电调产品 CAN 接口使用手册

V2.2.0

探索世界的动力

POWER MAKES YOUR EXPLORATION

目　录

1 总述

1.1 简介

本公司开发的电调产品，CAN 底层通信协议遵循 CAN 2.0B 协议，B 版本协议为 29 位标识符（扩展帧）。兼容开源 DRONECAN、及我司自定义的 CUBECAN。对接开发中只能选择其中一种协议使用。

CUBECAN 协议是我司开发的一种形式简单，使用方便的 CAN 协议，直接使用 CAN 帧进行交互。与 DRONECAN 协议相比，不需要组包（多条 CAN 帧组成一条 DRONECAN 包），也不需要拆包（一条 DRONECAN 包拆分成多条 CAN 帧）。

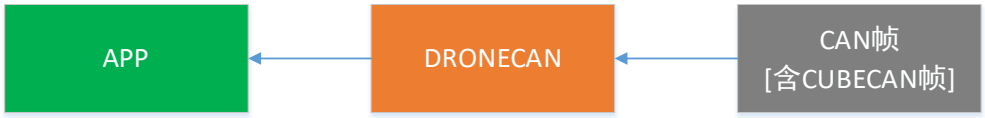


图 1　程序设计框图

1.2 CAN 总线波特率

波特率出厂默认是 1MHZ，可通过 CloudLink 进行修改

1.3 Node ID 设置

参考《Master 系列无人机电调快速手册》、《CloudLink 使用说明书》。

1.4 两种协议比较

1.4.1 DRONECAN: UAVCAN V0.9 版本的继承者。

ID field

Message frame

Field name	Priority					Message type ID												Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	1...127						
CAN ID bytes	3					2								1								0							

Anonymous message frame

Field name	Priority					Discriminator								Lower bits of message type ID								Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	0						
CAN ID bytes	3					2								1								0							

Service frame

Field name	Priority					Service type ID								Request not response	Destination node ID							Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values															1...127							1	1...127						
CAN ID bytes	3					2								1								0							

1.4.2 CUBECAN: 该协议是我司自定义的协议，基于普通的 CAN 扩展帧。对用户来说，使用简单。

2 DRONECAN 协议概述（UAVCAN V0.9）

2.1 通信模型

两种通信模型：

- 服务：服务器/客户端模式（RPC），点对点
- 消息：发布/订阅模式，一对多，广播

本产品使用的是消息模型。

2.2 CAN ID 定义

ID field

Message frame

Field name	Priority					Message type ID												Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	1...127						
CAN ID bytes	3					2								1								0							

Anonymous message frame

Field name	Priority					Discriminator								Lower bits of message type ID								Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	0						
CAN ID bytes	3					2								1								0							

Service frame

Field name	Priority					Service type ID								Request not response	Destination node ID							Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values															1...127							1	1...127						
CAN ID bytes	3					2								1								0							

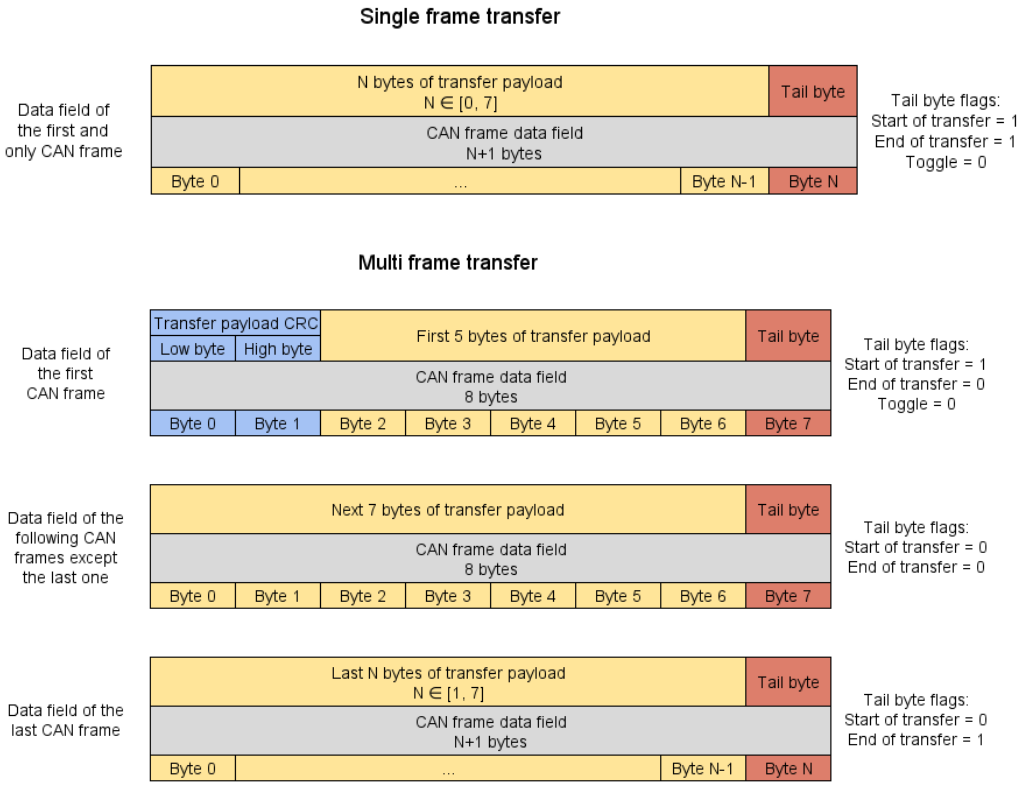
BIT24~BIT28: 优先级。

BIT8~BIT23: 消息 ID。

BIT7: 0。

BIT0~BIT6: NODE ID。

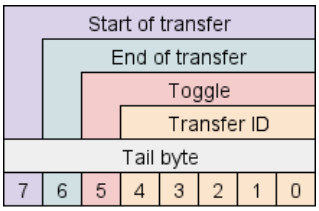
2.3 单帧/多帧传输



DRONECAN 协议(UAVCAN V0.9)的 CRC 字节在包消息的头部。

2.4 尾部字节定义

尾部 1 字节 8 个位拆分成了几个字段，用于携带协议信息。



Field	Bits	Description
Start of transfer	BIT7(1)	标识该帧是此次传输的第一帧，对于单帧传输，该字段始终为 1； 对于多帧传输，第一帧该字段为 1，其余为 0。
End of transfer	BIT6(1)	标识该帧是此次传输的最后一帧，对于单帧传输，该字段始终为 1； 对于多帧传输，最后一帧该字段为 1，其余为 0.
Toggle bit	BIT5(1)	对于单帧传输，该字段始终为 0； 对于多帧传输，该字段将在各帧之间切换，第一帧该位为 0，下一帧为 1，如此翻转下去，用途是检测和避免重复 CAN 帧的错误。
Transfer ID	BIT0~BIT4(5)	该字段包含当前传输的传输 ID 值（适用于所有类型的传输）。 该值为 5 位宽，允许值范围为 0 至 31（含）。

2.5 参考链接

官方文档：

https://legacy.uavcan.org/Specification/4._CAN_bus_transport_layer/

开源代码：

https://github.com/dronecan/libcanard

3 DRONECAN 协议交互

该章节主要描述了电调的交互消息，适用 DRONECAN 协议(UAVCAN V0.9)。使用时的消息 ID 描述： message-type-ID 即代表消息 ID，占用位段是 BIT8-BIT23。

