

Referee System Serial Port Protocol Appendix

Publisher: RoboMaster Organizing Committee (RMOC)

Version: V1.1

Date: 2019-03-08

Release Notes

Date	Version	Changes
2019.2.25	V1.0	Release
2019.3.8	V1.1	<ol style="list-style-type: none">1. Add the projectile number of the action identification data of the Battlefield Projectile Supplier2. Add the supply robot ID of the Projectile Supplier3. Fix the content ID of the interactive data between student robots4. Update byte offset description of Battlefield event data5. Update remarks of the client custom data

1. Serial Port Configuration

The communication interface is serial port, which is configured with 115200 baud rate, 8 data bits and 1 stop bit while there is no hardware flow control or parity bit.

2. Port Protocol Description

Communication protocol format:

frame_header (5-byte)	cmd_id (2-byte)	data (n-byte)	frame_tail (2-byte, CRC16, whole package check)
-----------------------	-----------------	---------------	---

Table 1 frame_header Format

SOF	data_length	seq	CRC8
1-byte	2-byte	1-byte	1-byte

Table 2 Frame Header Definition

Domain	Offset Position	Size (byte)	Description
SOF	0	1	Starting byte of data frame, whose fixed value is 0xA5
data_length	1	2	The length of data inside the data frame
seq	3	1	Package sequence number
CRC8	4	1	Frame header CRC8 Check

Table 3 cmd_id Command Code IDs Description

Command Code	Data Segment Length	Function Description
0x0001	3	Competition status data, whose transmitting cycle is 1Hz
0x0002	1	Competition result data, which is transmitted at the end of the competition
0x0003	2	Competition robots survival data, whose transmitting cycle is 1Hz

Command Code	Data Segment Length	Function Description
0x0101	4	Battlefield event data, which is transmitted after the event has happened
0x0102	3	Battlefield Projectile Supplier action identification data, which is transmitted after the action is changed
0x0103	2	Request projectile supply data from the Projectile Supplier. The data is then transmitted by the team and the upper limit is 10Hz. (RoboMaster Robotics Competition is not yet available)
0x0201	15	Robot status data, whose transmitting cycle is 10Hz
0x0202	14	Real-time power and barrel heat data, whose transmitting cycle is 50Hz
0x0203	16	Robot's position data, whose transmitting cycle is 10Hz
0x0204	1	Robot gain data, which is transmitted after the gain status is changed
0x0205	3	Aerial energy status data, whose transmitting cycle is 10Hz and it is only transmitted by the Aerial's main controller
0x0206	1	Damage status data, which is transmitted after the damage occurs
0x0207	6	Real-time shooting data, which is transmitted after the projectile is launched
0x0301	n	Interaction data between robots, which is triggered to transmit by the sender and the upper limit is 10Hz

Detailed Description

1. Competition status data: 0x0001. Transmission frequency: 1Hz

Byte Offset	Size	Description
0	1	0-3 bit: Competition Type <ul style="list-style-type: none"> • 1: RoboMaster Robotics Competition; • 2: RoboMaster Technical Challenge; • 3: RoboMaster ICRA 4-7 bit: Current Competition Stage <ul style="list-style-type: none"> • 0: Pre-match; • 1: Setup Period; • 2: Referee System Initialization Period; • 3: 5-second Countdown; • 4: Round Period; • 5: Calculation Period
1	2	Remaining time of the current period (unit: s)

```
typedef __packed struct
{
    uint8_t game_type : 4;
    uint8_t game_progress : 4;
    uint16_t stage_remain_time;
} ext_game_state_t;
```

2. Competition result data: 0x0002. Transmission frequency: send after the competition

Byte Offset	Size	Description
0	1	0: Draw; 1: Red win;

Byte Offset	Size	Description
		2: Blue win

```
typedef __packed struct
{
    uint8_t winner;
} ext_game_result_t;
```

