Title	**FLIPSKY FTESC CAN Communication Protocol V1.4**	**Version**	V1.4	
		FT Firmware Version	V1.4	
		Issue Date	2025/04/23	
Number	FT-CM-CAN-20250423-04			
Product Model	**Flipsky FT Series ESC**	**Total 22 pages**	**Level**	**Public**

FLIPSKY FTESC CAN
Communication Protocol V1.4

Guangdong FLIPSKY Technology Co., Ltd.

Revision Control

Version	Revision Date	Revised by	Revision Summary
1.4	2024-11-04	Lu	
	2025-04-23	Lu	

Catalog

1 Summary

This document describes the CAN communication protocol between the Motor Control Unit (MCU) and the Electronic Control Unit (ECU).

In this document, MCU specifically refers to the FT series ESC developed by FLIPSKY and ECU specifically refers to the CAN communication device developed by the user.

This protocol is intended to assist customers to use the CAN communication port to access and control the FT series ESC.

2 Message Format Description

FT series ESC are all equipped with CAN ports, so users can use the CAN ports to communicate with the MCU when developing their own ECU.

2.1 Communication Parameters

Baud Rate : 250K、500K、750K、1000K
IDE: Extended ID
EID: 29 bits. Bit0~bit7 are used for CAN ID command, bit8~bit15 are used for ESC ID.
RTR: Data Frame
DLC: Determined by the actual data length
The specific frame formats as below:

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field	CRC	ACK	EOF
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	8 bytes	15 bits	2 bit	7bits
1	0	0	0	1	0	MCU ID	CAN COMMAND ID	0	0	length	data[0-8]			

2.2 Implementation Annotation

In the message and response frame protocol,

(1) Data of **uint16_t** , **int16_t**, **uint32_t** and **int32_t** involved are transferred in Big-Endian Mode, i.e., high order byte data transfers first(low address), low order byte data comes second(high address).

(2) Single byte data without special marks are all **uint8_t** data.

2.3 CRC

```
static const unsigned char aucCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40
};

static const unsigned char aucCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7,
0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E,
0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9,
0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,
0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,
0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D,
0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,
0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF,
0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,
0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,
0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB,
0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA,
0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97,
0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E,
0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89,
0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,
0x41, 0x81, 0x80, 0x40
};
```

```
uint16_t crc16(uint8_t *Buffer, uint32_t Length)
{
  uint8_t ucCRCHi = 0xFF;
  uint8_t ucCRCLo = 0xFF;
  int i = 0;
  while( Length-- )
  {
    i = ucCRCLo ^ *( Buffer++ );
    ucCRCLo = ( uint8_t )( ucCRCHi ^ aucCRCHi[i] );
    ucCRCHi = aucCRCLo[i];
  }
  return (uint16_t)((((uint16_t)ucCRCHi) << 8) | ucCRCLo);
}
```

2.4 CAN COMMAND

```
typedef enum {
    CAN_SET_CURRENT_GEAR_CRC16              = 0, // Current control command(gear, CRC16 checksum)
    CAN_SET_CURRENT_BRAKE_CRC16             = 1, // Brake current control command（CRC16 checksum）
    CAN_SET_DUTY_GEAR_CRC16                 = 2, // Duty cycle control command(gear, CRC16 checksum)
    CAN_SET_CURRENT_PERCENT_CRC16           = 3, // Current percent control(CRC16 checksum)
    CAN_SET_CURRENT_PERCENT_GEAR_CRC16      = 4, // Current percent control(gear, CRC16 checksum)
    CAN_SET_CURRENT_BRAKE_PERCENT_CRC16     = 5, // Brake current percent control command（CRC16
                                                 //checksum）

    CAN_ESC_REALTIME_DATA_0_CRC16           = 11,// Return motor current and battery current(auto return
                                                 //every 20ms)
    CAN_ESC_REALTIME_DATA_1_CRC16           = 12,// Return motor erpm and duty
    CAN_ESC_REALTIME_DATA_2_CRC16           = 13,// Return MOSFET temperature, motor temperature and
                                                 //battery voltage
    ...
    CAN_SET_ERPM_GEAR_CRC16                 = 18,// Erpm control command with gear and CRC16
    CAN_SET_POSITION_GEAR_CRC16             = 19,// Position control command with gear and CRC16
    CAN_SET_ID_CURRENT_CRC16                = 20,// Motor D-axis current control command

    // Commands below without CRC16 checksum
    CAN_SET_CURRENT_GEAR                    = 21,// Current control command(gear)
    CAN_SET_CURRENT_PERCENT_GEAR            = 22,// Current percent control command(gear)
    CAN_SET_DUTY_GEAR                       = 23,// Duty cycle control command(gear)
    CAN_SET_ERPM_GEAR                       = 24,// Erpm control command(gear)
    CAN_SET_POSITION_GEAR                   = 25,// Position control command(gear)
    CAN_SET_CURRENT_BRAKE                   = 26,// Brake current control command
    CAN_SET_CURRENT_BRAKE_PERCENT           = 27,// Brake current percent control command
} CAN_COMMAND;
```

