Dobot API 接口文档

Dobot Magician

TN01010101 V1.2.0 Date: 2017/01/04

深圳市越疆科技有限公司

修订历史

版本	日期	原因
V1.0.0	2016/07/29	创建文档
V1.0.1	2016/08/09	修改协议分类、说明等
V1.0.2	2016/08/22	修改设置末端参数接口说明，也支持队列方式
V1.0.3	2016/08/26	增加搜索 Dobot 的功能；修改 EIO 的部分定义
V1.0.4	2016/08/27	修改搜索的 API 定义等
V1.0.5	2016/08/29	修正重设实时位姿 API 接口错误
V1.0.6	2016/08/31	增加 Wi-Fi 配置功能
V1.0.7	2016/09/09	修改末端夹具的控制接
V1.0.8	2016/09/13	修改回零参数及 maxJumpHeight
V1.0.9	2016/09/19	增加 DNS 的接口
V1.1.0	2016/09/27	增加检测-Fi 模块是否在位的 API 接口
V1.1.1	2016/10/24	修改 ConnectDobot 中输入参数说明
V1.1.2	2016/11/10	增加连续轨迹灰度雕刻功能
V1.1.3	2016/11/11	增加扩展点击控制接口
V1.1.4	2016/12/22	添加返回结构体说明；添加新队列控制接口
V1.2.0	2017/01/04	新版动态库，全面支持多线程，无需调用 API 背景任务

目 录

1. 适用范围

本文档旨在对 Dobot API 接口进行详细说明，并给出基于 Dobot API 接口开发应用程序的一般流程。

2. API 接口说明

2.1 Dobot 指令简介

在与 Dobot 控制器通信时，其通信指令具有以下两个特点：

1. 控制器对所有的指令都有返回；对于设置指令，控制器将指令参数域去掉，并返回；对于获取指令，控制器将该指令将要获取的参数填入到参数域，并返回；
2. 控制器支持两类指令：立即指令与队列指令：
   - 立即指令：Dobot 控制器将在收到立即指令时立即处理该指令，而无论当前控制器是否还有其余指令运行中；
   - 队列指令：Dobot 控制器在收到队列指令时，将该命令压入控制器内部指令队列中。Dobot 控制器将根据指令压入队列的顺序执行指令。

关于 Dobot 指令更具体的内容，可查询 Dobot 通信协议手册。

2.2 连接/断开

2.2.1 搜索 Dobot

表 2.1 获取 Dobot 数量接口说明

原型	int SearchDobot(char *dobotList, uint32_t maxLen)
描述	搜索 Dobot，动态库会将已连接的 Dobot 信息存储，并使用 ConnectDobot 连接 Dobot。
参数	dobotList:外部传入的数组首地址，动态库会将搜索到的串口/UDP 信息写入到 dobotList，比如一个典型的 dobotList 的返回是："COM1 COM3 COM6 192.168.0.5"，不同的接口以空格分开。 maxLen:外部传入的 dobotList 支持的最大长度，以避免内存溢出
返回	Dobot 数量

2.2.2 连接 Dobot

表 2.2 连接 Dobot 控制器接口说明

原型	int ConnectDobot(const char * portName, int baudrate)
描述	连接 Dobot 控制器。其中，portName 从上个 API 的 dobotList 得到。 注：若不调用 SearchDobot 而直接调用 ConnectDobot（portName 为空），则动态库将自动连接第一个找到的 Dobot 控制器。
参数	portName，Dobot 端口名；对于串口，可能是”COM3”；对于 UDP，可能是“192.168.0.5” baudrate，波特率
返回	DobotConnect_NoError：连接 Dobot 控制器成功 DobotConnect_NotFound：未找到 Dobot 控制器接口 DobotConnect_Occupied：Dobot 控制器接口被占用

注：为了使 API 接口能识别 Dobot 控制器接口，请提前安装所需的驱动，详情请查询 Dobot 用户手册。

2.2.3 断开 Dobot

表 2.3 断开 Dobot 接口说明

原型	int DisconnectDobot(void)
描述	断开 Dobot 控制器
参数	无
返回	DobotConnect_NoError：无错误 DobotConnect_NotFound：未找到 Dobot 控制器接口（本接口不会返回该值） DobotConnect_Occupied：Dobot 控制器接口被占用（本接口不会返回该值）

2.3 指令超时

如 2.1 中介绍，发往 Dobot 控制器的所有指令都带有返回。当由于通信链路干扰等造成指令错误时，控制器将无法识别该条指令且无法返回。因此，每条下发给控制器的指令都有一个超时时间。该指令超时时间可以通过以下的 API 进行设置。

表 2.4 设置指令超时接口说明

原型	int SetCmdTimeout(uint32_t cmdTimeout)
描述	设置指令超时
参数	cmdTimeout:指令超时时间；单位：ms
返回	DobotCommunicate_NoError:无错误 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时（本接口不会返回该值）

2.4 指令队列控制

Dobot 控制器支持启动、停止队列指令的执行。

2.4.1 启动指令队列运行

表 2.5 启动指令队列运行接口说明

原型	int SetQueuedCmdStartExec(void)
描述	启动指令队列运行
参数	无
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.4.2 停止指令队列运行

表 2.6 停止指令队列运行接口说明

原型	int SetQueuedCmdStopExec(void)
描述	停止指令队列运行。若当前指令队列正在运行一条指令，则其将会在这条指令运行完成后，停止指令队列运行。
参数	无
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.4.3 强制停止指令队列运行

表 2.7 强制停止指令队列运行接口说明

原型	int SetQueuedCmdForceStopExec(void)
描述	强制停止指令队列运行。无论指令队列是否正在运行一条指令，控制器都会强制其停止运行。
参数	无
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.4.4 启动指令队列下载