ID	DRONECAN (UAVCAN V0.9)
消息 ID	BIT8-BIT23
NODE ID	BIT0-BIT6
优先级	BIT24-BIT28

3.1 使用约定

(1) NODE_ID 约定：

- 协议栈的 NODE_ID 取值范围是 1~127。
- 电调 NODE_ID 取值范围是 1~63。
- 飞控 NODE_ID 取值范围是 1~127。
- CAN 总线上，NODE_ID 不能相同，NODE_ID 具有唯一性。（飞控与电调之间、电调与电调之间、电调与挂载之间等等）

(2) 消息 ID 约定：

指令描述	Message-type-ID	传输方向
状态一上报	1150	电调 → 飞控
状态二上报	1151	电调 → 飞控
状态三上报	1152	电调 → 飞控
状态四上报	1153	电调 → 飞控
状态五上报	1154	电调 → 飞控
通用指令	999	电调 → 飞控
	1000	飞控 → 电调
指令下发	1160	飞控 → 电调

以上消息 ID 已经被占用，严禁其他设备使用。（例如挂载设备等）

3.2 通用指令

通用指令的消息 ID 是 1000

通用指令的消息内部又包含了一层协议，这层协议是我司自定义的一套协议，为了支持多种数据的交互，由于交互频率比较低，故使用同一个消息 ID。

通用指令携带的消息 = 内部协议头 + 内部消息结构体

对应消息结构体：

```
#define CAN_APP_PROTO_MAGIC 0xabcd
```

```
typedef struct
{
    uint16_t magic;
    uint16_t msg_id;
    uint16_t len;
} CAN_APP_PROTO_HEAD;
```

内部协议头由 magic、内部协议消息 ID 和消息长度构成。

3.2.1 使能上报

电调程序上电默认不上报状态，需要使能上报，电调才会上报状态。

内部协议消息 ID：ENA_ESC_STAT_REP_MSG_ID = 4670

对应消息结构体：

```
typedef struct
{
    uint32_t enable;    // 0--OFF, 1--ON
} ENA_ESC_STAT_REP_T;
```

enable = 0: 关闭电调状态上报；

enable = 1: 打开电调状态上报。

3.2.2 航灯控制

电调航灯是否通过 CAN 接口控制，需通过客户端进行设置,具体参考《CloudLink 使用说明书》。航灯状态默认为 CAN 控制。

内部协议消息 ID：ARM_LED_CTRL_MSG_ID = 4669

对应消息结构体：

```
struct  CAN_LED_BITS {          // bits  description
    uint16_t state:10;          // 9:0   refer to ARM_LED_STATE
    uint16_t node_id:6;         // 15:10  node_id, range 0~63
};

union CAN_LED_UN {
    uint16_t                    all;
    struct CAN_LED_BITS  bit;
};

typedef struct
{
    union CAN_LED_UN esc[8];    //ESC numbers, e.g. 8
} CAN_ESC_LED;
```

航灯枚举定义：

```
typedef enum
{
    CUBECAN_LED_STATE_OFF = 0,     //ABC OFF

    CUBECAN_LED_STATE_A_ON,         //A ON
    CUBECAN_LED_STATE_B_ON,         //B ON
    CUBECAN_LED_STATE_C_ON,         //C ON

    CUBECAN_LED_STATE_AB_ON,        //AB ON
    CUBECAN_LED_STATE_AC_ON,        //AC ON
    CUBECAN_LED_STATE_BC_ON,        //BC ON

    CUBECAN_LED_STATE_A_BLINK,     //A Flash
    CUBECAN_LED_STATE_B_BLINK,     //B Flash
    CUBECAN_LED_STATE_C_BLINK,     //C Flash

    CUBECAN_LED_STATE_AB_BLINK,    //AB Flash alternately
    CUBECAN_LED_STATE_AC_BLINK,    //AC Flash alternately
    CUBECAN_LED_STATE_BC_BLINK,    //BC Flash alternately

    CUBECAN_LED_STATE_ABC_BLINK, //ABC Flash alternately
} CUBECAN_LED_STATE;
```

ABC 代表 3 种颜色的灯，默认为 A – Red，B – Green，C -White。

具体请查阅硬件型号说明书，与实际硬件型号绑定。

3.3 状态上报

电调状态上报，有 4 种状态，分别是状态一/状态二/状态三/状态四/状态五，消息 ID 分别是 1150/1151/1152/1153/1154，采用单帧传输，can 帧负载字节长度是 8，有效字节长度是 7，最后 1 个字节是 Dronecan 单帧协议的尾部字节。上报频率为 10HZ。

指令描述	Message-type-ID	结构体
状态一上报	1150	S_CAN_MSG_ESC_STAT1
状态二上报	1151	S_CAN_MSG_ESC_STAT2
状态三上报	1152	S_CAN_MSG_ESC_STAT3
状态四上报	1153	S_CAN_MSG_ESC_STAT4
状态五上报	1154	S_CAN_MSG_ESC_STAT5

定义如下：

```
//电调状态上报(单帧传输)
#define S_ESC_STAT1_PORT_ID ((uint32_t)1150)//电调 -> 外部, S_CAN_MSG_ESC_STAT1
```

```
#define S_ESC_STAT2_PORT_ID ((uint32_t)1151)//电调 -> 外部, S_CAN_MSG_ESC_STAT2
#define S_ESC_STAT3_PORT_ID ((uint32_t)1152)//电调 -> 外部, S_CAN_MSG_ESC_STAT3
#define S_ESC_STAT4_PORT_ID ((uint32_t)1153)//电调 -> 外部, S_CAN_MSG_ESC_STAT4
#define S_ESC_STAT5_PORT_ID ((uint32_t)1154)//电调 -> 外部, S_CAN_MSG_ESC_STAT5

//order: bit0 -> bit15
struct    CAN_MSG_ESC_MODE_BITS       // bits    description
{
      uint16_t esc_mode: 8;//low 8 bit
      uint16_t pwm_thr_online: 1;//0->lost, 1->online
      uint16_t can_thr_online: 1;//0->lost, 1->online
      uint16_t thr_pri: 1;//0->pwm thr first, 1->can thr first
      uint16_t resv: 5;
};

union CAN_MSG_ESC_MODE_UN
{
      uint16_t                        all;
      struct CAN_MSG_ESC_MODE_BITS    bit;
};

typedef struct
{
      union CAN_MSG_ESC_MODE_UN esc_mode; //控制模式
      int16_t esc_cmd;   //油门指令
      int16_t spd_rpm;   //电机转速
      uint16_t resv: 8;
      uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT1;

typedef struct
{
      int16_t     vdc_100mv;           // 母线电压
      int16_t     irms_100ma;          // 相电流有效值
      int16_t     idq0_100ma;       // d-q 轴电流
      uint16_t resv: 8;
      uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT2;

typedef struct
{
```

```
    int16_t alg_err;              //算法错误字
    int16_t alg_warn;             //算法警告字
    int16_t vdq0_duty;        // d-q 轴指令
    uint16_t resv: 8;
    uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT3;

typedef struct
{
    int16_t mos_temp; //mos 管温度
    int16_t     idq1_100ma;       // d-q 轴电流
    int16_t vdq1_duty;        // d-q 轴指令
    uint16_t resv: 8;
    uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT4;

typedef struct
{
   int16_t idc; //母线估测电流
   int16_t cap_temp; //电容温度，单位 0.1 摄氏度
   int16_t motor_temp;//电机温度，单位 0.1 摄氏度
   uint16_t resv: 8;
   uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT5;
```