3 Command/Response Message Details

3.1 Command Analysis

Note: Please set the "Input Signal Type" to "Off" through the MCU before communication control when using CAN communication to control the FT ESC. Ensure the CAN baud rate parameter settings of the ECU and MCU are consistent.

3.1.1 CAN_SET_CURRENT_GEAR_CRC16 (00H)

1）Frame Format and Function Description

Send a motor current control command with gear setting and CRC16 checksum.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 00H	0	0	7	0	crc16_high	crc16_low	gear	motor current*100000			
															Int32_t			

2）Data Field Parameter Explanation

Item	Value	Explanation
D0 D1 D2 D3	Motor current*100000	The actual needed drive motor current value (unit: A) is amplified by 100,000 times. For example, if the needed current value is 50A, the actual sending data is 5,000,000.
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note:Please set “D4” to “00H” if the gear limit control is not required.
D5 D6	CRC 16 Checksum	For specific calculation methods, refer to 3.2.1

3.1.2 CAN_SET_CURRENT_BRAKE_CRC16 (01H)

1）Frame Format and Function Description

Send a brake current control command with CRC16 checksum.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 01H	0	0	6	0	0	crc16_high	crc16_low	Brake*100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Brake*100000	The actual needed drive braking current value (unit: A) is magnified 100,000 times.
D4 D5	CRC 16 Check Code	

3.1.3 CAN_SET_DUTY_GEAR_CRC16 (02H)

1）Frame Format and Function Description

Send a duty cycle control command with gear setting and CRC16.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 02H	0	0	6	0	crc16_high	crc16_low	gear	Duty*1000000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Duty*1000000	The actual needed drive duty cycle value is magnified 100,000 times. For example, if the duty cycle value is 0.2, the actual sending value is 0.2*1000000=200000.
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note:Please set “D4” to “00H” if the gear limit control is not required.
D5 D6	CRC 16	

3.1.4 CAN_SET_CURRENT_PERCENT_CRC16 (03H)

1）Frame Format and Function Description

Send a current percentage control command with CRC16.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 03H	0	0	6	0	0	crc16_high	crc16_low	Current Percent *100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Current Percent *100000	The actual drive current percentage value needs to be magnified by 100000 times. For example, if the value is 0.2, the actual sending value is 0.2*100000=20000, and the actual motor running current is 0.2*Motor Current. Note: The Motor Current parameter value can be set by MCU.
D4 D5	CRC 16	

3.1.5 CAN_SET_CURRENT_PERCENT_GEAR_CRC16 (04H)

1）Frame Format and Function Description

Send a current percentage control instruction with CRC16.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field								
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	0	1	0	MCU ID	CMD: 04H	0	0	7	0	crc16_high	crc16_low	gear	Current Percent *100000				
															Int32_t				