3. Robot survival data: 0x0003. Transmission frequency: 1Hz

Byte Offset	Size	Description
0	2	bit 0: Red Hero; bit 1: Red Engineer; bit 2: Red Standard #1; bit 3: Red Standard #2; bit 4: Red Standard #3; bit 5: Red Aerial; bit 6: Red Sentry; bit 7: Reserved bit 8: Blue Hero; bit 9: Blue Engineer; bit 10: Blue Standard #1; bit 11: Blue Standard #2; bit 12: Blue Standard #3; bit 13: Blue Aerial; bit 14: Blue Sentry; bit 15: Reserved

Byte Offset	Size	Description
		A corresponding bit value of 1 indicates that the robot is alive, and a value of 0 indicates that the robot is defeated or has not entered the stage.

```
typedef __packed struct
{
    uint16_t robot_legion;
} ext_game_robot_survivors_t;
```

4. Battlefield event data: 0x0101. Transmission frequency: send after the event changes

Byte Offset	Size	Description
0	4	<p>bit 0-1: The occupation status of your Landing Pad</p> <ul style="list-style-type: none"> • 0 indicates no robot occupies; • 1 indicates that Aerial has occupied the Landing Pad but does not stop the propeller; • 2 indicates that Aerial has occupied the Landing Pad and stopped the propeller <p>bit 2: The occupation status of the #1 Restoration Zone of your Projectile Supplier and 1 indicates as occupied;</p> <p>bit 3: The occupation status of the #2 Restoration Zone of your Projectile Supplier and 1 indicates as occupied;</p> <p>bit 4: The occupation status of the #3 Restoration Zone of your Projectile Supplier and 1 indicates as occupied;</p> <p>bit 5-6: Your Large Power Rune status:</p> <ul style="list-style-type: none"> • 0 indicates that the striking point is not occupied and Large Power Rune is not activated; • 1 indicates that the striking point is occupied but Large Power Rune is not activated;

Byte Offset	Size	Description
		<ul style="list-style-type: none"> • 2 indicates that Large Power Rune is activated; • 3 indicates that Large Power Rune is activated and the striking point is occupied. <p>bit 7: The occupation status of your Bridge End Platform and 1 indicates as occupied;</p> <p>bit 8: The occupation status of your Bunker and 1 indicates as occupied;</p> <p>bit 9: The occupation status of your Resource Island and 1 indicates as occupied;</p> <p>bit 10-11: The defense status of your Base:</p> <ul style="list-style-type: none"> • 2 indicates that Base is 100% defense; • 1 indicates that Base is defended with Sentry; • 0 indicates that Base has no defense <p>bit 12-13: ICRA Red Team Defense Bonus</p> <ul style="list-style-type: none"> • 0 indicates that Defense Bonus has not been activated • 1 indicates that Defense Bonus 5s has been triggered and is activating • 2 indicates that Defense Bonus has been activated <p>bit 14-15: ICRA Blue Team Defense Bonus</p> <ul style="list-style-type: none"> • 0 indicates that Defense Bonus has not been activated • 1 indicates that Defense Bonus 5s has been triggered and is activating • 2 indicates that Defense Bonus has been activated <p>Other bits are reserved.</p>

```
typedef __packed struct
{
    uint32_t event_type;
} ext_event_data_t;
```

5. Projectile Supplier Zone action identification: 0x0102. Transmission frequency: send after the action changes

Byte Offset	Size	Description
0	1	Projectile Supplier outlet ID: 1: Projectile Supplier outlet #1; 2: Projectile Supplier outlet #2
1	1	Projectile Supply robot ID: 0 indicates that no robot supplies projectile; 1 indicates that Red Hero supplies; 2 Red Engineer; 3/4/5 Red Standard; 11 Blue Hero; 12 Blue Engineer; 13/14/15 Blue Standard
2	1	The open and close mode of Projectile outlet: 0 indicates close; 1 indicates preparing for projectiles, 2 indicates falling projectiles
3	1	Quantity of Projectile Supply: 50: 50 projectiles 100: 100 projectiles 150: 150 projectiles 200: 200 projectiles

```
typedef __packed struct
{
    uint8_t supply_projectile_id;
    uint8_t supply_robot_id;
    uint8_t supply_projectile_step;
} ext_supply_projectile_action_t;
```

6. Request Projectile Supplier to supply projectiles: cmd_id (0x0103). Transmission frequency: upper limit 10Hz. RoboMaster Robotics Competition is not yet available.

