ESCs CAN Communication Manual

V2.2.0

POWER MAKES YOUR EXPLORATION

Directory

1 General

1.1 Introduction

The ESCs are developed according to the CAN 2.0B protocol with 29-bit identifier (Extended frame).

The ESCs support DRONECAN and CUBECAN which is designed by TMOTOR. Only one protocol can be selected according to your needs.

The CUBECAN protocol is an easy-to-use CAN protocol developed by TMOTOR, which allows direct interaction using CAN frames. Compared with the DRONECAN protocols, it does not require not only combine packets (multiple CAN frames to form a single DRONECAN packet), but also unpacking (one DRONECAN packet is split into multiple CAN frames).

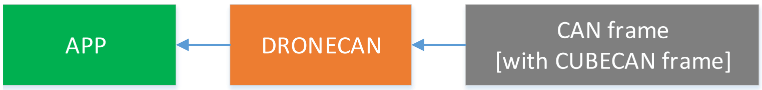


Fig. 1 Block diagram of program

1.2 CAN Bus Baud Rate

The baud rate is 1MHz, which can be modified via CloudLink.

1.3 Node ID Setting

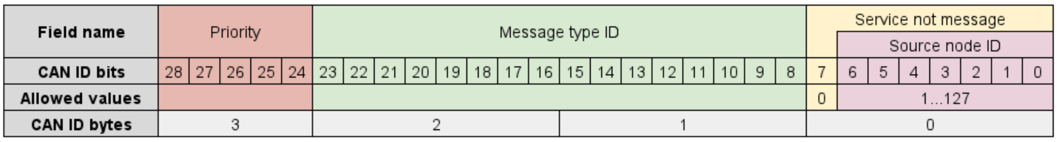
Refer to " Master Series ESC QuickStart Guide " and "CloudLink User manual ".

1.4 Comparison of two protocols

1.4.1 DRONECAN: Successor to UAVCAN V0.9.

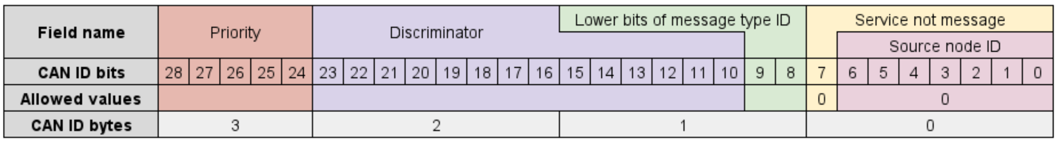
ID field

Message frame



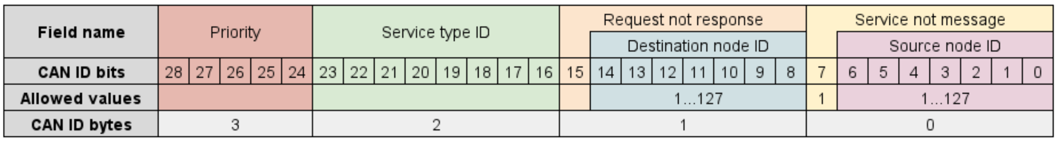
Field name	Priority					Message type ID																Service not message							
																							Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	1...127						
CAN ID bytes	3					2								1								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								Service not message							
															Destination node ID								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: The protocol is based on normal CAN extended frames. It is easy to use for users.

2. DRONECAN Protocol Overview (UAVCAN V0.9)

2.1 communications model

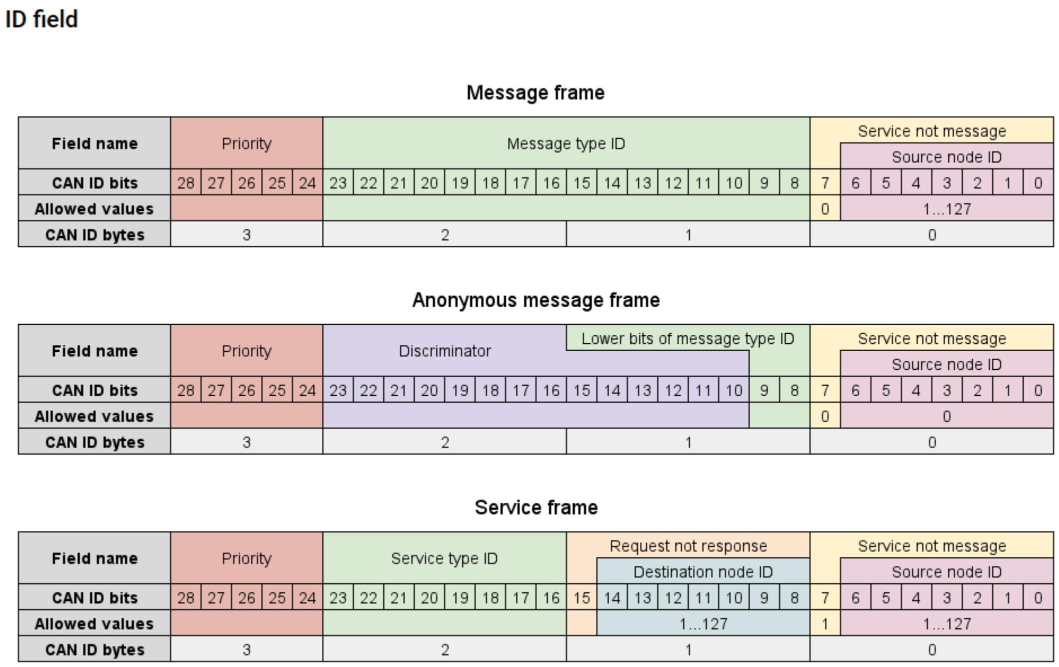
The nodes of the UAVCAN network can communicate using any of the following communication methods:

1) Message broadcasting – The primary method of data exchange with publish/subscribe semantics.

2) Service invocation – The communication method for peer-to-peer request/response interactions.

There uses message broadcasting.

2.2 Definition of CAN ID



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								Service not message							
															Destination node ID								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							

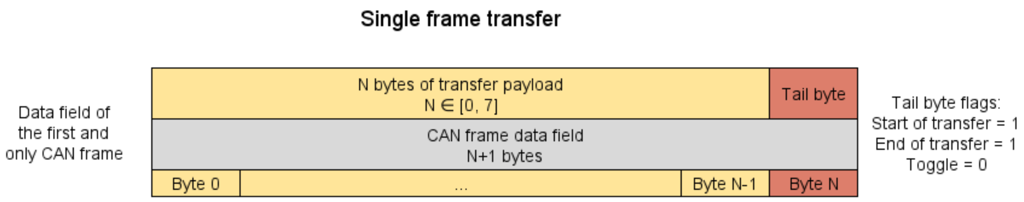
BIT24 ~ BIT28: Priority.