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Current Percent*100000	The actual drive current percentage value needs to be magnified by 100000 times. For example, if the value is 0.2, the actual value sent is 0.2*100000=20000, and the actual running current of the motor is 0.2*Motor Current. Note: Motor Current value can be set by MCU.
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note: Please set “D4” to “00H” if the gear limit control is not required.
D5 D6	CRC 16	

3.1.6 CAN_SET_CURRENT_BRAKE_PERCENT_CRC16 (05H)

1）Frame Format and Function Description

Send a brake current percentage control command with CRC16.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 05H	0	0	6	0	0	crc16_high	crc16_low	Brake Current Percent *100000			
															Int32_t			

2）Data Field Parameter Explanation

Item	Value	Explanation
D0 D1 D2 D3	Brake Current Percent*100000	The actual needed drive braking current percentage value needs to be magnified by 100000 times. For example, if the value is 0.2, the actual sending value is 0.2*100000=20000, and the actual motor running braking current is 0.2*Motor Current Brake. Note: Motor Current Brake value can be set by MCU.
D4 D5	CRC 16	

3.1.7 CAN_ESC_REALTIME_DATA_0_CRC16 (0BH)

1） Frame Format and Function Description

The MCU issues a real-time data_0 information frame command, and the returned data includes the motor current, battery current, and a CRC16.

Note:

①The command is automatically sent to the CAN bus by the MCU every 10ms. The user can obtain real-time data _0 information by parsing the command;

② Users can identify every real-time data of ESC through the MCU ID parameter.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 0BH	0	0	8	crc16_high	crc16_low	battery current*1000			motor current*1000		
													Int32_t			Int32_t		

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2	Motor Current Percent*1000	The reading data is the motor current value amplified by 1000 times, it needs to be divided by 1000 to get the actual motor current value.
D3 D4 D5	Motor Current Percent*1000	The reading data is the battery current value amplified by 1000 times, it needs to be divided by 1000 to get the actual battery current value.
D6 D7	CRC 16	

3.1.8 CAN_ESC_REALTIME_DATA_1_CRC16 (0CH)

1） Frame Format and Function Description

The MCU issues a real-time data_1 information frame command, and the returned data includes the motor erpm, duty cycle, and CRC16.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 0CH	0	0	8	crc16_high	crc16_low	duty*1000			motor erpm		
													Int32_t			Int32_t		

Note：

①The command is automatically sent to the CAN bus by the MCU every 10ms. The user can obtain real-time data_1 information by parsing the command;

②Users can identify every real-time data of ESC through the MCU ID parameter.

2）Data Field Parameter Explanation

Item	Value	Explanation
D0 D1 D2	Motor erpm	The motor erpm(electronic revolutions per minute) value, i.e., the motor mechanical rpm* the number of motor magnetic pole pairs
D3 D4 D5	Duty*1000	Th duty cycle value is magnified by 1000 times. The value divided by 1000 is the actual duty cycle value.
D6 D7	CRC 16	

3.1.9 CAN_ESC_REALTIME_DATA_2_CRC16 (0DH)

1） Frame format and function description

The MCU sends a real-time data_2 information frame command, and the returned data includes the MOSFET temperature, motor temperature and battery voltage, and CRC16 check code.

Note：

①The command is automatically sent to the CAN bus by the MCU every 10ms. The user can obtain data_2 information in real time by parsing the command;

②Users can identify every real-time data of ESC through the MCU ID parameter.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 0DH	0	0	8	crc16_high	crc16_low	Battery voltage*100		motor temp*100		mosfet temp*100	
													Int16_t		Int16_t		Int16_t	

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1	MOSFET temp*100	MOSFET temperature, magnified by 100 times.
D2 D3	Motor temp*100	Motor temperature, magnified by 100 times.
D4 D5	Battery voltage*100	Battery voltage, magnified by 100 times.
D6 D7	CRC 16	