Byte Offset	Size	Description
0	1	Projectile Supplier outlet ID:

Byte Offset	Size	Description
		1: Projectile Supplier outlet #1
1	1	Projectile Supply robot ID: 1 indicates that Red Hero supplies; 2 Red Engineer; 3/4/5 Red Standard; 11 Blue Hero; 12 Blue Engineer; 13/14/15 Blue Standard
1	1	Quantity of Projectile Supply: 50: request 50 projectiles

```
typedef __packed struct
{
    uint8_t supply_projectile_id;
    uint8_t supply_robot_id;
    uint8_t supply_num;
} ext_supply_projectile_booking_t;
```

7. Match robot status: 0x0201. Transmission frequency: 10Hz

Byte Offset	Size	Description
0	1	Robot ID: 1: Red Hero; 2: Red Engineer; 3/4/5: Red Standard; 6: Red Aerial; 7: Red Sentry; 11: Blue Hero; 12: Blue Engineer; 13/14/15: Blue Standard; 16: Blue Aerial;

Byte Offset	Size	Description
		17: Blue Sentry
1	1	Robot level: 1: level one; 2: level two; 3: level three
2	2	Robot Remaining HP
4	2	Robot Maximum HP
6	2	17 mm barrel cooling value per second
8	2	17 mm barrel heat limit
10	2	42 mm barrel cooling value per second
12	2	42 mm barrel heat limit
14	1	Main controller power output status: 0 bit: gimbal port output: 1 indicates 24V output, 0 indicates no 24V output; 1 bit: chassis port output: 1 indicates 24V output, 0 indicates no 24V output; 2 bit: shooter port output: 1 indicates 24V output, 0 indicates no 24V output;

```
typedef __packed struct
{
    uint8_t robot_id;
    uint8_t robot_level;
    uint16_t remain_HP;
    uint16_t max_HP;
    uint16_t shooter_heat0_cooling_rate;
    uint16_t shooter_heat0_cooling_limit;
    uint16_t shooter_heat1_cooling_rate;
    uint16_t shooter_heat1_cooling_limit;
    uint8_t mains_power_gimbal_output : 1;
    uint8_t mains_power_chassis_output : 1;
};
```

```
uint8_t mains_power_shooter_output : 1;
} ext_game_robot_state_t;
```

8. Real-time power and barrel heat data: 0x0202. Transmission frequency: 50Hz

Byte Offset	Size	Description
0	2	Chassis output voltage (unit: mV)
2	2	Chassis output current (unit: mA)
4	4	Chassis output power (unit: W)
8	2	Chassis power buffer (unit: J)
10	2	17 mm barrel heat
12	2	42 mm barrel heat

```
typedef __packed struct
{
    uint16_t chassis_volt;
    uint16_t chassis_current;
    float chassis_power;
    uint16_t chassis_power_buffer;
    uint16_t shooter_heat0;
    uint16_t shooter_heat1;
} ext_power_heat_data_t;
```

9. Robot position: 0x0203. Transmission frequency: 10Hz

Byte Offset	Size	Description
0	4	Position x coordinate (unit: m)
4	4	Position y coordinate (unit: m)
8	4	Position z coordinate (unit: m)
12	4	Barrel position (unit: degree)

```
typedef __packed struct
{
    float x;
    float y;
    float z;
    float yaw;
} ext_game_robot_pos_t;
```

10. Robot gain: 0x0204. Transmission frequency: send after the buff status changes

Byte Offset	Size	Description
0	1	bit 0: robot HP restoration status bit 1: barrel heat cooling rate accelerates bit 2: robot defense bonus bit 3: robot attack bonus Other bits are reserved

```
typedef __packed struct
{
    uint8_t power_rune_buff;
} ext_buff_musk_t;
```

11. Aerial energy status: 0x0205. Transmission frequency: 10Hz

Byte Offset	Size	Description
0	1	Accumulated energy points
1	2	Attack time (unit: s). Drop to 0 in 50s

```
typedef __packed struct
{
    uint8_t energy_point;
    uint8_t attack_time;
} aerial_robot_energy_t;
```

