裁判系统学生串口协议附录

发布者：RoboMaster 组委会

发布版本：V1.1

发布日期：2019-03-08

修改日志

日期	版本	改动记录
2019.2.25	V1.0	首次发布
2019.3.8	V1.1	1. 增加场地补给站动作标识数据中的子弹数 2. 增加请求补给站补弹机器人 ID 3. 修正学生机器人间通信数据的内容 ID 4. 更新场地事件数据字节偏移量说明 5. 更新客户端自定义数据备注

1. 串口配置

通信方式是串口，配置为波特率 115200，8 位数据位，1 位停止位，无硬件流控，无校验位。

2. 接口协议说明

通信协议格式：

frame_header (5-byte)	cmd_id (2-byte)	data (n-byte)	frame_tail (2-byte，CRC16，整包校验)

表 1 frame_header 格式

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

表 2 帧头详细定义

域	偏移位置	大小（字节）	详细描述
SOF	0	1	数据帧起始字节，固定值为 0xA5
data_length	1	2	数据帧中 data 的长度
seq	3	1	包序号
CRC8	4	1	帧头 CRC8 校验

表 3 cmd_id 命令码 ID 说明

命令码	数据段长度	功能说明
0x0001	3	比赛状态数据，1Hz 周期发送
0x0002	1	比赛结果数据，比赛结束后发送
0x0003	2	比赛机器人存活数据，1Hz 周期发送
0x0101	4	场地事件数据，事件改变后发送
0x0102	3	场地补给站动作标识数据，动作改变后发送

命令码	数据段长度	功能说明
0x0103	2	请求补给站补弹数据，由参赛队发送，上限 10Hz。（RM 对抗赛尚未开放）
0x0201	15	机器人状态数据，10Hz 周期发送
0x0202	14	实时功率热量数据，50Hz 周期发送
0x0203	16	机器人位置数据，10Hz 发送
0x0204	1	机器人增益数据，增益状态改变后发送
0x0205	3	空中机器人能量状态数据，10Hz 周期发送，只有空中机器人主控发送
0x0206	1	伤害状态数据，伤害发生后发送
0x0207	6	实时射击数据，子弹发射后发送
0x0301	n	机器人间交互数据，发送方触发发送，上限 10Hz

详细说明

1. 比赛状态数据：0x0001。发送频率：1Hz

字节偏移量	大小	说明
0	1	0-3 bit：比赛类型 • 1：RoboMaster 机甲大师赛； • 2：RoboMaster 机甲大师单项赛； • 3：ICRA RoboMaster 人工智能挑战赛 4-7 bit：当前比赛阶段 • 0：未开始比赛； • 1：准备阶段； • 2：自检阶段；

字节偏移量	大小	说明
		• 3：5s 倒计时； • 4：对战中； • 5：比赛结算中
1	2	当前阶段剩余时间，单位 s

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

2. 比赛结果数据：0x0002。发送频率：比赛结束后发送

字节偏移量	大小	说明
0	1	0 平局 1 红方胜利 2 蓝方胜利

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

3. 机器人存活数据：0x0003。发送频率：1Hz

字节偏移量	大小	说明
0	2	bit 0：红方英雄机器人； bit 1：红方工程机器人； bit 2：红方步兵机器人 1； bit 3：红方步兵机器人 2； bit 4：红方步兵机器人 3；

字节偏移量	大小	说明
		bit 5：红方空中机器人； bit 6：红方哨兵机器人； bit 7：保留 bit 8：蓝方英雄机器人； bit 9：蓝方工程机器人； bit 10：蓝方步兵机器人 1； bit 11：蓝方步兵机器人 2； bit 12：蓝方步兵机器人 3； bit 13：蓝方空中机器人； bit 14：蓝方哨兵机器人； bit 15：保留 对应的 bit 数值置 1 代表机器人存活，数值置 0 代表机器人死亡或者未上场。

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

4. 场地事件数据：0x0101。发送频率：事件改变后发送

字节偏移量	大小	说明
0	4	bit 0-1：己方停机坪占领状态 • 0 为无机器人占领； • 1 为空中机器人已占领但未停桨； • 2 为空中机器人已占领并停桨 bit 2：己方补给站 1 号补血点占领状态 1 为已占领；