3.1.10 CAN_SET_ERPM_GEAR_CRC16 (12H)

1） Frame format and function description

Send a speed control command with gear setting and CRC16 check.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 12H	0	0	6	0	crc16_high	crc16_low	gear	motor erpm			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Motor erpm	Motor erpm(electronic revolutions per minute), i.e., motor rpm*number of motor magnetic pole pairs
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note: Please set “D4” to “00H” if the gear limit control is not required.
D5 D6	CRC 16	

3.1.11 CAN_SET_POSITION_GEAR_CRC16 (13H)

1）Frame format and function description

Send a position control command with gear setting and CRC16 check.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 13H	0	0	6	0	crc16_high	crc16_low	gear	Position*100000			
															Int32_t			

2）Data Field Parameter Explanation

Item	Value	Explanation
D0 D1 D2 D3	Position*100000	Angle position control, -180°~180°, magnified by 100000 times. For example, if the angle needed is 45.6°, the actual sending value is 45.6*100000=4560000
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note:Please set “D4” to “00H” if the gear limit

		control is not required.
D5 D6	CRC 16	

3.1.12 CAN_SET_ID_CURRENT_CRC16 (14H)

1）Frame Format and Function Description

Send an inject current control command to the motor D-axis with gear setting and CRC16 check code. This command can be used in seamless mode or Hall mode. Injecting current to the motor D-axis can enable the motor to achieve the parking function, but it will consume current and the motor will get hot. It is not recommended to use it for a long time.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 14H	0	0	6	0	0	crc16_high	crc16_low	current*100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Current*100000	The current value of the D-axis of the motor that needs to be injected is amplified 100,000 times
D4 D5	CRC 16	Same as above

3.1.13 CAN_SET_CURRENT_GEAR (15H)

1）Frame format and function description

Send a motor current control command with gear setting.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 15H	0	0	7	0	0	0	gear	Current*100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Current*100000	The actual needed drive motor current value (unit: A) is magnified by 100,000 times. If the controlling current value is 50A, it needs to be magnified by 100000 times when actually sent, that is, 5,000,000.
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note: Please set “D4” to “00H” if the gear limit control is not required.

3.1.14 CAN_SET_CURRENT_PERCENT_GEAR (16H)

1）Frame Format and Function Description

Send a current percentage control command with gear setting.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 16H	0	0	6	0	crc16_high	crc16_low	gear	Current Percent *100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Current Percent*100000	The actual needed drive motor current percentage value. If the value is 0.2, the actual sending value is 0.2*100,000=20,000, and the actual motor current is 0.2*Motor Current. Note: The Motor Current parameter value can be set by MCU.
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note: Please set “D4” to “00H” if the gear limit control is not required.

3.1.15 CAN_SET_DUTY_GEAR (17H)

1） Frame Format and Function Description

Send a duty cycle control command with gear setting.

Note: Please send it at least every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 17H	0	0	6	0	0	0	gear	Duty*1000000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Duty*1000000	The duty cycle value actually needs to be amplified by 100,000 times. For example, if the duty cycle is 0.2, the actual value sent is 0.2*1000000=200000
D4 D5	CRC16	

3.1.16 CAN_SET_ERPM_GEAR (18H)

1）Frame Format and Function Description

Send a erpm control command with gear setting.

Note: Please send it at least once every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 18H	0	0	6	0	0	0	gear	motor erpm			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Motor erpm	Erpm=rpm* motor magnetic pole pairs
D4	Gear	00H: No gear 01H: Low speed gear 02H: Medium speed gear 03H: High speed gear 04H: Reverse gear Note: If gear speed limit control is not required, please set “D4” to “00H”;

3.1.17 CAN_SET_POSITION_GEAR (19H)

1）Frame Format and Function Description

Send a position control command with gear setting.

Note: Please send it at least once every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 19H	0	0	6	0	0	0	gear	Position*100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Position*100000	Angle position control, -180°~180°, magnified by 100,000 times. For example, if the angle needs to be controlled to 45.6°, the actual sending value is 45.6*100,000=4,560,000
D4	Gear	00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse Note: If gear speed limit control is not required, please set “D4” to “00H”;

3.1.18 CAN_SET_CURRENT_BRAKE (20H)

1）Frame Format and Function Description

Send a brake current control command.