Dobot 的控制器支持将指令存储到控制器外部 Flash 中，而后可以通过控制器上的按键触发执行，也即脱机运行功能。

指令下载的一般流程是：

1. 调用启动指令队列下载 API。其中，指定 totalLoop 为指令脱机运行时的总次数；
2. 发送队列指令；
3. 重复，直至队列指令发送完成；
4. 调用停止指令队列下载控制 API。

表 2.8 启动指令队列下载接口说明

原型	int SetQueuedCmdStartDownload(uint32_t totalLoop, uint32_t linePerLoop)
描述	启动指令队列下载
参数	totalLoop:脱机运行总次数 linePerLoop:单次循环的指令条数
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.4.5 完成指令队列下载

表 2.9 完成指令队列下载接口说明

原型	int SetQueuedCmdStopDownload(void)
描述	完成指令队列下载
参数	无
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.4.6 清空指令队列

该接口可以清空 Dobot 控制器中缓存的指令队列。

表 2.10 清除指令队列

原型	int SetQueuedCmdClear(void)
描述	清空指令队列
参数	无
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.4.7 获取指令队列索引

在 Dobot 控制器指令队列机制中，有一个 64 位内部计数索引。当控制器每执行完一条命令时，该计数器将自动加一。通过该内部索引，可以查询控制器已经执行了多少队列指令，以及当前已经执行到哪条指令（指示运行进度时）。

表 2.11 获取指令队列当前索引接口说明

原型	int GetQueuedCmdCurrentIndex(uint64_t *queuedCmdCurrentIndex)
描述	获取指令队列已执行到的索引
参数	queuedCmdCurrentIndex:队列指令索引变量指针；
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.5 设备信息

2.5.1 设置设备序列号

表 2.12 设置设备序列号接口说明

原型	int SetDeviceSN(const char *deviceSN)
描述	设置设备序列号
参数	deviceSN:设备序列号字符串指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

注：设备序列号设置接口仅在出厂时有效（需要特殊密码）。

2.5.2 获取设备序列号

表 2.13 获取设备序列号接口说明

原型	int GetDeviceSN(char *deviceSN, uint32_t maxLen)
描述	获取设备序列号
参数	deviceSN:设备序列号字符串指针 maxLen:传入外部缓冲区长度，以避免溢出
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.5.3 设置设备名称

表 2.14 设置设备名称接口说明

原型	int SetDeviceName(const char *deviceName)
描述	设置设备名称。当有多台机器时，可使用本接口设置设备名以作区分
参数	deviceName:设备名称字符串指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.5.4 获取设备名称

表 2.15 获取设备名称接口说明

原型	int GetDeviceName(char *deviceName, uint32_t maxLen)
描述	获取设备名称
参数	deviceName:设备名称字符串指针 maxLen:传入外部缓冲区长度，以避免溢出
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.5.5 获取设备版本号

表 2.16 获取设备版本号接口说明

原型	int GetDeviceVersion(uint8_t *majorVersion, uint8_t *minorVersion, uint8_t *revision)
描述	获取设备版本信息
参数	majorVersion:主版本 minorVersion:次版本 revision:修订版本
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.6 实时位姿

在 DobotV2.0 中，Dobot 控制器根据以下信息，计算出实时位姿初始值：

- 底座码盘位置（可以通过回零可以得到）；
- 大臂传感器角度（上电或者按小臂 UNLOCK 按键时）；
- 小臂传感器角度（上电或者按小臂 UNLOCK 按键时）。

而后在控制 Dobot 时，Dobot 控制器将基于实时位姿初始值，以及实时运动状态，更新实时位姿。

2.6.1 获取实时位姿

表 2.17 获取实时位姿接口说明

原型	int GetPose(Pose *pose)
描述	获取实时位姿

参数	Pose 定义：
	typedef struct tagPose {
	float x; //机械臂坐标系 x
	float y; //机械臂坐标系 y
	float z; //机械臂坐标系 z
	float r; //机械臂坐标系 r
	float jointAngle[4];//机械臂 4 轴(底座、大臂、小臂、末端)角度
	}Pose;
	
	pose:实时位姿指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.6.2 重设实时位姿

在以下情况，可以重设实时位姿（初始值）：

- 角度传感器损坏，必须借助外部的角度测量手段；
- 角度传感器精度太差，借助外部的角度直接/间接测量手段确认其精确值。

表 2.18 设置初始姿态接口说明

原型	int ResetPose(bool manual, float rearArmAngle, float frontArmAngle)
描述	重设姿态
参数	manual:为 0 时,自动重设姿态，不用传入 rearArmAngle 及 frontArmAngle；为 1 时，传入 rearArmAngle 和 frontArmAngle rearArmAngle:大臂角度值 frontArmAngle:小臂角度值
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.6.3 获取运动学参数

表 2.19 获取运动学参数接口

原型	int GetKinematics(Kinematcis *kinematics)
描述	获取运动学参数
参数	typedef struct tagKinematics { float velocity; float acceleration; }Kinematics; kinematics:运动学参数变量指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.7 ALARM 功能

2.7.1 获取系统报警状态

表 2.20 获取系统报警状态接口说明

原型	int GetAlarmsState(uint8_t *alarmsState, uint32_t *len, uint32_t maxLen)
描述	获取系统报警状态
参数	alarmsState:数组首地址，用于接收各报警位 len:报警所占字节 maxLen:传入外部缓冲区长度，以避免溢出
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

注：关于具体报警与索引的对应关系，请查询 Dobot 用户手册。数组 alarmsState 中的每一个字节可以标识 8 个报警项的报警状态，且 MSB（Most Significant Bit）在高位，LSB（Least Significant Bit）在低位。

2.7.2 清除系统所有报警

表 2.21 清除系统所有报警接口说明

原型	int ClearAllAlarmsState(void)
描述	清除系统所有报警
参数	无
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.8 HOME 功能

2.8.1 设置回零参数

表 2.22 设置回零参数接口说明

原型	int SetHOMEParams(HOMEParams *homeParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置回零参数