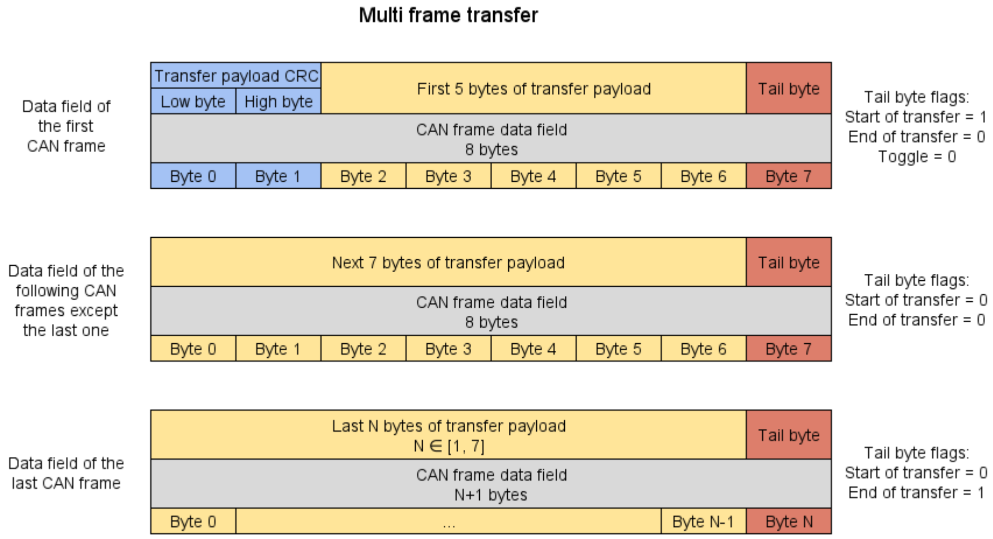
BIT8 ~ BIT23: Message ID.

BIT7: 0.

BIT0 ~ BIT6: NODE ID.

2.3 Single/multi frame transfer

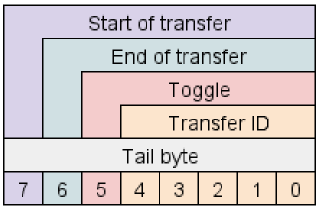




The CRC byte of the DRONECAN protocol (UAVCAN V0.9) is in the header of the packet.

2.4 Tail Byte

The 8 bits of the trailing 1-byte are split into fields to carry protocol information.



Field	Bits	Description
Start of transfer	BIT7(1)	For single-frame transfers, the value of this field is always 1. For multi-frame transfers, the value of this field is 1 if the current frame is the first frame of the transfer, and 0 otherwise.
End of transfer	BIT6(1)	For single-frame transfers, the value of this field is always 1. For multi-frame transfers, the value of this field is 1 if the current frame is the last frame of the transfer, and 0 otherwise.
Toggle bit	BIT5(1)	For single-frame transfers, the value of this field is always 0.

		For multi-frame transfers, this field contains the value of the toggle bit. As specified above this will alternate value between frames, starting at 0 for the first frame.
Transfer ID	BIT0~BIT4(5)	This field contains the transfer ID value of the current transfer (for all types of transfers). The value is 5 bits wide, therefore the allowed values range from 0 to 31, inclusively.

2.5 Reference

Official Documentation:

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

Open-source code:

https://github.com/DRONECAN/libcanard

3. DRONECAN Protocol Interaction

This section describes the ESC interaction messages for the DRONECAN protocol (UAVCAN V0.9). Description of the message ID when used: message-type-ID represents the message ID and occupies the bit segment BIT8-BIT23.

ID	DRONECAN (UAVCAN V0.9)
Message ID	BIT8-BIT23
NODE ID	BIT0-BIT6
Priority	BIT24-BIT28

3.1 Usage agreement

(1) NODE_ID convention:

- The NODE_ID of the protocol stack takes values in the range of 1~127.
- The range of the ESC NODE_ID is 1~63.
- The Flight Control (FC) node ID value range is 1~127.
- On CAN bus, NODE_ID cannot be the same, node ID has uniqueness. (Between FC and ESC, ESC and ESC, ESC and mount, etc.)

(2) Message ID convention:

Instruction Description	Message-type-ID	Transmission Direction
Status 1 reporting	1150	ESC → FC
Status 2 reporting	1151	ESC → FC
Status 3 reporting	1152	ESC → FC
Status 4 reporting	1153	ESC → FC
Status 5 reporting	1154	ESC → FC
Generic instruction	999	ESC → FC
	1000	FC → ESC
Command	1160	FC → ESC

The above message-type-ID is already occupied and is strictly forbidden for other devices. (e.g. mounting devices, etc.)

3.2 General Instruction ID

The Message ID for the throttle command is 1000.

The message of the generic command contains another layer of protocol, which is a set of protocols customized by our company. In order to support multiple data interactions and the frequency of interaction is relatively low, so the same message ID is used. The message carried by a general-purpose instruction consists of an internal protocol header and an internal message structure.

The structure is defined as follows:

```
#define CAN_APP_PROTO_MAGIC 0xabcd

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

The internal protocol header consists of MAGIC, the internal protocol Message-ID, and the message length.

3.2.1 Enable Reporting

The ESC program does not report its status by default upon powering up. It must be enabled to report its status before doing so.

Internal Protocol Message ID： ENA_ESC_STAT_REP_MSG_ID = 4670.

The corresponding message structure:

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

enable = 0: Turn off ESC status reporting

enable = 1: Turn on ESC status reporting

3.2.2 LED Control

Whether the ESC’s LEDs is controlled by CAN interface needs to be set by the user through the client, please refer to the "CloudLink User Manual" for details. The default state of the LEDs is CAN controlled.

Internal Protocol Message ID： ARM_LED_CTRL_MSG_ID = 4669.

The corresponding message structure.

```
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;
```

LED enumeration:

```
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 stands for 3 colors of lights, default is A - Red, B - Green, C - White.

Please refer to the hardware model specification for details, which is bound to the actual hardware model.

3.3 Status Reporting ID

There are 4 states for ESC status reporting, which are state 1/state 2/state 3/state 4/state 5. Message-IDs are 1150/1151/1152/1153/1154 and are transmitted in a single frame. The load byte length of the CAN frame is 8, the effective byte length is 7, and the last 1 byte is the trailing byte of the DRONECAN single frame protocol. The reporting frequency of the ESC is 10 HZ.