Note: Please send it at least once every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 20H	0	0	6	0	0	0	0	Brake*100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Brake*100000	The actual needed drive brake current value (unit: A) is magnified 100,000 times.

3.1.19 CAN_SET_CURRENT_BRAKE_PERCENT (21H)

1）Frame Format and Function Description

Send a brake current percentage control command.

Note: Please send it at least once every 450ms, otherwise the command will be timeout and invalid.

SOF	Top 11 Bits of ID		SRR	IDE	Bottom 18 Bits of ID			RTR	Reserved	DLC	Data Field							
1 bit	bits 26-28	bits 18-25	1 bit	1 bit	bits 16-17	bits 8-15	bits 0-7	1 bit	2 bits	4bits	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	MCU ID	CMD: 21H	0	0	6	0	0	0	0	Brake Current Percent*100000			
															Int32_t			

2）Data Field Parameters Explanation

Item	Value	Explanation
D0 D1 D2 D3	Brake Current Percent*100000	The actual motor current percentage value needs to be amplified by 100,000 times. If the value is 0.2, the actual sending value is 0.2*100,000=20,000, and the actual motor current is 0.2*Motor Current. Note: The Motor Current parameter value can be set by MCU.

3.2 Command/Response Operation Example(C Language)

3.2.1 ECU Sends Control Commands

```
typedef enum {
     REMOTE_SPEED_NON = 0,
     REMOTE_SPEED_LOW,
     REMOTE_SPEED_MIDDLE,
     REMOTE_SPEED_HIGH,
     SMART_REVERSE
} Motor_Gear;

// Duty cycle control command
// ftescId: Controlled ESC ID
// duty: Duty cycle[-1.0, 1.0]
// gear: Speed gear, set as REMOTE_SPEED_NON if it is not required.
void canControlDutyGear(uint8_t ftescId, float duty, Motor_Gear gear) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_I32(can_signalframe_buff, (int32_t)(duty * 1000000.0f), &buff_index_t);
     can_signalframe_buff[buff_index_t++] = gear;
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_DUTY_GEAR_CRC16, can_signalframe_buff,
                                buff_index_t);
}

// Rpm control command
// ftescId: Controlled ESC ID
// erpm: Electronic rpm
// gear: Speed gear, set as REMOTE_SPEED_NON if it is not required.
void canControlErpmGear(uint8_t ftescId, float erpm, Motor_Gear gear) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_I32(can_signalframe_buff, (int32_t)(erpm), &buff_index_t);
     can_signalframe_buff[buff_index_t++] = gear;
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_ERPM_GEAR_CRC16, can_signalframe_buff,
                                buff_index_t);
}

// Position control command
// ftescId: Controlled ESC ID
// pos: Position,[-180°, 180°]
// gear: Speed gear, set as REMOTE_SPEED_NON if it is not required.
void canControlPositionGear(uint8_t ftescId, float pos, Motor_Gear gear) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
```

```
    valueDecompose_I32(can_signalframe_buff, (int32_t)(pos * 100000.0f), &buff_index_t);
    can_signalframe_buff[buff_index_t++] = gear;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_POSITION_GEAR_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// Current control command
// ftescId: Controlled ESC ID
// current: Motor current, unit:A
// gear: Speed gear, set as REMOTE_SPEED_NON if it is not required.
void canControlCurrentGear(uint8_t ftescId, float current, Motor_Gear gear) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(current * 100000.0f), &buff_index_t);
    can_signalframe_buff[buff_index_t++] = gear;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_GEAR_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// Motor D-axis current control command
// ftescId: Controlled ESC ID
// current: Motor D-axis current, unit:A
void canControlIdCurrent(uint8_t ftescId, float current) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(current * 100000.0f), &buff_index_t);
    can_signalframe_buff[buff_index_t++] = 0;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_ID_CURRENT_CRC16, can_signalframe_buff, buff_index_t);
}

// Motor brake current control command
// ftescId: Controlled ESC ID
// current: Motor brake current, unit: A
void canControlCurrentBrake(uint8_t ftescId, float current) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(current * 100000.0f), &buff_index_t);
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_BRAKE_CRC16, can_signalframe_buff,
                               buff_index_t);
```

```
}

// Motor current percentage control command
// ftescId: Controlled ESC ID
// current_percent: motor current percentage, [-1.0, 1.0]
void comm_can_set_current_rel(uint8_t ftescId, float current_percent) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_F32(can_signalframe_buff, current_percent, 1e5, &buff_index_t);
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_PERCENT_CRC16, can_signalframe_buff,
                                buff_index_t);
}

// Motor current percentage control command with gear setting
// ftescId: Controlled ESC ID
// current_percent: motor current percentage, [-1.0, 1.0]
// gear: Speed gear, set as REMOTE_SPEED_NON if it is not required.
void canControlCurrentPercentGear(uint8_t ftescId, float current_percent, Motor_Gear gear) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_F32(can_signalframe_buff, current_percent, 1e5, &buff_index_t);
     can_signalframe_buff[buff_index_t++] = gear;
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_PERCENT_GEAR_CRC16, can_signalframe_buff,
                                buff_index_t);
}

// Motor brake current percentage control command
// ftescId: Controlled ESC ID
// current_percent: Motor brake current percentage, [-1.0, 1.0]
void canControlCurrentBrakePercent(uint8_t ftescId, float current_percent) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_F32(can_signalframe_buff, current_percent, 1e5, &buff_index_t);
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_BRAKE_PERCENT_CRC16, can_signalframe_buff,
                                buff_index_t);
}
```