参数	HOMEParams 定义：
	typedef struct tagHOMEParams {
	float x; //机械臂坐标系 x
	float y; //机械臂坐标系 y
	float z; //机械臂坐标系 z
	float r; //机械臂坐标系 r
	}HOMEParams;
	
	homeParams:回零参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注:在某些机型中可能不支持回零功能，具体请查询用户手册。

2.8.2 获取回零参数

表 2.23 获取回零参数接口说明

原型	int GetHOMEParams(HOMEParams *homeParams)
描述	获取回零参数

参数	HOMEParams 定义：
	typedef struct tagHOMEParams {
	float x; //机械臂坐标系 x
	float y; //机械臂坐标系 y
	float z; //机械臂坐标系 z
	float r; //机械臂坐标系 r
	}HOMEParams;
	
	homeParams:回零参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

注:在某些机型中可能不支持回零功能，具体请查询用户手册。

2.8.3 执行回零功能

表 2.24 执行回零功能接口说明

原型	int SetHOMECmd(HOMECmd *homeCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行回零功能

参数	HOMECmd 定义：
	typedef struct tagHOMECmd {
	uint32_t reserved; // 预留未来使用
	}HOMECmd;
	
	homeCmd:回零指令变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注:在某些机型中可能不支持回零功能，具体请查询用户手册。

2.9 HHT 功能

2.9.1 设置触发模式

表 2.25 设置手持示教触发模式接口说明

原型	int SetHHTTrigMode (HHTTrigMode hhtTrigMode)
描述	设置手持示教时的存点触发模式
参数	typedef enum tagHHTTrigMode { TriggeredOnKeyReleased, // 按键释放时更新 TriggeredOnPeriodicInterval // 定时触发 }HHTTrigMode; hhtTrigMode:UNLOCK 按键按下后，手持示教存点的触发模式
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.9.2 获取触发模式

表 2.26 获取手持示教触发模式接口说明

原型	int GetHHTTrigMode (HHTTrigMode *hhtTrigMode)
描述	获取手持示教时的存点触发模式
参数	typedef enum tagHHTTrigMode { TriggeredOnKeyReleased, // 按键释放时更新 TriggeredOnPeriodicInterval // 定时触发 }HHTTrigMode; hhtTrigMode:UNLOCK 按键按下后，手持示教存点的触发模式
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.9.3 设置触发输出使能/禁止

表 2.27 设置触发输出使能/禁止接口说明

原型	int SetHHTTrigOutputEnabled (bool isEnabled)
描述	设置手持示教触发输出使能状态
参数	isEnabled:触发输出使能状态
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.9.4 获取触发输出使能/禁止

表 2.28 获取触发输出使能/禁止接口说明

原型	int GetHHTTrigOutputEnabled (bool *isEnabled)
描述	获取手持示教触发输出使能状态
参数	isEnabled:触发输出使能状态
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.9.5 获取触发输出

表 2.29 获取触发输出接口说明

原型	int GetHHTTrigOutput(bool *isTriggered)
描述	获取触发输出状态
参数	isTriggered:触发输出状态
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10 末端执行器

2.10.1 设置末端执行器参数

表 2.30 设置末端执行器参数接口说明

原型	int SetEndEffectorParams(EndEffectorParams *endEffectorParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置末端参数

参数	EndEffectorParams 定义：
	typedef struct tagEndEffectorParams {
	float xBias; //末端 x 方向长度
	float yBias; //末端 y 方向长度
	float zBias; //末端 z 方向长度
	}EndEffectorParams;
	
	endEffectorParams:末端执行器参数指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注：当使用标准的末端执行器时，需要请查询 Dobot 用户手册，得到其 X 轴与 Y 轴偏置，并调用本接口设置。其他情况下的末端执行器参数，需自行确认结构参数。

2.10.2 获取末端执行器参数

表 2.31 获取末端执行器参数接口说明

原型	int GetEndEffectorParams(EndEffectorParams *endEffectorParams)
描述	获取当前末端参数

参数	EndEffectorParams 定义：
	typedef struct tagEndEffectorParams {
	float xBias; //末端 x 方向长度
	float yBias; //末端 y 方向长度
	float zBias; //末端 z 方向长度
	}EndEffectorParams;
	
	endEffectorParams:末端执行器参数指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10.3 设置激光输出

表 2.32 设置激光输出接口说明

原型	int SetEndEffectorLaser(bool enableCtrl, bool on, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置激光开关
参数	enableCtrl:使能控制 on:激光是否开启 isQueued:是否将该指令指定为队列命令 queuedCmdIndex:队列命令索引（控制器返回）
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10.4 获取激光输出

表 2.33 获取激光输出接口说明

原型	int GetEndEffectorLaser(bool *isCtrlEnabled, bool *isOn)
描述	获取激光开关状态
参数	isCtrlEnabled:控制是否使能 isOn:激光是否开启
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10.5 设置吸盘输出

表 2.34 设置吸盘输出接口说明

原型	int SetEndEffectorSuctionCup(bool enableCtrl, bool suck, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置吸盘吸放
参数	enableCtrl:使能控制 suck:是否吸住 isQueued:是否将该指令指定为队列命令 queuedCmdIndex:队列命令索引（控制器返回）
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10.6 获取吸盘输出

表 2.35 获取吸盘输出接口说明

原型	int GetEndEffectorSuctionCup(bool *isCtrlEnabled, bool *isSucked)
描述	获取吸盘吸放状态
参数	isCtrlEnabled:控制是否使能 isSucked:吸盘是否吸住
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10.7 设置爪子输出

表 2.36 设置爪子输出接口说明

原型	int SetEndEffectorGripper(bool enableCtrl, bool grip, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置爪子抓住/释放
参数	enableCtrl:使能控制 grip:是否抓住 isQueued:是否将该指令指定为队列命令 queuedCmdIndex:队列命令索引（控制器返回）
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.10.8 获取爪子输出

