Dobot 通讯协议文档

Dobot Magician

TN01010101 V1.1.3 Date: 2017/9/6

深圳市越疆科技有限公司

修订历史

版本	日期	原因
V1.0.0	2016/09/22	创建文档
V1.0.1	2016/11/18	增加通信参数说明
V1.0.2	2016/11/21	增加 BLE 读/写 Characteristic UUID
V1.0.3	2016/11/30	增加运动控制相关参数说明（JOG PTP CP ARC 等）
V1.0.4	2016/12/20	获取指令队列剩余空间的补充
V1.1.0	2017/05/05	修复执行回零命令的错误、增加导轨相关接口、增加 UID 端口、增加颜色传感器端口…
V1.1.1	2017/5/25	获取 EIO 状态下发端口参数修改
V1.1.2	2017/6/26	颜色传感器和光电开关、丢步检测功能添加 EIO 输入增加上拉输入和下拉输入
V1.1.3	2017/9/6	IO 复用顺序错误修复、IO 触发条件错误修复、设置 EMotor 文本错误修复

目 录

1. 适用范围

本文档适用于 Dobot Magician 产品上位机与 Dobot Magician 机械臂之间命令/数据交互的通信协议。

2. 通讯协议

2.1 通信参数

1. USB 转串口
- 波特率：115200bps；
- 数据位：8 位；
- 停止位：1 位；
- 校验位：无。

2. Wi-Fi
- IP：路由等分配；
- 端口：8899。

3. BLE
- Service UUID：0003CDD0-0000-1000-8000-00805F9B0131；
- 读（指令）端口 Characteristic UUID：0003CDD1-0000-1000-8000-00805F9B0131；
- 写（指令）端口 Characteristic UUID：0003CDD2-0000-1000-8000-00805F9B0131。

4. 扩展串口（TTL）
- 波特率：115200bps；
- 数据位：8 位；
- 停止位：1 位；
- 校验位：无。

2.2 协议简介

Dobot Magician 机械臂可通过 PC/Android/iOS 客户端控制，这些设备与机械臂之间通过特定的通信协议实现数据传输，进而实现控制。目前 Dobot Magician 可以通过默认 USB 转串口、TLL 电平串口、Wi-Fi（UDP）控制。

物理层每次接收的数据是 8 位原始数据，需要制定通信协议来确定数据传输的开始与结束、以及校验数据的准确性。通信协议一般需要包括数据包的包头、数据负载、负载校验和等，来保证数据的准确传输。

2.2.1 协议特点

Dobot 的通信协议的特点如下：

1. 协议指令不定长；
2. 协议指令由包头、负载帧长、负载帧、校验组成；
3. 所有的通信都由主机主动发起，且对于所有通信指令，下位机都返回（无论读写）；对于队列指令，其返回带有 64 位的执行索引值；
4. 指令分为立即指令和队列指令。立即指令将立即被调用执行，队列指令将被放入下位机队列中，串行地执行；所有的读操作都是立即指令；对于写（或设置）操作，其中运动类型的指令都应该是队列命令（比如回零、JOG、PTP 等），设置参数的指令既可以是立即指令，也可以是队列指令；
5. 在主机向下位机发送队列命令前，应查询下位机命令队列的剩余空间（可查询一次，发送多条命令）；
6. 立即指令总是立即执行完成；队列指令的执行完成情况可通过查询当前下位机正在执行的队列命令索引，与该条命令的索引（第 3 点中提到的命令中返回的）的比较得到；

7. 命令中的参数使用小端模式。

2.2.2 校验计算

在 Dobot Magician 的通信协议中，发送端校验计算方法如下：

1. 将负载帧（Payload）中的所有内容按照字节（8 位）的方式逐字节相加，得到一个结果 R（8 位）；
2. 对结果 R（8 位）求二补数，存入校验字节中。

二补数：Two's complement。对于一个 N 位的数字，其二补数等于2^N减去该数。在本协议中，假设上述的结果 R 为 0x0A，其二补数也即上述的校验结果等于(2^8 - 0x0A) = (256 - 10) = 246 = 0xF6。

在接收端，校验一帧数据是否正确的方法是：

1. 将负载帧（Payload）中的所有内容按照字节（8 位）的方式逐字节相加，得到一个结果 A；
2. 结果 A 与校验字节相加，如果等于 0，则说明校验正确。

2.2.3 协议分类

根据实现功能不同，又可分为以下几部分：队列执行控制命令、设备信息相关命令、公共参数命令、回零功能命令、手持示教命令、点动模式命令、PTP 模式命令、CP 模式命令、TRACK 模式命令、WAIT 模式命令、TRIG 触发相关命令、IO 控制命令等。

通过分类，将通信协议功能 ID 分成表 1 中的几个大项：

表 1 功能项划分

功能项划分	Function ID 区间	可用 ID 个数
ProtocolFunctionDeviceInfoBase	[0, 10)	10
ProtocolFunctionPoseBase	[10, 20)	10
ProtocolFunctionALARMBase	[20, 30)	10
ProtocolFunctionHOMEBase	[30, 40)	10
ProtocolFunctionHHTBase	[40, 50)	10
ProtocolFunctionArmOrientationBase	[50, 60)	10
ProtocolFunctionEndEffectorBase	[60, 70)	10
ProtocolFunctionJOGBase	[70, 80)	10
ProtocolFunctionPTPBase	[80, 90)	10
ProtocolFunctionCPBase	[90, 100)	10
ProtocolFunctionARCBase	[100, 110)	10
ProtocolFunctionWAITBase	[110, 120)	10
ProtocolFunctionTRIGBase	[120, 130)	10
ProtocolFunctionEIOBase	[130, 140)	10
ProtocolFunctionCALBase	[140, 150)	10
ProtocolFunctionWIFIBase	[150, 160)	10
ProtocolFunctionQueuedCmdBase	[240, 250)	10
ProtocolMax	256	1

2.2.4 其他说明

1. 在下文的每条指令说明中，都带有 ID 说明；

2. 下文中 Ctrl 字节中，rw 为 Ctrl 字节的第 0 位，isQueued 为 Ctrl 字节的第 1 位。

2.3 设备信息

该部分命令用于设置设备 SN 号、设备名称和设备版本号，并可以通过命令读回设备当前信息。

2.3.1 设置和获取设备序列号（Set/Get DeviceSN）

1. 设置设备序列号（SetDeviceSN），下发的指令包格式如表 2 所示，返回指令包格式如表 3 所示；

表 2 设置设备序列号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	0	1	0	char* DeviceSN	Payload checksum

表 3 设置设备序列号返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	0	1	0	Empty	Payload checksum

2. 获取设备序列号（GetDeviceSN），下发的指令包格式如表 4 所示，返回指令包格式如表 5 所示。

表 4 获取设备序列号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	0	0	0	Empty	Payload checksum

表 5 获取设备序列号返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	0	0	0	char* DeviceSN	Payload checksum

2.3.2 设置和获取设备名称（Set/Get DeviceName）

1. 设置设备名称（SetDeviceName），下发的指令包格式如表 6 所示，返回指令包格式如表 7 所示；

表 6 设置设备名称指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	1	1	0	char* DeviceName	Payload checksum

表 7 设置设备名称返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	1	1	0	Empty	Payload checksum

2. 获取设备名称（GetDeviceName），下发的指令包格式如表 8 所示，返回指令包格式如表 9 所示。

表 8 获取设备名称指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	1	0	0	Empty	Payload checksum