3.4 油门控制

电调具有上电 0 油门自检逻辑。先连续发送 3 秒 0 油门信号，让电调通过自检。DRONECAN 协议指令下发（飞控→电调），采用单帧传输。CAN 帧负载字节长度是 8，有效字节长度是 7，最后 1 个字节是 DRONECAN 单帧协议的尾部字节。

CAN Frame	Command	Tail Byte
29BITs message frame	Payload [0-6]	Payload [7]

单帧传输，每次可发送最多 4 组电调的油门指令。有效字节长度是 7，即有效位数是 7*8=56Bit，包含了 4 组电调的油门指令，那么每组电调占用位数是 56/4=14Bit。

56Bit 的位域划分如下：

Field Name	e4	id4			cmd4					e3	id3			cmd3					e2	id2			
BITs	55	54	53	52	51	···	48	47	···	42	41	40	39	38	37	···	32	31	···	28	27	26	25	24
Bytes	payload[6]							payload[5]				payload[4]						payload[3]						

Field Name	cmd2					e1	id1			cmd1				
BITs	23	···	16	15	14	13	12	11	10	9	8	7	···	0
Bytes	payload[2]					payload[1]						payload[0]		

第 4 通道	BIT 55	第 4 通道生效开关：0 不生效；1 生效。
	BIT 54- BIT 52	第 4 通道电调 NODE ID 的个位数：取值范围 0-7。电调端，根据该 3bit 的信息，与自身 NODE ID 的个位数进行匹配，如果匹配上，则响应该指令。
	BIT 51- BIT 42	第 4 通道电调油门指令：取值范围 0-1000
第 3 通道	BIT 41	第 3 通道生效开关：0 不生效；1 生效。
	BIT 40- BIT 38	第 3 通道电调 NODE ID 的个位数：取值范围 0-7。电调端，根据该 3bit 的信息，与自身 NODE ID 的个位数进行匹配，如果匹配上，则响应该指令。
	BIT 37- BIT 28	第 3 通道电调油门指令：取值范围 0-1000
第 2 通道	BIT 27	第 2 通道生效开关：0 不生效；1 生效。
	BIT 26- BIT 24	第 2 通道电调 NODE ID 的个位数：取值范围 0-7。电调端，根据该 3bit 的信息，与自身 NODE ID 的个位数进行匹配，如果匹配上，则响应该指令。
	BIT 23- BIT 14	第 2 通道电调油门指令：取值范围 0-1000
第 1 通道	BIT 13	第 1 通道生效开关：0 不生效；1 生效。
	BIT 12- BIT 10	第 1 通道电调 NODE ID 的个位数：取值范围 0-7。电调端，根据该 3bit 的信息，与自身 NODE ID 的个位数进行匹配，如果匹配上，则响应该指令。
	BIT 9- BIT 0	第 1 通道电调油门指令：取值范围 0-1000

需注意，电调 CAN NODE ID 必须选择以下范围中的某一组，不能跨范围设置。

电调 CAN ID 取值范围					
CAN NODE ID	10~17	20~27	30~37	40~47	50~57
THROTTLE CHANNEL ID	0~7	0~7	0~7	0~7	0~7
MOTOR INDEX	1~8	1~8	1~8	1~8	1~8

定义结构体

```
//order: bit0 -> bit63
typedef struct
{
    uint64_t chan1_thr_cmd: 10;
    uint64_t chan1_node_id: 3;
    uint64_t chan1_enable: 1;

    uint64_t chan2_thr_cmd: 10;
    uint64_t chan2_node_id: 3;
    uint64_t chan2_enable: 1;
```

```
    uint64_t chan3_thr_cmd: 10;
    uint64_t chan3_node_id: 3;
    uint64_t chan3_enable: 1;

    uint64_t chan4_thr_cmd: 10;
    uint64_t chan4_node_id: 3;
    uint64_t chan4_enable: 1;

    uint64_t tail_byte: 8;
} S_CAN_ESC_CMD;
```

以四轴为例，假设电调 CAN NODE ID 被设置为 20/21/22/23。我们需要向四个通道发送数值为 1000(0x03E8)的数字油门，第一个通道是 20（Chan1_node_id = 0），第二个通道是 21（Chan2_node_id = 1），第三个通道是 22（Chan3_node_id = 2），第四个通道是 23（Chan4_node_id = 3）。代码如下：

```
S_CAN_ESC_CMD cmd;

    memset(&cmd, 0, sizeof(cmd));

    cmd.chan1_enable = 1;//使生效
    cmd.chan1_node_id = 0;//匹配电调 0,10,20,30,40,50
    cmd.chan1_thr_cmd = 1000;//油门指令

    cmd.chan2_enable = 1; //使生效
    cmd.chan2_node_id = 1; //匹配电调 1,11,21,31,41,51
    cmd.chan2_thr_cmd = 1000; //油门指令

    cmd.chan3_enable = 1; //使生效
    cmd.chan3_node_id = 2; //匹配电调 2,12,22,32,42,52
    cmd.chan3_thr_cmd = 1000; //油门指令

    cmd.chan4_enable = 1; //使生效
    cmd.chan4_node_id = 3; //匹配电调 3,13,23,33,43,53
    cmd.chan4_thr_cmd = 1000; //油门指令
```

定义了一个 uint64_t 大小的结构体，即 8 个字节，有效字节是 7。

3.5 油门控制（超过 8 轴）

油门指令的消息 ID 是 1130

结构体定义如下：

```
Struct      CAN_CMD_BITS {              // bits   description
    uint16_t cmd:10;                    // 9:0          cmd, range 0~1000
    uint16_t node_id:6;                 // 15:10node_id, range 1~63
};

union CAN_CMD_UN {
    uint16_t            all;
    struct CAN_CMD_BITS       bit;
};

typedef struct
{
     union CAN_CMD_UN esc[8];    // ESC numbers, e.g.8
} CAN_ESC_CMD;
```

结构体名称是 CAN_ESC_CMD，包含所使用电调的控制信息（上述例程中为 8 个），每个电调的控制信息是 1 个 uint16_t 类型的变量，低 10 位表示油门指令，取值范围是 0~1000，对应油门大小是 0%~100%，高 6 位表示电调节点 ID，取值范围是 1~63。