3.2.2 CAN Real-time Data Frame Processing(0BH、0CH、0DH)

```
#define ESC_NUM_MAX 4

typedef struct {
     uint8_t esc_id;
     float motorCurrent;
     float BatteryCurrent;
} can_esc_data_0;

typedef struct {
     uint8_t esc_id;
     float erpm;
     float duty;
} can_esc_data_1;

typedef struct {
     uint8_t esc_id;
     float mosfetTemp;
     float motorTemp;
     float battVoltage;
} can_esc_data_2;

can_esc_data_0 esc_data_0[ESC_NUM_MAX];
can_esc_data_1 esc_data_1[ESC_NUM_MAX];
can_esc_data_2 esc_data_2[ESC_NUM_MAX];


// extern_id:Extended frame ID
// buff:CAN data pack
// length:length of data pack
static void canCommIDHandle(uint32_t extern_id, uint8_t *buff, int length) {
     int32_t buff_index_t = 0;
     uint8_t crc16_low = 0;
     uint8_t crc16_high = 0;

     uint8_t esc_id = (extern_id >> 8);//Obtain ESC ID
     CAN_COMMAND can_comm_id = (CAN_COMMAND)(extern_id & 0xFF);//Obtain ESC command

     switch (can_comm_id) {
     case CAN_ESC_REALTIME_DATA_0_CRC16:
     {
          crc16_high = buff[6];
          crc16_low = buff[7];
          if (crc16(buff, 6) == ((unsigned short) crc16_high << 8 | (unsigned short) crc16_low))
          {
               for (int i = 0; i < ESC_NUMBER_MAX; i++) {
                    can_esc_data_0 *esc_data_status_0 = &esc_data_0[i];
                    //"-2" indicates that a new ESC has sent data over. This ID value will be saved after
                    //the first update.
                    if (esc_data_status_0->esc_id == esc_id ||
```

```
                    esc_data_status_0->esc_id == -2)
                {                       buff_index_t = 0;
                                    esc_data_status_0->esc_id = esc_id;
                    esc_data_status_0->motorCurrent =
                                     (float)valueCompose_I24(buff, &buff_index_t)*0.001f;
                    esc_data_status_0->BatteryCurrent =
                                    (float)valueCompose_I24(buff, &buff_index_t)*0.001f;
                    break;
                }
            }
        }
    }
    break;

    case CAN_ESC_REALTIME_DATA_1_CRC16:
    {
        crc16_high = buff[6];
        crc16_low = buff[7];
        if (crc16(buff, 6) == ((unsigned short) crc16_high << 8 | (unsigned short) crc16_low))
        {
            for (int i = 0; i < ESC_NUMBER_MAX; i++) {
                can_esc_data_1 *esc_data_status_1 = &esc_data_1[i];
                if (esc_data_status_1->esc_id == esc_id ||
                    esc_data_status_1->esc_id == -2) {
                    buff_index_t = 0;
                    esc_data_status_1->esc_id = esc_id;
                    esc_data_status_1->erpm =
                      (float)valueCompose_I24(buff, &buff_index_t);
                    esc_data_status_1->duty =
                      (float)valueCompose_I24(buff, &buff_index_t)*0.001f;
                    break;
                }
            }
        }
    }
    break;

    case CAN_ESC_REALTIME_DATA_2_CRC16:
    {
        crc16_high = buff[6];
        crc16_low = buff[7];
        if (crc16(buff, 6) == ((unsigned short) crc16_high << 8 | (unsigned short) crc16_low))
        {
            for (int i = 0; i < ESC_NUMBER_MAX; i++) {
                can_esc_data_2 *esc_data_status_2 = &esc_data_2[i];