Instruction Description	Message-type-ID	Structural
Status 1 reporting	1150	S_CAN_MSG_ESC_STAT1

Status 2 reporting	1151	S_CAN_MSG_ESC_STAT2
Status 3 reporting	1152	S_CAN_MSG_ESC_STAT3
Status 4 reporting	1153	S_CAN_MSG_ESC_STAT4
Status 5 reporting	1154	S_CAN_MSG_ESC_STAT5

Definitions are as follows:

```
// ESC status reporting (single frame transmission)
#define S_ESC_STAT1_PORT_ID ((uint32_t)1150)// ESC -> External, S_CAN_MSG_ESC_STAT1
#define S_ESC_STAT2_PORT_ID ((uint32_t)1151)// ESC -> External, S_CAN_MSG_ESC_STAT2
#define S_ESC_STAT3_PORT_ID ((uint32_t)1152)// ESC -> External, S_CAN_MSG_ESC_STAT3
#define S_ESC_STAT4_PORT_ID ((uint32_t)1153)// ESC -> External, S_CAN_MSG_ESC_STAT4
#define S_ESC_STAT5_PORT_ID ((uint32_t)1154)// ESC -> External, 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; // Control modes
        int16_t esc_cmd;    // Throttle commands
        int16_t spd_rpm;    // Motor speed
        uint16_t resv: 8;
        uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT1;

typedef struct
{
        int16_t      vdc_100mv;             // Busbar voltage
        int16_t      irms_100ma;            // RMS phase current
        int16_t      idq0_100ma;        // d-q axis current
        uint16_t resv: 8;
        uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT2;

typedef struct
{
        int16_t alg_err;                  // Algorithmic error word
        int16_t alg_warn;                 // Algorithmic Warning Word
        int16_t vdq0_duty;          // d-q axis commands
        uint16_t resv: 8;
        uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT3;

typedef struct
{
        int16_t mos_temp; // Mos tube temperature
        int16_t      idq1_100ma;        // d-q axis current
        int16_t vdq1_duty;          // d-q axis commands
        uint16_t resv: 8;
        uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT4;

typedef struct
{
    int16_t idc; // Estimated busbar current
    int16_t cap_temp; // Capacitor temperature in 0.1 degrees Celsius
    int16_t motor_temp;// Motor temperature in 0.1 degrees Celsius
    uint16_t resv: 8;
    uint16_t tail: 8;
} S_CAN_MSG_ESC_STAT5;
```

3.4 Throttle Command Control

The ESC has a power up 0 throttle self-test logic. First send 3 seconds of 0 throttle continuously to get the ESC through the self-test. DRONECAN protocol commands are sent down (FC → ESC) using single frame transmission. the CAN frame load byte length is 8, the effective byte length is 7, and the last 1 byte is the tail byte of the DRONECAN single frame protocol.

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

Single frame transmission, each time can send up to 4 groups of ESC's throttle commands. The effective byte length is 7, i.e., the effective number of bits is 7*8=56Bit, which contains the throttle commands of 4 groups of ESCs, then each group of ESCs occupies 56/4=14Bit. The 56Bit bit field is divided as follows:

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]		

Channel 4	BIT 55	Channel 4 entry-into-force switch: 0 no entry-into-force; 1 entry-into-force.
	BIT 54- BIT 52	The number of digits of the NODE ID of the 4th channel ESC: the value range is 0-7. The ESC terminal, according to the information of this 3bit, matches with the number of digits of its own NODE ID, and if it matches, it responds to this command.
	BIT 51- BIT 42	Channel 4 ESC throttle command: value range 0-1000
Channel 3	BIT 41	Channel 3 entry-into-force switch: 0 no entry-into-force; 1 entry-into-force.
	BIT 40- BIT 38	The number of digits of the NODE ID of the ESC of the 3rd channel: the value range is 0-7. The ESC terminal, according to the information of this 3bit, matches with the number of digits of its own NODE ID, and if it matches, it responds to this instruction
	BIT 37- BIT 28	Channel 3 ESC throttle command: value range 0-1000
Channel 2	BIT 27	Channel 2 entry-into-force switch: 0 no entry-into-force; 1 entry-into-force.
	BIT 26- BIT 24	The number of digits of the NODE ID of the 2nd channel ESC: the value range is 0-7. The ESC terminal, according to the information of this 3bit, matches with the number of digits of its own NODE ID, and if it matches, it responds to this command.
	BIT 23- BIT 14	Channel 2 ESC throttle command: value range 0-1000
Channel 1	BIT 13	Channel 1 entry-into-force switch: 0 no entry-into-force; 1 entry-into-force.
	BIT 12- BIT 10	The number of digits of the NODE ID of the 1st channel ESC: the value range is 0-7. The ESC side, according to the information of this 3bit, matches with the number of digits of its own NODE ID, and if it matches, it responds to this command.
	BIT 9- BIT 0	Channel 1 ESC throttle command: value range 0-1000

Note that the NODE ID of the ESC CAN must be selected from one of the following ranges and cannot be set across ranges.

Range of CAN ID Values for ESCs					
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

Defining Structures.

```
//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;
```

Taking 4-axis as an example, suppose the ESC CAN NODE ID is set to 20/21/22/23. we need to send digital throttles with a value of 1000 (0x03E8) to four channels, the first one is 20 (Chan1_node_id = 0), the second one is 21 (Chan2_node_id = 1), the third one is 22 (Chan3_node_id = 2), and the fourth channel is 23 (Chan4_node_id = 3).

The code is as follows:

```
S_CAN_ESC_CMD cmd;

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

        cmd.chan1_enable = 1;// Put into effect
        cmd.chan1_node_id = 0;// Matching ESCs 0,10,20,30,40,50
        cmd.chan1_thr_cmd = 1000;// Throttle instruction

        cmd.chan2_enable = 1; // Put into effect
        cmd.chan2_node_id = 1; // Matching ESCs 1,11,21,31,41,51
        cmd.chan2_thr_cmd = 1000; // Throttle instruction

        cmd.chan3_enable = 1; // Put into effect
        cmd.chan3_node_id = 2; // Matching ESCs 2,12,22,32,42,52
        cmd.chan3_thr_cmd = 1000; // Throttle instruction

        cmd.chan4_enable = 1; // Put into effect
        cmd.chan4_node_id = 3; // Matching ESCs 3,13,23,33,43,53
        cmd.chan4_thr_cmd = 1000; // Throttle instruction
```

A structure of size uint64_t is defined, i.e. 8 bytes, the effective byte is 7.

3.5 Throttle Control (over 8 axes)

The message ID for the throttle command is 1130.