表 9 获取设备名称返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	1	0	0	char* DeviceName	Payload checksum

2.3.3 获取设备版本号（GetDeviceVersion）

获取设备版本号（GetDeviceVersion），下发的指令包格式如表 10 所示，返回指令包格式如表 11 所示。

表 10 获取设备版本号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	2	0	0	Empty	Payload checksum

表 11 获取设备版本号返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+3	2	0	0	uint8_t: majorVersion	uint8_t: minorVersion	uint8_t : revision	Payload checksum

2.3.4 设置设备导轨状态（Set DeviceWithL）

1. 设置设备导轨状态（Set DeviceWithL），下发的指令包格式如表 12 所示，返回指令包格式如表 13 所示；

表 12 设置设备导轨状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	3	1	0	uint8_t:WithL	Payload checksum

表 13 设置设备导轨状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	3	1	0	Empty	Payload checksum

2. 获取设备名称（GetDeviceName），下发的指令包格式如表 14 所示，返回指令包格式如表 15 所示。

表 14 获取设备导轨状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	3	0	0	Empty	Payload checksum

表 15 获取设备导轨状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	3	0	0	uint8_t:WithL	Payload checksum

2.3.5 获取设备系统滴答时钟（Get DeviceTime）

1. 获取设备系统滴答时钟（GetDeviceTime），下发的指令包格式如表 16 所示，返回指令包格式如表 17 所示。

表 16 获取设备 UID 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	4	0	0	Empty	Payload checksum

表 17 获取设备 UID 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	4	0	0	uint32_t：gSystick	Payload checksum

2.3.6 获取设备 UID（Get DeviceID）

1. 获取设备 UID（GetDeviceID），下发的指令包格式表 18 所示，返回指令包格式如表 19 所示。

表 18 获取设备 UID 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	5	0	0	Empty	Payload checksum

表 19 获取设备 UID 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+12	5	0	0	uint32_t[3]：DeviceID	Payload checksum

2.4 实时位姿

实现设置初始姿态、获取实时位姿、运动学参数等功能。

2.4.1 获取实时位姿（GetPose）

获取实时位姿（GetPose），下发的指令包格式如表 20 所示，返回指令包格式如表 21 所示。

表 20 获取实时位姿指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	10	0	0	Empty	Payload

						checksum

表 21 获取实时位姿返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	10	0	0	Pose（见 程序 1）	Payload checksum

程序 1 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;
```

2.4.2 重设实时姿态（ResetPose）

重设实时姿态（ResetPose），下发的指令包格式如表 22 所示，返回指令包格式如表 23 所示。

表 22 重设实时位姿指令包

Header	Len	Payload						Checksum
		ID	Ctrl		Params			
			rw	isQueued				
0xAA 0xAA	2+9	11	1	0	uint8_t: manual	float: rearArm Angle	float: frontArm Angle	Payload checksum

表 23 重设实时位姿返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	11	1	0	Empty	Payload checksum

说明：manual 为 0 时，自动重设姿态，不用传入 rearArmAngle 及 frontArmAngle；manual 为 1 时，传入 rearArmAngle（大臂角度）和 frontArmAngle（小臂角度）。

2.4.3 获取导轨实时位姿（GetPoseL）

获取导轨实时位姿（GetPoseL），下发的指令包格式如表 24 所示，返回指令包格式如表 25 所示。

表 24 获取导轨实时位姿指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	13	0	0	Empty	Payload checksum

表 25 获取导轨实时位姿返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	13	0	0	float：PoseL	Payload checksum

2.5 报警功能

2.5.1 获取报警状态（GetAlarmsState）

获取报警状态（GetAlarmsState），下发的指令包格式如表 26 所示，返回指令包格式如表 27 所示。

表 26 获取系统报警状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	20	0	0	Empty	Payload checksum

表 27 获取系统报警状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	20	0	0	uint8_t[16]:alarmsState	Payload checksum

说明：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 在高位，LSB 在低位。报警每个位的具体含义参考报警说明文档。

2.5.2 清除系统的所有报警（ClearAllAlarmsState）

清除系统的所有报警（ClearAllAlarmsState），下发的指令包格式如表 28 所示，返回指令包格式如表 29 所示。

表 28 清除系统报警状态指令包

Header	Len	Payload	Checksum

		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	20	1	0	Empty	Payload checksum

表 29 清除系统报警状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	20	1	0	Empty	Payload checksum

2.6 回零功能

该部分为回零位功能，包括设置零位参数、获取零位参数、设置回零位命令。机械臂默认零位为（0°，45°，45°，0°）所对应的坐标值，用户可根据自己需求调用 SetHOMEParams 重新设置零位坐标。执行回零命令后，机械臂会运动到设置的零位位置。

注意：回零过程中机械臂指示灯为蓝色闪烁，运动到零位附近时会进行微调，蜂鸣器响并且指示灯变绿色后表明回零命令执行成功。

2.6.1 设置和获取零位参数（Set/Get HOMEParams）

1. 设置零位参数（SetHOMEParams），下发的指令包格式如表 30 所示，返回指令包格式如表 31 所示；

表 30 设置零位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	30	1	0 or 1	HOMEParams（见 程序 2）	Payload checksum

表 31 设置零位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	30	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取零位参数（GetHOMEParams），下发的指令包格式如表 32 所示，返回指令包格式如表 33 所示。

表 32 获取零位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+0	30	0	0	Empty	Payload checksum

表 33 获取零位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	30	0	0	HOMEParams（见 程序 2）	Payload checksum

程序 2 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;
```

2.6.2 执行回零命令（SetHOMECmd）

执行回零（SetHOMECmd），下发的指令包格式如表 34 所示，返回指令包格式如表 35 所示。

表 34 执行回零指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	31	1	1	HOMECmd（见 程序 3）	Payload checksum

表 35 执行回零返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	31	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 3 HOMECmd 定义

```
typedef struct tagHOMECmd {
    uint32_t reserved;  // 预留未来使用
} HOMECmd;
```

2.6.3 执行和获取自动调平（Set/Get AutoLeveling）

1. 执行自动调平（Set AutoLeveling），下发的指令包格式如表 36 所示，返回指令包格式如表 37 所示；

表 36 设置零位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+5	30	1	1	AutoLeveling（见 程序 4）	Payload checksum

表 37 设置零位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	30	1	1	uint64_t:queuedCmdIndex	Payload checksum

2. 获取自动调平结果（GetAutoLeveling），下发的指令包格式如表 38 所示，返回指令包格式如表 39 所示。

表 38 获取零位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	30	0	0	float：AutoLevelingResult	Payload checksum

表 39 获取零位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	30	0	0	float：AutoLevelingResult	Payload checksum

程序 4 AutoLevelingParams 定义

```
typedef struct tagAutoLevelingParams {
    uint8_t IsAutoleveling;       //调平执行标记
    float Accuracy;               //回零精度
} AutoLevelingParams;
```

2.7 手持示教功能

手持示教功能的指令，用于手持示教相关的命令配置与信息获取，包括使能/禁止手持示教模式、获取手持示教使能信息、获取手持示教是否有新的点增加。

2.7.1 设置和获取手持示教触发模式（Set/Get HHTTrigMode）

1. 设置手持示教时的存点触发模式（SetHHTTrigMode），下发的指令包格式如表 40 所示，返回指令包格式如表 41 所示；

表 40 设置手持示教触发模式指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	40	1	0	HHTTrigMode（见 程序 5）	Payload checksum