                if (esc_data_status_2->esc_id == esc_id ||
                   esc_data_status_2->esc_id == -2) {
                    buff_index_t = 0;
                    esc_data_status_2->esc_id = esc_id;
                    esc_data_status_2->mosfetTemp =
                        (float)valueCompose_I24(buff, &buff_index_t)*0.01f;
```

```
                        esc_data_status_2->motorTemp =
                            (float)valueCompose_I24(buff, &buff_index_t)*0.01f;
                        esc_data_status_2->battVoltage =
                            (float)valueCompose_I24(buff, &buff_index_t)*0.01f;
                        break;
                    }
                }
            }
        }
        break;

        default:
            break;
        }
}

// Split the signed short int data(int16_t) into 2 bytes to transmit with the high-order data first
//and the low-order data second.
void valueDecompose_I16(uint8_t* dataNum, int16_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// Split the unsigned short int data(uint16_t) into 2 bytes to transmit with the high-order data
// first and the low-order data second.
void valueDecompose_U16(uint8_t* dataNum, uint16_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// Split the signed long int data(int32_t) into 4 bytes with the maximum 8 bits truncated and deleted
// Reserved the low-order 24 bits with the high-order digit first and  the low-order digit second
// range:-8388608~8388607
void valueDecompose_I24(uint8_t* dataNum, int32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// Split the unsigned long int data(uint32_t) into 4 bytes with the maximum 8 bits truncated and deleted
// Reserved the low-order 24 bits with the high-order digit first and the low-order digit second
// range:0~16777215
void valueDecompose_U24(uint8_t* dataNum, uint32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// Split the signed long int data(int32_t) into 4 bytes to transmit with the high-order digit first
// and the low-order digit second
void valueDecompose_I32(uint8_t* dataNum, int32_t value, int16_t *numInd) {
```

```
  dataNum[(*numInd)++] = value >> 24;
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// Split the unsigned long int data(uint32_t) into 4 bytes to transmit with the high-order digit
// first and the low-order digit second
void valueDecompose_U32(uint8_t* dataNum, uint32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 24;
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// Combine 2 bytes into signed short int data(int16_t) with the high-order digit first and the
// low-order digit second
int16_t valueCompose_I16(const uint8_t *dataNum, int16_t *numInd) {
  int16_t retval =  ((uint16_t) dataNum[*numInd]) << 8 |
          ((uint16_t) dataNum[*numInd + 1]);
  *numInd += 2;
  return retval;
}

// Combine 2 bytes into unsigned short int data(uint16_t) with the high-order digit first and the
// low-order digit second
uint16_t valueCompose_U16(const uint8_t *dataNum, int16_t *numInd) {
  uint16_t retval =   ((uint16_t) dataNum[*numInd]) << 8 |
          ((uint16_t) dataNum[*numInd + 1]);
  *numInd += 2;
  return retval;
}

// Combine 3 bytes into signed short int data(int32_t)
// With the high-order digit first and the low-order digit second
