
11.8 Getting the Velocity Ratio and Acceleration Ratio when Jogging .....	33
11.9 Executing the Jogging Command .....	33
<b>12. PTP .....</b>	<b>35</b>
12.1 Setting the Velocity and Acceleration of the Joint Coordinate Axis in PTP Mode .....	36
12.2 Getting the Velocity and Acceleration of the Joint Coordinate Axis in PTP Mode.....	36
12.3 Setting the Velocity and Acceleration of the Cartesian Coordinate Axis in PTP Mode ..	37
12.4 Getting the Velocity and Acceleration of the Cartesian Coordinate Axis in PTP Mode ..	37
12.5 Setting the Lifting Height and the Maximum Lifting Height in JUMP mode .....	38
12.6 Getting the Lifting Height and the Maximum Lifting Height in JUMP mode.....	38
12.7 Setting the Extended Parameters in JUMP mode.....	39
12.8 Getting the Extended Parameters in JUMP mode.....	39
12.9 Setting the Velocity and Acceleration of the Sliding Rail in PTP Mode.....	40
12.10 Getting the Velocity and Acceleration of the Sliding rail in PTP Mode.....	40
12.11 Setting the Velocity Ratio and Acceleration Ratio in PTP Mode .....	41
12.12 Getting the Velocity Ratio and Acceleration Ratio in PTP Mode.....	41
12.13 Executing a PTP Command .....	42
12.14 Executing a PTP Command with the I/O Control .....	43
12.15 Executing a PTP Command with the Sliding Rail .....	45
12.16 Executing a PTP Command with the Sliding Rail and I/O Control .....	46
<b>13. CP .....</b>	<b>49</b>

13.1	Setting the Velocity and Acceleration in CP Mode .....	49
13.2	Getting the Velocity and Acceleration in CP Mode .....	49
13.3	Executing the CP Command .....	50
13.4	Executing the CP Command with the Laser Engraving.....	51
<b>14.</b>	<b>ARC .....</b>	<b>53</b>
14.1	Setting the Velocity and Acceleration in ARC Mode.....	53
14.2	Getting the Velocity and Acceleration in ARC Mode .....	53
14.3	Executing the ARC Command.....	54
14.4	Executing the CIRCLE Command.....	55
<b>15.</b>	<b>Losing-Step Detection .....</b>	<b>57</b>
15.1	Setting the losing-step threshold .....	57
15.2	Executing the Losing-Step Command .....	57
15.3	Demo: Executing the Losing-Step Command.....	57
<b>16.</b>	<b>WAITING .....</b>	<b>59</b>
16.1	Executing the Waiting Command .....	59
<b>17.</b>	<b>TRIGGERING .....</b>	<b>60</b>
17.1	Executing the Triggering Command .....	60
<b>18.</b>	<b>EIO .....</b>	<b>61</b>
18.1	Setting the I/O Multiplexing .....	61
18.2	Getting the I/O multiplexing.....	62
18.3	Setting the I/O Output.....	62
18.4	Getting the I/O Output .....	63
18.5	Setting the PWM Output.....	63
18.6	Getting the PWM Output .....	64
18.7	Getting the I/O Input .....	64
18.8	Getting the A/D Input.....	65
18.9	Setting the Velocity of the Extended Motor.....	65
18.10	Setting the Velocity of the Extended Motor and the Movement Distance .....	66
18.11	Enabling the Photoelectric Sensor.....	66
18.12	Getting the Photoelectric Sensor Value .....	67
18.13	Enabling the Color Sensor.....	67
18.14	Getting the Color Sensor Value .....	68
<b>19.</b>	<b>CAL .....</b>	<b>69</b>
19.1	Setting the Angle Sensor Static Error .....	69
19.2	Getting the Angle Sensor Static Error .....	69
19.3	Setting the Linearization Parameter of the Angle Sensor .....	69
19.4	Getting the Linearization Parameter of the Angle Sensor.....	70
19.5	Setting the Static Error of the Base Encoder.....	70
19.6	Getting the Static Error of the Base Encoder .....	70
<b>20.</b>	<b>WIFI .....</b>	<b>72</b>
20.1	Enabling WIFI .....	72
20.2	Getting the WIFI Status .....	72
20.3	Setting the SSID.....	72
20.4	Getting the SSID .....	73

---

20.5 Setting the Network Password .....	73
20.6 Getting the Network Password .....	73
20.7 Setting the IP Address .....	73
20.8 Getting the IP Address .....	74
20.9 Setting the Sub Netmask .....	74
20.10 Getting the Sub Netmask.....	74
20.11 Setting the Gateway .....	75
20.12 Getting the Gateway.....	76
20.13 Setting the DNS.....	76
20.14 Getting the DNS.....	77
20.15 Getting the WIFI Connection Status .....	77
<b>21. Other functions.....</b>	<b>78</b>
21.1 Event Loop.....	78

## 1. Dobot Commands

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

- Immediate 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*.



## 2. Command Timeout

### 2.1 Setting Command Timeout

As described in *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 2.1 Set timeout

Prototype	<code>void SetCmdTimeout( unsigned int cmdTimeout)</code>
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

#### NOTE

API returns result as an enumerate type. You can view them in the **DobotType.h** file.

### 3. Connect/Disconnect

#### 3.1 Searching for the Dobot

Table 3.1 Search for the Dobot

Prototype	<code>int SearchDobot(char *dobotNameList, uint32_t maxLen)</code>
Description	Search for Dobot, DLL will store the information of Dobot that has been searched for and use <code>ConnectDobot</code> to connect the searched Dobot
Parameter	<p><code>dobotNameList</code>: String pointer, DLL will write serial port/UDP searched into <code>dobotNameList</code>. For example, a specific <code>dobotNameList</code> is "<b>COM1 COM3 COM6 192.168.0.5</b>", different serial port or IP address should be separated by the space</p> <p><code>maxLen</code>: Maximum String length, to avoid memory overflow</p>
Return	The number of Dobot

#### 3.2 Connecting to the Dobot

Table 3.2 Connect to the Dobot

Prototype	<code>int ConnectDobot(const char *portName, uint32_t baudrate, char *fwType, char *version float *time)</code>
Description	<p>Connecting to the Dobot. In this process, <code>portName</code> can be obtained from <code>dobotList</code> in the <code>SearchDobot(char *dobotList, uint32_t maxLen)</code> API.</p> <p>If <code>portName</code> is empty, and <code>ConnectDobot</code> is called directly, DLL will connect the random searched Dobot automatically</p>
Parameter	<p><code>portName</code>: Dobot port. As for the serial port, <code>portName</code> is <b>COM3</b>; While for UDP, <code>portName</code> is <b>192.168.0.5</b></p> <p><code>baudrate</code>: Baud rates. Value: 115200</p> <p><code>fwType</code>: Firmware type. Dobot or Marlin</p> <p><code>version</code>: Version</p> <p><code>time</code>: Timeout</p>
Return	<p><code>DobotConnect_NoError</code>: The connection is successful</p> <p><code>DobotConnect_NotFound</code>: Dobot interface was not found</p> <p><code>DobotConnect_Occupied</code>: Dobot interface is occupied or unavailable</p>