表 41 设置手持示教触发模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	40	1	0	Empty	Payload checksum

2. 获取手持示教时的存点触发模式（GetHHTTrigMode），下发的指令包格式如表 42 所示，返回指令包格式如表 43 所示。

表 42 获取手持示教触发模式指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	40	0	0	Empty	Payload checksum

表 43 获取手持示教触发模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	40	1	0	HHTTrigMode（见 程序 5）	Payload checksum

程序 5 HHTTrigMode 的定义

```
typedef enum tagHHTTrigMode {
    TriggeredOnKeyReleased,        //按键释放时更新
    TriggeredOnPeriodicInterval    //定时更新
} HHTTrigMode;
```

2.7.2 设置和获取触发模式输出使能/禁止（Set/Get HHTTrigOutputEnabled）

1. 设置手持示教触发输出使能状态（SetHHTTrigOutputEnabled），下发的指令包格式如表 44 所示，返回指令包格式如表 45 所示；

表 44 设置触发模式输出使能/禁止指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	41	1	0	uint8_t: isEnabled	Payload checksum

表 45 设置触发模式输出使能/禁止返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	41	1	0	Empty	Payload checksum

2. 获取触发输出使能/禁止状态（GetHHTTrigOutputEnabled），下发的指令包格式如表 46 所示，返回指令包格式如表 47 所示。

表 46 获取触发模式输出使能/禁止指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	41	0	0	Empty	Payload checksum

表 47 获取触发模式输出使能/禁止返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	41	0	0	uint8_t: isEnabled	Payload checksum

2.7.3 获取触发输出状态（GetHHTTrigOutput）

获取触发输出状态（GetHHTTrigOutput），下发的指令包格式如表 48 所示，返回指令包格式如表 49 所示。

表 48 获取触发输出状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	42	0	0	Empty	Payload checksum

表 49 获取触发输出状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+1	42	0	0	uint8_t: isTriggered	Payload checksum

2.8 手臂方向

2.8.1 设置/获取手臂方向（Set/Get ArmOrientation）

注意：该命令当前仅适用于 SCARA 机型。

1. 设置手臂方向（SetArmOrientation），下发的指令包格式如表 50 所示，返回指令包格式如表 51 所示；

表 50 设置手臂方向指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	50	1	0 or 1	ArmOrientation（见 程序 6）	Payload checksum

表 51 设置手臂方向返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	50	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取手臂方向（GetArmOrientation），下发的指令包格式如表 52 所示，返回指令包格式如表 53 所示。

表 52 获取手臂方向指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	50	0	0	Empty	Payload checksum

表 53 获取手臂方向返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	50	0	0	ArmOrientation（见 程序 6）	Payload checksum

程序 6 ArmOrientation 定义

```
typedef enum tagArmOrientation {
    LeftyArmOrientation,
```

```
    RightyArmOrientation
} ArmOrientation;
```

2.9 末端执行器

2.9.1 设置和获取末端执行器参数（Set/Get EndEffectorParams）

1. 设置末端执行器参数（SetEndEffectorParams），下发的指令包格式如表 54 所示，返回指令包格式如表 55 所示。

表 54 设置末端执行器参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+12	60	1	0 or 1	EndEffectorParams（见 程序 7）	Payload checksum

表 55 设置末端执行器参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	60	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取末端执行器参数（GetEndEffectorParams），下发的指令包格式如表 56 所示，返回指令包格式如表 57 所示。

表 56 获取末端执行器参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	60	0	0	Empty	Payload checksum

表 57 获取末端执行器参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+12	60	0	0	EndEffectorParams（见 程序 7）	Payload checksum

程序 7 EndEffectorParams 定义

```
typedef struct tagEndEffectorParams {
    float xBias;                //末端 x 轴方向长度
    float yBias;                //末端 y 轴方向长度
```

```
    float zBias;                //末端 z 轴方向长度
} EndEffectorParams;
```

2.9.2 设置和获取激光输出（Set/Get EndEffectorLaser）

1. 设置激光开关（SetEndEffectorLaser），下发的指令包格式如表 58 所示，返回指令包格式如表 59 所示；

表 58 设置激光开关指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	61	1	0 or 1	uint8_t: isCtrlEnabled	uint8_t: isOn	Payload checksum

表 59 设置激光开关返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	61	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

说明：控制是否使能（isCtrlEnabled），激光是否开启（isOn）。

2. 获取激光开关状态（GetEndEffectorLaser），下发的指令包格式如表 60 所示，返回指令包格式如表 61 所示。

表 60 获取激光开关状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	61	0	0	Empty	Payload checksum

表 61 获取激光开关状态指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	61	0	0	uint8_t: isCtrlEnabled	uint8_t: isOn	Payload checksum

说明：控制是否使能（isCtrlEnabled），激光是否开启（isOn）。

2.9.3 设置和获取吸盘输出（Set/Get EndEffectorSuctionCup）

1. 设置吸盘吸放（SetEndEffectorSuctionCup），下发的指令包格式如表 62 所示，返回指令包格式如表 63 所示；

表 62 设置吸盘吸放指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	62	1	0 or 1	uint8_t: isCtrlEnabled	uint8_t: issucked	Payload checksum

表 63 设置吸盘吸放返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	62	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

说明：控制是否使能（isCtrlEnabled），吸盘是否吸住（isSucked）。

2. 获取吸盘吸放状态（GetEndEffectorSuctionCup），下发的指令包格式如表 64 所示，返回指令包格式如表 65 所示。

表 64 获取吸盘吸放状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	62	0	0	Empty	Payload checksum

表 65 获取吸盘吸放状态返回指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	62	0	0	uint8_t: isCtrlEnable	uint8_t: isSuck	Payload checksum

说明：控制是否使能（isCtrlEnabled），吸盘是否吸住（isSucked）。

2.9.4 设置和获取爪子输出（Set/Get EndEffectorGripper）

1. 设置爪子抓住/释放（SetEndEffectorGripper），下发的指令包格式如表 66 所示，返回指令包格式如表 67 所示；

表 66 设置爪子抓住/释放指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	63	1	0 or 1	uint8_t:	uint8_t:	Payload

					isCtrlEnable	isGriped	checksum

表 67 设置爪子抓住/释放返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	63	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

说明：控制是否使能（isCtrlEnabled），爪子是否抓住（isGripped）

2. 获取爪子夹住状态（SetEndEffectorGripper），下发的指令包格式如表 68 所示，返回指令包格式如表 69 所示。

表 68 获取爪子夹住状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	63	0	0	Empty	Payload checksum

表 69 获取爪子夹住状态返回指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	63	0	0	uint8_t: isCtrlEnable	uint8_t: isGriped	Payload checksum

说明：控制是否使能（isCtrlEnabled），爪子是否抓住（isGripped）

2.10 JOG 功能

设置/获取其中包括关节参数、坐标系参数，点动公共参数和执行点动功能。

2.10.1 设置和获取关节点动参数（Set/Get JOGJointParams）

1. 设置关节点动参数（SetJOGJointParams），下发的指令包格式如表 70 所示，返回指令包格式如表 71 所示；

表 70 设置关节点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	70	1	0 or 1	JOGJointParams（见 程序 8）	Payload checksum

表 71 设置关节点动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	70	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