12. Damage status: 0x0206. Transmission frequency: send after damage happens

Byte Offset	Size	Description
0	1	<p>bit 0-3: when the HP change type is armor damage, it indicates the armor ID and the value 0-4 represents the five armor modules of the robot. As for other HP change types, the variable value is 0.</p> <p>bit 4-7: HP Change Type</p> <p>0x0 HP deduction from armor damage;</p> <p>0x1 HP deduction from module offline;</p> <p>0x2 HP deduction from exceeding the barrel heat limit;</p> <p>0x3 HP deduction from exceeding the chassis power.</p>

```
typedef __packed struct
{
    uint8_t armor_id : 4;
    uint8_t hurt_type : 4;
} ext_robot_hurt_t;
```

13. Real-time shooting data: 0x0207. Transmission frequency: send after shooting

Byte Offset	Size	Description
0	1	<p>Projectile type:</p> <ul style="list-style-type: none"> • 1: 17 mm projectile • 2: 42 mm projectile
1	1	Projectile frequency of launch (unit: Hz)
2	4	Projectile speed of launch (unit: m/s)

```
typedef __packed struct
{
    uint8_t bullet_type;
    uint8_t bullet_freq;
    float bullet_speed;
} ext_shoot_data_t;
```

3. Interactive data between robots

The interactive data includes a unified data segment header structure. The data segment consists of the content ID, the sender and the receiver's ID and the content data segment. The total length of the entire interactive data packet is up to 128 bytes, with the subtraction of the 9 bytes of frame_header, cmd_id and frame_tail and the 6 bytes of the data segment header structure, thus the content data segment that is sent is 113 at most. The upload frequency of the entire interactive data package 0x0301 is 10Hz.

1. Interactive data receiving information: 0x0301. Transmission frequency: maximum 10Hz

Byte Offset	Size	Description	Remarks
0	2	Content ID of data segment	
2	2	Sender ID	Need to verify the correctness of the sender ID. For example, if Red 1 is sent to Red 5, this item needs to check Red 1.
4	2	Receiver ID	Need to verify the correctness of the receiver ID. For example, you cannot send to the enemy robot's ID.
6	x	Content data segment	x is 113 atmost

```
typedef __packed struct
{
    uint16_t data_cmd_id;
    uint16_t send_ID;
    uint16_t receiver_ID;
}ext_student_interactive_header_data_t;
```

Content ID	Length (head structure length + content data segment length)	Function Description
0xD180	6 + 13	Client custom data
0x0200~0x02FF	6+n	Communication between your robots

Since there are multiple content IDs while the entire cmd_id upload frequency is up to 10Hz, please arrange the bandwidth reasonably.

ID Description

1. Robot ID: 1, Hero (Red) ; 2, Engineer (Red) ; 3/4/5, Standard (Red) ; 6, Aerial (Red) ; 7, Sentry (Red) ; 11, Hero (Blue) ; 12, Engineer (Blue) ; 13/14/15, Standard (Blue) ; 16, Aerial (Blue) ; 17, Sentry (Blue) .
2. Client ID: 0x0101 for Hero operator's client (Red); 0x0102, Engineer operator's client (Red); 0x0103/0x0104/0x0105, Standard operator's client (Red); 0x0106, Aerial operator's client (Red); 0x0111, Hero operator's client (Blue); 0x0112, Engineer operator's client (Blue); 0x0113/0x0114/0x0115, Standard operator's client (Blue); 0x0116, Aerial operator's client (Blue).

Client custom data: cmd_id: 0x0301. Content ID: 0xD180.

1. Client. Client custom data: cmd_id: 0x0301. Content ID: 0xD180. Transmission frequency: Maximum 10Hz

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0xD180
2	2	Sender ID	Need to verify the correctness of the sender's robot ID
4	2	Client ID	Can only be the corresponding client of the sender's robot
6	4	Custom float point data 1	Display the float point data on the Client custom data display panel
10	4	Custom float point data 2	Display the float point data on the Client custom data display panel

Byte Offset	Size	Description	Remarks
14	4	Custom float point data 3	Display the float point data on the Client custom data display panel
18	1	Custom 8-bit data 4	Bit 0-5: control the six indicators on the Client custom data display panel individually. When the value is 1, the indicator turns solid green and 0 turns solid red Bit 6-7: reserved

```
typedef __pack struct
{
float data1;
float data2;
float data3;
uint8_t masks;
} client_custom_data_t
```