#### 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*.

### 3.3 Disconnecting the Dobot

Table 3.3 Disconnect the Dobot

Prototype	<code>void DisconnectDobot(void)</code>
Description	Disconnect the Dobot
Parameter	None
Return	DobotConnect_NoError :There is no error

### 3.4 Demo: Connection Example

Program 3.1 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;
}
```

## 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.

### 4.1 Starting Command in Command queue

Table 4.1 Start command in command queue

Prototype	<code>int SetQueuedCmdStartExec(void)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 4.2 Stopping Command in Command queue

Table 4.2 Stop command in command queue

Prototype	<code>int SetQueuedCmdStopExec(void)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 4.3 Stopping Command in Command queue Forcedly

Table 4.3 Stop command in command queue forcedly

Prototype	<code>int SetQueuedCmdForceStopExec(void)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout (aka error)

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

For details about PTP, please refer to *12 PTP*.

Program 4.1 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();
}
```

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

Program 4.2 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);
}
```

## 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 4.4 Download commands

Prototype	<code>int SetQueuedCmdStartDownload(uint32_t totalLoop, uint32_t linePerLoop)</code>
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 linePerLoop: loops of per command in offline mode. The number of the issued commands must be the same as <b>linePerLoop</b> . The issued commands should be added to the command queue.
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 4.7 Stopping Downloading Commands.

Table 4.5 Stop to download commands

Prototype	<code>int SetQueuedCmdStopDownload(void)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 4.8 Demo: Downloading PTP Command

Program 4.3 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.

#### 4.9 Clearing Command queue

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

Table 4.6 Clear command queue

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

#### 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 4.7 Get command index

Prototype	<code>int GetQueuedCmdCurrentIndex(uint64_t *queuedCmdCurrentIndex)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Program 4.4 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();  
}
```

## 5. Device Information

### 5.1 Setting the Device Serial Number

Table 5.1 Set the device serial number

Prototype	<code>int SetDeviceSN(const char *deviceSN)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 5.2 Getting the Device Serial Number

Table 5.2 Get the device serial number

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

### 5.3 Setting the Device Name

Table 5.3 Set the device name

Prototype	<code>int SetDeviceName(const char *deviceName)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 5.4 Getting the Device Name

Table 5.4 Get the device name

Prototype	<code>int GetDeviceName(char *deviceName, uint32_t maxLen)</code>
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 maxLen: Maximum string length, to avoid overflow
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 5.5 Getting the Device Version

Table 5.5 Get the device version

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

## 5.6 Setting the Sliding Rail Status

Table 5.6 Set the sliding rail status

Prototype	<code>int SetDeviceWithL(bool isEnabled, bool isQueued, uint64_t *queuedCmdIndex, uint8_t version)</code>
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	isEnabled: <b>0</b> , Disabled. <b>1</b> , Enabled isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid. version: Version flag of sliding rail. <b>0</b> : The version is <b>V1.0</b> . <b>1</b> : The version is

	<b>V2.0</b>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 5.7 Getting the Sliding Rail Status

Table 5.7 Get the sliding rail status

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

## 5.8 Getting the Device Clock

Table 5.8 Get the device clock

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

## 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.

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

Table 6.1 Get the real-time pose of Dobot

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

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

Table 6.2 Get the real-time pose of sliding rail

Prototype	<code>int GetPose(Pose *pose)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a

timeout
---------

### 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 6.3 Reset the reference value of the real-time pose

Prototype	<code>int ResetPose(bool manual, float rearArmAngle, float frontArmAngle)</code>
Description	Reset the reference value of the real-time pose
Parameter	<p>manual: Indicate whether to reset reference value of real-time pose automatically. <b>0</b>, reset the reference value automatically and <b>rearArmAngle</b> and <b>frontArmAngle</b> are not to set. <b>1</b>, <b>rearArmAngle</b> and <b>frontArmAngle</b> need to be set</p> <p>rearArmAngle: Rear Arm angle</p> <p>frontArmAngle: Forearm angle</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>



## 7. ALARM

### 7.1 Getting the Alarm Status

Table 7.1 Get the alarm status

Prototype	<code>int GetAlarmsState(uint8_t *alarmsState, uint32_t *len, uint32_t maxLen)</code>
Description	Get the alarm status
Parameter	<p><code>alarmsState</code>: The first address of the array. Each byte in the array <b>alarmsState</b> 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.</p> <p><code>len</code>: The byte occupied by the alarm.</p> <p><code>maxLen</code>: Maximum array length, to avoid overflow</p>
Return	<p><code>DobotCommunicate_NoError</code>: The command returns with no error</p> <p><code>DobotCommunicate_Timeout</code>: The command does not return, resulting in a timeout</p>

### 7.2 Clearing the Statuses of All Alarms

Table 7.2 Clear the statuses of all alarms

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

## 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.

### 8.1 Setting the Homing Position

Table 8.1 Set the homing position

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

### 8.2 Getting the Homing Position

Table 8.2 Get the homing position

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

	<pre>float r;           //Cartesian coordinate system R-axis }HOMEParams; homeParams: HOMEParams pointer</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 8.3 Executing the Homing Function

Table 8.3 Execute the homing function

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

### 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 8.4 Execute the Automatic leveling function

Prototype	<pre>int SetAutoLevelingCmd(AutoLevelingCmd *autoLevelingCmd, bool isQueued, uint64_t *queuedCmdIndex)</pre>
-----------	--------------------------------------------------------------------------------------------------------------

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

## 8.5 Getting the Automatic Leveling Results

Table 8.5 Get the automatic leveling results

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

## 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.

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

Table 9.1 Set the hand-hold teaching mode

Prototype	<code>int SetHHTTrigMode (HHTTrigMode hhtTrigMode)</code>
Description	Set the hand-hold teaching triggering mode. If this API is not called, Dobot will save points when releasing the <b>UNLOCK</b> key on Forearm
Parameter	<p>HHTTrigMode:</p> <pre>typedef enum tagHHTTrigMode {     TriggeredOnKeyReleased,           //Trigger when releasing the                                      UNLOCK key     TriggeredOnPeriodicInterval      //Trigger when pressing the                                      UNLOCK key }HHTTrigMode; hhtTrigMode: HHTTrigMode enum</pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 9.2 Get the hand-hold teaching trigger mode