表 2.37 获取爪子输出接口说明

原型	int GetEndEffectorGripper(bool *isCtrlEnabled, bool *isGripped)
描述	获取爪子夹住状态
参数	isCtrlEnabled:控制是否使能 isGripped:爪子是否夹住
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.11 ARM 方向

2.11.1 设置手臂方向

表 2.38 设置手臂方向接口说明

原型	int SetArmOrientation(ArmOrientation armOrientation, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置手臂方向
参数	ArmOrientation 定义： typedef enum tagArmOrientation { LeftyArmOrientation, RightyArmOrientation }ArmOrientation; armOrientation:手臂方向 isQueued:是否将该指令指定为队列命令 queuedCmdIndex:队列命令索引（控制器返回）
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注:该命令当前仅适用于 SCARA 机型。

2.11.2 获取手臂方向

表 2.39 获取手臂方向接口说明

原型	int GetArmOrientation(ArmOrientation *armOrientation)
描述	设置手臂方向
参数	ArmOrientation 定义： typedef enum tagArmOrientation { LeftyArmOrientation, RightyArmOrientation }ArmOrientation; armOrientation:手臂方向变量指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.12 JOG 功能

2.12.1 设置关节点动参数

表 2.40 设置关节点动参数接口说明

原型	int SetJOGJointParams(JOGJointParams *jogJointParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置关节点动参数

参数	JOGJointParams 定义：
	typedef struct tagJOGJointParams {
	float velocity[4]; //4 轴关节速度
	float acceleration[4]; //4 轴关节加速度
	}JOGJointParams;
	
	jogJointParams:关节点动参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.12.2 获取关节点动参数

表 2.41 获取关节点动参数接口说明

原型	int GetJOGJointParams(JOGJointParams *jogJointParams)
描述	获取关节点动参数

参数	JOGJointParams 定义：
	typedef struct tagJOGJointParams {
	float velocity[4]; //4 轴关节速度
	float acceleration[4]; //4 轴关节加速度
	}JOGJointParams;
	
	jogJointParams:关节点动参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.12.3 设置坐标轴点动参数

表 2.42 设置坐标轴点动参数接口说明

原型	int SetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置坐标轴点动参数

参数	JOGCoordinateParams 定义：
	typedef struct tagJOGCoordinateParams {
	float velocity[4]; //4 轴坐标轴（x,y,z,r）速度
	float acceleration[4]; //4 轴坐标轴（x,y,z,r）加速度
	}JOGCoordinateParams;
	
	jogCoordinateParams:坐标轴点动参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.12.4 获取坐标轴点动参数

表 2.43 获取坐标轴点动参数接口说明

原型	int GetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams)
描述	获取坐标轴点动参数

参数	JOGCoordinateParams 定义：
	typedef struct tagJOGCoordinateParams {
	float velocity[4]; //4 轴坐标轴（x,y,z,r）速度
	float acceleration[4]; //4 轴坐标轴（x,y,z,r）加速度
	}JOGCoordinateParams;
	
	jogCoordinateParams:坐标轴点动参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.12.5 设置点动公共参数

表 2.44 设置点动公共参数接口说明

原型	int SetJOGCommonParams(JOGCommonParams *jogCommonParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置点动公共参数

参数	JOGCommonParams 定义：
	typedef struct tagJOGCommonParams {
	float velocityRatio; //速度比例，关节点动和坐标轴点动共用
	float accelerationRatio; //加速度比例，关节点动和坐标轴点动共用
	}JOGCommonParams;
	
	jogCommonParams:点动公共参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.12.6 获取点动公共参数

表 2.45 获取点动公共参数接口说明

原型	int GetJOGCommonParams(JOGCommonParams *jogCommonParams)
描述	获取点动公共参数

参数	JOGCommonParams 定义：
	typedef struct tagJOGCommonParams {
	float velocityRatio; //速度比例，关节点动和坐标轴点动共用
	float accelerationRatio; //加速度比例，关节点动和坐标轴点动共用
	}JOGCommonParams;
	
	jogCommonParams:点动公共参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.12.7 执行点动功能

表 2.46 执行点动功能接口说明

原型	int SetJOGCmd(JOGCmd *jogCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行

参数	JOGCmd 定义：
	typedef struct tagJOGCmd {
	uint8_t isJoint; //点动方式：0——坐标轴点动； 1——关节点动
	uint8_t cmd; //点动命令（取值范围 0~8）
	}JOGCmd;
	
	//点动命令详细说明
	enum {
	IDEL, //无效状态
	AP_DOWN, // X+/Joint1+
	AN_DOWN, // X-/Joint1-
	BP_DOWN, // Y+/Joint2+
	BN_DOWN, // Y-/Joint2-
	CP_DOWN, // Z+/Joint3+
	CN_DOWN, // Z-/Joint3-
	DP_DOWN, // R+/Joint4+
	DN_DOWN // R-/Joint4-
	};
	
	jogCmd:点动功能变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.13 PTP 功能

2.13.1 设置关节点位参数

表 2.47 设置关节点位参数接口说明

原型	int SetPTPJointParams(PTPJointParams *ptpJointParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置关节点位参数

参数	PTPJointParams 定义：
	typedef struct tagPTPJointParams {
	float velocity[4]; //PTP 模式下 4 轴关节速度
	float acceleration[4]; //PTP 模式下 4 轴关节加速度
	}PTPJointParams;
	
	ptpJointParams:PTP 关节点位参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.13.2 获取关节点位参数

表 2.48 获取关节点位参数接口说明

原型	int GetPTPJointParams(PTPJointParams *ptpJointParams)
描述	获取关节点位参数

参数	PTPJointParams 定义：
	typedef struct tagPTPJointParams {
	float velocity[4]; //PTP 模式下 4 轴关节速度
	float acceleration[4]; //PTP 模式下 4 轴关节加速度
	}PTPJointParams;
	
	ptpJointParams:PTP 关节点位参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.13.3 设置坐标轴点位参数

表 2.49 设置坐标轴点位参数接口说明

原型	int SetPTPCoordinateParams(PTPCoordinateParams *ptpCoordinateParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置坐标轴点位参数