Communication between student robots: cmd_id 0x0301; content ID: 0x0200~0x02FF

- Interactive data. Communication between robots: 0x0301. Transmission frequency: Maximum 10Hz

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0200~0x02FF Can be selected in the above ID segments and the specific ID definition is customized by the team
2	2	Sender ID	Need to verify the correctness of the sender ID
4	2	Receiver ID	Need to verify the correctness of the receiver ID. For example, you cannot send to the enemy robot's ID.
6	n	Data segment	n should be smaller than 113

```

typedef __pack struct
{
uint8_t data[]
} robot_interactive_data_t

```

CRC Check Code Example

```

//crc8 generator polynomial: G(x)=x8+x5+x4+1
const unsigned char CRC8_INIT = 0xff;
const unsigned char CRC8_TAB[256] =
{
0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83, 0xc2, 0x9c, 0x7e, 0x20, 0xa3, 0xfd, 0x1f, 0x41,
0x9d, 0xc3, 0x21, 0x7f, 0xfc, 0xa2, 0x40, 0x1e, 0x5f, 0x01, 0xe3, 0xbd, 0x3e, 0x60, 0x82, 0xdc,
0x23, 0x7d, 0x9f, 0xc1, 0x42, 0x1c, 0xfe, 0xa0, 0xe1, 0xbf, 0x5d, 0x03, 0x80, 0xde, 0x3c, 0x62,
0xbe, 0xe0, 0x02, 0x5c, 0xdf, 0x81, 0x63, 0x3d, 0x7c, 0x22, 0xc0, 0x9e, 0x1d, 0x43, 0xa1, 0xff,
0x46, 0x18, 0xfa, 0xa4, 0x27, 0x79, 0x9b, 0xc5, 0x84, 0xda, 0x38, 0x66, 0xe5, 0xbb, 0x59, 0x07,
0xdb, 0x85, 0x67, 0x39, 0xba, 0xe4, 0x06, 0x58, 0x19, 0x47, 0xa5, 0xfb, 0x78, 0x26, 0xc4, 0x9a,
0x65, 0x3b, 0xd9, 0x87, 0x04, 0x5a, 0xb8, 0xe6, 0xa7, 0xf9, 0x1b, 0x45, 0xc6, 0x98, 0x7a, 0x24,
0xf8, 0xa6, 0x44, 0x1a, 0x99, 0xc7, 0x25, 0x7b, 0x3a, 0x64, 0x86, 0xd8, 0x5b, 0x05, 0xe7, 0xb9,
0x8c, 0xd2, 0x30, 0x6e, 0xed, 0xb3, 0x51, 0x0f, 0x4e, 0x10, 0xf2, 0xac, 0x2f, 0x71, 0x93, 0xcd,
0x11, 0x4f, 0xad, 0xf3, 0x70, 0x2e, 0xcc, 0x92, 0xd3, 0x8d, 0x6f, 0x31, 0xb2, 0xec, 0x0e, 0x50,
0xaf, 0xf1, 0x13, 0x4d, 0xce, 0x90, 0x72, 0x2c, 0x6d, 0x33, 0xd1, 0x8f, 0x0c, 0x52, 0xb0, 0xee,
0x32, 0x6c, 0x8e, 0xd0, 0x53, 0x0d, 0xef, 0xb1, 0xf0, 0xae, 0x4c, 0x12, 0x91, 0xcf, 0x2d, 0x73,
0xca, 0x94, 0x76, 0x28, 0xab, 0xf5, 0x17, 0x49, 0x08, 0x56, 0xb4, 0xea, 0x69, 0x37, 0xd5, 0x8b,
0x57, 0x09, 0xeb, 0xb5, 0x36, 0x68, 0x8a, 0xd4, 0x95, 0xcb, 0x29, 0x77, 0xf4, 0xaa, 0x48, 0x16,
0xe9, 0xb7, 0x55, 0x0b, 0x88, 0xd6, 0x34, 0x6a, 0x2b, 0x75, 0x97, 0xc9, 0x4a, 0x14, 0xf6, 0xa8,
0x74, 0x2a, 0xc8, 0x96, 0x15, 0x4b, 0xa9, 0xf7, 0xb6, 0xe8, 0x0a, 0x54, 0xd7, 0x89, 0x6b, 0x35,
};
unsigned char Get_CRC8_Check_Sum(unsigned char *pchMessage,unsigned int
dwLength,unsigned char ucCRC8)
{
unsigned char ucIndex;
while (dwLength--)
{
ucIndex = ucCRC8^(*pchMessage++);
ucCRC8 = CRC8_TAB[ucIndex];
}
return(ucCRC8);
}
/*
** Descriptions: CRC8 Verify function