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

timeout	
---------	--

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

Table 9.3 Set the status of the hand-hold teaching function

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

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

Table 9.4 Get the status of the hand-hold teaching function

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

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

Table 9.5 Get the hand-hold teaching trigger single

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

### 9.6 Demo: Hand-Hold Teaching

## Program 9.1 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();
}
```

## 10. End-effector

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

Table 10.1 Set the offset of the end-effector

Prototype	<code>int SetEndEffectorParams(EndEffectorParams *endEffectorParams, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	<p>Set the offset of the end-effector. If the end-effector is installed, this API is required</p> <p>If a standard end-effector is used, please refer to <i>Dobot Magician User Guide</i> to obtain the X-axis offset and Y-axis offset and call this API. Otherwise, please confirm the structural parameters.</p>
Parameter	<p>EndEffectorParams:</p> <pre>typedef struct tagEndEffectorParams {     float xBias;           //X-axis offset of end-effector     float yBias;           //Y-axis offset of end-effector     float zBias;           //Z-axis offset of end-effector }EndEffectorParams;</pre> <p>endEffectorParams: EndEffectorParams pointer</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid.</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 10.2 Get offset of end-effector

Prototype	<code>int GetEndEffectorParams(EndEffectorParams *endEffectorParams)</code>
Description	Get the offset of the end-effector
Parameter	<p>EndEffectorParams:</p> <pre>typedef struct tagEndEffectorParams {     float xBias;           //X-axis offset of end-effector     float yBias;           //Y-axis offset of end-effector     float zBias;           //Z-axis offset of end-effector }</pre>



	}EndEffectorParams; endEffectorParams: EndEffectorParams pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 10.3 Setting the Status of the Laser

Table 10.3 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. <b>0</b> : Disabled. <b>1</b> : Enabled on: Start or stop laser. <b>0</b> , Off. <b>1</b> , On isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid.
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 10.4 Getting the Status of the Laser

Table 10.4 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. <b>0</b> : Disabled. <b>1</b> : Enabled isOn: If the status of the laser is on. <b>0</b> , Off. <b>1</b> , On
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 10.5 Setting the Status of the Air Pump

Table 10.5 Set the status of the air pump

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

## 10.6 Getting the Status of the Air Pump

Table 10.6 Get the status of the air pump

Prototype	<code>int GetEndEffectorSuctionCup(bool *isCtrlEnabled, bool *isSucked)</code>
Description	Get the status of the air pump
Parameter	<p>isCtrlEnabled: If the status of the end-effector is enabled. <b>0</b>: Disabled. <b>1</b>: Enabled</p> <p>isSucked: If the status of the air pump is intake or outtake. <b>0</b>: Outtake. <b>1</b>: Intake</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 10.7 Setting the Status of the Gripper

Table 10.7 Set the status of the gripper

Prototype	<code>int SetEndEffectorGripper(bool enableCtrl, bool grip, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the status of the gripper
Parameter	<p>enableCtrl: Control end-effector. <b>0</b>: Disabled. <b>1</b>: Enabled</p> <p>grip: Control the gripper to grip or release. <b>0</b>: Released, <b>1</b>: Grabbed</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>

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

## 10.8 Getting the Status of the Gripper

Table 10.8 Get the status of the gripper

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

## 11. JOG

When doing jogging, the method calculating the velocity and acceleration for each axis (in Joint or Cartesian coordinate system) is shown as follows.

- Actual jogging velocity = the set jogging velocity \* the set jogging velocity rate
- Actual jogging acceleration = the set jogging acceleration \* the set jogging acceleration rate

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

Table 11.1 Set the velocity and acceleration of the joints coordinate axis when jogging

Prototype	<code>int SetJOGJointParams(JOGJointParams *jogJointParams, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the velocity( $^{\circ}$ /s) and acceleration ( $^{\circ}$ /s <sup>2</sup> ) of the joint coordinate axis when jogging
Parameter	<p>JOGJointParams:</p> <pre>typedef struct tagJOGJointParams {     float velocity[4];           //Joint velocity     float acceleration[4];       //Joint acceleration }JOGJointParams;</pre> <p>jogJointParams: JOGJointParams pointer</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 11.2 Get the velocity and acceleration of joint coordinate axis when jogging

Prototype	<code>int GetJOGJointParams(JOGJointParams *jogJointParams)</code>
Description	Get the velocity( $^{\circ}$ /s) and acceleration ( $^{\circ}$ /s <sup>2</sup> ) of the joint coordinate axis when jogging

Parameter	JOGJointParams: <pre>typedef struct tagJOGJointParams {     float velocity[4];           //Joint velocity     float acceleration[4];       //Joint acceleration }JOGJointParams; jogJointParams: JOGJointParams pointer</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 11.3 Set the velocity and acceleration of the Cartesian coordinate axis when jogging

Prototype	<pre>int SetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams, bool isQueued, uint64_t *queuedCmdIndex)</pre>
Description	Set the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the Cartesian coordinate axis when jogging
Parameter	JOGCoordinateParams: <pre>typedef struct tagJOGCoordinateParams {     float velocity[4];           //Cartesian coordinate axis (X,Y,Z,R)velocity     float acceleration[4];       //Cartesian coordinate axis (X,Y,Z,R)                                 acceleration }JOGCoordinateParams; jogCoordinateParams: JOGCoordinateParams pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid.</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 11.4 Get the velocity and acceleration of the Cartesian coordinate axis when jogging

Prototype	<pre>int GetJOGCoordinateParams(JOGCoordinateParams</pre>
-----------	-----------------------------------------------------------

	<code>*jogCoordinateParams)</code>
Description	Get the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the Cartesian coordinate axis when jogging
Parameter	<pre>typedef struct tagJOGCoordinateParams {     float velocity[4];        //Cartesian coordinate axis (X,Y,Z,R)velocity     float acceleration[4];    //Cartesian coordinate axis (X,Y,Z,R)                              acceleration }JOGCoordinateParams;  jogCoordinateParams: JOGCoordinateParams pointer</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 11.5 Set the velocity and acceleration of the sliding rail when jogging

