版本：V1.0.0

密级：保密

**惠州市几米物联技术有限公司**

**SMT防错料系统**

**通讯协议**

**修订记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 作者 | 日期 | 版本 | 审核 | 批准 | 描述 |
| 曾康 | 2017-11-28 | 1.0.0 |  |  | 初始版发行 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**目录**

[一、 协议包格式 3](#_Toc499651493)

[1. 协议号详解 3](#_Toc499651494)

[二、 协议包分解 4](#_Toc499651495)

[1. 登录包 5](#_Toc499651496)

[a) 登录包 5](#_Toc499651497)

[a) 登录包回复（服务器回复） 5](#_Toc499651498)

[2. 心跳包 6](#_Toc499651499)

[a) 终端发送心跳包 6](#_Toc499651500)

[b) 服务器心跳包回复 6](#_Toc499651501)

[3. 上传板子数包 8](#_Toc499651502)

[c) 中控发送板子数量包 8](#_Toc499651503)

[d) 服务器上传板子数包回复 8](#_Toc499651504)

[4. 中控控制包 9](#_Toc499651505)

[a) 服务器发送至中控的控制包 9](#_Toc499651506)

[b) 中控回复服务器 9](#_Toc499651507)

[6. 板子数清零包 10](#_Toc499651508)

[c) 服务器发送至中控的控制包 10](#_Toc499651509)

[d) 中控回复服务器 10](#_Toc499651510)

[三、 附件 11](#_Toc499651511)

[1. 附件1 CRC-ITU 查表算法 C 语言代码片段 11](#_Toc499651512)

1. **协议包格式**

|  |  |  |
| --- | --- | --- |
| 格式 | 长度(Byte) | 说明 |
| 起始位 | 2 | 0x80 0x80 |
| 包长度 | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | 1 | 代表传输包的类型（详见下表） |
| 信息内容 | N | 按不同的应用，对应相应的“协议号”，确定具体的内容 |
| 信息序列号 | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附表1） |
| 停止位 | 2 | 固定值，统一为0x0D 0x0A |

1. 协议号详解

|  |  |
| --- | --- |
| 登录包 | 0x4C |
| 心跳包 | 0x48 |
| 板子数量包 | 0x42 |
| 重置板子数 | 0x52 |
| 控制器 | 0x43 |

1. **协议包分解**
2. **登录包**

**关于登录包的说明：**

* + - * **登录包是SMT中控与服务器建立连接的信息包，会向服务器发送中控的信息**
      * **网络连接建立成功并向服务器发送一条登录包，5 秒内收到返回包则认为连接正常，未收到会继续发送登录包**
      * **超过 5 秒没有收到服务器的返回包，则认为登录包回复超时**
      * **超时 3 次后SMT中控启动定时重启功能**

* 1. 登录包

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x4C |
| 信息内容 | 中控MAC | 6 | 例：MAC地址为00:50:BA:CE:07:0C，则MAC地址为：0x00 0x50 0xBA 0xCE 0x07 0x0C |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 0B 4C 0x00 0x50 0xBA 0xCE 0x07 0x0C 00 01 50 F3 0D 0A

* 1. 登录包回复（服务器回复）

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x4C |
| 信息内容 | 线号 | 1 | 例：线号01，则终端 ID 为：0x01 |
| 时间戳 | 4 | Unix时间戳 如: 北京时间2017/11/27 14:16:3则时间戳为0x5A 0x1B 0xAD 0xA3 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 0A 4C 01 5A 1B AD A3 00 01 D6 DA 0D 0A

1. **心跳包**

**关于心跳包的说明：**

* + - * **为了维持SMT中控与服务器连接不被断开而进行发送维持链路；**
      * **SMT中控连接建立成功并向服务器发送一条心跳包，5 秒内收到返回包则认为连接正常，周期后会发送下一个心跳包**
      * **超过 5 秒没有收到服务器的返回包，则认为心跳超时**
      * **心跳超时 3 次后终端启动定时重启功能**
      * **心跳包周期暂定为3分钟**
  1. 终端发送心跳包

心跳包

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x48 |
| 信息内容 | 线号 | 1 | 发送登录包时收到服务器回复的线号 如:0x01 |
| 时间戳 | 4 | Unix时间戳 如: 北京时间2017/11/27 14:16:3则时间戳为0x5A 0x1B 0xAD 0xA3 |
| 中控信息内容 | 1 | 详见下表 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 09 48 01 5A 1B AD A3 03 00 02 17 5B 0D 0A

