Referee System Serial Port Protocol Appendix

Publisher: RoboMaster Organizing Committee (RMOC)

Version: V1.0

Date: 2020-02-25

Release Notes

Date	Version	Changes
2020.2.25	V1.0	Initial release

1. Serial Port Configuration

The communication interface is serial port, which is configured with 115200 baud rate, 8 data bits and 1 stop bit and 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 and the fixed value is 0xA5
data_length	1	2	The length of data inside the data frame
seq	3	1	The sequence number of package
CRC8	4	1	The CRC8 checksum of frame header

Table 3 cmd_id Command Code IDs Description

Command Code	Data Segment Length	Function Description
0x0001	3	Status data of the competition and the transmitting cycle is 1Hz
0x0002	1	Result data of the competition, which is transmitted at the end of the competition
0x0003	32	Robot HP data of the competition and the transmitting cycle is 1Hz

Command Code	Data Segment Length	Function Description
0x0004	3	Dart launching status，which is transmitted when the dart is fired
0x0005	3	Buff and debuff zone status of the AI Challenge and the transmitting cycle is 1Hz
0x0101	4	Battlefield event data and the transmitting cycle is 1Hz
0x0102	3	Battlefield Projectile Supplier action identification data, which is transmitted after the action has changed
0x0104	2	Referee warning data, which is transmitted after the warning has been issued
0x0105	1	Dart barrel countdown and the transmitting cycle is 1Hz
0x0201	18	Robot status data, whose transmitting cycle is 10Hz
0x0202	16	Real-time power and barrel heat data and the transmitting cycle is 50Hz
0x0203	16	Robot’s position data and the transmitting cycle is 10Hz
0x0204	1	Robot gain data and the transmitting cycle is 1Hz
0x0205	3	Aerial energy status data, which is only transmitted by the Aerial’s Main Controller Module, and the transmitting cycle is 10Hz
0x0206	1	Damage status data, which is transmitted after the damage has occurred
0x0207	6	Real-time shooting data, which is transmitted after the projectile is launched
0x0208	2	The remaining launch quantity of projectile, which is transmitted only by the Main Controller Module of Aerial, Sentry and ICRA robots, and the transmitting cycle is 10Hz

Command Code	Data Segment Length	Function Description
0x209	4	RFID card information detected by the robot and the transmitting cycle is 1Hz
0x0301	n	Interaction data between robots, which is triggered to transmit by the sender

Detailed Description

1. Competition status data: 0x0001. Transmission frequency: 1Hz. Transmission scope: all robots.

Byte Offset	Size	Description

0	1	0-3 bit: Competition Type
		1: RoboMaster Robotics Competition;
		2: RoboMaster Technical Challenge;
		3: RoboMaster AI Challenge
		4-7 bit: Current Competition Stage
		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_status_t;
```

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

Byte Offset	Size	Description

0	1	0: Draw;
		1: Red win;
		2: Blue win

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

3. Robot HP data: 0x0003. Transmission frequency: 1Hz. Transmission scope: all robots.

Byte Offset	Size	Description
0	2	Red 1 Hero HP. If the robot has not entered the stage or is defeated, its HP will be 0.
2	2	Red 2 Engineer HP
4	2	Red 3 Standard HP
6	2	Red 4 Standard HP
8	2	Red 5 Standard HP
10	2	Red 7 Sentry HP
12	2	Red outpost HP
14	2	Red Base HP
16	2	Blue 1 Hero HP
18	2	Blue 2 Engineer HP
20	2	Blue 3 Standard HP
22	2	Blue 4 Standard HP
24	2	Blue 5 Standard HP
26	2	Blue 7 Sentry HP
28	2	Blue outpost HP

Byte Offset	Size	Description
30	2	Blue Base HP

```
typedef __packed struct
{
   uint16_t red_1_robot_HP;
   uint16_t red_2_robot_HP;
   uint16_t red_3_robot_HP;
   uint16_t red_4_robot_HP;
   uint16_t red_5_robot_HP;
   uint16_t red_7_robot_HP;
   uint16_t red_outpost_HP;
   uint16_t red_base_HP;
   uint16_t blue_1_robot_HP;
   uint16_t blue _2_robot_HP;
   uint16_t blue _3_robot_HP;
   uint16_t blue _4_robot_HP;
   uint16_t blue _5_robot_HP;
   uint16_t blue _7_robot_HP;
   uint16_t blue_outpost_HP;
   uint16_t blue _base_HP;
} ext_game_robot_HP_t;
```

4. Dart launching status: 0x0004. Transmission frequency: send after the dart is launched. Transmission scope: all robots.

Byte Offset	Size	description
0	1	Dart launchinging team: 1: From Red Team 2: From Blue Team
1	2	Remaining competition time when the dart is launched (s)