Prototype	<code>int SetJOGLParams(JOGLParams *jogLParams, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the sliding rail when jogging
Parameter	JOGLParams: <pre>typedef struct tagJOGLParams {     float velocity;          //Sliding rail velocity     float acceleration;     //Sliding rail acceleration }JOGLParams;  jogLParams: JOGLParams  isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid.</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 11.6 Get the velocity and acceleration of the sliding rail when jogging

Prototype	<code>int GetJOGLParams(JOGLParams * jogLParams )</code>
Description	Get the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the sliding rail when jogging
Parameter	JOGLParams: <pre>typedef struct tagJOGLParams {     float velocity;        //Sliding rail velocity     float acceleration;    //Sliding rail acceleration }JOGLParams; jogLParams: JOGLParams</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 11.7 Set the velocity ratio and acceleration ratio when jogging

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

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

Table 11.8 Get the velocity ratio and acceleration ratio when jogging

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

## 11.9 Executing the Jogging Command

Table 11.9 Execute the Jogging command

Prototype	<code>int SetJOGCmd(JOGCmd *jogCmd, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Execute the Jogging command. Please call this API after setting the related parameters of jogging
Parameter	JOGCmd: <pre>typedef struct tagJOGCmd {     uint8_t isJoint;    //Jogging mode: 0, Jog in Cartesian coordinate system. 1, Jog in Joint coordinate system     uint8_t cmd;        //Jogging command: 0-10 }JOGCmd; //Details for jogging commands enum {     IDLE,    // Idle     AP_DOWN, // X+/Joint1+     AN_DOWN, // X-/Joint1-     BP_DOWN, // Y+/Joint2+     BN_DOWN, // Y-/Joint2-     CP_DOWN, // Z+/Joint3+</pre>



	<p>                     CN_DOWN, // Z-/Joint3-                      DP_DOWN, // R+/Joint4+                      DN_DOWN, // R-/Joint4-                      LP_DOWN, // L+. Only when the parameter isJoint=1, the LP_DOWN is available                      LN_DOWN // L-. Only when the parameter isJoint=1, the LN_DOWN is available                      };                      jogCmd: JOGCmd pointer                      isQueued: Whether to add this command to the queue                      queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid                 </p>
Return	<p>                     DobotCommunicate_NoError: The command returns with no error                      DobotCommunicate_BufferFull: The command queue is full                      DobotCommunicate_Timeout: The command does not return, resulting in a timeout                 </p>

## 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 12.1.

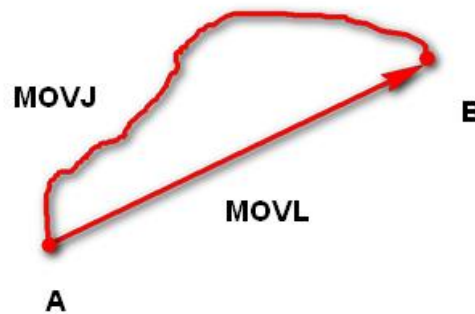


Figure 12.1 MOVL/MOVJ mode

- **MOVL:** Rectilinear movement. The joints will perform a straight line trajectory from point A to point B, as shown in Figure 12.1.
- **JUMP:** From point A to point B, the trajectory is shown in Figure 12.2., 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 12.2 JUMP mode

When doing playback, the method calculating the velocity and acceleration for each axis (in Joint or Cartesian coordinate system) is shown as follows.

- Actual playback velocity = the set playback velocity \* the set playback velocity rate
- Actual playback acceleration = the set playback acceleration \* the set playback acceleration rate

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

Table 12.1 Set the velocity and acceleration of the joint coordinate axis in PTP mode

Prototype	<code>int SetPTPJointParams(PTPJointParams *ptpJointParams, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the velocity( $^{\circ}/s$ ) and acceleration( $^{\circ}/s^2$ ) of the joint coordinate axis in PTP mode
Parameter	PTPJointParams: <pre>typedef struct tagPTPJointParams {     float velocity[4];           // Joint velocity in PTP mode     float acceleration[4];      // Joint acceleration in PTP mode }PTPJointParams;</pre> ptpJointParams: PTPJointParams pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 12.2 Get the velocity and acceleration of the joint coordinate axis in PTP mode

Prototype	<code>int GetPTPJointParams(PTPJointParams *ptpJointParams)</code>
Description	Get the velocity( $^{\circ}/s$ ) and acceleration( $^{\circ}/s^2$ ) of the joint coordinate axis in PTP mode
Parameter	PTPJointParams <pre>typedef struct tagPTPJointParams {     float velocity[4];           // Joint velocity in PTP mode     float acceleration[4];      // Joint acceleration in PTP mode }PTPJointParams;</pre> ptpJointParams: PTPJointParams pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a

timeout	
---------	--

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

Table 12.3 Set the velocity and acceleration of the Cartesian coordinate axis in PTP mode

Prototype	<code>int SetPTPCoordinateParams(<a href="#">PTPCoordinateParams</a> *ptpCoordinateParams, <code>bool</code> isQueued, <code>uint64_t</code> *queuedCmdIndex)</code>
Description	Set the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the Cartesian coordinate axis in PTP mode
Parameter	<p>PTPCoordinateParams:</p> <pre>typedef struct tagPTPCoordinateParams {     float xyzVelocity;           //Cartesian coordinate axis (X,Y,Z) velocity     float rVelocity;            // Cartesian coordinate axis (R) velocity     float xyzAcceleration;      //Cartesian coordinate axis (X,Y,Z) acceleration     float rAcceleration;        //Cartesian coordinate axis (R) acceleration }PTPCoordinateParams;</pre> <p>ptpCoordinateParams: PTPCoordinateParams pointer          isQueued: Whether to add this command to the queue          queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error          DobotCommunicate_BufferFull: The command queue is full          DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 12.4 Get the velocity and acceleration of the Cartesian coordinate axis in PTP mode

Prototype	<code>int GetPTPCoordinateParams(<a href="#">PTPCoordinateParams</a> *ptpCoordinateParams)</code>
Description	Get the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the Cartesian coordinate axis in PTP mode
Parameter	<p>PTPCoordinateParams:</p> <pre>typedef struct tagPTPCoordinateParams {</pre>

	<pre> float xyzVelocity;           //Cartesian coordinate axis (X,Y,Z) velocity float rVelocity;            // Cartesian coordinate axis (R) velocity float xyzAcceleration;      //Cartesian coordinate axis (X,Y,Z)                              acceleration float rAcceleration;        //Cartesian coordinate axis (R) acceleration }PTPCoordinateParams; ptpCoordinateParams: PTPCoordinateParams pointer                 </pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 12.5 Set the lifting height and the maximum lifting height in JUMP mode

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

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

Table 12.6 Get the lifting height and the maximum lifting height in JUMP mode

Prototype	<pre> int GetPTPJumpParams(PTPJumpParams *ptpJumpParams)                 </pre>
Description	Get the lifting height and the maximum lifting height in JUMP mode