说明：在示教的关节运动中，需要设定关节的速度和加速度参数，这一组指令都是与此相关的指令在关节运动的时候需要预先设定好。此指令中会设置四个关节的速度和加速度。

2. 获取关节点动参数（GetJOGJointParams），下发的指令包格式如表 72 所示，返回指令包格式如表 73 所示。

表 72 获取关节点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	70	0	0	Empty	Payload checksum

表 73 获取关节点动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	70	0	0	JOGJointParams（见 程序 8）	Payload checksum

程序 8 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;
```

2.10.2 设置和获取坐标轴点动参数（Set/Get JOGCoordinateParams）

1. 设置坐标系参数（SetJOGCoordinateParams），下发的指令包格式如表 74 所示，返回指令包格式如表 75 所示；

表 74 设置坐标系参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	71	1	0 or 1	JOGCoordinateParams（见 程序 9）	Payload checksum

表 75 设置坐标系参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	71	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

说明：此指令与单关节运动参数指令类似不同的是，本条指令设置的是坐标系的参数，分别为 X、Y、Z、R 轴的速度和加速度。

2. 获取坐标轴点动参数（GetJOGCoordinateParams），下发的指令包格式如表 76 所示，返回指令包格式如表 77 所示。

表 76 获取坐标轴点动参数指令宝

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	71	0	0	Empty	Payload checksum

表 77 获取坐标轴点动参数返回指令宝

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	71	0	0	JOGCoordinateParams（见 程序 9）	Payload checksum

程序 9 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;
```

2.10.3 设置和获取点动公共参数（Set/Get JOGCommonParams）

1. 设置点动公共参数（SetJOGCommonParams），下发的指令包格式如表 78 所示，返回指令包格式如表 79 所示；

表 78 设置点动公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	72	1	0 or 1	JOGCommonParams（见 程序 10）	Payload checksum

表 79 设置点动公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	72	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取点动公共参数（GetJOGCommonParams），下发的指令包格式如表 80 所示，返回指令包格式如表 81 所示。

表 80 获取点动公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	72	0	0	Empty	Payload checksum

表 81 获取点动公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	72	0	0	JOGCommonParams（见 程序 10）	Payload checksum

程序 10 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;
```

2.10.4 执行点动功能（SetJOGCmd）

执行点动功能，下发的指令包格式如表 82 所示，返回指令包格式如表 83 所示。

表 82 执行点动功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	73	1	1	JOGCmd（见 程序 11）	Payload checksum

表 83 执行点动功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+8	73	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 11 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-
};
```

2.10.5 设置和获取导轨 L 点动参数（Set/Get JOGLParams）

3. 设置导轨 L 点动参数（SetJOGLParams），下发的指令包格式如表 84 所示，返回指令包格式如表 85 所示；

表 84 设置导轨 L 点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	74	1	0 or 1	JOGLParams（见 程序 12）	Payload checksum

表 85 设置导轨 L 点动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	74	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

说明：在示教的关节运动中，需要设定关节的速度和加速度参数，这一组指令都是与此

相关的指令在关节运动的时候需要预先设定好。此指令中会设置四个关节的速度和加速度。

4. 获取关节点动参数（GetJOGJointParams），下发的指令包格式如表 86 所示，返回指令包格式如表 87 所示。

表 86 获取导轨 L 点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	74	0	0	Empty	Payload checksum

表 87 获取导轨 L 点动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	74	0	0	JOGLParams（见程序 12）	Payload checksum

程序 12 JOGJointParams 定义

```
typedef struct tagJOGLParams{
    float velocity;       //导轨关节速度
    float acceleration;   //导轨关节加速度
}JOGLParams;
```

2.11 再现(PTP)功能

再现功能的指令，用于再现相关的运动设置和配置。其中包括关节参数、坐标系参数，比例参数和其他相关的参数。

2.11.1 设置和获取关节点位参数（Set/Get PTPJointParams）

这两条命令用于设置和获取再现速度参数，包括单关节的速度加速度以及直线速度和加速度，此命令设置的速度仅适用于再现运动，对于示教运动是不起作用的。

1. 设置关节点位参数（SetPTPJointParams），用于控制再现运动的速度实现运动的快慢控制。下发的指令包格式如表 88 所示，返回指令包格式如表 89 所示；

表 88 设置关节点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	80	1	0 or 1	PTPJointParams（见 程序 13）	Payload checksum

表 89 设置关节点位参数返回指令包

Header	Len	Payload	Checksum

		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenghr	80	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取关节点位参数（GetPTPJointParams），下发的指令包格式如表 90 所示，返回指令包格式如表 91 所示。

表 90 获取关节点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	80	0	0	Empty	Payload checksum

表 91 获取关节点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	80	0	0	PTPJointParams（见 程序 13）	Payload checksum

程序 13 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;
```

2.11.2 设置和获取坐标轴点位参数（Set/Get PTPCoordinateParams）

1. 设置坐标轴点位参数（SetPTPCoordinateParams），下发的指令包格式如表 92 所示，返回指令包格式如表 93 所示；

表 92 设置坐标轴点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	81	1	0 or 1	PTPCoordinateParams（见 程序 14）	Payload checksum

表 93 设置坐标轴点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	Payload lenght	81	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取坐标轴点位参数（GetPTPCoordinateParams），下发的指令包格式如表 94 所示，返回指令包格式如表 95 所示。

表 94 获取坐标轴点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	81	0	0	Empty	Payload checksum

表 95 获取坐标轴点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	81	0	0	PTPCoordinateParams（见 程序 14）	Payload checksum

程序 14 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;
```

2.11.3 设置和获取门型模式点位参数（Set/Get PTPJumpParams）

1. 设置门型模式点位参数（SetPTPJumpParams），下发的指令包格式如表 96 所示，返回指令包格式如表 97 所示；

表 96 设置门型模式点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	82	1	0 or 1	PTPJumpParams（见 程序 15）	Payload checksum

表 97 设置门型模式点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	Payload lenght	82	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取门型模式点位参数（GetPTPJumpParams），下发的指令包格式如表 98 所示，返回指令包格式如表 99 所示。

表 98 获取门型模式点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	82	0	0	Empty	Payload checksum

表 99 获取门型模式点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	82	0	0	PTPJumpParams（见 程序 15）	Payload checksum

程序 15 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;
```

2.11.4 设置和获取点位公共参数（Set/Get PTPCommonParams）

1. 设置点位公共参数（SetPTPJointParams），下发的指令包格式如表 100 所示，返回指令包格式如表 101 所示；

表 100 设置点位公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	83	1	0 or 1	PTPCommonParams（见 程序 16）	Payload checksum

表 101 设置点位公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	83	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdI	Payload checksum

					ndex	

2. 获取点位公共参数（GetPTPJointParams），下发的指令包格式如表 102 所示，返回指令包格式如表 103 所示。

表 102 获取点位公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	83	0	0	Empty	Payload checksum

表 103 获取点位公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	83	0	0	PTPCommonParams（见 程序 16）	Payload checksum

程序 16 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;
```

2.11.5 执行点位功能（SetPTPCmd）

执行点位功能（PTPCmd），下发的指令包格式如表 104 所示，返回指令包格式如表 105 所示。

表 104 执行点位功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	84	1	1	PTPCmd（见 程序 17）	Payload checksum