```
typedef __packed struct
{
   uint8_t dart_belong;
   uint16_t stage_remaining_time;
} ext_dart_status_t;
```

5. Buff and Debuff Zone status of the AI Challenge: 0x0005. Transmission frequency: 1Hz. Transmission scope: all robots.

Byte Offset	Size	description
0	3	bit [0, 4, 8, 12, 16, 20]: activation status for F1-F6: 0: unactivated 1: activated bit [1-3, 5-7, 9-11, 13-15, 17-19, 21-23]: status information for F1-F1: 1: Red Restoration Zone 2: Red Projectile Supplier Zone 3: Blue Restoration Zone 4: Blue Projectile Supplier Zone 5: Launch Penalty Zone 6; Movement Penalty Zone

```
typedef __packed struct
{
  uint8_t F1_zone_status:1;
  uint8_t F1_zone_buff_debuff_status:3;
  uint8_t F2_zone_status:1;
  uint8_t F2_zone_buff_debuff_status:3;
  uint8_t F3_zone_status:1;
  uint8_t F3_zone_buff_debuff_status:3;
  uint8_t F4_zone_status:1;
  uint8_t F4_zone_buff_debuff_status:3;
  uint8_t F5_zone_status:1;
  uint8_t F5_zone_buff_debuff_status:3;
  uint8_t F6_zone_status:1;
  uint8_t F6_zone_buff_debuff_status:3;
} ext_ICRA_buff_debuff_zone_status_t;
```

6. Battlefield event data: 0x0101. Transmission frequency: 1Hz. Transmission scope: own side robots.

Byte Offset	Size	Description
0	4	bit 0-1: The occupation status of Landing Pad of one's own side ● 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 bit 2-3: Power Rune status of one's own side: ● bit 2 is the activation status of Small Power Rune and 1 indicates it has been activated; ● bit 3 is the activation status of Large Power Rune and 1 indicates it has been activated; bit 4: Base Shield status of one's own side: ● 1 indicates that Base has Virtual Shield HP; ● 0 indicates that Base has no Virtual Shield HP; bit 5-31: Reserved

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

7. Projectile Supplier Zone action identification: 0x0102. Transmission frequency: send after the action is triggered. Transmission scope: own side robots.

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; 101 Blue Hero; 102 Blue Engineer; 103/104/105 Blue Standard

Byte Offset	Size	Description
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;
```

8. Referee warning: cmd_id (0x0104). Transmission frequency: transmitted after the warning has been issued. Transmission scope: own side robots.

Byte Offset	Size	Description
0	1	Warning level:
1	1	Offending robot ID: For Level 1 Warning and Level 5 Warning, robot ID is 0 For Level 2 to 4 Warning, robot ID is the offending robot’s ID

```
typedef __packed struct
{
  uint8_t level;
  uint8_t foul_robot_id;
} ext_referee_warning_t;
```

9. Dart barrel countdown: cmd_id (0x0105). Transmission frequency: 1Hz. Transmission scope: own side robots.

Byte Offset	Size	Description
0	1	15s countdown

```
typedef __packed struct
{
   uint8_t dart_remaining_time;
} ext_dart_remaining_time_t;
```

10. Match robot status: 0x0201. Transmission frequency: 10Hz. Transmission scope: single robot.

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;
		8: Red Dart Robot;
		9: Red Radar Station;
		101: Blue Hero;
		102: Blue Engineer;
		103/104/105: Blue Standard;
		106: Blue Aerial;
		107: Blue Sentry
		108: Blue Dart Robot;
		109: Blue Radar Station.

1	1	Robot level:
		1: level one;
		2: level two;

Byte Offset	Size	Description
		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	17 mm barrel maximum speed (m/s)
15	1	42 mm barrel maximum speed (m/s)
16	1	Maximum chassis power (w)

16	1	Main Controller Module 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 shooter_heat0_speed_limit;
   uint8_t shooter_heat1_speed_limit;
   uint8_t max_chassis_power;
   uint8_t mains_power_gimbal_output : 1;
   uint8_t mains_power_chassis_output : 1;
   uint8_t mains_power_shooter_output : 1;
} ext_game_robot_status_t;
```

11. Real-time power and barrel heat data: 0x0202. Transmission frequency: 50Hz. Transmission scope: single robot.

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) Note: Launch Ramp will increase to 250 J according to the rule
10	2	17 mm barrel heat
12	2	42 mm barrel heat
14	2	17 mm mobile 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;
   uint16_t mobile_shooter_heat2;
} ext_power_heat_data_t;
```

12. Robot position: 0x0203. Transmission frequency: 10Hz. Transmission scope: single robot.

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;
```

13. Robot gain: 0x0204. Transmission frequency: 1Hz. Transmission scope: single robot.

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;
```

14. Aerial energy status: 0x0205. Transmission frequency: 10Hz. Transmission scope: single robot.

Byte Offset	Size	Description
0	2	Accumulated energy points
2	1	Attack time (unit: s). Drop to 0 from 30 seconds

```
typedef __packed struct
{
    uint16_t energy_point;
    uint8_t attack_time;
} aerial_robot_energy_t;
```

15. Damage status: 0x0206. Transmission frequency: send after damage happens. Transmission scope: single robot.

Byte Offset	Size	Description
0	1	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.