Parameter	PTPJumpParams: <pre>typedef struct tagPTPJumpParams {     float jumpHeight;           //Lifting height     float zLimit;               //Maximum lifting height }PTPJumpParams; ptpJumpParams: PTPJumpParams pointer</pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 12.7 Setting the Extended Parameters in JUMP mode

Table 12.7 Set the extended parameters in JUMP mode

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

## 12.8 Getting the Extended Parameters in JUMP mode

Table 12.8 Get extended parameters in JUMP mode

Prototype	<pre>int GetPTPJump2Params(PTPJumpParams *ptpJump2Params)</pre>
-----------	-----------------------------------------------------------------

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

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

Table 12.9 Set the velocity and acceleration of the sliding rail in PTP mode

Prototype	<pre>int SetPTPLParams(PTPLParams * ptpLParams, bool isQueued, uint64_t *queuedCmdIndex)</pre>
Description	Set the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the sliding rail in PTP mode
Parameter	PTPLParams: <pre>typedef struct tagPTPJointParams {     float velocity;          //Sliding rail velocity in PTP mode     float acceleration;     //Sliding rail acceleration in PTP mode }PTPLParams;</pre> ptpLParams: PTPLParams pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 12.10 Get the velocity and acceleration of the Sliding rail s in PTP mode

Prototype	<code>int GetPTPLParams(PTPLParams *ptpLParams)</code>
Description	Get the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) of the sliding rail in PTP mode
Parameter	PTPLParams: <pre>typedef struct tagPTPJointParams {     float velocity;           //Sliding rail velocity in PTP mode     float acceleration;      //Sliding rail acceleration in PTP mode }PTPLParams;</pre> ptpLParams: PTPLParams pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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

Table 12.11 Set the velocity ratio and the acceleration ratio in PTP mode

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

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

Table 12.12 Get the velocity ratio and acceleration ratio in PTP mode



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

## 12.13 Executing a PTP Command

Table 12.13 Execute a PTP command

Prototype	<code>int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)</code>
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: <pre>typedef struct tagPTPCmd {     uint8_t ptpMode;           //PTP mode (0-9)     float x;                   //Coordinate parameters in PTP mode. (x,y,z,r)                                //can be set to Cartesian coordinate, joints                                //angle, or increment of them     float y;     float z;     float r; }PTPCmd;</pre> Details for ptpMode: <pre>enum {     JUMP_XYZ,                 //JUMP mode, (x,y,z,r) is the target point in                                //Cartesian coordinate system     MOVJ_XYZ,                 //MOVJ mode, (x,y,z,r) is the target point in                                //Cartesian coordinate system     MOVL_XYZ,                 //MOVL mode, (x,y,z,r) is the target point in                                //Cartesian coordinate system }</pre>

	<pre> 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 MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in               Joint coordinate system MOVJ_INC,    //MOVJ mode, (x,y,z,r) is the angle increment               in Joint coordinate system MOVL_INC,    //MOVL mode, (x,y,z,r) is the Cartesian               coordinate increment in Joint coordinate               system MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian               coordinate increment in Cartesian               coordinate system JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian               coordinate increment in Cartesian               coordinate system  }; ptpCmd: PTPCmd pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid                 </pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 12.14 Execute a PTP command with the I/O control

Prototype	<pre> int SetPTPPOCmd(PTPCmd *ptpCmd, ParallelOutputCmd *parallelCmd, int parallelCmdCount, bool isQueued, uint64_t *queuedCmdIndex)                 </pre>
Description	<p>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 <i>Dobot Magician User Guide</i></p>
Parameter	<p>PTPCmd:</p> <pre> typedef struct tagPTPCmd {     uint8_t ptpMode; //PTP mode (0-9)     float x; //Coordinate parameters in PTP mode. (x,y,z,r)                 </pre>

	<p>can be set to Cartesian coordinate, joints angle, or increment of them</p> <pre>float y; float z; float r; }PTPCmd; Details for ptpMode: enum {     JUMP_XYZ,           //JUMP mode, (x,y,z,r) is the target point in Cartesian coordinate system     MOVJ_XYZ,          //MOVJ mode, (x,y,z,r) is the target point in Cartesian coordinate system     MOVL_XYZ,          //MOVL mode, (x,y,z,r) is the target point in Cartesian coordinate system     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     MOVL_ANGLE,        //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system     MOVJ_INC,          //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system     MOVL_INC,          //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system     MOVJ_XYZ_INC,      //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system     JUMP_MOVL_XYZ,     //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system }; ParallelOutputCmd: typedef struct tagParallelOutputCmd {     uint8_t ratio;      //The distance ratio between the two points in PTP mode, namely, the position where I/O is triggered     uint16_t address;   //I/O address (0-20)     uint8_t level;      //Output value</pre>
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<pre> }ParallelOutputCmd; ptpCmd: PTPCmd pointer parallelCmd: ParallelOutputCmd pointer parallelCmdCount::I/O number isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid                 </pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 12.15 Executing a PTP Command with the Sliding Rail

Table 12.15 Execute a PTP command with the sliding rail

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

	<pre> Joint coordinate system MOVJ_ANGLE, //MOVJ mode, (x,y,z,r) is the target point in Joint coordinate system MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system MOVJ_INC, //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system MOVL_INC, //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system  }; ptpWithLCmd : PTPWithLCmd pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid                 </pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 12.16 Execute a PTP command with the sliding rail and I/O control

Prototype	<pre> int SetPTPPOWithLCmd(PTPWithLCmd *ptpWithLCmd, ParallelOutputCmd *parallelCmd, int parallelCmdCount, bool isQueued, uint64_t *queuedCmdIndex)                 </pre>
Description	Execute a PTP command with the sliding rail and I/O control
Parameter	<p>PTPWithLCmd</p> <pre> typedef struct tagPTPWithL {     uint8_t ptpMode; //PTP mode (0-9)     float x; //Coordinate parameters in PTP mode. (x,y,z,r) can be set to Cartesian coordinate, joints angle, or increment of them                 </pre>