如果需要控制 16 轴，则发送 2 包。需要控制 32 轴，则发送 4 包。注意电调的 node_id 要保持唯一。

3.6 数字签名

对于 dronecan 协议，需要用到数字签名。数字签名定义如下：

指令描述	传输方向	Message-type-ID	数字签名
油门控制	飞控 → 电调	1160	0x5362ab78cd12f03aULL
油门控制（超过 8 轴）	飞控 → 电调	1130	0x5362ab78cd12f03aULL
状态一上报	电调 → 飞控	1150	0x2362ab78cd12f03aULL
状态二上报	电调 → 飞控	1151	0x2362ab78cd12f03aULL
状态三上报	电调 → 飞控	1152	0x2362ab78cd12f03aULL
状态四上报	电调 → 飞控	1153	0x2362ab78cd12f03aULL
状态五上报	电调 → 飞控	1154	0x2362ab78cd12f03aULL
通用指令	电调 → 飞控	999	0x1362ab78cd12f03aULL
	飞控 → 电调	1000	

4 CUBECAN 协议交互

该协议是我司基于普通的 CAN 扩展帧自定义的协议。对用户来说，使用简单。

目前 CANID 使用范围是 0x10000000~0x10000146。所有 CANID 定义如下：

```
#define CUBECAN_ESC_CMD_ID ((uint32_t)0x10000000)//fc -> esc, CUBECAN_ESC_CMD
```

```
#define CUBECAN_ESC0_STAT1_ID ((uint32_t)0x10000001)//esc -> fc, CUBECAN_ESC_STAT1
//1 号电调至 62 号电调上报状态一，CAN ID 值递增，此处省略。
#define CUBECAN_ESC63_STAT1_ID ((uint32_t)0x10000040)//esc -> fc, CUBECAN_ESC_STAT1

#define CUBECAN_ESC0_STAT2_ID ((uint32_t)0x10000041)//esc -> fc, CUBECAN_ESC_STAT2
//1 号电调至 62 号电调上报状态二，CAN ID 值递增，此处省略。
#define CUBECAN_ESC63_STAT2_ID ((uint32_t)0x10000080)//esc -> fc, CUBECAN_ESC_STAT2

#define CUBECAN_ESC0_STAT3_ID ((uint32_t)0x10000081)//esc -> fc, CUBECAN_ESC_STAT3
//1 号电调至 62 号电调上报状态三，CAN ID 值递增，此处省略。
#define CUBECAN_ESC63_STAT3_ID ((uint32_t)0x100000C0)//esc -> fc, CUBECAN_ESC_STAT3

#define CUBECAN_ESC_LED_ID ((uint32_t)0x100000C1)//fc -> esc, CUBECAN_ESC_LED

#define CUBECAN_ESC_ENA_ID ((uint32_t)0x100000C2)//fc -> esc, CUBECAN_ESC_ENA

#define CUBECAN_ANG_SET_ID ((uint32_t)0x100000C3)//fc -> esc, CUBECAN_ANG_SET

#define CUBECAN_ESC0_STAT4_ID ((uint32_t)0x100000C4)//esc -> fc, CUBECAN_ESC_STAT4
//1 号电调至 62 号电调上报状态四，CAN ID 值递增，此处省略。
#define CUBECAN_ESC63_STAT4_ID ((uint32_t)0x10000103)//esc -> fc, CUBECAN_ESC_STAT4

#define CUBECAN_ESC _QUERY_STATE_ID ((uint32_t)0x10000104)//fc -> esc,
CUBECAN_QUERY_STATE
#define CUBECAN_ESC _RESV_ID ((uint32_t)0x10000105)//reserved

#define CUBECAN_OPE_ID ((uint32_t)0x10000106)//fc -> esc, CUBECAN_BATCH_OPE
#define CUBECAN_ESC0_ACK_ID ((uint32_t)0x10000107)//esc -> fc, CUBECAN_OPE_ACK
//1 号电调至 62 号电调的操作应答，CAN ID 值递增，此处省略。
#define CUBECAN_ESC63_ACK_ID ((uint32_t)0x10000146)//esc -> fc, CUBECAN_OPE_ACK
```

4.1 油门控制

电调具有上电 0 油门自检逻辑。先连续发送 3 秒 0 油门信号，让电调通过自检。

4.1.1 CAN ID 定义

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC_CMD_ID	发送电调油门指令	CUBECAN_ESC_CMD	8	0x10000000	飞控→电调

4.1.2 负载结构体定义

```
#define ESCx_UNUSED ((uint16_t)0xffff)

struct    CUBECAN_CMD_BITS              // bits    description
{
      uint16_t cmd: 10;                 // 9:0      cmd, range 0~1000, throttle range 0%~100%
      uint16_t node_id: 6;              // 15:10    node_id, range 0~63
};

union CUBECAN_CMD_UN
{
      uint16_t                    all;
      struct CUBECAN_CMD_BITS     bit;
};

typedef struct
{
      union CUBECAN_CMD_UN esc[4];//单帧可最多控制 4 组电调
} CUBECAN_ESC_CMD;
```

①一条 CAN 帧可发送 4 组电调指令；

②4 组指令没有顺序要求；

③4 组指令的电调 ID 不能重复；

④Node_id 的范围是 1 到 63；

⑤指令范围是 0 到 1000,对应油门是 0%-100%；

⑥每组指令可独立设置；

⑦某个通道可不使用，设为 0xffff，表示该通道不起作用。

4.1.3 示例代码

```
void demo_cubecan_send_esc_cmd(void)
{
      CUBECAN_ESC_CMD esc_cmd;
      uint16_t cmd_len = sizeof(esc_cmd);

      memset(&esc_cmd, 0, cmd_len);

      /*** 指令赋值 ***/
      //第 0 组: 向 1 号电调发送指令
      esc_cmd.esc[0].bit.node_id = 1;
```

```
    esc_cmd.esc[0].bit.cmd = 100;

    //第 1 组: 向 0 号电调发送指令
    esc_cmd.esc[1].bit.node_id = 0;
    esc_cmd.esc[1].bit.cmd = 50;

    //第 2 组: 不使用
    esc_cmd.esc[2].all = ESCx_UNUSED;

    //第 3 组: 向 63 号电调发送指令
    esc_cmd.esc[3].bit.node_id = 63;
    esc_cmd.esc[3].bit.cmd = 150;

    /***  调用 CAN 底层发送函数  ***/

    if (cmd_len != 8)
    {
        return;
    }

    can2_send(CUBECAN_ESC_CMD_ID, (uint8_t*)&esc_cmd, cmd_len);
}
```

一帧 CAN 帧最多可控制 4 组电调。因此，如果是四轴飞机，则一个控制周期只需要发送一帧 CAN 帧。如果是六轴或者八轴飞机，则一个控制周期需要发送 2 帧 CAN 帧。

4.2 航灯控制

电调航灯是否通过 CAN 接口控制，需通过云盒客户端进行设置。航灯状态默认为受 CAN 控制，如果上位机设置灯不受 CAN 控制，则飞控发送 CAN 指令也无法改变灯的状态。

4.2.1 CAN ID 定义

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC_LED_ID	控制电调灯状态	CUBECAN_ESC_LED	8	0x100000C1	飞控→电调

4.2.2 负载结构体定义

```
struct    CUBECAN_LED_BITS       // bits    description
{
    uint16_t led: 10;              // 9:0     ESC_LED_STATE
    uint16_t node_id: 6;           // 15:10    node_id, range 0~63
};
```

```
union CUBECAN_LED_UN
{
    uint16_t                    all;
    struct CUBECAN_LED_BITS    bit;
};

typedef struct
{
    union CUBECAN_LED_UN esc[4];//单帧可最多控制 4 组电调
} CUBECAN_ESC_LED;
```