Byte Offset	Size	Description
		bit 4-7: HP Change Type 0x0 HP deduction from armor damage; 0x1 HP deduction from module offline; 0x2 HP deduction from exceeding the speed limit; 0x3 HP deduction from exceeding the barrel heat limit; 0x4 HP deduction from exceeding the chassis power; 0x5 HP deduction for armor collision

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

16. Real-time shooting data: 0x0207. Transmission frequency: send after shooting. Transmission scope: single robot.

Byte Offset	Size	Description
0	1	Projectile type: ● 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;
```

17. Quantity of remaining projectiles: 0x0208. Transmission frequency: 1Hz and is transmitted by the Main Controller Module of Aerial, Sentry and ICRA Robots. Transmission scope: single robot.

Byte Offset	Size	Description
0	2	Quantity of remaining projectiles that can be launched

```
typedef __packed struct
{
   uint16_t bullet_remaining_num;
} ext_bullet_remaining_t;
```

18. Robot RFID status: 0x0209. Transmission frequency: 1Hz. Transmission scope: single robot.

Byte Offset	Size	Description

0	4	bit 0: Base Gain Zone RFID status;
		bit 1: Highland Gain Zone RFID status;
		bit 2: Power Rune Activation Point RFID status;
		bit 3: Launch Ramp Gain Zone RFID status;
		bit 4: Outpost Gain Zone RFID status;
		bit 5: Resource Island Gain Zone RFID status;
		bit 6: Restoration Zone Gain Zone RFID status;
		bit 7: Engineer Restoration Card RFID status;
		bit 8-25: reserved
		bit 26-31: AI Challenge F1-F6 RFID status;
		RFID status does not necessarily represent the corresponding buff and debuff status. For instance, when the enemy has occupied Highland Gain Zone, its corresponding buff status cannot be obtained.

```
typedef __packed struct
{
   uint32_t rfid_status
} ext_rfid_status_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 following table shows the overall byte limit of the interactive data 0x0301, where the data volume includes the byte volume of frame-header, cmd_id, frame_tail and data segment header structure.

Robot type	**Maximum uplink data volume (byte/s)**	**Maximum downlink data volume (byte/s)**
Radar Station	5120	5120
Hero	3720	3720
Engineer	3720	3720
Infantry	3720	3720
Sentry	3720	5120
Ariel	3720	3720

1. Interactive data receiving information: 0x0301.

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.

Byte Offset	Size	Description	Remarks
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
0x0200~0x02FF	6+n	Communication between your robots
0x0100	6+2	Client deletes graphics
0x0101	6+15	Client draws one graphic
0x0102	6+30	Client draws two graphics
0x0103	6+75	Client draws five graphics
0x0104	6+105	Client draws seven graphics
0x0110	6+45	Client draws character graphics

Since there are multiple content IDs and the transmit data volume per second is limited, 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) ; 101, Hero (Blue) ; 102, Engineer (Blue) ; 103/104/105, Standard (Blue) ; 106, Aerial (Blue) ; 107, 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);

0x0165, Hero operator’s client (Blue); 0x0166, Engineer operator’s client (Blue); 0x0167/0x0168/0x0169, Standard operator’s client (Blue); 0x016A, Aerial operator’s client (Blue).

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

1. Interactive data. Communication between robots: 0x0301.

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
```

2. Client deletes graphics, communication between robots: 0x0301.

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0100
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 since the data can only be sent to the corresponding client of robot.

Byte Offset	Size	Description	Remarks
6	1	Graphics operation type	Including: 0: null; 1: delete allgraphics on a layer; 2: delete all graphics
7	1	Layer number	Layer Number: 0-9

```
typedef __packed struct
{
uint8_t operate_tpye;
uint8_t layer;
} ext_client_custom _graphic_delete_t
```

Graphics data

Byte Offset	Size	Description	Remarks
0	3	Graphics name	Use the graphics name as an index to conduct delete or edit operation.
3	4	Graphic configuration 1	bit 0-2: Graphic operation; 0: null 1: add a graphic 2: edit a graphic 3: delete a graphic Bit 3-5: graphic type: 0: straight line 1: rectangle 2: round 3: ellipse

Byte Offset	Size	Description	Remarks
			4: arc 5: floating-point number; 6: integer number; 7: character Bit 6-9: layer number, 0~9 Bit 10-13: color: 0: Red and Blue as main colors 1: yellow 2: green 3: orange 4: magenta, 5: pink 6: cyan 7: black 8: white Bit 14-22: starting angle，the unit is degree and the range [0,360]; Bit 23-31: ending angle，the unit is degree and the range [0,360];
7	4	Graphic configuration 2	Bit 0-9: line width; Bit 10-20: starting point x coordinate; Bit 21-31: starting point y coordinate.

Byte Offset	Size	Description	Remarks
11	4	Graphic configuration3	Bit 0-9: font size or radius; Bit 10-20: ending point x coordinate; Bit 21-31: ending point y coordinate.