```

float y;
float z;
float r;
float l;           //The distance that sliding rail moves
}PTPWithLCmd;
Details for ptpMode:
enum {
    JUMP_XYZ,       //JUMP mode, (x,y,z,r) is the target point in
                    //Cartesian coordinate system
    MOVJ_XYZ,       //MOVJ mode, (x,y,z,r) is the target point in
                    //Cartesian coordinate system
    MOVL_XYZ,       //MOVL mode, (x,y,z,r) is the target point in
                    //Cartesian coordinate system
    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
    MOVL_ANGLE,     //MOVL mode, (x,y,z,r) is the target point in
                    //Joint coordinate system
    MOVJ_INC,       //MOVJ mode, (x,y,z,r) is the angle increment
                    //in Joint coordinate system
    MOVL_INC,       //MOVL mode, (x,y,z,r) is the Cartesian
                    //coordinate increment in Joint coordinate
                    //system
    MOVJ_XYZ_INC,   //MOVJ mode, (x,y,z,r) is the Cartesian
                    //coordinate increment in Cartesian
                    //coordinate system
    JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian
                    //coordinate increment in Cartesian
                    //coordinate system
};
ParallelOutputCmd:
typedef struct tagParallelOutputCmd {
    uint8_t ratio;   //The distance ratio between the two points in
                    //PTP mode, namely, the position where I/O
                    //is triggered
    uint16_t address; //I/O address (0-20)
    uint8_t level;   //Output value
}ParallelOutputCmd;

```

	<p>ptpWithLCmd : PTPWithLCmd pointer</p> <p>parallelCmd: ParallelOutputCmd pointer</p> <p>parallelCmdCount: I/O number</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 13. CP

CP: Continuous Path.

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

Table 13.1 Set the velocity and acceleration in CP mode

Prototype	<code>int SetCPPParams(CPPParams *cpParams, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) in CP mode
Parameter	<p>CPPParams</p> <pre>typedef struct tagCPPParams {     float planAcc;           //The maximum planning acceleration     float junctionVel;      //The maximum junction velocity     union {         float acc;          //The maximum actual acceleration. It is                             //valid only when <b>realTimeTrack</b> is set to                             //0         float period;      //Interpolation period. It is valid only when                             //<b>realTimeTrack</b> is set to 1     };     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. }CPPParams;</pre> <p>cpParams: CPPParams pointer                  isQueued: Whether to add this command to the queue                  queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error                  DobotCommunicate_BufferFull: The command queue is full                  DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 13.2 Get the velocity and acceleration in CP mode



Prototype	<code>int GetCPPParams(CPPParams *cpParams)</code>
Description	Get the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) in CP mode
Parameter	<p>CPPParams</p> <pre>typedef struct tagCPPParams {     float planAcc;           //The maximum planning acceleration     float junctionVel;      //The maximum junction velocity     union {         float acc;          //The maximum actual acceleration. It is                             //valid only when <b>realTimeTrack</b> is set to                             //0         float period;      //Interpolation period. It is valid only when                             //<b>realTimeTrack</b> is set to 1     };     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. }CPPParams;</pre> <p>cpParams: CPPParams pointer</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

### 13.3 Executing the CP Command

Table 13.3 Execute the CP command

Prototype	<code>int SetCPCmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Execute the CP commands
Parameter	<p>CPCmd</p> <pre>typedef struct tagCPCmd {     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     float x;               //(x,y,z )can be set to Cartesian coordinate,                             //or Cartesian coordinate increment </pre>

	<pre>float y; float z; union { float velocity;    //Reserved float power;      //Reserved }CPCmd; cpCmd: CPCmd pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

### 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.

## 13.4 Executing the CP Command with the Laser Engraving

Table 13.4 Execute the CP command with laser engraving

Prototype	<pre>int SetCPLECmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex)</pre>
Description	Execute the CP command with the laser engraving.
Parameter	<pre>typedef struct tagCPCmd { 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  float x;           //(x,y,z )can be set to Cartesian coordinate,                     or Cartesian coordinate increment  float y; float z; union { float velocity;    // Reserved</pre>

	<pre>float power;           //Laser power 0-100 }CPCmd;</pre> <p>cpCmd: CPCmd pointer</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 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 14.1.

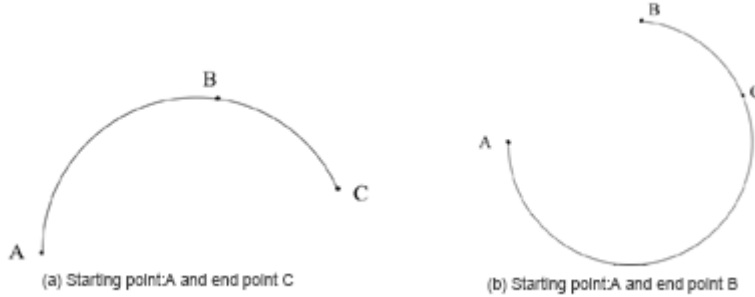


Figure 14.1 ARC mode

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

Table 14.1 Set the velocity and acceleration in ARC mode

Prototype	<code>int SetARCParams(ARCParams *arcParams, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) in PTP mode
Parameter	<p>ARCParams</p> <pre>typedef struct tagARCParams {     float xyzVelocity;           //Cartesian coordinate axis (X,Y,Z) velocity     float rVelocity;            //Cartesian coordinate axis (R) velocity     float xyzAcceleration;      //Cartesian coordinate axis (X,Y,Z) acceleration     float rAcceleration;        //Cartesian coordinate axis (R) acceleration }ARCParams;</pre> <p>arcParams: ARCParams pointer</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

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

Table 14.2 Get the velocity and acceleration in ARC mode

Prototype	<code>int GetARCParams(ARCParams *arcParams)</code>
Description	Get the velocity(mm/s) and acceleration(mm/s <sup>2</sup> ) in ARC mode
Parameter	ARCParams <pre>typedef struct tagARCParams {     float xyzVelocity;           //Cartesian coordinate axis (X,Y,Z) velocity     float rVelocity;            //Cartesian coordinate axis (R) velocity     float xyzAcceleration;      //Cartesian coordinate axis (X,Y,Z)                                 acceleration     float rAcceleration;       //Cartesian coordinate axis (R) acceleration }ARCParams;</pre> arcParams: ARCParams pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 14.3 Executing the ARC Command

Table 14.3 Execute the ARC command

Prototype	<code>int SetARCCmd(ARCCmd *arcCmd, bool isQueued, uint64_t *queuedCmdIndex)</code>
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. In ARC mode, it is necessary to confirm the three points with other motion modes.
Parameter	ARCCmd: <pre>typedef struct tagARCCmd {     struct {         float x;         float y;         float z;         float r;     }cirPoint ;           //Middle point. (x,y,z,r) can be set to                            Cartesian coordinate      struct {         float x;</pre>

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

## 14.4 Executing the CIRCLE Command

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

Table 14.4 Execute the CIRCLE command

Prototype	<pre> int SetCircleCmd(CircleCmd *circleCmd, bool isQueued, uint64_t *queuedCmdIndex)                 </pre>
Description	<p>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.</p> <p>In CIRCLE mode, it is necessary to confirm the three points with other motion modes.</p>
Parameter	<pre> CircleCmd typedef struct tagCircleCmd {     struct {         float x;         float y;         float z;         float r;     }cirPoint ;           //Middle point.(x,y,z,r) can be set to                         Cartesian coordinate      struct {         float x;         float y;                 </pre>

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

## 15. Losing-Step Detection

### 15.1 Setting the losing-step threshold

Table 15.1 Set the losing-step threshold

Prototype	<code>int SetLostStepParams(float threshold, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the losing-step threshold, checking for whether the position error exceeds this threshold. If this threshold is exceeded, the motor loses step  If you do not call this API, the default threshold is 5
Parameter	threshold: Losing-step threshold  isQueued: Whether to add this command to the queue  queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid
Return	DobotCommunicate_NoError: The command returns with no error  DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 15.2 Executing the Losing-Step Command

Table 15.2 Execute the losing-step command

Prototype	<code>int SetLostStepCmd(bool isQueued, uint64_t *queuedCmdIndex)</code>
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.  This command must be added to the command queue, namely, <b>isQueued</b> must be set to <b>1</b> .
Parameter	isQueued: Whether to add this command to the queue  queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid
Return	DobotCommunicate_NoError: The command returns with no error  DobotCommunicate_BufferFull: The command queue is full  DobotCommunicate_Timeout: The command does not return, resulting in a timeout

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



## Program 15.1 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();
}
```