表 105 执行点位功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	84	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 17 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
};
```

2.11.6 设置和获取导轨关节点位参数（Set/Get PTPLParams）

这两条命令用于设置和获取导轨的再现速度参数，包括速度加速度以及直线速度和加速度，此命令设置的速度仅适用于再现运动，对于示教运动是不起作用的。

1. 设置导轨关节点位参数（SetPTPLParams），用于控制再现运动的速度实现运动的快慢控制。下发的指令包格式如表 106 所示，返回指令包格式如表 107 所示；

表 106 设置导轨关节点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	85	1	0 or 1	PTPLParams（见程序 18）	Payload checksum

表 107 设置关节点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	Payload Length	85	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取关节点位参数（GetPTPJointParams），下发的指令包格式如表 108 所示，返回指令包格式如表 109 所示。

表 108 获取关节点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	85	0	0	Empty	Payload checksum

表 109 获取关节点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	85	0	0	PTPJointParams（见 程序 18）	Payload checksum

程序 18 PTPJointParams 定义

```
typedef struct tagPTPJointParams {
    float velocity;        //PTP 模式下 4 轴关节速度
    float acceleration;    //PTP 模式下 4 轴关节加速度
} PTPLParams;
```

2.11.7 执行带导轨点位功能（SetPTPWithLCmd）

执行带导轨点位功能（SetPTPJointParams），下发的指令包格式如表 110 所示，返回指令包格式如表 111 所示。

表 110 执行点位功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+21	86	1	1	PTPWithLCmd（见 程序 19）	Payload checksum

表 111 执行点位功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	86	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 19 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;
    float l;
} 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
};
```

2.11.8 设置和获取门型模式点位参数 2（Set/Get PTPJump2Params）

设置和获取导轨门型移动参数 2，把抬升高度分成开始和结束，可以设置不同高度。

1. 设置导轨门型模式点位参数（SetPTPJumpLParams），下发的指令包格式如表 112 所示，返回指令包格式如表 113 所示；

表 112 设置门型模式点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	87	1	0 or 1	PTPJump2Params（见 程序 20）	Payload checksum

表 113 设置门型模式点位参数返回指令包

Header	Len	Payload	Checksum

		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	87	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取门型模式点位参数（GetPTPJumpParams），下发的指令包格式如表 114 所示，返回指令包格式如表 115 所示。

表 114 获取门型模式点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	87	0	0	Empty	Payload checksum

表 115 获取门型模式点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	87	0	0	PTPJump2Params（程序 20）	Payload checksum

程序 20 PTPJumpParams 定义

```
typedef struct tagPTPJump2Params {
    float startjumpHeight;          //门型模式开始抬升高度
    float endJumpHeight;            //门型模式结束抬升高度
    float zLimit;                   //门型模式运动最大抬升高度限制
} PTPJump2Params;
```

2.11.9 执行平行输出点位功能（SetPTPPOCmd）

执行平行输出点位功能（SetPTPPOCmd），主要实现的是在运行 PTP 命令中间某一个时刻时可以对 IO 进行操作，而不必在运动完成之后才能通过 EIO 的命令来控制 IO 输出状态。

下发的指令包格式如表 116 所示，返回指令包格式如表 117 所示。

表 116 执行平行输出点位功能指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+17+4*n	88	1	1	PTPCmd（见 程序 21）	POCmd（n）	Payload checksum

表 117 执行平行输出点位功能返回指令包

Header	Len	Payload	Checksum

		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	88	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 21 PTPCmd 和 POCmd 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、
    float y;          //关节角度、或者坐标/角度增量
    float z;
    float r;
} PTPPOCmd;

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
};
结构体 POCmd 定义如下：
Typedef struct tagPOCmd{
    Uint8_t ratio;   //运动完成百分比
    Uint16_t address //EIO 编号
    Uint8_t   level  //输出状态
}POCmd;
```

2.11.10 执行导轨平行输出点位功能（SetPTPPOWithLCmd）

执行导轨平行输出点位功能（SetPTPPOWithLCmd），主要实现的是在运行 PTP 命令中间某一个时刻时可以对 IO 进行操作，而不必在运动完成之后才能通过 EIO 的命令来控制 IO 输出状态。

下发的指令包格式如表 118 所示，返回指令包格式如表 119 所示。

表 118 执行导轨平行输出点位功能指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+21+4*n	89	1	1	PTPCmd（见 程序 22）	POCmd（n）	Payload checksum

表 119 执行导轨平行输出点位功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	89	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 22 PTPCmd 和 POCmd 定义

```
typedef struct tagPTPCmd {
    uint8_t ptpMode;  //PTP 模式（取值范围 0~8）
    float x;          //x,y,z,r,l 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;
    float l;
} PTPPOCmd;

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
};
结构体 POCmd 如下：
```

```
Typedef struct tagPOCmd{
    Uint8_t ratio;    //运动完成百分比
    Uint16_t address //EIO 编号
    Uint8_t   level  //输出状态
}POCmd;
```

2.12 CP 功能

连续轨迹功能的指令，用于连续轨迹相关的运动设置和配置。其中包括关节参数、坐标系参数、功能设置参数等。连续轨迹功能和上位机 Dobot CP 对应，可以实现写字、画画、激光雕刻等需要连续轨迹的功能。

2.12.1 设置和获取连续轨迹功能参数（Set/Get CPParams）

这两条命令用于设置和获取连续轨迹参数，包括预设加速度、关节速度及加速度，此命令设置的速度仅适用于连续轨迹运动。

1. 设置连续轨迹运动参数（SetCPParams），用于控制连续轨迹运动的速度实现运动的快慢控制，下发的指令包格式如表 120 所示，返回指令包格式如表 121 所示；

表 120 设置连续轨迹运动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+13	90	1	0 or 1	CPParams（见 程序 23）	Payload checksum

表 121 设置连续轨迹运动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	90	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取连续运动轨迹参数（GetCPParams），下发的指令包格式如表 122 所示，返回指令包格式如表 123 所示。

表 122 获取连续运动轨迹参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	90	0	0	Empty	Payload checksum

表 123 获取连续运动轨迹参数返回指令包

Header	Len	Payload	Checksum

		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+13	90	0	0	CPParams（见 程序 23）	Payload checksum

程序 23 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;
```

2.12.2 执行连续轨迹功能（SetCPCmd）

执行连续轨迹功能（SetCPCmd），下发的指令包格式如表 124 所示，返回指令包格式如表 125 所示。

表 124 执行连续轨迹功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	91	1	1	CPCmd（见 程序 24）	Payload checksum

表 125 执行连续轨迹功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	91	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 24 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;
```

2.12.3 执行连续轨迹灰度雕刻功能（SetCPLECmd）

执行连续轨迹灰度雕刻功能（SetCPLECmd），下发的指令包格式如表 126 所示，返回指令包格式如表 127 所示。

表 126 执行连续轨迹功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	92	1	1	CPCmd（见程序 25）	Payload checksum

表 127 执行连续轨迹功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	92	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 25 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;
```

2.13 ARC 功能

2.13.1 设置和获取圆弧插补功能参数（Set/Get ARCParams）

1. 设置圆弧插补功能参数（SetARCParams），下发的指令包格式如表 128 所示，返回指令包格式如表 129 所示；

表 128 设置圆弧插补功能参数指令包

Header	Len	Payload	Checksum

		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	100	1	0 or 1	ARCParams（见 程序 26）	Payload checksum

表 129 设置圆弧插补功能参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	100	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 获取圆弧插补功能参数（GetARCParams），下发的指令包格式如表 130 所示，返回指令包格式如表 131 所示。

表 130 获取圆弧插补功能参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	100	0	0	Empty	Payload checksum