```
typedef __packed struct
{
uint8_t graphic_name[3];
uint32_t operate_tpye:3;
uint32_t graphic_tpye:3;
uint32_t layer:4;
uint32_t color:4;
uint32_t start_angle:9;
uint32_t end_angle:9;
uint32_t width:10;
uint32_t start_x:11;
uint32_t start_y:11;
uint32_t radius:10;
uint32_t end_x:11;
uint32_t end_y:11;
} graphic_data_struct_t
```

The following table shows the graphic configurations, where “null” indicates that the field has no influence on the graphics. The recommended font size to line width ratio is 10:1.

Type	start_angle	end_angle	width	start_x	start_y	radius	end_x	end_y
Straight line	Null	Null	Line width	Starting point x coordinate	Starting point y coordinate	Null	Starting point x coordinate	Starting point y coordinate

Type	start_angle	end_angle	width	start_x	start_y	radius	end_x	end_y
Rectangle	Null	Null	Line width	Starting point x coordinate	Starting point y coordinate	Null	Diagonal apex x coordinate	Diagonal apex y coordinate
Round	Null	Null	Line width	center x coordinate	center y coordinate	radius	Null	Null
Ellipse	Null	Null	Line width	center x coordinate	center y coordinate	Null	x semiaxis length	y semiaxis length
Arc	Start angle	End angle	Line width	center x coordinate	center y coordinate	Null	x semiaxis length	y semiaxis length
Floating-point number	Character size	Significant decimal place number	Line width	Start point x coordinate	point y coordinate	32-digit floating-point nuber, float		
Integer	Character size	Null	Line width	Start point x coordinate	point y coordinate	32-digit integer number, int32_t		
Character	Character size	Character length	Line width	Start point x coordinate	point y coordinate	Null	Null	Null

3. Client draws one graphic, communication between robots: 0x0301.

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0101
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 since the data can only be sent to the corresponding client of robot.
6	15	Graphic 1	See graphic data introduction

```
typedef __packed struct
{
   graphic_data_struct_t   grapic_data_struct;
} ext_client_custom_graphic_single_t;
```

4. Client draws two graphics, communication between robots: 0x0301.

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0102
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 since the data can only be sent to the corresponding client of robot.
6	15	Graphic 1	See graphic data introduction
21	15	Graphic 2	See graphic data introduction

```
typedef __packed struct
{
   graphic_data_struct_t   grapic_data_struct[2];
} ext_client_custom_graphic_double_t;
```

5. Client draws five graphics, communication between robots: 0x0301.

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0103
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 since the data can only be sent to the corresponding client of robot.
6	15	Graphic 1	See graphic data introduction
21	15	Graphic 2	See graphic data introduction
36	15	Graphic 3	See graphic data introduction
51	15	Graphic 4	See graphic data introduction
66	15	Graphic 5	See graphic data introduction

```
typedef __packed struct
{
   graphic_data_struct_t    grapic_data_struct[5];
} ext_client_custom_graphic_five_t;
```

6. Client draws seven graphics, communication between robots: 0x0301.

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0104
2	2	Sender ID	Need to verify the correctness of the sender ID

Byte Offset	Size	Description	Remarks
4	2	Receiver ID	Need to verify the correctness of the receiver ID since the data can only be sent to the corresponding client of robot.
6	15	Graphic 1	See graphic data introduction
21	15	Graphic 2	See graphic data introduction
36	15	Graphic 3	See graphic data introduction
51	15	Graphic 4	See graphic data introduction
66	15	Graphic 5	See graphic data introduction
81	15	Graphic 6	See graphic data introduction
96	15	Graphic 7	See graphic data introduction

```
typedef __packed struct
{
   graphic_data_struct_t    grapic_data_struct[7];
} ext_client_custom_graphic_seven_t;
```

7. Client draws characters, communication between robots: 0x0301.

Byte Offset	Size	Description	Remarks
0	2	Data content ID	0x0110
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 since the data can only be sent to the corresponding client of robot.

Byte Offset	Size	Description	Remarks
6	15	Character configuration	See graphic data introduction
21	30	character	

```
typedef __packed struct
{
   graphic_data_struct_t   grapic_data_struct;
   uint8_t data[30];
} ext_client_custom_character_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);
```