int32_t valueCompose_I32(const uint8_t *dataNum, int16_t *numInd) {
  int32_t retval = ((uint32_t) dataNum[*numInd]) << 16  |
          ((uint32_t) dataNum[*numInd + 1]) << 8 |
          ((uint32_t) dataNum[*numInd + 2]);
  *numInd += 3;
  return retval;
}

// Combine 3 bytes into unsigned short int data(uint32_t)
// With the high-order digit first and the low-order digit second

uint32_t valueCompose_U32(const uint8_t *dataNum, int16_t *numInd) {
  uint32_t retval = ((uint32_t) dataNum[*numInd]) << 16 |
          ((uint32_t) dataNum[*numInd + 1]) << 8 |
          ((uint32_t) dataNum[*numInd + 2]);
  *numInd += 3;
```

```
  return retval;
}

// Combine 4 bytes into unsigned long int data(uint32_t) with the high-order digit first and
// the low-order digit second
uint32_t valueCompose_U32(const uint8_t *dataNum, int16_t *numInd) {
  uint32_t retval = ((uint32_t) dataNum[*numInd]) << 24 |
          ((uint32_t) dataNum[*numInd + 1]) << 16 |
          ((uint32_t) dataNum[*numInd + 2]) << 8 |
          ((uint32_t) dataNum[*numInd + 3]);
  *numInd += 4;
  return retval;
}

// Combine 4 bytes into signed long int data(int32_t) with the high-order digit first and
// the low-order digit second
int32_t valueCompose_I32(const uint8_t *dataNum, int16_t *numInd) {
  int32_t retval =  ((uint32_t) dataNum[*numInd]) << 24 |
          ((uint32_t) dataNum[*numInd + 1]) << 16  |
          ((uint32_t) dataNum[*numInd + 2]) << 8 |
          ((uint32_t) dataNum[*numInd + 3]);
  *numInd += 4;
  return retval;
}

// Combine 4 bytes into unsigned long int data(uint32_t) with the high-order digit first and
// the low-order digit second
uint32_t valueCompose_U32(const uint8_t *dataNum, int16_t *numInd) {
  uint32_t retval = ((uint32_t) dataNum[*numInd]) << 24 |
          ((uint32_t) dataNum[*numInd + 1]) << 16 |
          ((uint32_t) dataNum[*numInd + 2]) << 8 |
          ((uint32_t) dataNum[*numInd + 3]);
  *numInd += 4;
  return retval;
}

// Convert float data into int16_t data by magnification factor and split it into two bytes for
// transmission, with the high-order digit first and the low-order digit second

void valueDecompose_F16(uint8_t* dataNum, float value, float multiple, int32_t *numInd) {
    valueDecompose_I16(dataNum, (int16_t)(value * multiple), numInd);
}

// Convert float data into int32_t data by magnification factor and split it into four bytes for
// transmission, with the high-order digit first and the low-order digit second
void valueDecompose_F32(uint8_t* dataNum, float value, float multiple, int32_t *numInd) {
    valueDecompose_I32(dataNum, (int32_t)(value * multiple), numInd);
}

// Combine two bytes into signed short int data (int16_t), with the high-order digit first
// and the low-order digit second, and convert them into float data by magnification factor
```

```
// float valueCompose_F16(const uint8_t *dataNum, float multiple, int16_t *numInd) {
    return (float)valueCompose_I16(dataNum, numInd) / multiple;
}

// Combine four bytes into signed short int data (int32_t), with the high-order digit first
// and the low-order digit second, and convert them into float data by magnification factor
float valueCompose_F32(const uint8_t *dataNum, float multiple, int16_t *numInd) {
    return (float)valueCompose_I32(dataNum, numInd) / multiple;
}
```