中控信息内容详解

占用1个字节，用来表示中控的各种状态信息。

|  |  |  |
| --- | --- | --- |
| 位 | | 代码含义 |
| BYTE | Bit2 | 1：报警灯打开 |
| 0：报警灯断开 |
| Bit1 | 1：接驳台接通 |
| 0：接驳台断开 |
| Bit0 | 1：红外线正常 |
| 0：红外线异常 |

* 1. 服务器心跳包回复

|  |  |  |
| --- | --- | --- |
|  | 长度 | 详解 |
| 起始位 | 2 | 0x80 0x80 |
| 包长度 | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | 1 | 0x48 |
| 信息序列号 | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 05 48 00 02 71 2F 0D 0A

1. **上传板子数包**

**关于上传板子数包的说明：**

* + - * **为了中控能上传当前探测PCB板总数上传至服务器；**
      * **中控应每隔30分钟上传一次PCB板总数，5 秒内收到返回包则认为连接正常，周期后会发送下一个板子数包**
  1. 中控发送板子数量包

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x42 |
| 信息内容 | 线号 | 1 | 发送登录包时收到服务器回复的线号 如:0x01 |
| 时间戳 | 4 | Unix时间戳 如: 北京时间2017/11/27 14:16:3则时间戳为0x5A 0x1B 0xAD 0xA3 |
| 板子数 | 3 | 板子总数 例:1000台　0x 00 0x03 0xE8 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 0D 42 01 5A 1B AD A3 00 03 E8 00 03 3E 99 0D 0A

* 1. 服务器上传板子数包回复

|  |  |  |
| --- | --- | --- |
|  | 长度 | 详解 |
| 起始位 | 2 | 0x80 0x80 |
| 包长度 | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | 1 | 0x42 |
| 信息序列号 | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 05 42 00 03 13 DC 0D 0A

1. **中控控制包**

**关于中控控制包的说明：**

* + - * **用于控制中控、接驳台、报警灯等工作状态的数据包**

1. 服务器发送至中控的控制包

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x43 |
| 信息内容 | 设备标志位 | 1 | 用于区分发送方的设备是APP、PC还是SERVER |
| 线号 | 1 | 要控制的线号 |
| 类型 | 1 | 0x01 中控器  0x02 接驳台  0x03 报警器 |
| 工作方式 | 1 | 0x00 关  0x01 开  0x02 重启 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 09 43 FF 08 02 01 00 05 D3 15 0D 0A

1. 中控回复服务器

中控将控制结果回复至服务器

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x43 |
| 信息内容 | 设备标志位 | 1 | 用于区分发送方的设备是APP、PC还是SERVER |
| 控制结果 | 1 | 例：0x00:失败 0x01:成功 |
| 错误码 | １ | 例：0x01:继电器故障 成功返回0x00 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 08 43 FF 01 00 00 05 E2 7C 0D 0A

1. **板子数清零包**

**关于板子数清零包的说明：**

* + - * **用于服务器通知中控清除板子数**

1. 服务器发送至中控的控制包

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x52 |
| 信息内容 | 设备标志位 | 1 | 用于区分发送方的设备是APP、PC还是SERVER |
| 线号 | 1 | 要清除板子数的线号 |
| 清除原因 | 1 | 0x00 工单开始  0x01 线号开始 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 08 52 FF 01 00 00 06 96 7C 0D 0A

1. 中控回复服务器

中控将控制结果回复至服务器

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 长度 | 详解 |
| 起始位 | | 2 | 0x80 0x80 |
| 包长度 | | 1 | 长度=协议号+信息内容+信息序列号+错误校验 |
| 协议号 | | 1 | 0x52 |
| 信息内容 | 设备标志位 | 1 | 用于区分发送方的设备是APP、PC还是SERVER |
| 控制结果 | 1 | 例：0x00:失败 0x01:成功 |
| 错误码 | １ | 例：成功返回0x00 其它错误码待定 |
| 信息序列号 | | 2 | 从开机后，每次发送数据序列号都自动加1 |
| 错误校验 | | 2 | “包长度”到“信息序列号”的CRC-ITU值。接收方若收到的信息计算有CRC错误，则忽略，抛弃这个数据包（算法详见附件1） |
| 停止位 | | 2 | 固定值，统一为0x0D 0x0A |

示例数据：80 80 08 52 FF 01 00 00 06 96 7C 0D 0A

1. **附件**
   1. 附件1 CRC-ITU 查表算法 C 语言代码片段

static const U16 crctab16[] =

{

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, 0x80DF, 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,

};

// 计算给定长度数据的 16 位 CRC。

U16 GetCrc16(const U8\* pData, int nLength)

{

U16 fcs = 0xffff; // 初始化

while(nLength>0){

fcs = (fcs >> 8) ^ crctab16[(fcs ^ \*pData) & 0xff];

nLength--;

pData++;

}

return ~fcs; // 取反

}