The structure is defined as follows:

```
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;
```

The structure name is CAN_SSC_CMD, which contains control information for the ESC used (8 in the example program above). Each ESC control information is a variable of type uint16ut, with the lower 10 bits representing throttle commands, ranging from 0 to 1000, corresponding to throttle sizes of 0% to 100%, and the upper 6 bits representing ESC node IDs, ranging from 1 to 63.

If 16 axes need to be controlled, send 2 packets. If 32 axes need to be controlled, send 4 packets. Note that the node_id of ESC should be kept unique.

3.6 Digital Signature

For the DRONECAN protocol, a digital signature is required. A digital signature is defined as follows:

Command Description	Transmission Direction	Message-type-ID	Digital Signatures
Throttle Control	FC → ESC	1160	0x5362ab78cd12f03aULL
Throttle Control (over 8 axes)	FC → ESC	1130	0x5362ab78cd12f03aULL
Status 1 reporting	ESC → FC	1150	0x2362ab78cd12f03aULL
Status 2 reporting	ESC → FC	1151	0x2362ab78cd12f03aULL
Status 3 reporting	ESC → FC	1152	0x2362ab78cd12f03aULL
Status 4 reporting	ESC → FC	1153	0x2362ab78cd12f03aULL
Status 5 reporting	ESC → FC	1154	0x2362ab78cd12f03aULL
Generic instruction	ESC → FC	999	0x1362ab78cd12f03aULL
	FC → ESC	1000	

4. CUBECAN Protocol Interaction

This protocol is our customized protocol based on normal CAN extension frames. It is simple to use for users. Currently CANID is used in the range of 0x10000000~0x10000146. all CANIDs are defined as follows:

```
#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
//ESC 1 to ESC 62 report Status 1 with increasing CAN ID values, which are omitted here.
#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
//ESC 1 to ESC 62 report Status 2 with increasing CAN ID values, which are omitted here.
#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
//ESC 1 to ESC 62 report Status 3 with increasing CAN ID values, which are omitted here.
#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
//ESC 1 to ESC 62 report Status 4 with increasing CAN ID values, which are omitted here.
#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
//The CAN ID value is incremented in response to the operation of ESC 1 to ESC 62 and is omitted here.
#define CUBECAN_ESC63_ACK_ID ((uint32_t)0x10000146)//esc -> fc, CUBECAN_OPE_ACK
```

4.1 Throttle Control

The ESC has a power up 0 throttle self-test logic. First send 3 seconds of 0 throttle continuously to get the ESC through the self-test.

4.1.1 CAN_ID Definition

CAN_ID Name	**CAN_ID Description**	**Load Structure Definition**	**Load length**	**CAN_ID value**	**Transmission Direction**
CUBECAN_ESC_CMD_ID	Sending ESC throttle commands	CUBECAN_ESC_CMD	8	0x10000000	FC → ESC

4.1.2 Load Structure Definition

```
#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];//Up to 4 ESCs can be controlled in a single frame.
} CUBECAN_ESC_CMD;
```

(1) One CAN frame can send 4 groups of ESC commands.

(2) There is no order requirement for the 4 groups of commands.

(3) The ESC IDs of the 4 groups of instructions cannot be repeated.

(4) The range of Node Id is 1 to 63.

(5) The range of instruction is 0 to 1000, and the corresponding throttle is 0%-100%.

(6) Each group of commands can be set independently.

(7) If a channel is not in use, it can be set to 0xffff, which means that the channel is not functional.

4.1.3 Sample Code

```
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);

    /*** Command Assignment ***/
    //  Group 0: Send command to ESC 1.
    esc_cmd.esc[0].bit.node_id = 1;
    esc_cmd.esc[0].bit.cmd = 100;

    //  Group 1: Send command to ESC 0.
    esc_cmd.esc[1].bit.node_id = 0;
    esc_cmd.esc[1].bit.cmd = 50;

    //  Group 2: Not used
    esc_cmd.esc[2].all = ESCx_UNUSED;

    //  Group 3: Send command to ESC 63.
    esc_cmd.esc[3].bit.node_id = 63;
    esc_cmd.esc[3].bit.cmd = 150;

    /*** Calling the underlying CAN transmitter function ***/

    if (cmd_len != 8)
    {
        return;
    }

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

One CAN frame can control up to 4 ESCs. Therefore, for a 4-axis aircraft, only one CAN frame needs to be sent in a control cycle. For a 6-axis or 8-axis aircraft, 2 CAN frames need to be sent in one control cycle.

4.2 LED Control

Whether the ESC's LEDs are controlled through the CAN or not needs to be set via the client. The default state of LED is CAN control. If the upper computer sets the light not to be controlled by CAN, the flight control cannot change the state of LED by sending CAN commands.

4.2.1 CAN_ID Definition

CAN_ID Name	CAN_ID Description	Load Structure Definition	Load length	CAN_ID value	Transmission Direction
CUBECAN_ESC_LED_ID	Control the LED status of the ESC	CUBECAN_ESC_LED	8	0x100000C1	FC → ESC

4.2.2 Load Structure Definition

```
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];// A single frame can control up to 4 sets of electronic control.
} CUBECAN_ESC_LED;
```

4.2.3 Definition of LED Enumeration

```
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 stands for 3 colors of lights, default is A - Red, B - Green, C - White.

Please refer to the hardware model specification for details, which is bound to the actual hardware model.

4.2.4 Sample code

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

    memset(&led, 0, len);

    /*** Instruction assignment ***/

    // Group 0: Send instructions to Electric Dispatch No.1.
    led.esc[0].bit.node_id = 1;
    led.esc[0].bit.led = CUBECAN_LED_STATE_A_ON;

    // Group 1: Send instructions to the No. 0 electric control center.
    led.esc[1].bit.node_id = 0;
    led.esc[1].bit.led = CUBECAN_LED_STATE_B_ON;

    // Group 2: Not used
    led.esc[2].all = ESCx_UNUSED;

    // Group 3: Send instructions to the 63rd ESC.
    led.esc[3].bit.node_id = 63;
    led.esc[3].bit.led = CUBECAN_LED_STATE_C_ON;

    /*** Call the CAN underlying sending function ***/

    if (len != 8)
    {
        return;
    }

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

4.3 Enable Reporting

4.3.1 CAN_ID Definition

CAN_ID Name	CAN_ID Description	Load Structure Definition	Load length	CAN_ID value	Transmission Direction
CUBECAN_ESC_ENA_ID	Enable ESC status reporting	CUBECAN_ESC_ENA	8	0x100000C2	FC → ESC