参数	PTPSpeedParams 定义：
	typedef struct tagPTPCoordinateParams {
	float xyzVelocity; //PTP 模式下 xyz 3 轴坐标轴速度
	float rVelocity; //PTP 模式下末端速度
	float xyzAcceleration; //PTP 模式下 xyz 3 轴坐标轴加速度
	float rAccleration; //PTP 模式下末端加速度
	} PTPCoordinateParams;
	
	ptpCoordinateParams:PTP 坐标轴点位参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.13.4 获取坐标轴点位参数

表 2.50 获取坐标轴点位参数接口说明

原型	int GetPTPCoordinateParams(PTPCoordinateParams *ptpCoordinateParams)
描述	获取坐标轴点位参数

参数	PTPCoordinateParams 定义：
	typedef struct tagPTPCoordinateParams {
	float xyzVelocity; //PTP 模式下 xyz 3 轴坐标轴速度
	float rVelocity; //PTP 模式下末端速度
	float xyzAcceleration; //PTP 模式下 xyz 3 轴坐标轴加速度
	float rAccleration; //PTP 模式下末端加速度
	} PTPCoordinateParams;
	
	ptpCoordinateParams:PTP 坐标轴点位参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.13.5 设置门型模式点位参数

表 2.51 设置门型模式点位参数接口说明

原型	int SetPTPJumpParams(PTPJumpParams *ptpJumpParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置门型模式点位参数

参数	PTPJumpParams 定义：
	typedef struct tagPTPJumpParams {
	float jumpHeight; //门型模式运动抬升距离
	float zLimit; //门型模式运动最大抬升高度限制
	}PTPJumpParams;
	
	ptpJumpParams:PTP 门型模式参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.13.6 获取门型模式点位参数

表 2.52 获取门型模式点位参数接口说明

原型	int GetPTPJumpParams(PTPJumpParams *ptpJumpParams)
描述	获取门型模式点位参数

参数	PTPJumpParams 定义：
	typedef struct tagPTPJumpParams {
	float jumpHeight; //门型模式运动抬升距离
	float zLimit; //门型模式运动最大抬升高度限制
	}PTPJumpParams;
	
	ptpJumpParams: PTP 门型模式参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.13.7 设置点位公共参数

表 2.53 设置点位公共参数接口说明

原型	int SetPTPCommonParams(PTPCommonParams *ptpCommonParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置点位公共参数

参数	PTPCommonParams 定义：
	typedef struct tagPTPCommonParams {
	float velocityRatio; //PTP 模式速度比例，关节和坐标轴模式共用
	float accelerationRatio;//PTP 模式加速度比例，关节和坐标轴模式共用
	}PTPCommonParams;
	
	ptpCommonParams:PTP 公共参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.13.8 获取点位公共参数

表 2.54 获取点位公共参数接口说明

原型	int GetPTPCommonParams(PTPCommonParams *ptpCommonParams)
描述	获取点位速度参数

参数	PTPCommonParams 定义：
	typedef struct tagPTPCommonParams {
	float velocityRatio; //PTP 模式速度比例，关节和坐标轴模式共用
	float accelerationRatio;//PTP 模式加速度比例，关节和坐标轴模式共用
	}PTPCommonParams;
	
	ptpCommonParams:PTP 公共参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.13.9 执行点位功能

表 2.55 执行点位功能接口说明

原型	int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行点位功能

参数	PTPCmd 定义：
	typedef struct tagPTPCmd {
	uint8_t ptpMode; //PTP 模式（取值范围 0~8）
	float x; //x,y,z,r 为 ptpMode 运动方式的参数，可为坐标、
	//关节角度、或者坐标/角度增量
	float y;
	float z;
	float r;
	}PTPCmd;
	
	//其中， ptpMode 取值如下：
	enum {
	JUMP_XYZ, //门型运动，参数为目标点坐标
	MOVJ_XYZ, //关节运动，参数为目标点坐标
	MOVL_XYZ, //直线运动，参数为目标点坐标
	JUMP_ANGLE, //门型运动，参数为目标点关节角度
	MOVJ_ANGLE, //关节运动，参数为目标点关节角度
	MOVL_ANGLE, //直线运动，参数为目标点关节角度
	MOVJ_INC, //关节运动增量模式，参数为目标点关节角度增量
	MOVL_INC, //直线运动增量模式，参数为目标点坐标增量
	MOVJ_XYZ_INC, //关节运动增量模式，参数为目标点坐标增量
	JUMP_MOVL_XYZ, //门型运动，平移时运动模式为 MOVL
	};
	
	ptpCmd:PTP 命令变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.14 CP 功能

2.14.1 设置连续轨迹功能参数

表 2.56 设置连续轨迹功能参数接口说明

原型	int SetCPParams(CPParams *cpParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置连续轨迹功能参数

参数	CPParams 定义：
	typedef struct tagCPParams {
	float planAcc; //规划加速度最大值
	float junctionVel; //拐角加速度最大值
	union {
	float acc; //实际加速度最大值，非实时模式下使用
	float period; //插补周期， 实时模式下使用
	};
	uint8_t realTimeTrack; //0—非实时模式； 1—实时模式
	}CPParams;
	
	cpParams:连续轨迹功能参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.14.2 获取连续轨迹功能参数

表 2.57 获取连续轨迹功能参数接口说明

原型	int GetCPParams(CPParams *cpParams)
描述	获取连续轨迹功能参数

参数	CPParams 定义：
	typedef struct tagCPParams {
	float planAcc; //规划加速度最大值
	float junctionVel; //拐角加速度最大值
	union {
	float acc; //实际加速度最大值，非实时模式下使用
	float period; //插补周期， 实时模式下使用
	};
	uint8_t realTimeTrack; //0—非实时模式； 1—实时模式
	}CPParams;
	
	cpParams:连续轨迹功能参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.14.3 执行连续轨迹功能