```

```

** Input: Data to Verify,Stream length = Data + checksum
** Output: True or False (CRC Verify Result)
*/
unsigned int Verify_CRC8_Check_Sum(unsigned char *pchMessage, unsigned int dwLength)
{
    unsigned char ucExpected = 0;
    if ((pchMessage == 0) || (dwLength <= 2)) return 0;
    ucExpected = Get_CRC8_Check_Sum (pchMessage, dwLength-1, CRC8_INIT);
    return ( ucExpected == pchMessage[dwLength-1] );
}
/*
** Descriptions: append CRC8 to the end of data
** Input: Data to CRC and append,Stream length = Data + checksum
** Output: True or False (CRC Verify Result)
*/
void Append_CRC8_Check_Sum(unsigned char *pchMessage, unsigned int dwLength)
{
    unsigned char ucCRC = 0;
    if ((pchMessage == 0) || (dwLength <= 2)) return;
    ucCRC = Get_CRC8_Check_Sum ( (unsigned char *)pchMessage, dwLength-1, CRC8_INIT);
    pchMessage[dwLength-1] = ucCRC;
}

uint16_t CRC_INIT = 0xffff;
const uint16_t wCRC_Table[256] =
{
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
    0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
    0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
    0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
    0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
    0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
    0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
    0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
    0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,

```

```

0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};
/*
** Descriptions: CRC16 checksum function
** Input: Data to check,Stream length, initialized checksum
** Output: CRC checksum
*/
uint16_t Get_CRC16_Check_Sum(uint8_t *pchMessage,uint32_t dwLength,uint16_t wCRC)
{
    Uint8_t chData;
    if (pchMessage == NULL)
    {
        return 0xFFFF;
    }
    while(dwLength--)
    {
        chData = *pchMessage++;
        (wCRC) = ((uint16_t)(wCRC) >> 8) ^ wCRC_Table[(((uint16_t)(wCRC) ^ (uint16_t)(chData)) &
0x00ff)];
    }
}

```

```

}
return wCRC;
}

/*
** Descriptions: CRC16 Verify function
** Input: Data to Verify,Stream length = Data + checksum
** Output: True or False (CRC Verify Result)
*/
uint32_t Verify_CRC16_Check_Sum(uint8_t *pchMessage, uint32_t dwLength)
{
uint16_t wExpected = 0;
if ((pchMessage == NULL) || (dwLength <= 2))
{
return __FALSE;
}
wExpected = Get_CRC16_Check_Sum ( pchMessage, dwLength - 2, CRC_INIT);
return ((wExpected & 0xff) == pchMessage[dwLength - 2] && ((wExpected >> 8) & 0xff) ==
pchMessage[dwLength - 1]);
}

/*
** Descriptions: append CRC16 to the end of data
** Input: Data to CRC and append,Stream length = Data + checksum
** Output: True or False (CRC Verify Result)
*/
void Append_CRC16_Check_Sum(uint8_t * pchMessage,uint32_t dwLength)
{
uint16_t wCRC = 0;
if ((pchMessage == NULL) || (dwLength <= 2))
{
return;
}
wCRC = Get_CRC16_Check_Sum ( (U8 *)pchMessage, dwLength-2, CRC_INIT );
pchMessage[dwLength-2] = (U8)(wCRC & 0x00ff);
pchMessage[dwLength-1] = (U8)((wCRC >> 8)& 0x00ff);
}

```