4.3.2 Load Structure Definition

```
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];// A single frame can control up to 4 sets of electronic control.
} CUBECAN_ESC_ENA;
```

4.3.3 Sample code

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

    memset(&ena, 0, len);

    /*** Instruction assignment ***/
    // Group 0: Send instructions to Electric Dispatch No.1.
    ena.esc[0].bit.node_id = 1;
    ena.esc[0].bit.ena = 1;

    // Group 1: Send instructions to the No. 0 electric control center.
    ena.esc[1].bit.node_id = 0;
    ena.esc[1].bit.ena = 1;

    // Group 2: Not used.
    ena.esc[2].all = ESCx_UNUSED;

    // Group 3: Send instructions to the 63rd ESC.
    ena.esc[3].bit.node_id = 63;
    ena.esc[3].bit.ena = 1;

    /*** Call the CAN underlying sending function ***/

    if (len != 8)
    {
        return;
    }

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

4.4 Status Reporting

1. ESC power-up is not reported by default. To report ESC status, the flight control needs to send an enable command (please refer to section 4.3).

2. When the Enable Reporting command is sent to turn on the specified ESC to report, the specified ESC will report all the time, and the reporting frequency is 10HZ.

3. When the Enable Reporting command is sent to turn off reporting for the specified ESC, the specified ESC will stop reporting.

4. Each ESC can report 4 types of status information, STAT1/ STAT2/ STAT3/ STAT4.

4.4.1 CAN ID Definition

Status 1 is reported:

CAN_ID Name	CAN_ID Description	Load Structure Definition	Load length	CAN_ID value	Transmission Direction
CUBECAN_ESC0_STAT1_	ESC 0 reporting	CUBECAN_ESC_	8	0x10000001	ESC → FC

ID	status 1	STAT1			
ESC 1 to ESC 62 report status 1 with incremental CAN_ID values, which are omitted here.					
CUBECAN_ESC63_STAT1_ID	ESC 63 reporting status 1	CUBECAN_ESC_STAT1	8	0x10000040	ESC → FC

Status 2 is reported:

CAN_ID Name	**CAN_ID Description**	**Load Structure Definition**	**Load length**	**CAN_ID value**	**Transmission Direction**
CUBECAN_ESC0_STAT2_ID	ESC 0 reporting status 2	CUBECAN_ESC_STAT2	8	0x10000041	ESC → FC
ESC 1 to ESC 62 report status 2 with incremental CAN_ID values, which are omitted here.					
CUBECAN_ESC63_STAT2_ID	ESC 63 reporting status 2	CUBECAN_ESC_STAT2	8	0x10000080	ESC → FC

Status 3 is reported:

CAN_ID Name	**CAN_ID Description**	**Load Structure Definition**	**Load length**	**CAN_ID value**	**Transmission Direction**
CUBECAN_ESC0_STAT3_ID	ESC 0 reporting status 3	CUBECAN_ESC_STAT3	8	0x10000081	ESC → FC
ESC 1 to ESC 62 report status 3 with incremental CAN_ID values, which are omitted here.					
CUBECAN_ESC63_STAT3_ID	ESC 63 reporting status 3	CUBECAN_ESC_STAT3	8	0x100000C0	ESC → FC

Status 4 is reported:

CAN_ID Name	**CAN_ID Description**	**Load Structure Definition**	**Load length**	**CAN_ID value**	**Transmission Direction**
CUBECAN_ESC0_STAT4_ID	ESC 0 reporting status 4	CUBECAN_ESC_STAT4	8	0x100000C4	ESC → FC
ESC 1 to ESC 62 report status 4 with incremental CAN_ID values, which are omitted here.					
CUBECAN_ESC63_STAT4_ID	ESC 63 reporting status 4	CUBECAN_ESC_STAT4	8	0x10000103	ESC → FC

4.4.2 Load Structure Definition

```
//order: bit0 -> bit15
struct   ESC_MODE_BITS      // bits   description
{
    uint16_t esc_mode: 8;// ESC control mode
    uint16_t pwm_thr_online: 1;// 0 for PWM throttle lost, 1 for PWM throttle online.
    uint16_t can_thr_online: 1;// 0 for CAN throttle lost, 1 for CAN throttle online.
    uint16_t thr_pri: 1;// 0 for PWM throttle priority, 1 for CAN throttle priority.
    uint16_t resv: 5;// reserved bit.
};
```

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

typedef struct
{
        union ESC_MODE_UN esc_mode;// ESC control modes and associated status flags.
        int16_t esc_cmd;    // Reported throttle commands, same as input commands.
        int16_t spd_rpm;    // Motor speed in rpm.
        int16_t mos_temp; // mos tube temperature in 0.1 degrees Celsius.
} CUBECAN_ESC_STAT1;

typedef struct
{
        int16_t      vdc_100mv;              // Busbar voltage in 0.1V.
        int16_t      irms_100ma;             // RMS phase current, unit 0.1A.
        int16_t      idq_100ma[2];        // d-q-axis current in 0.1A.
} CUBECAN_ESC_STAT2;

typedef struct
{
        int16_t alg_err;          // Algorithm error word. 0 means no error, ! =0 means there is an error.
        int16_t alg_warn;    // Algorithm warning word. 0 means no warning, ! =0 means with warning.
        int16_t vdq_duty[2];      // d-q axis commands.
} CUBECAN_ESC_STAT3;

typedef struct
{
    int16_t idc;// Estimated busbar current in 0.1A.
    int16_t cap_temp; // Capacitor temperature in 0.1 degrees Celsius.
    int16_t motor_temp;// Motor temperature in 0.1 degrees Celsius.
    int16_t resv;// Reserved byte.
} CUBECAN_ESC_STAT4;
```

The flight control receives the reference code:

```
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];// Corresponds to the state of ESC 0 to ESC 63.

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 Enquiry Report

For some flight control applications that don't need ESC to report all the time, this protocol is added: when the flight control sends a query command once, ESC replies once with the status frames STAT1/STAT2/STAT3/STAT4, for the definition of the status frames, please refer to section 4.4.

4.5.1 CAN_ID Definition

CAN_ID Name	CAN_ID Description	Load Structure Definition	Load length	CAN_ID value	Transmission Direction
CUBECAN_ESC_QUERY_STATE_ID	Query ESC Status	CUBECAN_QUERY_STATE	8	0x10000104	FC → ESC

The length of the load structure is 8, internally it is a uint64_t type, bit0 to bit63 represent ESC 0 to ESC 63. when bit is 0, it represents not querying the current ESC status, ESC will not reply the status frame; when bit is 1, it represents querying the current ESC status, ESC will reply the status frame.