表 2.58 执行连续轨迹功能接口寿命

原型	int SetCPCmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行连续轨迹功能

参数	CPCmd 定义：
	typedef struct tagCPCmd {
	uint8_t cpMode; //CP 模式 0-相对模式 1-绝对模式
	float x; //x 坐标增量 / x 轴坐标
	float y; //y 坐标增量 / y 轴坐标
	float z; //z 坐标增量 / z 轴坐标
	union {
	float velocity; // Reserved
	float power; //激光功率
	};
	}CPCmd;
	
	cpCmd:连续轨迹功能功能变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注：当指令队列中有多条连续的 CP 指令时，Dobot 控制器将自动前瞻。前瞻的条件是，队列中这些 CP 指令之间没有 JOG、PTP、ARC、WAIT、TRIG 等指令。

2.14.4 执行连续轨迹灰度雕刻功能

表 2.59 执行连续轨迹功能接口说明

原型	int SetCPLECmd (CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行连续轨迹灰度雕刻功能 激光功率将在运动命令起始时应用

参数	CPCmd 定义：
	typedef struct tagCPCmd {
	uint8_t cpMode; //CP 模式 0-相对模式 1-绝对模式
	float x; //x 坐标增量(相对模式) / x 轴坐标(绝对模式)
	float y; //y 坐标增量(相对模式) / y 轴坐标(绝对模式)
	float z; //z 坐标增量(相对模式) / z 轴坐标(绝对模式)
	union {
	float velocity; //预留
	float power; // 激光功率 0~100
	}
	}CPCmd;
	
	cpCmd:连续轨迹功能功能变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.15 ARC 功能

2.15.1 设置圆弧插补功能参数

表 2.60 设置圆弧插补功能参数接口说明

原型	int SetARCParams(ARCParams *arcParams, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置圆弧插补功能参数

参数	ARCParams 定义：
	typedef struct tagARCParams {
	float xyzVelocity; //圆弧运动 xyz 三坐标轴速度
	float rVelocity; //圆弧运动末端旋转速度
	float xyzAcceleration; //圆弧运动 xyz 三坐标轴加速度
	float rAcceleration; //圆弧运动末端旋转加速度
	} ARCParams;
	
	arcParams:圆弧插补功能参数变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.15.2 获取圆弧插补功能参数

表 2.61 获取圆弧插补功能参数接口说明

原型	int GetARCParams(ARCParams *arcParams)
描述	获取圆弧插补功能参数

参数	ARCParams 定义：
	typedef struct tagARCParams {
	float xyzVelocity; //圆弧运动 xyz 三坐标轴速度
	float rVelocity; //圆弧运动末端旋转速度
	float xyzAcceleration; //圆弧运动 xyz 三坐标轴加速度
	float rAcceleration; //圆弧运动末端旋转加速度
	} ARCParams;
	
	arcParams:圆弧插补功能参数变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.15.3 执行圆弧插补功能

表 2.62 执行圆弧插补功能接口说明

原型	int SetARCCmd(ARCCmd *arcCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行圆弧插补功能

参数	ARCCmd 定义：
	typedef struct tagARCCmd {
	struct {
	float x;
	float y;
	float z;
	float r;
	}cirPoint;
	struct {
	float x;
	float y;
	float z;
	float r;
	}toPoint;
	}ARCCmd;
	
	arcCmd:圆弧插补功能变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

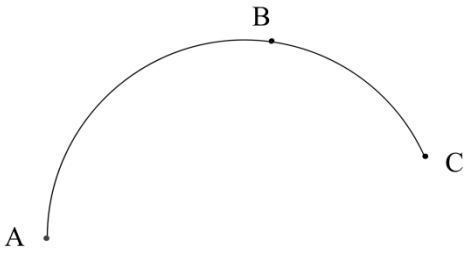
圆弧轨迹说明：

1. 圆弧轨迹是空间的圆弧，由当前点、圆弧上任一点和圆弧结束点三点共同确定的；
2. 圆弧总是从起点经过圆弧上一点再到结束点。

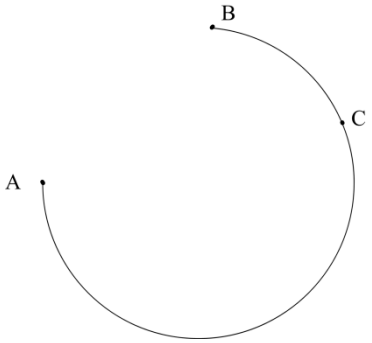
圆弧轨迹如下图示例：

（a） A 为当前点， B 为圆弧上任一点，C 为结束点；

（b） A 为当前点， C 为圆弧上任一点，B 为结束点。



（a） 起点 A 终点 C



（b） 起点 A 终点 B

2.16 WAIT 功能

2.16.1 执行时间等待功能

表 2.63 执行时间等待功能接口说明

原型	int SetWAITCmd(WAITCmd *waitCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行时间等待功能

参数	WAITCmd 定义：
	typedef struct tagWAITCmd {
	uint32_t timeout; // 单位:ms
	}WAITCmd;
	
	waitCmd:时间等待功能变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注：该指令只能作为队列指令，isQueued 必须设置为 true。将该命令设置为立即指令可能会导致正在执行的 WAIT 队列指令的超时时间变化。

2.17 TRIG 功能

2.17.1 执行触发功能

表 2.64 执行触发功能接口说明

原型	int SetTRIGCmd(TRIGCmd *trigCmd, bool isQueued, uint64_t *queuedCmdIndex)
描述	执行触发功能