表 131 获取圆弧插补功能参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	100	0	0	ARCParams（见 程序 26）	Payload checksum

程序 26 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
    flaot rAcceleration;        //圆弧运动末端旋转加速度
} ARCParams;
```

2.13.2 执行圆弧插补功能（SetARCCmd）

执行圆弧插补功能（SetARCCmd），下发的指令包格式如表 132 所示，返回指令包格式如表 133 所示。

表 132 执行圆弧插补功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	101	1	1	ARCCmd（见 程序 27）	Payload checksum

表 133 执行圆弧插补功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	101	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 27 ARCCmd 定义

```
typedef struct tagARCCmd {
    struct{
        float x;
        float y;
        float z;
        float r;
    } cirPoint;        //圆弧上任一点坐标

    struct {
        float x;
        float y;
        float z;
        float r;
    } toPoint;         //圆弧结束点坐标
} ARCCmd;
```

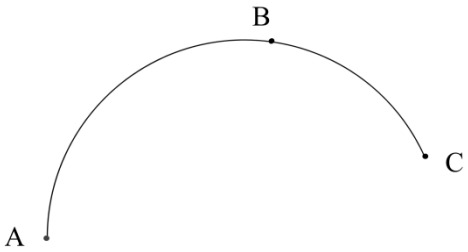
圆弧轨迹说明：

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。

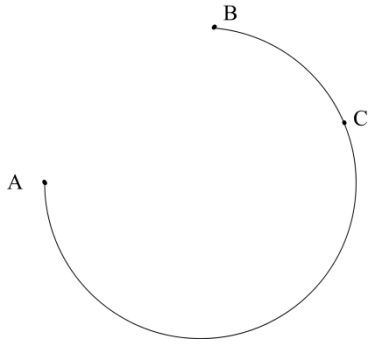
圆弧轨迹如下图示例：

（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

2.14 WAIT 功能

2.14.1 执行时间等待功能（SetWAITCmd）

执行时间等待功能（SetWAITCmd），下发的指令包格式如表 134 所示，返回指令包格式如表 135 所示。

表 134 执行时间等待功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	110	1	1	WAITCmd（见 程序 28）	Payload checksum

表 135 执行时间等待功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	110	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 28 WAITCmd 定义

```
typedef struct tagWAITCmd {
    uint32_t timeout;       //单位 ms
} WAITCmd;
```

2.15 TRIG 功能

2.15.1 执行触发功能（SetTRIGCmd）

执行触发功能（SetTRIGCmd），下发的指令包格式如表 136 所示，返回指令包格式如表 137 所示。

表 136 执行触发功能指令包

Header	Len	Payload			Checksum
		ID	Ctrl	Params	

			rw	isQueued		
0xAA 0xAA	2+4	120	1	1	TRIGCmd（见 程序 29）	Payload checksum

表 137 执行触发功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	120	1	1	uint64_t: queuedCmdIndex	Payload checksum

程序 29 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
    uint8_t condition;       //触发条件 IOCondition/ ADCCondition
    uint16_t threshold;      //触发值 IO 值—0/1  ADC 值—0~4095
} TRIGCmd;
typedef enum tagIOCondition {
    TRIGInputIOEqual,
    TRIGInputIONotEqual
} IOCondition;

typedef enum tagADCCondition {
    TRIGADCLT,     //Lower than
    TRIGADCLE,     //Lower than or Equal
    TRIGADCGE,     //Greater than or Equal
    TRIGADCGT      //Greater than
} ADCCondition;
```

2.16 EIO 功能

EIO（Extended I/O），也即扩展 I/O。在 Dobot 的产品中，其控制器中带有 EIO 并且绝大部分都有复用功能。

EIO 具体编址信息请参考 EIO 说明文档。

2.16.1 设置和读取 I/O 复用（Set/Get IOMultiplexing）

1. 设置 I/O 口复用（SetIOMultiplexing），下发的指令包格式如表 138 所示，返回指令包格式如表 139 所示；

表 138 设置 I/O 口复用指令包

Header	Len	Payload			Checksum
		ID	Ctrl	Params	

			rw	isQueued		
0xAA 0xAA	2+2	130	1	0 or 1	IOMultiplexing（见 程序 30）	Payload checksum

表 139 设置 I/O 口复用返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload length	130	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 读取复用 I/O（SetIOMultiplexing），下发的指令包格式如表 140 所示，返回指令包格式如表 141 所示。

表 140 读取复用 I/O 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	130	0	0	uint8_t address	Payload checksum

表 141 读取复用 I/O 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	130	0	0	IOMultiplexing（见 程序 30）	Payload checksum

程序 30 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如程序 31：

程序 31 IOFunction 定义

```
typedef enum tagIOFunction {
    IOFunctionDummy,     //不配置功能
    IOFunctionDO,        //IO 输出
    IOFunctionPWM,       //PWM 输出
    IOFunctionDI,        //IO 输入
```

```
    IOFunctionADC,        //AD 输入
    IOFunctionDIPU,       //上拉输入
    IOFunctionDIPD        //下拉输入
} IOFunction;
```

2.16.2 设置和读取 I/O 输出电平（Set/Get IODO）

1. 设置 I/O 口输出电平（SetIODO），下发的指令包格式如表 142 所示，返回指令包格式如表 143 所示；

表 142 设置 I/O 口输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	131	1	0 or 1	IODO（见 程序 32）	Payload checksum

表 143 设置 I/O 口输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload length	131	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 读取 I/O 输出电平（GetIODO），下发的指令包格式如表 144 所示，返回指令包格式如表 145 所示。

表 144 读取 I/O 口输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	131	0	0	uint8_t address	Payload checksum

表 145 读取 I/O 口输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	131	0	0	IODO（见 程序 32）	Payload checksum

程序 32 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;
```

2.16.3 设置和读取 PWM 输出（Set/Get IOPWM）

1. 设置 I/O PWM 输出（SetIOPWM），下发的指令包格式如表 146 所示，返回指令包格式如表 147 所示；

表 146 设置 I/O 口 PWM 输出指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+9	132	1	0 or 1	IOPWM（见 程序 33）	Payload checksum

表 147 设置 I/O 口 PWM 输出返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	132	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

2. 读取 I/O PWM（GetIOPWM），下发的指令包格式如表 148 所示，返回指令包格式如表 149 所示。

表 148 读取 I/O 口 PWM 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	132	0	0	uint8_t address	Payload checksum

表 149 读取 I/O 口 PWM 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+9	132	0	0	IOPWM（见 程序 33）	Payload checksum

程序 33 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;
```

2.16.4 读取 I/O 输入电平（GetIODI）

读取 I/O 输入电平（GetIODI），下发的指令包格式如表 150 所示，返回指令包格式如表 151 所示。

表 150 读取 I/O 输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	133	0	0	uint8_t address	Payload checksum

表 151 读取 I/O 输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	133	0	0	IODI（见 程序 34）	Payload checksum

程序 34 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;
```

2.16.5 读取 I/O 模数转换值（GetIOADC）

读取 I/O 模数转换值（GetIOADC），下发的指令包格式如表 152 所示，返回指令包格式如表 153 所示。

表 152 读取 I/O 模数转换值指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	134	0	0	uint8_t address	Payload checksum

表 153 读取 I/O 模数转换值返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+3	134	0	0	IOADC（见 程序 35）	Payload checksum

程序 35 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;
```