For example, if query_flg=0xFFFFFFFFFFFFFFFFFFFF, all ESCs are queried for their status and all ESCs will reply with a status frame; if query_flg=0x1, only the status of ESC 0 is queried and only ESC 0 will reply with a status frame; if query_flg=0x8000000000000000, only ESC query the status of ESC 63, and only ESC 63 will reply with a status frame.

4.5.2 Load Structure Definition

```
typedef struct
{
    uint64_t query_flg;// bit0~bit63 represents ESC0~ESC63: 0 represents no query state, 1 represents query state
} CUBECAN_QUERY_STATE;
```

4.6 Read/Write Parameters

The read/write parameter protocol is to facilitate the flight control to directly set various parameters of ESC, such as NODE_ID, motor direction, throttle priority, etc.. You can also use our cloud box device to set parameters for ESC.

The CAN ID for the flight control read/write parameters is CUBECAN_OPE_ID.

The ESC reply CAN IDs for numbers 0 to 63 correspond to CUBECAN_ESC0_ACK_ID to CUBECAN_ESC63_ACK_ID. the CS in the load structure, which stands for the specific command operator.

The protocol supports both batch reading and writing of ESC parameters and reading and writing of individual ESC parameters. When reading or writing ESC parameters in bulk, all ESCs on the CAN bus will reply; when reading or writing a single ESC parameter, only the specified ESC will reply, while all other ESCs will not reply.

After writing parameters successfully, it will take effect after ESC power off and restart. Currently there are 6 kinds of open parameter items, they are: ESC NODE_ID, Motor Direction, Throttle Priority, ESC Light Power-Up Default Status, Fixed Pitch Stop Angle, and Fixed Pitch Switch.

The following section explains the use of load structures in detail and gives examples for reference. If you need to open more parameters, please contact our technical support or sales!

4.6.1 CAN_ID Definition

CAN_ID Name	CAN_ID Description	Load Structure Definition	Load length	CAN_ID value	Transmission Direction
CUBECAN_OPE_ID	FC Operating Commands	CUBECAN_BATCH_OPE	8	0x10000106	ESC → FC
CUBECAN_ESC0_ACK_ID	ESC 0 operational response	CUBECAN_OPE_ACK	8	0x10000107	ESC → FC
Operational responses from ESC 1 to ESC 62 with incremental CAN ID values are omitted here.					
CUBECAN_ESC63_ACK_ID	ESC 63 operational response	CUBECAN_OPE_ACK	8	0x10000146	ESC → FC

4.6.2 Load Structure Definition

```
// Flight control transmitter
typedef struct
{
        uint16_t cs;// Command operator indicating a request to read or write a parameter.
        int16_t data;// Indicates the data written during a write operation. During read operation, the value is meaningless.
        uint16_t batch;// 0->Single Read or Write,1->Batch Read or Write.
        uint16_t target_node_id;// When read or written individually, indicates operation of the specified ESC. When batch read or write, the value is meaningless.
} CUBECAN_BATCH_OPE;


// ESC response
```

```
typedef struct
{
        uint16_t cs;// Command operator that indicates a response to read or write a parameter.
        int16_t src_node_id;// Indicates the responding ESC NODE_ID.
        int16_t ret;// Indicates whether a read or write operation was successful: 0 means success, <0 means failure.
        int16_t data;// During a read operation, it indicates the data read. During write operation, the value is meaningless.
} CUBECAN_OPE_ACK;
```

4.6.3 CS Command Operator Definitions

```
typedef enum
{
        CS_BATCH_SET_START = 16,
        CS_BATCH_SET_ESC_CAN_ID_REQ = 16,// Write ESC NODE_ID parameter request.
        CS_BATCH_SET_ESC_CAN_ID_RES,// Write ESC NODE_ID parameter response.
        CS_BATCH_SET_MOTOR_DIR_REQ,// Write motor direction parameter request.
        CS_BATCH_SET_MOTOR_DIR_RES,// Write motor direction parameter reply.
        CS_BATCH_SET_THR_PRI_REQ,// Write throttle priority parameter request.
        CS_BATCH_SET_THR_PRI_RES,// Write throttle priority parameter response.
        CS_BATCH_SET_LED_STA_REQ,// Write ESC lamp power-up default state parameter request.
        CS_BATCH_SET_LED_STA_RES,// Write ESC lamp power-up default state parameter response.
        CS_BATCH_SET_ANG_REQ,// Write paddle stop parameter request.
        CS_BATCH_SET_ANG_RES,// Write paddle stop parameter response.
        CS_BATCH_SET_LOCK_REQ,//write pitch switch parameter request.
        CS_BATCH_SET_LOCK_RES,//write pitch switch parameter reply.
        CS_BATCH_SET_END,

        CS_BATCH_GET_START = 256,
        CS_BATCH_GET_ESC_CAN_ID_REQ = 256,// Read ESC NODE_ID parameter request.
        CS_BATCH_GET_ESC_CAN_ID_RES,// Read ESC NODE_ID parameter reply.
        CS_BATCH_GET_MOTOR_DIR_REQ,// Read motor direction parameter request.
        CS_BATCH_GET_MOTOR_DIR_RES,// Read motor direction parameter response.
```

```
    CS_BATCH_GET_THR_PRI_REQ,// Read throttle priority parameter request.

    CS_BATCH_GET_THR_PRI_RES,// Read throttle priority parameter reply.

    CS_BATCH_GET_LED_STA_REQ,// Read ESC light parameter request.

    CS_BATCH_GET_LED_STA_RES,// Read ESC light parameter response.

    CS_BATCH_GET_ANG_REQ,// Request to read paddle stop parameters.

    CS_BATCH_GET_ANG_RES,// Read paddle stop parameter response.

    CS_BATCH_GET_LOCK_REQ,//read pitch switch parameter request.

    CS_BATCH_GET_LOCK_RES,//read pitch switch parameter reply.

    CS_BATCH_GET_END,

} CUBECAN _CS;
```

4.6.4 CS Parameter Value Legality Provisions

Parameter Name	Range of values
ESC NODE_ID	1 to 63.
Motor Direction	-1 or 1. -1 for reverse, 1 for forward.
Throttle Priority	0 or 1. 0 means PWM throttle priority, 1 means CAN throttle priority.
Arm Light Status	0 to 13. See section 4.2.3 for details.
Paddle Stop Angle	-900 to 900. represents -90 degrees to 90 degrees.
Paddle Stop Switch	0 or 1. 0 is off, 1 is on.