4.2.3 航灯枚举定义

```
typedef enum
{
    CUBECAN_LED_STATE_OFF = 0,        //ABC OFF

    CUBECAN_LED_STATE_A_ON,           //A ON
    CUBECAN_LED_STATE_B_ON,           //B ON
    CUBECAN_LED_STATE_C_ON,           //C ON

    CUBECAN_LED_STATE_AB_ON,          //AB ON
    CUBECAN_LED_STATE_AC_ON,          //AC ON
    CUBECAN_LED_STATE_BC_ON,          //BC ON

    CUBECAN_LED_STATE_A_BLINK,      //A Flash
    CUBECAN_LED_STATE_B_BLINK,      //B Flash
    CUBECAN_LED_STATE_C_BLINK,      //C Flash

    CUBECAN_LED_STATE_AB_BLINK,    //AB Flash alternately
    CUBECAN_LED_STATE_AC_BLINK,    //AC Flash alternately
    CUBECAN_LED_STATE_BC_BLINK,    //BC Flash alternately

    CUBECAN_LED_STATE_ABC_BLINK, //ABC Flash alternately
} CUBECAN_LED_STATE;
```

ABC 代表 3 种颜色的灯，默认为 A – Red，B – Green，C -White。

具体请查阅硬件型号说明书，与实际硬件型号绑定。

4.2.4 示例代码

```
void demo_cubecan_led_ctrl(void)
```

```
{
    CUBECAN_ESC_LED led;
    uint16_t len = sizeof(led);

    memset(&led, 0, len);

    /*** 指令赋值 ***/

    //第 0 组: 向 1 号电调发送指令
    led.esc[0].bit.node_id = 1;
    led.esc[0].bit.led = CUBECAN_LED_STATE_A_ON;

    //第 1 组: 向 0 号电调发送指令
    led.esc[1].bit.node_id = 0;
    led.esc[1].bit.led = CUBECAN_LED_STATE_B_ON;

    //第 2 组: 不使用
    led.esc[2].all = ESCx_UNUSED;

    //第 3 组: 向 63 号电调发送指令
    led.esc[3].bit.node_id = 63;
    led.esc[3].bit.led = CUBECAN_LED_STATE_C_ON;

    /*** 调用 CAN 底层发送函数 ***/

    if (len != 8)
    {
        return;
    }

    can2_send(CUBECAN_ESC_LED_ID, (uint8_t*)&led, len);
}
```

4.3 使能上报

4.3.1 CAN ID 定义

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC_ENA_ID	使能电调状态上报	CUBECAN_ESC_ENA	8	0x100000C2	飞控→电调

4.3.2 负载结构体定义

```
struct    CUBECAN_ENA_BITS       // bits    description
{
      uint16_t ena: 10;              // 9:0      0->disable, 1->enable
      uint16_t node_id: 6;          // 15:10    node_id, range 0~63
};

union CUBECAN_ENA_UN
{
      uint16_t                        all;
      struct CUBECAN_ENA_BITS    bit;
};

typedef struct
{
      union CUBECAN_ENA_UN esc[4];//单帧可最多控制 4 组电调
} CUBECAN_ESC_ENA;
```

4.3.3 示例代码

```
void demo_cubecan_enable_esc_status_report(void)
{
      CUBECAN_ESC_ENA ena;
      uint16_t len = sizeof(ena);

      memset(&ena, 0, len);

      /***  指令赋值  ***/

      //第 0 组:  向 1 号电调发送指令
      ena.esc[0].bit.node_id = 1;
      ena.esc[0].bit.ena = 1;

      //第 1 组:  向 0 号电调发送指令
      ena.esc[1].bit.node_id = 0;
      ena.esc[1].bit.ena = 1;

      //第 2 组:  不使用
      ena.esc[2].all = ESCx_UNUSED;

      //第 3 组:  向 63 号电调发送指令
```

```
    ena.esc[3].bit.node_id = 63;
    ena.esc[3].bit.ena = 1;

    /*** 调用 CAN 底层发送函数 ***/

    if (len != 8)
    {
        return;
    }

    can2_send(CUBECAN_ESC_ENA_ID, (uint8_t*)&ena, len);
}
```

4.4 状态上报

①电调上电默认不上报。如需上报电调状态，飞控需要发送使能命令（请参考 4.3 章节）。

②当发送使能上报指令，打开指定电调上报，指定电调会一直上报，上报频率为 10HZ。

③当发送使能上报指令，关闭指定电调上报，指定电调会停止上报。

④每个电调可以上报 4 种状态信息，分别是 STAT1/ STAT2/ STAT3/ STAT4。

4.4.1 CAN ID 定义

状态一上报：

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC0_STAT1_ID	0 号电调上报状态一	CUBECAN_ESC_STAT1	8	0x10000001	电调→飞控
1 号电调至 62 号电调上报状态一，CAN_ID 值递增，此处省略。					
CUBECAN_ESC63_STAT1_ID	63 号电调上报状态一	CUBECAN_ESC_STAT1	8	0x10000040	电调→飞控

状态二上报

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC0_STAT2_ID	0 号电调上报状态二	CUBECAN_ESC_STAT2	8	0x10000041	电调→飞控
1 号电调至 62 号电调上报状态二，CAN_ID 值递增，此处省略。					
CUBECAN_ESC63_STAT2_ID	63 号电调上报状态二	CUBECAN_ESC_STAT2	8	0x10000080	电调→飞控

状态三上报

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC0_STAT3_ID	0 号电调上报状态三	CUBECAN_ESC_STAT3	8	0x10000081	电调→飞控
1 号电调至 62 号电调上报状态三，CAN_ID 值递增，此处省略。					

CUBECAN_ESC63_STAT3_ID	63 号电调上报状态三	CUBECAN_ESC_STAT3	8	0x100000C0	电调→飞控

状态四上报

CANID 名称	**CANID 描述**	**负载结构体定义**	**负载长度**	**CAN_ID 值**	**传输方向**
CUBECAN_ESC0_STAT4_ID	0 号电调上报状态四	CUBECAN_ESC_STAT4	8	0x100000C4	电调→飞控
1 号电调至 62 号电调上报状态四，CAN_ID 值递增，此处省略。					
CUBECAN_ESC63_STAT4_ID	63 号电调上报状态四	CUBECAN_ESC_STAT4	8	0x10000103	电调→飞控