字节偏移量	大小	说明
		bit 3：己方补给站 2 号补血点占领状态 1 为已占领； bit 4：己方补给站 3 号补血点占领状态 1 为已占领； bit 5-6：己方大能量机关状态： • 0 为打击点未占领且大能量机关未激活； • 1 为打击点占领且大能量机关未激活； • 2 为大能量机关已激活； • 3 为大能量机关已激活且打击点被占领； bit 7：己方关口占领状态 1 为已占领； bit 8：己方碉堡占领状态 1 为已占领； bit 9：己方资源岛占领状态 1 为已占领； bit 10-11：己方基地防御状态： • 2 为基地 100%防御； • 1 为基地有哨兵防御； • 0 为基地无防御； bit 12-13：ICRA 红方防御加成 • 0：防御加成未激活； • 1：防御加成 5s 触发激活中； • 2：防御加成已激活 bit 14-15：ICRA 蓝方防御加成 • 0：防御加成未激活； • 1：防御加成 5s 触发激活中； • 2：防御加成已激活 其余保留

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

5. 补给站动作标识：0x0102。发送频率：动作改变后发送

字节偏移量	大小	说明
0	1	补给站口 ID： 1：1 号补给口； 2：2 号补给口
1	1	补弹机器人 ID：0 为当前无机器人补弹，1 为红方英雄机器人补弹，2 为红方工程机器人补弹，3/4/5 为红方步兵机器人补弹，11 为蓝方英雄机器人补弹，12 为蓝方工程机器人补弹，13/14/15 为蓝方步兵机器人补弹
2	1	出弹口开闭状态：0 为关闭，1 为子弹准备中，2 为子弹下落
3	1	补弹数量： 50：50 颗子弹； 100：100 颗子弹； 150：150 颗子弹； 200：200 颗子弹。

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

6. 请求补给站补弹子弹：cmd_id (0x0103)。发送频率：上限 10Hz。RM 对抗赛尚未开放

字节偏移量	大小	说明
0	1	补给站补弹口 ID： 1：1 号补给口
1	1	补弹机器人 ID：1 为红方英雄机器人补弹，2 为红方工程机器人补弹，3/4/5 为红方步兵机器人补弹，11 为蓝方英雄机器人补弹，12 为蓝方工程机器人补弹，13/14/15 为蓝方步兵机器人补弹
1	1	补弹数目： 50 ：请求 50 颗子弹下落

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

7. 比赛机器人状态：0x0201。发送频率：10Hz

字节偏移量	大小	说明
0	1	机器人 ID： 1：红方英雄机器人； 2：红方工程机器人； 3/4/5：红方步兵机器人； 6：红方空中机器人； 7：红方哨兵机器人； 11：蓝方英雄机器人； 12：蓝方工程机器人； 13/14/15：蓝方步兵机器人；

字节偏移量	大小	说明
		16：蓝方空中机器人； 17：蓝方哨兵机器人。
1	1	机器人等级： 1：一级；2：二级；3：三级。
2	2	机器人剩余血量
4	2	机器人上限血量
6	2	机器人 17mm 枪口每秒冷却值
8	2	机器人 17mm 枪口热量上限
10	2	机器人 42mm 枪口每秒冷却值
12	2	机器人 42mm 枪口热量上限
14	1	主控电源输出情况： 0 bit：gimbal 口输出： 1 为有 24V 输出，0 为无 24v 输出； 1 bit：chassis 口输出：1 为有 24V 输出，0 为无 24v 输出； 2 bit：shooter 口输出：1 为有 24V 输出，0 为无 24v 输出；

```
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. 实时功率热量数据：0x0202。发送频率：50Hz

字节偏移量	大小	说明
0	2	底盘输出电压 单位 毫伏
2	2	底盘输出电流 单位 毫安
4	4	底盘输出功率 单位 W 瓦
8	2	底盘功率缓冲 单位 J 焦耳
10	2	17mm 枪口热量
12	2	42mm 枪口热量

```
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. 机器人位置：0x0203。发送频率：10Hz

字节偏移量	大小	说明
0	4	位置 x 坐标，单位 m
4	4	位置 y 坐标，单位 m
8	4	位置 z 坐标，单位 m
12	4	位置枪口，单位度

```
typedef __packed struct
{
```

```
  float x;
  float y;
  float z;
  float yaw;
} ext_game_robot_pos_t;
```

10. 机器人增益：0x0204。发送频率：状态改变后发送

字节偏移量	大小	说明
0	1	bit 0：机器人血量补血状态 bit 1：枪口热量冷却加速 bit 2：机器人防御加成 bit 3：机器人攻击加成 其他 bit 保留