2.16.6 设置扩展电机接口（SetEMotor）

设置扩展电机接口（SetEMotor），下发的指令包格式如表 142 所示，返回指令包格式如表 143 所示；

表 154 设置 I/O 口输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	135	1	0 or 1	● EMotor（见程序 36）	Payload checksum

表 155 设置 I/O 口输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload length	135	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 36 EMotor 定义

```
typedef struct tagEMotor{
    uint8_t index;        //取值范围 0/1  0-Stepper1   1-Stepper2
    uint8_t insEnabled;   //电机控制使能
    float speed;          //电机控制速度(脉冲个数每秒)
}EMotor;
```

2.16.7 设置和读取颜色传感器（Set/Get ColorSensor）

1. 设置颜色传感器（SetColorSensor），下发的指令包格式如表 156 所示，返回指令包格式如表 157 所示；

表 156 设置颜色传感器指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	137	1	1	uint8_t : isEnable；uint8_t Port(见程序 33)	Payload checksum

表 157 设置颜色传感器返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	137	1	1	int64_t:queuedCmdIndex	Payload checksum

2. 读取颜色传感器（GetColorSensor），下发的指令包格式如表 158 所示，返回指令包格式如表 159 所示。

表 158 读取颜色传感器指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	137	0	0	Empty	Payload checksum

表 159 读取颜色传感器返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+3	137	0	0	Color（见 程序 33）	Payload checksum

程序 37 Port 及 Color 定义

```
typedef struct tagPort {
    uint8_t PORT_GP1;
    uint8_t PORT_GP2;
    uint8_t PORT_GP4;
    uint8_t PORT_GP5;
    } Port;
typedef struct tagColor {
    uint8_t r;
    uint8_t g;
    uint8_t b;
    } Clolor;
```

2.16.8 设置和读取红外开关（Set/Get IRSwitch）

3. 设置红外开关（Set IRSwitch），下发的指令包格式如表 160 所示，返回指令包格式如表 161 所示；

表 160 设置红外开关指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	138	1	1	uint8_t : isEnable；uint8_t IRPort (见程序 38)	Payload checksum

表 161 设置红外开关返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	138	1	1	int64_t:queuedCmdIndex	Payload checksum

4. 读取红外开关（Get IRSwitch），下发的指令包格式如表 162 所示，返回指令包格式如表 163 所示。

表 162 读取红外开关指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	137	0	0	Empty	Payload checksum

表 163 读取红外开关返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+3	137	0	0	Uint8_t state	Payload checksum

程序 38 IRPort 定义

```
typedef struct tagIRPort {
    uint8_t PORT_GP1;
    uint8_t PORT_GP2;
    } IRPort;
```

2.17 校准(CAL)功能

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），得到此静态误差，并通过此 API 写入到设备中。

2.17.1 设置和读取角度传感器静态误差（Set/Get AngleSensorStaticError）

1. 设置角度传感器静态误差（SetAngleSensorStaticError），下发的指令包格式如所示，返回指令包格式如所示表 165；

表 164 设置角度传感器静态误差指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+8	140	1	0	float: rearArmAngleError	float: frontArmAngleError	Payload checksum

表 165 设置角度传感器静态误差返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	140	1	0	Empty	Payload checksum

2. 读取角度传感器静态误差（GetAngleSensorStaticError），下发的指令包格式如表 166 所示，返回指令包格式如表 167 所示。

表 166 获取角度传感器静态误差指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	140	0	0	Empty	Payload checksum

表 167 获取角度传感器静态误差返回指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+8	140	0	0	float: rearArmAngleError	float: frontArmAngleError	Payload checksum

2.18 WIFI 功能

2.18.1 设置和获取 WIFI 配置模式（Set/Get WIFIConfigMode）

1. 设置 WIFI 配置模式（SetWIFIConfigMode），下发的指令包格式如表 168 所示，返回指令包格式如表 169 所示；

表 168 设置 WIFI 配置模式指令包

Header	Len	Payload			Checksum
		ID	Ctrl	Params	

			rw	isQueued		
0xAA 0xAA	2+1	150	1	0	uint8_t: enable	Payload checksum

表 169 设置 WIFI 配置模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	150	1	0	Empty	Payload checksum

2. 获取当前 WIFI 是否配置模式（GetWIFIConfigMode），下发的指令包格式如表 170 所示，返回指令包格式如表 171 所示。

表 170 获取当前 WIFI 是否配置模式指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	150	0	0	Empty	Payload checksum

表 171 获取当前 WIFI 是否配置模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	150	0	0	uint8_t: enable	Payload checksum

2.18.2 设置和获取 SSID（Set/Get WIFISSID）

1. 设置 SSID（SetWIFISSID ），下发的指令包格式如表 172 所示，返回指令包格式如表 173 所示；

表 172 设置 SSID 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	151	1	0	char* ssid	Payload checksum

表 173 设置 SSID 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	151	1	0	Empty	Payload checksum

2. 获取当前设置 SSID（GetWIFISSID），下发的指令包格式如表 174 所示，返回指令包格式如表 175 所示。

表 174 获取当前设置 SSID 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	151	0	0	Empty	Payload checksum

表 175 获取当前设置 SSID 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	151	0	0	char* ssid	Payload checksum

2.18.3 设置和获取网络密码（Set/Get WIFIPassword）

1. 设置网络密码（SetWIFIPassword ），下发的指令包格式如表 176 所示，返回指令包格式如表 177 所示；

表 176 设置网络密码指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	152	1	0	char* password	Payload checksum

表 177 设置网络密码返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	152	1	0	Empty	Payload checksum

2. 获取当前设置网络密码（GetWIFIPassword），下发的指令包格式如表 178 所示，返回指令包格式如表 179 所示。

表 178 获取当前设置网络密码指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	152	0	0	Empty	Payload checksum

表 179 获取当前设置网络密码返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	Payload lenght	152	0	0	char* password	Payload checksum

2.18.4 设置和获取 IP 地址（Set/Get WIFIIPAddress）

1. 设置 IP（SetWIFIIPAddress），下发的指令包格式如表 180 所示，返回指令包格式如表 181 所示；

表 180 设置 IP 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+5	153	1	0	WIFIIPAdress（见 程序 39）	Payload checksum

表 181 设置 IP 返回 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	153	1	0	Empty	Payload checksum

2. 获取当前设置 IP 地址（GetWIFIIPAddress），下发的指令包格式如表 182 所示，返回指令包格式如表 183 所示。

表 182 获取当前设置 IP 地址指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	153	0	0	Empty	Payload checksum

表 183 获取当前设置 IP 地址指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+5	153	0	0	WIFIIPAdress（见 程序 39）	Payload checksum

程序 39 WIFIIPAdress 定义

```
typedef struct tagWIFIIPAdress {
    uint8_t dhcp;
    uint8_t addr[4];
```

```
} WIFIIPAdress;
```

2.18.5 设置和获取子网掩码（Set/Get WIFINetmask）

1. 设置子网掩码（SetWIFINetmask），下发的指令包格式如表 184 所示，返回指令包格式如表 185 所示；

表 184 设置子网掩码指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	154	1	0	WIFINetmask（见 程序 40）	Payload checksum

表 185 设置子网掩码返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	154	1	0	Empty	Payload checksum

2. 获取当前设置子网掩码（GetWIFINetmask），下发的指令包格式如表 186 所示，返回指令包格式如表 187 所示。

表 186 获取当前设置子网掩码指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	154	0	0	Empty	Payload checksum