参数	TRIGCmd 定义：
	typedef struct tagTRIGCmd {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint8_t mode; //触发模式 0—IO 触发 1—AD 触发
	uint16_t threshold; //触发条件 IO 值—0/1 ADC 值—0~4095
	} TRIGCmd;
	
	trigCmd:触发功能变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

注：该指令只能作为队列指令，isQueued 必须设置为 true。将该命令设置为立即指令可能会导致正在执行的 TRIG 队列指令的触发条件变化。

2.18 EIO 功能

在 Dobot 控制器中，所有的扩展 I/O 都是统一编址的。根据现有情况，I/O 的功能可以有：

- 高低电平输出功能；
- PWM 输出功能；
- 读取输入高低电平功能；
- 读取输入模数转换值功能。

部分 I/O 可能同时具有以上的功能。在使用不同的功能时，需要先配置 I/O 的复用。

2.18.1 设置 I/O 复用

表 2.65 设置 I/O 复用功能接口说明

原型	int SetIOMultiplexing(IOMultiplexing *ioMultiplexing, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置 I/O 复用

参数	IOMultiplexing 定义：
	typedef struct tagIOMultiplexing {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint8_t multiplex; //EIO 功能
	} IOMultiplexing;
	
	其中 mutiplex 支持的取值如下所示：
	typedef enum tagIOFunction {
	IOFunctionDummy, //不配置功能
	IOFunctionPWM, //PWM 输出
	IOFunctionDO, //IO 输出
	IOFunctionDI, //IO 输入
	IOFunctionADC //AD 输入
	} IOFunction;
	
	ioMultiplexing:I/O 复用变量
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.18.2 读取 I/O 复用

表 2.66 读取 I/O 复用功能接口说明

原型	int GetIOMultiplexing(IOMultiplexing *ioMultiplexing)
描述	读取 I/O 复用

参数	IOMultiplexing 定义：
	typedef struct tagIOMultiplexing {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint8_t multiplex; //EIO 功能
	} IOMultiplexing;
	
	其中 mutiplex 支持的取值如下所示：
	typedef enum tagIOFunction {
	IOFunctionDummy, //不配置功能
	IOFunctionPWM, //PWM 输出
	IOFunctionDO, //IO 输出
	IOFunctionDI, //IO 输入
	IOFunctionADC //AD 输入
	} IOFunction;
	
	ioMultiplexing:I/O 复用变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.18.3 设置 I/O 输出电平

表 2.67 设置 I/O 输出电平接口说明

原型	int SetIODO(IODO *ioDO, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置 I/O 输出电平

参数	IODO 定义：
	typedef struct tagIODO {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint8_t level; //输出电平 0-低电平 1-高电平
	}IODO;
	
	ioDO:I/O 输出电平结构体变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.18.4 读取 I/O 输出电平

表 2.68 读取 I/O 输出电平

原型	int GetIODO(IODO *ioDO)
描述	读取 I/O 输出电平

参数	IODO 定义：
	typedef struct tagIODO {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint8_t level; //输出电平 0-低电平 1-高电平
	}IODO;
	
	ioDO:I/O 输出电平结构体变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.18.5 设置 PWM 输出

表 2.69 设置 PWM 输出接口说明

原型	int SetIOPWM(IOPWM *ioPWM, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置 I/O PWM 输出

参数	IOPWM 定义：
	typedef struct tagIOPWM {
	uint8_t address; //EIO 地址（取值范围 1~20）
	float frequency; //PWM 频率 10HZ~1MHz
	float dutyCycle; //PWM 占空比 0~100
	}IOPWM;
	
	ioPWM:I/O PWM 输出结构体变量指针
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.18.6 读取 PWM 输出

表 2.70 读取 PWM 输出接口说明

原型	int GetIOPWM(IOPWM *ioPWM)
描述	读取 I/O PWM 输出

参数	IOPWM 定义：
	typedef struct tagIOPWM {
	uint8_t address; //EIO 地址（取值范围 1~20）
	float frequency; //PWM 频率 10HZ~1MHz
	float dutyCycle; //PWM 占空比 0~100
	}IOPWM;
	
	ioPWM:I/O PWM 输出结构体变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.18.7 读取 I/O 输入电平

表 2.71 读取 I/O 输入电平接口说明

原型	int GetIODI(IODI *ioDI)
描述	读取 I/O 输入电平

参数	IODI 定义：
	typedef struct tagIODI {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint8_t level; //输入 IO 电平 0-低电平 1-高电平
	}IODI;
	
	ioDI:I/O 输入电平结构体变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.18.8 读取 I/O 模数转换值

表 2.72 读取 I/O 模数转换值

原型	int GetIOADC(IOADC *ioADC)
描述	读取 I/O 模数转换值

参数	IOADC 定义：
	typedef struct tagIOADC {
	uint8_t address; //EIO 地址（取值范围 1~20）
	uint16_t value; //输入 ADC 值，范围 0~4095
	}IOADC;
	
	ioADC:I/O 模数转换值结构体变量指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.18.9 设置扩展电机接口

表 2.73 设置 I/O 复用功能接口说明

原型	int SetEMotor(EMotor *eMotor, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置 I/O 复用

参数	EMotor 定义：
	typedef struct tagEMotor {
	uint8_t index; //取值范围 0/1 0-Stepper1 1-Stepper2
	uint8_t isEnabled; //电机控制使能
	float speed; //电机控制速度(脉冲个数每秒)
	}EMotor;
	
	eMotor:扩展电机控制结构体
	isQueued:是否将该指令指定为队列命令
	queuedCmdIndex:队列命令索引（控制器返回）

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满 DobotCommunicate_Timeout:指令无返回，导致超时

2.19 CAL 功能

2.19.1 设置角度传感器静态偏差

由于角度传感器焊接、机器状态等原因，大小臂上的角度传感器可能存在一个静态偏差。我们可以通过各种手段（如调平、与标准源比较），得到此静态偏差，并通过此 API 写入到设备中。

表 2.74 设置角度传感器静态偏差接口说明

原型	int SetAngleSensorStaticError(float rearArmAngleError, float frontArmAngleError)
描述	设置大小臂角度传感器静态偏差
参数	rearArmAngleError:大臂角度传感器静态偏差 frontArmAngleError:小臂角度传感器静态偏差
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.19.2 读取角度传感器静态偏差