4.4.2 负载结构体定义

```
//order: bit0 -> bit15
struct    ESC_MODE_BITS       // bits    description
{
      uint16_t esc_mode: 8;//电调控制模式
      uint16_t pwm_thr_online: 1;//0 代表 PWM 油门丢失，1 代表 PWM 油门在线
      uint16_t can_thr_online: 1;//0 代表 CAN 油门丢失，1 代表 CAN 油门在线
      uint16_t thr_pri: 1;//0 代表 PWM 油门优先，1 代表 CAN 油门优先
      uint16_t resv: 5;//保留位
};

union ESC_MODE_UN
{
      uint16_t                    all;
      struct ESC_MODE_BITS    bit;
};

typedef struct
{
      union ESC_MODE_UN esc_mode;//电调控制模式以及相关状态标志
      int16_t esc_cmd;   //上报的油门指令，与输入指令相同
      int16_t spd_rpm;   //电机转速，单位 rpm
      int16_t mos_temp; //mos 管温度，单位 0.1 摄氏度
} CUBECAN_ESC_STAT1;

typedef struct
{
      int16_t     vdc_100mv;          // 母线电压，单位 0.1V
      int16_t     irms_100ma;         // 相电流有效值，单位 0.1A
      int16_t     idq_100ma[2];       // d-q 轴电流，单位 0.1A
```

```
} CUBECAN_ESC_STAT2;

typedef struct
{
    int16_t alg_err;            //算法错误字。0 代表无错误，!=0 代表有错误
    int16_t alg_warn;           //算法警告字。0 代表无警告，!=0 代表有警告
    int16_t vdq_duty[2];        // d-q 轴指令
} CUBECAN_ESC_STAT3;

typedef struct
{
  int16_t idc;//母线估测电流，单位 0.1A
  int16_t cap_temp; //电容温度，单位 0.1 摄氏度
  int16_t motor_temp;//电机温度，单位 0.1 摄氏度
  int16_t resv;//保留字节
} CUBECAN_ESC_STAT4;
```

飞控接收参考代码：

```
typedef struct
{
    CUBECAN_ESC_STAT1 stat1;
    CUBECAN_ESC_STAT2 stat2;
    CUBECAN_ESC_STAT3 stat3;
    CUBECAN_ESC_STAT4 stat4;
    uint32_t stat1_recv_cnt;
    uint32_t stat2_recv_cnt;
    uint32_t stat3_recv_cnt;
    uint32_t stat4_recv_cnt;
} CUBECAN_ESC_STATUS;
CUBECAN_ESC_STATUS esc_sta[64];//对应电调 0 至电调 63 的状态

void init_esc_sta(void)
{
    memset(esc_sta, 0, sizeof(esc_sta));
}

void demo_cubecan_recv_esc_status(uint32_t ext_id, uint8_t *data)
{
    uint16_t idx = 0;

    if ((ext_id >= CUBECAN_ESC0_STAT1_ID) && (ext_id <= CUBECAN_ESC63_STAT1_ID))
    {
```

```
            idx = ext_id - CUBECAN_ESC0_STAT1_ID;
            memcpy(&esc_sta[idx].stat1, data, 8);
            esc_sta[idx].stat1_recv_cnt++;
        }
        else if ((ext_id >= CUBECAN_ESC0_STAT2_ID) && (ext_id <= CUBECAN_ESC63_STAT2_ID))
        {
            idx = ext_id - CUBECAN_ESC0_STAT2_ID;
            memcpy(&esc_sta[idx].stat2, data, 8);
            esc_sta[idx].stat2_recv_cnt++;
        }
        else if ((ext_id >= CUBECAN_ESC0_STAT3_ID) && (ext_id <= CUBECAN_ESC63_STAT3_ID))
        {
            idx = ext_id - CUBECAN_ESC0_STAT3_ID;
            memcpy(&esc_sta[idx].stat3, data, 8);
            esc_sta[idx].stat3_recv_cnt++;
        }
        else if ((ext_id >= CUBECAN_ESC0_STAT4_ID) && (ext_id <= CUBECAN_ESC63_STAT4_ID))
        {
            idx = ext_id - CUBECAN_ESC0_STAT4_ID;
            memcpy(&esc_sta[idx].stat4, data, 8);
            esc_sta[idx].stat4_recv_cnt++;
        }
        else
        {
        }
}
```

4.5 查询上报

针对某些不需要电调一直上报的飞控应用，新增此条协议：当飞控发送一次查询指令时，电调回复一次状态帧 STAT1/STAT2/STAT3/STAT4，状态帧定义请参考 4.4 章节。

4.5.1 CAN ID 定义

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_ESC_QUERY_STATE_ID	查询电调状态	CUBECAN_QUERY_STATE	8	0x10000104	飞控→电调

负载结构体长度是 8，内部是一个 uint64_t 类型，bit0 至 bit63 代表电调 0 至电调 63。当 bit 为 0 时，代表不查询当前电调状态，电调不会回复状态帧；当 bit 为 1 时，代表查询当前电调状态，电调会回复状态帧。

例如，如果 query_flg=0xFFFFFFFFFFFFFFFF，则查询所有电调状态，所有电调都会回复状态帧；如果 query_flg=0x1，则只查询 0 号电调的状态，只有 0 号电调会回复状态帧；如果 query_flg=0x8000000000000000，则只查询 63 号电调的状态，只有 63 号电调会回复状态帧。

4.5.2 负载结构体定义

```
typedef struct
{
    uint64_t query_flg;//bit0~bit63 代表 ESC0~ESC63：0 代表不查询状态，1 代表查询状态
} CUBECAN_QUERY_STATE;
```

4.6 读写参数

读写参数协议，是为了方便飞控直接设置电调各项参数，如 NODE_ID、电机方向、油门优先级等。也可以使用我司的云盒设备，对电调设置参数。

飞控读写参数的 CAN ID 是 CUBECAN_OPE_ID。

0 号至 63 号电调回复 CAN ID 分别是 CUBECAN_ESC0_ACK_ID 至 CUBECAN_ESC63_ACK_ID。负载结构体中的 CS，代表具体命令操作符。

该协议既支持批量读写电调参数，也支持读写单个电调参数。当批量读写电调参数时，CAN 总线上的所有电调均会回复；当读写单个电调参数时，只有指定的那个电调会回复，其他电调均不回复。

写入参数成功后，电调断电重启后生效。目前开放的参数项有 6 种，分别是：电调 NODE_ID、电机方向、油门优先级、电调灯上电默认状态、定桨停机角度和定桨开关。

下文会对负载结构体的使用做出详细解释，并给出示例以供参考。如需开放更多参数项，请联系我司技术支持或销售！

4.6.1 CAN ID 定义

CANID 名称	CANID 描述	负载结构体定义	负载长度	CAN_ID 值	传输方向
CUBECAN_OPE_ID	飞控操作命令	CUBECAN_BATCH_OPE	8	0x10000106	电调→飞控
CUBECAN_ESC0_ACK_ID	0 号电调操作回复	CUBECAN_OPE_ACK	8	0x10000107	电调→飞控
1 号电调至 62 号电调的操作回复，CAN ID 值递增，此处省略。					
CUBECAN_ESC63_ACK_ID	63 号电调操作回复	CUBECAN_OPE_ACK	8	0x10000146	电调→飞控

4.6.2 负载结构体定义

```
//飞控发送
typedef struct
{
    uint16_t cs;//命令操作符，表示读或写某项参数的请求
    int16_t data;//写操作时，表示写入的数据。读操作时，该值无意义
```

```
    uint16_t batch;//0->单个读或写,1->批量读或写
    uint16_t target_node_id;//当单个读或写时，表示操作指定电调。批量读或写时，该值无意义
} CUBECAN_BATCH_OPE;


//电调回复
typedef struct
{
    uint16_t cs;//命令操作符，表示读或写某项参数的回复
    int16_t src_node_id;//表示回复的电调 NODE_ID
    int16_t ret;//表示读操作或写操作是否成功：0 表示成功，<0 表示失败
    int16_t data;//读操作时，表示读取到的数据。写操作时，该值无意义
} CUBECAN_OPE_ACK;
```