表 187 获取当前设置子网掩码返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	154	0	0	WIFINetmask（见 程序 40）	Payload checksum

程序 40 WIFINetmask 定义

```
typedef struct tagWIFINetmask {
    uint8_t addr[4];
} WIFINetmask;
```

2.18.6 设置和获取网关（Set/Get WIFIGateway）

1. 设置网关（SetWIFIGateway），下发的指令包格式如表 188 所示，返回指令包格式如表 189 所示；

表 188 设置网关指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	155	1	0	WIFIGateway（见 程序 41）	Payload checksum

表 189 设置网关返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	155	1	0	Empty	Payload checksum

2. 获取当前设置网关（GetWIFIGateway），下发的指令包格式如表 190 所示，返回指令包格式如表 191 所示。

表 190 获取当前设置网关指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	155	0	0	Empty	Payload checksum

表 191 获取当前设置网关返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	155	0	0	WIFIGateway（见 程序 41）	Payload checksum

程序 41 WIFIGateway 定义

```
typedef struct tagWIFIGateway {
    uint8_t addr[4];
} WIFIGateway;
```

2.18.7 设置和获取 DNS（Set/Get WIFIDNS）

1. 设置 DNS（SetWIFIDNS），下发的指令包格式如表 192 所示，返回指令包格式如表 193 所示；

表 192 设置 DNS 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+4	156	1	0	WIFIDNS（见 程序 42）	Payload checksum

表 193 设置 DNS 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	156	1	0	Empty	Payload checksum

2. 获取当前设置 DNS（GetWIFIDNS），下发的指令包格式如表 194 所示，返回指令包格式如表 195 所示。

表 194 获取当前设置 DNS 指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	156	0	0	Empty	Payload checksum

表 195 获取当前设置 DNS 返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	156	0	0	WIFIDNS（见 程序 42）	Payload checksum

程序 42 WIFIDNS 定义

```
typedef struct tagWIFIDNS {
    uint8_t addr[4];
} WIFIDNS;
```

2.18.8 获取 WIFI 连接状态（GetWIFIConnectStatus）

获取当前 WI-FI 模块的连接状态（GetWIFIConnectStatus），下发的指令包格式如表 196 所示，返回指令包格式如表 197 所示。

表 196 获取当前 WI-FI 模块的连接状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	157	0	0	Empty	Payload checksum

表 197 获取当前 WI-FI 模块的连接状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	157	0	0	uint8_t: isConnected	Payload checksum

2.19 丢步功能功能

2.19.1 设置丢步检测功能（Set/Get LostStep）

3. 设置丢步检测偏离值（Set LostStepValue），下发的指令包格式如表 198 所示，返回指令包格式如表 199 所示；

表 198 设置丢步检测幅度指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	170	1	0	Float: value	Payload checksum

表 199 设置 WIFI 配置模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	170	1	0	Empty	Payload checksum

2.19.2 设置执行丢步检测功能（Set LostStep）

1. 获取当前 WIFI 是否配置模式（GetWIFIConfigMode），下发的指令包格式如表 200 所示，返回指令包格式如表 201 所示。

表 200 设置执行丢步检测功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	171	1	1	Empty	Payload checksum

表 201 设置执行丢步检测功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	171	1	1	uint64_t:queuedCmdIndex	Payload checksum

2.20 队列执行控制命令

队列执行控制命令主要是用于设置队列命令执行的相关参数，包括命令执行模式（在线/离线）、队列命令缓冲器当前状态、队列命令执行状态（TRUE / FALSE）、队列命令执行控制（START / PAUSE / STOP）。

2.20.1 启动指令队列运行（SetQueuedCmdStartExec）

启动指令队列运行（SetQueuedCmdStartExec），下发的指令包格式如表 202 所示，返回指令包格式如表 203 所示。

表 202 启动指令队列运行指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	240	1	0	Empty	Payload checksum

表 203 启动指令队列运行返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	240	1	0	Empty	Payload checksum

2.20.2 停止指令队列运行（SetQueuedCmdStopExec）

停止指令队列运行（SetQueuedCmdStopExec），下发的指令包格式如表 204 所示，返回指令包格式如表 205 所示。

表 204 停止指令队列运行指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	241	1	0	Empty	Payload checksum

表 205 停止指令队列运行返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	241	1	0	Empty	Payload checksum

2.20.3 强制停止指令队列运行（SetQueuedCmdForceStopExec）

强制停止指令队列运行（SetQueuedCmdForceStopExec），下发的指令包格式如表 206 所示，返回指令包格式如表 207 所示。

表 206 强制停止指令队列运行指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	242	1	0	Empty	Payload checksum

表 207 强制停止指令队列运行返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	242	1	0	Empty	Payload checksum

2.20.4 启动指令队列下载（SetQueuedCmdStartDownload）

启动指令队列下载（SetQueuedCmdStartDownload），下发的指令包格式如表 208 所示，返回指令包格式如表 209 所示。

表 208 启动指令队列下载指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+8	243	1	0	uint32_t: totalLoop	uint32: linePerLoop	Payload checksum

表 209 启动指令队列下载返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	243	1	0	Empty	Payload checksum

说明：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处罚执行，也即脱机运行功能。

2.20.5 完成指令队列下载（SetQueuedCmdStopDownload）

完成指令队列下载（SetQueuedCmdStopDownload），下发的指令包格式如表 210 所示，返回指令包格式如表 211 所示。

表 210 完成指令队列下载指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	244	1	0	Empty	Payload

						checksum

表 211 完成指令队列下载返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	244	1	0	Empty	Payload checksum

2.20.6 清空指令队列（SetQueuedCmdClear）

清空指令队列（SetQueuedCmdClear），下发的指令包格式如表 212 所示，返回指令包格式如表 213 所示。

表 212 清空指令队列指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	245	1	0	Empty	Payload checksum

表 213 清空指令队列返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	245	1	0	Empty	Payload checksum

2.20.7 获取指令队列索引（GetQueuedCmdCurrentIndex）

获取指令队列索引（GetQueuedCmdCurrentIndex），下发的指令包格式如表 214 所示，返回指令包格式如表 215 所示。

表 214 获取指令队列索引指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	246	0	0	Empty	Payload checksum

表 215 获取指令队列索引返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	246	0	0	uint64_t: queuedCmdCurrentIndex	Payload checksum

说明：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式，该计数器自动加一。通过该内部索引，可以查询控制器已经执行了多少条队列指令，以及当前执行到哪条指令（指示运行进度时）。

2.20.8 获取指令队列剩余空间（GetQueuedCmdLeftSpace）

获取指令队列剩余空间（GetQueuedCmdLeftSpace），下发的指令包格式如表 216 所示，返回指令包格式如表 217 所示。

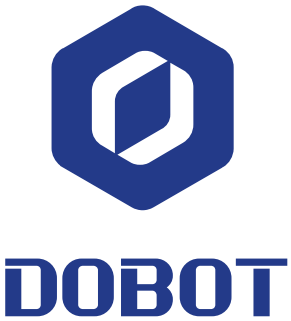
表 216 获取指令队列剩余空间指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	247	0	0	Empty	Payload checksum

表 217 获取指令队列剩余空间指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	247	0	0	uint32_t:leftSpace	Payload checksum

说明：在 Dobot 控制器指令队列机制中，有一个指令队列。在发送队列指令时，应先查询指令队列的剩余空间，若非零，才能向 Dobot 控制器发送队列指令。



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