## 16. WAITING

### 16.1 Executing the Waiting Command

Table 16.1 Execute the Waiting command

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

## 17. TRIGGERING

### 17.1 Executing the Triggering Command

Table 17.1 Execute the Triggering command

Prototype	<pre>int SetTRIGCmd(TRIGCmd *trigCmd, bool isQueued, uint64_t *queuedCmdIndex)</pre>
Description	<p>Execute the triggering command.</p> <p>This command must be added to the command queue, namely, <b>isQueued</b> must be set to <b>1</b>. If not, the parameter <b>condition</b> of the <b>Triggering</b> command in the queue command being executed may be changed because the <b>TRIGCmd</b> memory is shared</p>
Parameter	<p>TRIGCmd:</p> <pre>typedef struct tagTRIGCmd {     uint8_t address;           // EIO address:1-20     uint8_t mode;              //Triggering mode. 0: Level                               trigger.1:A/D trigger     uint8_t condition;        //Triggering condition                               Level: 0, equal. 1, unequal                               A/D: 0, less than. 1,less than or equal                               2, greater than or equal. 3, greater                               than     uint16_t threshold;       //Triggering threshold. Level : 0,1 .A/D:                               0-4095 }TRIGCmd;</pre> <p>trigCmd: TRIGCmd pointer</p> <p>isQueued: Whether to add this command to the queue</p> <p>queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 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*.

### 18.1 Setting the I/O Multiplexing

Table 18.1 Set the I/O multiplexing

Prototype	<code>int SetIOMultiplexing(IOMultiplexing *ioMultiplexing, bool isQueued, uint64_t *queuedCmdIndex)</code>
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	<p>IOMultiplexing:</p> <pre>typedef struct tagIOMultiplexing {     uint8_t address;           //I/O address:1-20     uint8_t multiplex;        //I/O multiplexing function: 0-6 }IOMultiplexing;</pre> <p>The values supported by <b>multiplex</b> are shown as follows:</p> <pre>typedef enum tagIOFunction {     IOFunctionDummy;         //Invalid     IOFunctionDO;            // I/O output     IOFunctionPWM;           // PWM output     IOFunctionDI;            //I/O input     IOFunctionADC;           //A/D input     IOFunctionDIPU;          //Pull-up input     IOFunctionDIPD           //Pull-down input }IOFunction;</pre> <p>ioMultiplexing: IOMultiplexing pointer  isQueued: Whether to add this command to the queue  queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p>

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

## 18.2 Getting the I/O multiplexing

Table 18.2 Getting the I/O multiplexing

Prototype	<code>int GetIOMultiplexing(IOMultiplexing *ioMultiplexing)</code>
Description	Get the I/O multiplexing
Parameter	<p>IOMultiplexing:</p> <pre>typedef struct tagIOMultiplexing {     uint8_t address;           //I/O address     uint8_t multiplex;        //I/O multiplexing function: <b>0-6</b> }IOMultiplexing;</pre> <p>The values supported by <b>multiplex</b> are as follows.</p> <pre>typedef enum tagIOFunction {     IOFunctionDummy;         //Invalid     IOFunctionDO;            // I/O output     IOFunctionPWM;           // PWM output     IOFunctionDI;            //I/O input     IOFunctionADC;           //A/D input     IOFunctionDIPU;          //Pull-up input     IOFunctionDIPD           //Pull-down input }IOFunction;</pre> <p>ioMultiplexing: IOMultiplexing pointer</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.3 Setting the I/O Output

Table 18.3 Set the I/O output

Prototype	<code>int SetIODO(IODO *ioDO, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the I/O output
Parameter	<p>IODO:</p> <pre>typedef struct tagIODO {     uint8_t address;          //I/O address</pre>

	<pre>uint8_t level;           //0: Low level.1: High level }IODO; ioDO: IODO pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.4 Getting the I/O Output

Table 18.4 Get the I/O output

Prototype	<code>int GetIODO(IODO *ioDO)</code>
Description	Get the I/O output
Parameter	<p>IODO:</p> <pre>typedef struct tagIODO {     uint8_t address;           //I/O address     uint8_t level;           //0: Low level.1: High level }IODO; ioDO: IODO pointer</pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.5 Setting the PWM Output

Table 18.5 Set PWM output

Prototype	<code>int SetIOPWM(IOPWM *ioPWM, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the PWM output
Parameter	<p>IOPWM:</p> <pre>typedef struct tagIOPWM {     uint8_t address;           //I/O address     float frequency;           // PWM frequency: 10Hz-1MHz</pre>

	<pre>float dutyCycle;           // PWM duty cycle: <b>0-100</b> }IOPWM; ioPWM: IOPWM pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid</pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_BufferFull: The command queue is full</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.6 Getting the PWM Output

Table 18.6 Get the PWM output

Prototype	<code>int GetIOPWM(IOPWM *ioPWM)</code>
Description	Get the PWM output
Parameter	<p>IOPWM:</p> <pre>typedef struct tagIOPWM {     uint8_t address;           //I/O address     float frequency;          // PWM frequency: <b>10Hz-1MHz</b>     float dutyCycle;          // PWM duty cycle: <b>0-100</b> }IOPWM; ioPWM: IOPWM pointer</pre>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.7 Getting the I/O Input

Table 18.7 Get the I/O input

Prototype	<code>int GetIODI(IODI *ioDI)</code>
Description	Get the I/O input
Parameter	<p>IODI:</p> <pre>typedef struct tagIODI {     uint8_t address;           //I/O address     uint8_t level;            //<b>0</b>: Low level. <b>1</b>: High-level</pre>

	<pre> }IODI; ioDI: IODO pointer                 </pre>
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 18.8 Getting the A/D Input

Table 18.8 Get the A/D Input

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

## 18.9 Setting the Velocity of the Extended Motor

Table 18.9 Set the velocity of the extended motor

Prototype	<code>int SetEMotor(EMotor *eMotor, bool isQueued, uint64_t *queuedCmdIndex)</code>
Description	Set the velocity of the extended motor. The motor will always be operated at a constant velocity after calling this API
Parameter	EMotor: <pre> typedef struct tagEMotor {     uint8_t index;           //Motor index. 0: Stepper1. 1:Stepper2     uint8_t isEnabled;      //Control motor. 0: Disabled. 1: Enabled     uint32_t speed;         //Motor velocity (Pulse number per second) }EMotor;                 </pre> eMotor: EMotor pointer isQueued: Whether to add this command to the queue



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

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

Table 18.10 Set the velocity of extended motor and the movement distance

Prototype	<code>int SetEMotorS(EMotorS *eMotorS, bool isQueued, uint64_t *queuedCmdIndex)</code>
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: <pre>typedef struct tagEMotorS{     uint8_t index;           //Motor index. 0: Stepper1. 1:Stepper2     uint8_t isEnabled;     //Control motor. 0: Disabled. 1: Enabled     uint32_t speed;        //Motor velocity (Pulse number per second)     uint32_t distance      //Movement distance (Pulse number) }EMotorS;</pre> eMotorS: EMotorS pointer isQueued: Whether to add this command to the queue queuedCmdIndex: If this command is added to the queue, <b>queuedCmdIndex</b> indicates the index of this command in the queue. Otherwise, it is invalid
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 18.11 Enabling the Photoelectric Sensor