表 2.75 读取角度传感器静态偏差接口说明

原型	int SetAngleSensorStaticError(float rearArmAngleError, float frontArmAngleError)
描述	设置大小臂角度传感器静态偏差
参数	rearArmAngleError:大臂角度传感器静态偏差 frontArmAngleError:小臂角度传感器静态偏差
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20 WIFI 功能

2.20.1 设置 WIFI 配置模式

表 2.76 设置 WIFI 配置模式接口说明

原型	int SetWIFIConfigMode(bool enable)
描述	设置 WIFI 配置模式
参数	enable:是否使能配置模式
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.2 获取当前 WIFI 是否配置模式

表 2.77 获取当前 WIFI 是否配置模式接口说明

原型	int GetWIFIConfigMode(bool *isEnabled)
描述	获取当前 WIFI 是否配置模式
参数	isEnabled:传入变量指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.3 设置 SSID

表 2.78 设置网络 SSID 接口说明

原型	int SetWIFISSID(const char *ssid)
描述	设置网络 SSID
参数	ssid：网络 SSID 字符串指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.4 获取当前设置 SSID

表 2.79 获取当前设置 SSID

原型	int GetWIFISSID(char *ssid, uint32_t maxLen)
描述	获取当前设置 SSID
参数	ssid：网络 SSID 字符串指针 maxLen：传入外部缓冲区长度，以避免溢出
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.5 设置网络密码

表 2.80 设置网络密码接口说明

原型	int SetWIFIPassword(const char *password)
描述	设置网络密码
参数	password：网络密码字符串指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.6 获取当前设置网络密码

表 2.81 获取当前设置网络密码

原型	int GetWIFIPassword(char *password, uint32_t maxLen)
描述	获取当前设置网络密码
参数	password：网络密码字符串指针 maxLen：传入外部缓冲区长度，以避免溢出
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.7 设置 IP 地址

表 2.82 设置 IP 地址接口说明

原型	int SetWIFIIPAddress(WIFIIPAddress *wifiIPAddress)
描述	设置 IP 地址
参数	typedef struct tagWIFIIPAddress { uint8_t dhcp; uint8_t addr[4]; }WIFIIPAddress; wifiIPAddr：IP 地址结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.8 获取当前设置 IP 地址

表 2.83 获取当前设置 IP 地址接口说明

原型	int GetWIFIIPAddress(WIFIIPAddress *wifiIPAddress)
描述	获取当前设置 IP 地址
参数	typedef struct tagWIFIIPAddress { uint8_t dhcp; uint8_t addr[4]; }WIFIIPAddress; wifiIPAddr：IP 地址结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.9 设置子网掩码

表 2.84 设置子网掩码接口说明

原型	int SetWIFINetmask(WIFINetmask *wifiNetmask)
描述	设置子网掩码
参数	typedef struct tagWIFINetmask { uint8_t addr[4]; }WIFINetmask; wifiNetmask：子网掩码结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.10 获取当前设置子网掩码

表 2.85 获取当前设置子网掩码接口说明

原型	int GetWIFINetmask(WIFINetmask *wifiNetmask)
描述	获取当前设置子网掩码
参数	typedef struct tagWIFINetmask { uint8_t addr[4]; }WIFINetmask; wifiNetmask：子网掩码结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.11 设置网关

表 2.86 设置子网掩码接口说明

原型	int SetWIFIGateway(WIFIGateway *wifiGateway)
描述	设置网关
参数	typedef struct tagWIFIGateway { uint8_t addr[4]; }WIFIGateway; wifiGateway：网关结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.12 获取当前设置网关

表 2.87 获取当前设置子网掩码接口说明

原型	int GetWIFIGateway(WIFIGateway *wifiGateway)
描述	获取当前设置网关
参数	typedef struct tagWIFIGateway { uint8_t addr[4]; }WIFIGateway; wifiGateway：网关结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.13 设置 DNS

表 2.88 设置 DNS 接口说明

原型	int SetWIFIDNS(WIFIDNS *wifiDNS)
描述	设置 DNS
参数	typedef struct tagWIFIDNS { uint8_t addr[4]; }WIFIDNS; wifiDNS：DNS 结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.14 获取当前设置 DNS

表 2.89 获取当前设置 DNS 接口说明

原型	int GetWIFIDNS(WIFIDNS *wifiDNS)
描述	获取当前设置 DNS
参数	typedef struct tagWIFIDNS { uint8_t addr[4]; }WIFIDNS; wifiDNS：DNS 结构体指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

2.20.15 获取当前 Wi-Fi 模块的连接状态

表 2.90 获取当前 Wi-Fi 模块连接状态接口说明

原型	int GetWIFIConnectStatus(bool *isConnected)
描述	获取当前 Wi-Fi 模块连接状态
参数	isConnected：Wi-Fi 连接状态变量指针
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_BufferFull:指令队列已满（本接口不会返回该值） DobotCommunicate_Timeout:指令无返回，导致超时

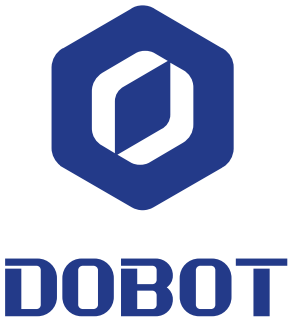
2.21 其他功能

2.21.1 事件循环功能

在某些语言中，当调用 API 接口后，由于没有事件循环，应用程序将直接退出，因此将导致指令没有下发到 Dobot 控制器中。为了避免这种情况，我们提供了事件循环接口，在应用程序退出前调用（目前已知的需要做此处理的语言有 Python）。

表 2.91 事件循环功能接口说明

原型	void DobotExec(void)
描述	事件循环功能
参数	无
返回	无



深圳市越疆科技有限公司

邮编：510630

网址：www.dobot.cc

电话：(0755)38730916

地址：深圳市南山区西丽桃源街道塘朗工业区 A 区 8 栋 4 楼