4.6.5 Sample code

(1) When writing the NODE_ID of a single ESC, the CAN ID and load structure sent by the flight control is as follows:

```
CAN ID：CUBECAN_OPE_ID= 0x10000106

Load structure：CUBECAN_BATCH_OPE
```

Assuming that the NODE_ID of ESC #1 is set to 10, the load structure is assigned as follows:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_SET_ESC_CAN_ID_REQ;// Write ESC NODE_ID parameter request.

batch_ope.data = 10;// Setting the ESC NODE_ID to 10.

batch_ope.batch = 0;// 0 means set the specified ESC.

batch_ope.target_node_id = 1;// Specify the NODE_ID of the ESC.
```

Only ESC #1 will reply to the flight control, sending CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. Load structure for ESC reply: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_SET_ESC_CAN_ID_RES;// Write ESC NODE_ID parameter response.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 indicates successful write.

//ack.data， for write parameter replies, this value is meaningless.
```

CAN messages:

	Frame Type	Frame format	Frame ID	Data Length	Data
FC sends frame	Extended Frame	Data frame	10000106	8	10 00 0A 00 00 00 01 00
ESC reply frame	Expanded frame	Data frame	10000108	8	11 00 01 00 00 00 00 00

(2) When batch writing the ESC NODE_ID, the CAN ID and load structure sent by the flight control is as follows:

```
CAN ID：CUBECAN_OPE_ID= 0x10000106

Load Structure：CUBECAN_BATCH_OPE
```

Assuming that the NODE_ID of all ESCs on the CAN bus is set to 10, the load structure is assigned as follows:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_SET_ESC_CAN_ID_REQ;// Write ESC NODE_ID parameter request.

batch_ope.data = 10;// Set the NODE_ID for all ESCs on the CAN bus to 10.

batch_ope.batch = 1;// 1 means set all ESCs.

//batch_ope.target_node_id    // This value is meaningless when set in batch.
```

Suppose there are 2 ESCs on the CAN bus, ESC #1 and ESC #2, both replying to the flight control.

ESC #1 sends CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. ESC replies with load structure: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_SET_ESC_CAN_ID_RES;// Write ESC NODE_ID parameter response.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 indicates successful write.
```

```
//ack.data    // For write parameter replies, this value is meaningless.
```

ESC #2 sends CAN ID: CUBECAN_ESC2_ACK_ID= 0x10000109. ESC replies with load structure: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_SET_ESC_CAN_ID_RES;// Write ESC NODE_ID parameter response.

ack. src_node_id = 2;// The NODE_ID of the responding ESC is 2.

ack. ret = 0;// 0 indicates successful write.

//ack.data    // For write parameter replies, this value is meaningless.
```

CAN messages:

	Frame Type	Frame format	Frame ID	Data Length	Data
FC sends frame	Extended Frame	Data frame	10000106	8	10 00 0A 00 01 00 01 00
ESC reply frame	Expanded frame	Data frame	10000109	8	11 00 02 00 00 00 00 00
ESC reply frame	Expanded frame	Data frame	10000108	8	11 00 01 00 00 00 00 00

(3) Read the NODE_ID of a single ESC. The flight control sends the CAN ID: CUBECAN_OPE_ID= 0x10000106. The flight control sends the load structure: CUBECAN_BATCH_OPE.

Assuming that the NODE_ID of ESC #1 is read, the load structure is assigned as follows:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_GET_ESC_CAN_ID_REQ;// Read ESC NODE_ID parameter request.

//batch_ope.data // The value is meaningless when reading the parameter.

batch_ope.batch = 0;// 0 means read the specified ESC.

batch_ope.target_node_id = 1;// Specify the NODE_ID of the ESC.
```

Only ESC #1 will reply to the flight control, sending CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. Load structure for ESC reply: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_GET_ESC_CAN_ID_RES;// Read ESC NODE_ID parameter reply.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 means read successfully.

ack.data=1;// The NODE_ID of the ESC read is 1.
```

CAN messages:

	Frame Type	Frame format	Frame ID	Data Length	Data
FC sends frame	Extended Frame	Data frame	10000106	8	00 01 00 00 00 00 01 00
ESC reply frame	Expanded frame	Data frame	10000108	8	01 01 01 00 00 00 01 00

(4) Batch read ESC NODE_ID. The flight control sends the CAN ID: CUBECAN_OPE_ID= 0x10000106. The flight control sends the load structure: CUBECAN_BATCH_OPE.

Assuming that the NODE_IDs of all ESCs on the CAN bus are read, the load structure is assigned as follows:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_GET_ESC_CAN_ID_REQ;// Read ESC NODE_ID parameter request

//batch_ope.data // The value is meaningless when reading the parameter.

batch_ope.batch = 1;// 1 means read all ESCs.

//batch_ope.target_node_id    // This value is meaningless for batch reads.
```

Suppose there are 2 ESCs on the CAN bus, ESC #1 and ESC #2, both replying to the flight control.

ESC #1 sends CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. ESC replies with load structure: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_GET_ESC_CAN_ID_RES;// Read ESC NODE_ID parameter reply.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 means read successfully.

ack.data=1;// The NODE_ID of the ESC read is 1.
```

ESC #2 sends CAN ID: CUBECAN_ESC2_ACK_ID= 0x10000109. ESC replies with load structure: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_GET_ESC_CAN_ID_RES;// Read ESC NODE_ID parameter reply.

ack. src_node_id = 2;// The NODE_ID of the responding ESC is 2.

ack. ret = 0;// 0 means read successfully.

ack.data=2;// The NODE_ID of the ESC is read as 2.
```

CAN messages:

	Frame Type	Frame format	Frame ID	Data Length	Data
FC sends frame	Extended Frame	Data frame	10000106	8	00 01 00 00 01 00 00 00
ESC reply frame	Expanded frame	Data frame	10000109	8	01 01 02 00 00 00 02 00
ESC reply frame	Expanded frame	Data frame	10000108	8	01 01 01 00 00 00 01 00

(5) Write to individual ESC motor directions. The flight control sends the CAN ID: CUBECAN_OPE_ID= 0x10000106. The flight control sends the load structure: CUBECAN_BATCH_OPE.

Assuming that the motor direction of ESC #1 is set to positive, the load structure body is assigned as follows:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_SET_MOTOR_DIR_REQ;// Write ESC motor direction parameter request.

batch_ope.data = 1;// 1 for positive motor direction.

batch_ope.batch = 0;// 0 means set the specified ESC.

batch_ope.target_node_id = 1;// Specify the NODE_ID of the ESC.
```

Only ESC #1 will reply to the flight control, sending CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. Load structure for ESC reply: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_SET_MOTOR_DIR_RES;// Write ESC Motor Direction Parameter Response.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 indicates successful write.

// ack.data, which is meaningless for write parameter replies.
```

CAN messages:

	Frame Type	Frame format	Frame ID	Data Length	Data
FC sends frame	Extended Frame	Data frame	10000106	8	12 00 01 00 00 00 01 00
ESC reply frame	Expanded frame	Data frame	10000108	8	13 00 01 00 00 00 00 00

(6) Batch Write ESC Motor Direction. The flight control sends the CAN ID: CUBECAN_OPE_ID= 0x10000106. The flight control sends the load structure: CUBECAN_BATCH_OPE.