Table 18.11 Enable the photoelectric sensor

Prototype	<code>int SetInfraredSensor(bool enable, InfraredPort infraredPort, uint8_t version)</code>
Description	Enable the photoelectric sensor
Parameter	InfraredPort:

	<pre>enum InfraredPort {     IF_PORT_GP1;     IF_PORT_GP2;     IF_PORT_GP4;     IF_PORT_GP5; };</pre> <p>enable: <b>0</b>, Disabled. <b>1</b>, Enabled</p> <p>infraredPort: The Dobot interface that the photoelectric sensor is connected to. Please select the corresponding interface</p> <p>version: Version flag of photoelectric sensor. <b>0</b>: The version is <b>V 1.0</b>. <b>1</b>: The version is <b>V2.0</b></p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.12 Getting the Photoelectric Sensor Value

Table 18.12 Get the photoelectric sensor value

Prototype	<code>int GetInfraredSensor (InfraredPort infraredPort, uint8_t *value)</code>
Description	Get the photoelectric sensor value
Parameter	<p>InfraredPort:</p> <pre>enum InfraredPort {     IF_PORT_GP1;     IF_PORT_GP2;     IF_PORT_GP4;     IF_PORT_GP5; };</pre> <p>infraredPort: The Dobot interface that the photoelectric sensor is connected to. Please select the corresponding interface</p> <p>value: The value of the photoelectric sensor</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.13 Enabling the Color Sensor

Table 18.13 Enable the color sensor

Prototype	<code>int SetColorSensor(bool enable, ColorPort colorPort, uint8_t version)</code>
Description	Enable the color sensor
Parameter	<p>ColorPort:</p> <pre>enum ColorPort {     IF_PORT_GP1;     IF_PORT_GP2;     IF_PORT_GP4;     IF_PORT_GP5; };</pre> <p>enable: <b>0</b>, Disabled. <b>1</b>, Enabled</p> <p>colorPort: The Dobot interface that the color sensor is connected to. Please select the corresponding interface</p> <p>version: Version flag of color sensor. <b>0</b>: The version is <b>V1.0</b>. <b>1</b>: The version is <b>V2.0</b></p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 18.14 Getting the Color Sensor Value

Table 18.14 Get the color sensor value

Prototype	<code>int GetColorSensor( uint8_t *r, uint8_t *g, uint8_t *b )</code>
Description	Get the color sensor value
Parameter	<p>r: Red. The range is: 0-255</p> <p>g: Green. The range is: 0-255</p> <p>b: Blue. The range is: 0-255</p>
Return	<p>DobotCommunicate_NoError: The command returns with no error</p> <p>DobotCommunicate_Timeout: The command does not return, resulting in a timeout</p>

## 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 sensor value of Forearm/Rear Arm \*Linearization parameter of Forearm/Rear Arm angle sensor

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

### 19.1 Setting the Angle Sensor Static Error

Table 19.1 Set the angle sensor static error

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

### 19.2 Getting the Angle Sensor Static Error

Table 19.2 Get the angle sensor static error

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

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

Table 19.3 Set the linearization parameter of the angle sensor

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

## 19.4 Getting the Linearization Parameter of the Angle Sensor

Table 19.4 Get the linearization parameter of the angle sensor

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

## 19.5 Setting the Static Error of the Base Encoder

Table 19.5 Set static error of the base Encoder

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

## 19.6 Getting the Static Error of the Base Encoder

Table 19.6 Get the static error of the base Encoder

Prototype	<code>int GetBaseDecoderStaticError (float *baseDecoderError)</code>
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 DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 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.

### 20.1 Enabling WIFI

Table 20.1 Enable WIFI

Prototype	<code>int SetWIFIConfigMode(bool enable)</code>
Description	Enable WIFI
Parameter	enable: <b>0</b> , Disabled. <b>1</b> , Enabled
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

### 20.2 Getting the WIFI Status

Table 20.2 Get the WIFI Status

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

### 20.3 Setting the SSID

SSID (Service Set Identifier): WIFI network name.

Table 20.3 Set the SSID

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

## 20.4 Getting the SSID

Table 20.4 Get the SSID

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

## 20.5 Setting the Network Password

Table 20.5 Set the Network password

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

## 20.6 Getting the Network Password

Table 20.6 Get the Network password

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

## 20.7 Setting the IP Address

Table 20.7 Set the IP Address



Prototype	<code>int SetWIFIIPAddress(WIFIIPAddress *wifiIPAddress)</code>
Description	Set the IP address
Parameter	WIFIIPAddress <pre>typedef struct tagWIFIIPAddress {     uint8_t dhcp;           //Whether to enable DHCP. 0: Disabled1:Enabled     uint8_t addr[4];       // The IP address is divided into 4 segments, the value range of each segment is 0-255 }WIFIIPAddress;</pre> wifiIPAddr: WIFIIPAddress pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 20.8 Getting the IP Address

Table 20.8 Get the IP Address

Prototype	<code>int GetWIFIIPAddress(WIFIIPAddress *wifiIPAddress)</code>
Description	Get the IP address
Parameter	WIFIIPAddress <pre>typedef struct tagWIFIIPAddress {     uint8_t dhcp;           //Whether to enable DHCP. 0: Disabled1:Enabled     uint8_t addr[4];       // The IP address is divided into 4 segments, the value range of each segment is 0-255 }WIFIIPAddress;</pre> wifiIPAddr: WIFIIPAddress pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 20.9 Setting the Sub Netmask

Table 20.9 Set the sub netmask

## 20.10 Getting the Sub Netmask

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

Table 20.10 Get the sub netmask

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

## 20.11 Setting the Gateway

Table 20.11 Set the gateway

Prototype	<code>int SetWIFIGateway(WIFIGateway *wifiGateway)</code>
Description	Set the gateway
Parameter	<p>WIFIGateway</p> <pre>typedef struct tagWIFIGateway {     uint8_t addr[4];           //The IP address is divided into 4                                segments, the value range of each</pre>

	segment is 0-255 }WIFIGateway; wifiGateway: WIFIGateway pointer
Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_Timeout: The command does not return, resulting in a timeout

## 20.12 Getting the Gateway

Table 20.12 Get the gateway

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

## 20.13 Setting the DNS

Table 20.13 Set the DNS

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

timeout	
---------	--

## 20.14 Getting the DNS

Table 20.14 Get the DNS

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

## 20.15 Getting the WIFI Connection Status

Table 20.15 Get the WIFI connection status

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

## 21. Other functions

### 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 21.1 Event loop

Prototype	<code>void DobotExec(void)</code>
Description	Event loop
Parameter	None
Return	Void