4.6.3 CS 命令操作符定义

```
typedef enum
{
    CS_BATCH_SET_START = 16,
    CS_BATCH_SET_ESC_CAN_ID_REQ = 16,//  写电调 NODE_ID 参数请求
    CS_BATCH_SET_ESC_CAN_ID_RES,//  写电调 NODE_ID 参数回复
    CS_BATCH_SET_MOTOR_DIR_REQ,//写电机方向参数请求
    CS_BATCH_SET_MOTOR_DIR_RES,//写电机方向参数回复
    CS_BATCH_SET_THR_PRI_REQ,//写油门优先级参数请求
    CS_BATCH_SET_THR_PRI_RES,//写油门优先级参数回复
    CS_BATCH_SET_LED_STA_REQ,//写电调灯上电默认状态参数请求
    CS_BATCH_SET_LED_STA_RES,//写电调灯上电默认状态参数回复
    CS_BATCH_SET_ANG_REQ,//写定桨停机参数请求
    CS_BATCH_SET_ANG_RES,//写定桨停机参数回复
    CS_BATCH_SET_LOCK_REQ,//写定桨开关参数请求
    CS_BATCH_SET_LOCK_RES,//写定桨开关参数回复
    CS_BATCH_SET_END,


    CS_BATCH_GET_START = 256,
```

```
    CS_BATCH_GET_ESC_CAN_ID_REQ = 256,//读电调 NODE_ID 参数请求
    CS_BATCH_GET_ESC_CAN_ID_RES,//读电调 NODE_ID 参数回复
    CS_BATCH_GET_MOTOR_DIR_REQ,//读电机方向参数请求
    CS_BATCH_GET_MOTOR_DIR_RES,//读电机方向参数回复
    CS_BATCH_GET_THR_PRI_REQ,//读油门优先级参数请求
    CS_BATCH_GET_THR_PRI_RES,//读油门优先级参数回复
    CS_BATCH_GET_LED_STA_REQ,//读电调灯参数请求
    CS_BATCH_GET_LED_STA_RES,//读电调灯参数回复
    CS_BATCH_GET_ANG_REQ,//读定桨停机参数请求
    CS_BATCH_GET_ANG_RES,//读定桨停机参数回复
    CS_BATCH_GET_LOCK_REQ,//读定桨开关参数请求
    CS_BATCH_GET_LOCK_RES,//读定桨开关参数回复
    CS_BATCH_GET_END,
} CUBECAN _CS;
```

4.6.4 CS 参数值合法性规定

参数名称	取值范围
电调 NODE_ID	1 到 63。
电机方向	-1 或者 1。-1 代表反向，1 代表正向。
油门优先级	0 或者 1。0 代表 PWM 油门优先，1 代表 CAN 油门优先。
机臂灯状态	0 到 13。详情参见 4.2.3 章节。
定桨停机角度	-900 到 900，代表-90 度到 90 度。
定桨开关	0 或者 1。0 代表关闭，1 代表打开。

4.6.5 示例代码

(1) 写入单个电调的 NODE_ID 时，飞控发送的 CAN ID 和负载结构体如下：

```
CAN ID：CUBECAN_OPE_ID= 0x10000106
负载结构体：CUBECAN_BATCH_OPE
```

假设设置 1 号电调的 NODE_ID 为 10，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_SET_ESC_CAN_ID_REQ;// 写电调 NODE_ID 参数请求
batch_ope.data = 10;//设置电调 NODE_ID 为 10
```

```
batch_ope.batch = 0;//0 表示设置指定电调
batch_ope.target_node_id = 1;//指定电调的 NODE_ID
```

只有 1 号电调会回复飞控，发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_SET_ESC_CAN_ID_RES;// 写电调 NODE_ID 参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示写入成功
//ack.data，对于写参数回复，该值无意义。
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	10 00 0A 00 00 00 01 00
电调回复帧	扩展帧	数据帧	10000108	8	11 00 01 00 00 00 00 00

(2) 批量写入电调 NODE_ID 时，飞控发送的 CAN ID 和负载结构体如下：

```
CAN ID：CUBECAN_OPE_ID= 0x10000106
负载结构体：CUBECAN_BATCH_OPE
```

假设设置 CAN 总线上的所有电调的 NODE_ID 为 10，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_SET_ESC_CAN_ID_REQ;// 写电调 NODE_ID 参数请求
batch_ope.data = 10;// 设置 CAN 总线上的所有电调的 NODE_ID 为 10
batch_ope.batch = 1;//1 表示设置所有电调
//batch_ope.target_node_id  //批量设置时，该值无意义
```

假设 CAN 总线上有 2 个电调，分别是 1 号电调和 2 号电调，都会回复飞控。

1 号电调发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
```

```
ack.cs = CS_BATCH_SET_ESC_CAN_ID_RES;// 写电调 NODE_ID 参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示写入成功
//ack.data  //对于写参数回复，该值无意义。
```

2 号电调发送 CAN ID：CUBECAN_ESC2_ACK_ID= 0x10000109。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_SET_ESC_CAN_ID_RES;// 写电调 NODE_ID 参数回复
ack. src_node_id = 2;//回复的电调 NODE_ID 为 2
ack. ret = 0;//0 表示写入成功
//ack.data  //对于写参数回复，该值无意义。
```

CAN 报文：

	帧类型	**帧格式**	**帧 ID**	**数据长度**	**数据**
飞控发送帧	扩展帧	数据帧	10000106	8	10 00 0A 00 01 00 01 00
电调回复帧	扩展帧	数据帧	10000109	8	11 00 02 00 00 00 00 00
电调回复帧	扩展帧	数据帧	10000108	8	11 00 01 00 00 00 00 00

(3) 读单个电调的 NODE_ID

飞控发送 CAN ID：CUBECAN_OPE_ID= 0x10000106。

飞控发送负载结构体：CUBECAN_BATCH_OPE。

假设读取 1 号电调的 NODE_ID，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_GET_ESC_CAN_ID_REQ;// 读电调 NODE_ID 参数请求
//batch_ope.data //读参数时，该值无意义
batch_ope.batch = 0;//0 表示读取指定电调
batch_ope.target_node_id = 1;//指定电调的 NODE_ID
```

只有 1 号电调会回复飞控，发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
```

```
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_GET_ESC_CAN_ID_RES;// 读电调 NODE_ID 参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示读取成功
ack.data=1;//读取到的电调 NODE_ID 是 1
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	00 01 00 00 00 00 01 00
电调回复帧	扩展帧	数据帧	10000108	8	01 01 01 00 00 00 01 00

(4) 批量读电调 NODE_ID

飞控发送 CAN ID：CUBECAN_OPE_ID= 0x10000106。

飞控发送负载结构体：CUBECAN_BATCH_OPE。

假设读取 CAN 总线上所有电调的 NODE_ID，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_GET_ESC_CAN_ID_REQ;// 读电调 NODE_ID 参数请求
//batch_ope.data //读参数时，该值无意义
batch_ope.batch = 1;//1 表示读取所有电调
//batch_ope.target_node_id  //批量读取时，该值无意义
```