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

11. 空中机器人能量状态：0x0205。发送频率：10Hz

字节偏移量	大小	说明
0	1	积累的能量点
1	2	可攻击时间 单位 s。50s 递减至 0

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

12. 伤害状态：0x0206。发送频率：伤害发生后发送

字节偏移量	大小	说明
0	1	bit 0-3：当血量变化类型为装甲伤害，代表装甲 ID，其中数值为 0-4 号代表机器人的五个装甲片，其他血量变化类型，该变量数值为 0。 bit 4-7：血量变化类型 0x0 装甲伤害扣血； 0x1 模块掉线扣血； 0x2 超枪口热量扣血； 0x3 超底盘功率扣血。

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

13. 实时射击信息：0x0207。发送频率：射击后发送

字节偏移量	大小	说明
0	1	子弹类型: 1：17mm 弹丸 2：42mm 弹丸
1	1	子弹射频 单位 Hz
2	4	子弹射速 单位 m/s

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

3. 机器人间交互数据

交互数据包括一个统一的数据段头结构。数据段包含了内容 ID，发送者以及接收者的 ID 和内容数据段，整个交互数据的包总共长最大为 128 个字节，减去 frame_header,cmd_id 和 frame_tail 共 9 个字节以及数据段头结构的 6 个字节，故而发送的内容数据段最大为 113。整个交互数据 0x0301 的包上行频率为 10Hz。

1. 交互数据接收信息：0x0301。发送频率：上限 10Hz

字节偏移量	大小	说明	备注
0	2	数据段的内容 ID	
2	2	发送者的 ID	需要校验发送者的 ID 正确性，例如红 1 发送给红 5，此项需要校验红 1
4	2	接收者的 ID	需要校验接收者的 ID 正确性，例如不能发送到敌对机器人的 ID
6	x	内容数据段	x 最大为 113

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

内容 ID	长度 （头结构长度+内容数据段长度）	功能说明
0xD180	6 + 13	客户端自定义数据
0x0200~0x02FF	6+n	己方机器人间通信

由于存在多个内容 ID，但整个 cmd_id 上行频率最大为 10Hz，请合理安排带宽。

ID 说明

1. 机器人 ID：1，英雄(红)；2，工程(红)；3/4/5，步兵(红)；6，空中(红)；7，哨兵(红)；11，英雄(蓝)；12，工程(蓝)；13/14/15，步兵(蓝)；16，空中(蓝)；17，哨兵(蓝)。
2. 客户端 ID：0x0101 为英雄操作手客户端(红)；0x0102，工程操作手客户端((红)；0x0103/0x0104/0x0105，步兵操作手客户端(红)；0x0106，空中操作手客户端((红)； 0x0111，英雄操作手客户端(蓝)；0x0112，工程操作手客户端(蓝)；0x0113/0x0114/0x0115，操作手客户端步兵(蓝)；0x0116，空中操作手客户端(蓝)。

客户端自定义数据：cmd_id:0x0301。内容 ID:0xD180。

1. 客户端 客户端自定义数据：cmd_id:0x0301。内容 ID:0xD180。发送频率：上限 10Hz

字节偏移量	大小	说明	备注
0	2	数据的内容 ID	0xD180
2	2	发送者的 ID	需要校验发送者机器人的 ID 正确性
4	2	客户端的 ID	只能为发送者机器人对应的客户端
6	4	自定义浮点数据 1	在客户端自定义数据显示面板上显示该浮点数
10	4	自定义浮点数据 2	在客户端自定义数据显示面板上显示该浮点数
14	4	自定义浮点数据 3	在客户端自定义数据显示面板上显示该浮点数
18	1	自定义 8 位数据 4	bit 0-5：分别控制客户端自定义数据显示面板上的六个指示灯，值为 1 时显示绿色，值为 0 是显示红色。 Bit 6-7：保留

```
typedef __pack struct
{
float data1;
float data2;
float data3;
```

```
uint8_t masks;
} client_custom_data_t
```

学生机器人间通信 cmd_id 0x0301，内容 ID:0x0200~0x02FF

2. 交互数据 机器人间通信：0x0301。发送频率：上限 10Hz

字节偏移量	大小	说明	备注
0	2	数据的内容 ID	0x0200~0x02FF 可以在以上 ID 段选取，具体 ID 含义由参赛队自定义
2	2	发送者的 ID	需要校验发送者的 ID 正确性
4	2	接收者的 ID	需要校验接收者的 ID 正确性，例如不能发送到敌对机器人的 ID
6	n	数据段	n 需要小于 113

```
typedef __pack struct
{
uint8_t data[]
} robot_interactive_data_t
```

CRC 校验代码示例

```
//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);
```