Assuming that the motor direction of all ESCs on the CAN bus is set to positive, the load structure body is assigned as follows:

```
CUBECAN_BATCH_OPE batch_ope;

memset(&batch_ope, 0, sizeof(batch_ope));
```

```
batch_ope.cs = CS_BATCH_SET_MOTOR_DIR_REQ;// Write motor direction parameter request.

batch_ope.data = 1;// Setting the motor direction of all ESCs on the CAN bus to be positive.

batch_ope.batch = 1;// 1 means set all ESCs.

//batch_ope.target_node_id    // This value is meaningless when set in batch.
```

Suppose there are 2 ESCs on the CAN bus, ESC #1 and ESC #2, both replying to the flight control.

ESC #1 sends CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. ESC replies with load structure: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_SET_MOTOR_DIR_RES;// Write motor direction parameter reply.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 indicates successful write.

//ack.data    // For write parameter replies, this value is meaningless.
```

ESC #2 sends CAN ID: CUBECAN_ESC2_ACK_ID= 0x10000109. Load structure for ESC reply: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_SET_MOTOR_DIR_RES;// Write motor direction parameter reply.

ack. src_node_id = 2;// The NODE_ID of the responding ESC is 2.

ack. ret = 0;// 0 indicates successful write.

//ack.data    // For write parameter replies, this value is meaningless.
```

CAN messages:

	Frame Type	Frame format	Frame ID	Data Length	Data
FC sends frame	Extended Frame	Data frame	10000106	8	12 00 01 00 01 00 00 00
ESC reply frame	Expanded frame	Data frame	10000109	8	13 00 02 00 00 00 00 00
ESC reply frame	Expanded frame	Data frame	10000108	8	13 00 01 00 00 00 00 00

(7) Reading individual ESC motor directions. The flight control sends the CAN ID: CUBECAN_OPE_ID= 0x10000106. The flight control sends the load structure: CUBECAN_BATCH_OPE.

Assuming that the motor direction of ESC #1 is set to positive, the load structure is assigned the following value:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_GET_MOTOR_DIR_REQ;// Read ESC Motor Direction Parameter Requests.

//batch_ope.data // The value is meaningless when reading the parameter.

batch_ope.batch = 0;// 0 means read the specified ESC.

batch_ope.target_node_id = 1;// Specify the NODE_ID of the ESC.
```

Only ESC #1 will reply to the flight control, sending CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. Load structure for ESC reply: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_GET_MOTOR_DIR_RES;// Read ESC Motor Direction Parameter Requests.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 means read successfully.

ack.data=1;// The value of the parameter read is 1, which means the motor direction is positive.
```

CAN messages:

	Frame Type	**Frame format**	**Frame ID**	**Data Length**	**Data**
FC sends frame	Extended Frame	Data frame	10000106	8	02 01 00 00 00 00 01 00
ESC reply frame	Expanded frame	Data frame	10000108	8	03 01 01 00 00 00 01 00

(8) Batch reading of ESC motor direction. The flight control sends CAN ID: CUBECAN_OPE_ID= 0x10000106. The flight control sends the load structure: CUBECAN_BATCH_OPE.

Assuming that the motor direction of all ESCs on the CAN bus is set to positive, the load structure is assigned the following value:

```
CUBECAN_BATCH_OPE batch_ope;

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

batch_ope.cs = CS_BATCH_GET_MOTOR_DIR_REQ;// Read motor direction parameter request.

//batch_ope.data // The value is meaningless when reading the parameter.

batch_ope.batch = 1;// 1 means read all ESCs.

//batch_ope.target_node_id    // This value is meaningless for batch reads.
```

Suppose there are 2 ESCs on the CAN bus, ESC #1 and ESC #2, both replying to the flight control.

ESC #1 sends CAN ID: CUBECAN_ESC1_ACK_ID= 0x10000108. ESC replies with load structure: cubecan_ope_ack.

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_GET_MOTOR_DIR_RES;// Read motor direction parameter response.

ack. src_node_id = 1;// The NODE_ID of the responding ESC is 1.

ack. ret = 0;// 0 means read successfully.

ack.data=1;// The value of the parameter read is 1, which means the motor direction is positive.
```

ESC #2 sends CAN ID: CUBECAN_ESC2_ACK_ID= 0x10000109. Load Structures for ESC Response: CUBECAN_OPE_ACK。

```
CUBECAN_OPE_ACK ack;

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

ack.cs = CS_BATCH_GET_MOTOR_DIR_RES;// Read motor direction parameter response.

ack. src_node_id = 2;// The NODE_ID of the responding ESC is 2.

ack. ret = 0;// 0 means read successfully.

ack.data=1;// The value of the parameter read is 1, which means the motor direction is positive.
```

CAN messages:

	Frame Type	**Frame format**	**Frame ID**	**Data Length**	**Data**
FC sends frame	Extended Frame	Data frame	10000106	8	02 01 00 00 01 00 00 00
ESC reply frame	Expanded frame	Data frame	10000109	8	03 01 02 00 00 00 01 00
ESC reply frame	Expanded frame	Data frame	10000108	8	03 01 01 00 00 00 01 00

5 Revision History

Number	**Date of change**	**Versions**	**Descriptions**
1	2022-12-08	V1.0.0	Establish
2	2022-12-17	V1.0.1	Add enumeration for navigation light control
3	2023-05-16	V2.0.0	Add CUBECAN protocol
4	2024-03-25	V2.1.0	Add DRONECAN protocol
5	2024-04-23	V2.1.1	Add digital signature
6	2024-06-07	V2.1.2	Adding Message-ID definition
7	2024-08-08	V2.1.3	DRONECAN protocol status report (ESC→FC), using single frame transmission.
8	2024-08-19	V2.1.4	DRONECAN protocol commands are sent out (FC→ESC), using single frame transmission.
9	2025-04-29	V2.1.8	CUBECAN protocol supports reading and writing

			parameters
10	2025-06-21	V2.1.9	CUBECAN protocol support for reading and writing pitch switch parameters
11	2025-09-04	V2.2.0	Add new throttle control command, supporting over 8 axes.