假设 CAN 总线上有 2 个电调，分别是 1 号电调和 2 号电调，都会回复飞控。

1 号电调发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_GET_ESC_CAN_ID_RES;// 读电调 NODE_ID 参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示读取成功
ack.data=1;//读取到的电调 NODE_ID 是 1
```

2 号电调发送 CAN ID：CUBECAN_ESC2_ACK_ID= 0x10000109。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_GET_ESC_CAN_ID_RES;// 读电调 NODE_ID 参数回复
ack. src_node_id = 2;//回复的电调 NODE_ID 为 2
ack. ret = 0;//0 表示读取成功
ack.data=2;//读取到的电调 NODE_ID 是 2
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	00 01 00 00 01 00 00 00
电调回复帧	扩展帧	数据帧	10000109	8	01 01 02 00 00 00 02 00
电调回复帧	扩展帧	数据帧	10000108	8	01 01 01 00 00 00 01 00

(5) 写入单个电调电机方向

飞控发送 CAN ID：CUBECAN_OPE_ID= 0x10000106。

飞控发送负载结构体：CUBECAN_BATCH_OPE。

假设设置 1 号电调的电机方向为正向，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_SET_MOTOR_DIR_REQ;// 写电调电机方向参数请求
batch_ope.data = 1;//1 代表电机方向为正向
batch_ope.batch = 0;//0 表示设置指定电调
batch_ope.target_node_id = 1;//指定电调的 NODE_ID
```

只有 1 号电调会回复飞控，发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_SET_MOTOR_DIR_RES;// 写电调电机方向参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示写入成功
```

```
//ack.data，对于写参数回复，该值无意义。
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	12 00 01 00 00 00 01 00
电调回复帧	扩展帧	数据帧	10000108	8	13 00 01 00 00 00 00 00

(6) 批量写入电调电机方向

飞控发送 CAN ID：CUBECAN_OPE_ID= 0x10000106。

飞控发送负载结构体：CUBECAN_BATCH_OPE。

假设设置 CAN 总线上的所有电调的电机方向为正向，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_SET_MOTOR_DIR_REQ;// 写电机方向参数请求
batch_ope.data = 1;// 设置 CAN 总线上的所有电调的电机方向为正向
batch_ope.batch = 1;//1 表示设置所有电调
//batch_ope.target_node_id  //批量设置时，该值无意义
```

假设 CAN 总线上有 2 个电调，分别是 1 号电调和 2 号电调，都会回复飞控。

1 号电调发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_SET_MOTOR_DIR_RES;// 写电机方向参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示写入成功
//ack.data  //对于写参数回复，该值无意义。
```

2 号电调发送 CAN ID：CUBECAN_ESC2_ACK_ID= 0x10000109。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
```

```
ack.cs = CS_BATCH_SET_MOTOR_DIR_RES;// 写电机方向参数回复
ack. src_node_id = 2;//回复的电调 NODE_ID 为 2
ack. ret = 0;//0 表示写入成功
//ack.data  //对于写参数回复，该值无意义。
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	12 00 01 00 01 00 00 00
电调回复帧	扩展帧	数据帧	10000109	8	13 00 02 00 00 00 00 00
电调回复帧	扩展帧	数据帧	10000108	8	13 00 01 00 00 00 00 00

(7) 读取单个电调电机方向

飞控发送 CAN ID：CUBECAN_OPE_ID= 0x10000106。

飞控发送负载结构体：CUBECAN_BATCH_OPE。

假设设置 1 号电调的电机方向为正向，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_GET_MOTOR_DIR_REQ;// 读电调电机方向参数请求
//batch_ope.data //读参数时，该值无意义
batch_ope.batch = 0;//0 表示读取指定电调
batch_ope.target_node_id = 1;//指定电调的 NODE_ID
```

只有 1 号电调会回复飞控，发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_GET_MOTOR_DIR_RES;// 读电调电机方向参数请求
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示读取成功
ack.data=1;//读取到的参数值是 1，代表电机方向是正向
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	02 01 00 00 00 00 01 00
电调回复帧	扩展帧	数据帧	10000108	8	03 01 01 00 00 00 01 00

(8) 批量读取电调电机方向

飞控发送 CAN ID：CUBECAN_OPE_ID= 0x10000106。

飞控发送负载结构体：CUBECAN_BATCH_OPE。

假设设置 CAN 总线上的所有电调的电机方向为正向，则负载结构体赋值如下：

```
CUBECAN_BATCH_OPE batch_ope;
memset(&batch_ope, 0, sizeof(batch_ope));
batch_ope.cs = CS_BATCH_GET_MOTOR_DIR_REQ;// 读电机方向参数请求
//batch_ope.data //读参数时，该值无意义
batch_ope.batch = 1;//1 表示读取所有电调
//batch_ope.target_node_id  //批量读取时，该值无意义
```

假设 CAN 总线上有 2 个电调，分别是 1 号电调和 2 号电调，都会回复飞控。

1 号电调发送 CAN ID：CUBECAN_ESC1_ACK_ID= 0x10000108。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_GET_MOTOR_DIR_RES;// 读电机方向参数回复
ack. src_node_id = 1;//回复的电调 NODE_ID 为 1
ack. ret = 0;//0 表示读取成功
ack.data=1;//读取到的参数值是 1，代表电机方向是正向
```

2 号电调发送 CAN ID：CUBECAN_ESC2_ACK_ID= 0x10000109。

电调回复的负载结构体：CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;
memset(&ack, 0, sizeof(ack));
ack.cs = CS_BATCH_GET_MOTOR_DIR_RES;// 读电机方向参数回复
ack. src_node_id = 2;//回复的电调 NODE_ID 为 2
ack. ret = 0;//0 表示读取成功
ack.data=1;//读取到的参数值是 1，代表电机方向是正向
```

CAN 报文：

	帧类型	帧格式	帧 ID	数据长度	数据
飞控发送帧	扩展帧	数据帧	10000106	8	02 01 00 00 01 00 00 00
电调回复帧	扩展帧	数据帧	10000109	8	03 01 02 00 00 00 01 00
电调回复帧	扩展帧	数据帧	10000108	8	03 01 01 00 00 00 01 00

5 修订记录

序号	变更时间	版本	描述
1	2022-12-08	V1.0.0	创建
2	2022-12-17	V1.0.1	增加航灯控制枚举
3	2023-05-16	V2.0.0	增加 CUBECAN 协议
4	2024-03-25	V2.1.0	增加 DRONECAN 协议
5	2024-04-23	V2.1.1	增加数字签名
6	2024-06-07	V2.1.2	增加消息 ID 定义
7	2024-08-08	V2.1.3	DRONECAN 协议状态上报（电调→飞控），采用单帧传输
8	2024-08-19	V2.1.4	DRONECAN 协议指令下发（飞控→电调），采用单帧传输
9	2025-04-29	V2.1.8	CUBECAN 协议支持读写参数
10	2025-06-21	V2.1.9	CUBECAN 协议支持读写定桨开关参数
11	2025-09-04	V2.2.0	添加新的油门控制指令，支持超过 8 轴