EMC测试监控通讯协议

# 通讯方式

本通信协议采用TCP/IP，PC端上位机作为服务器端，设备端作为客户端

# 传输规则

本协议中采用小端模式(Little-Endian)的网络字节序来传递双字节和四字节

# 消息的数据格式组成



# 标识位

采用0x7e 表示，若校验码、消息头以及消息体中出现0x7e，则需要进行转义处理，转义规则定义如下：

* 0x7e <-----> 0x7d 后紧跟一个0x02；
* 0x7d <-----> 0x7d 后紧跟一个0x01；

转义处理过程如下：

* 发送消息时：消息封装——>计算并填充校验码——>转义
* 接收消息时：转义还原——>验证校验码——>解析消息

例：发送一包内容为0x30 0x7e 0x08 0x7d 0x55 的数据包，则经过封装如下：

0x7e 0x30 7d 0x01 0x08 0x7d 0x02 0x55 0x7e

# 消息内容

## 消息头

|  |  |  |  |
| --- | --- | --- | --- |
| 起始字节 | 字段 | 字段长度 | 说明 |
| 0 | 消息ID | 2字节 | 表明消息体的功能 |
| 2 | 消息体长度 | 2字节 | 表明消息体的长度 |
| 4 | 设备ID | 8字节 | 唯一设备号 |
| 12 | 消息体流水号 | 4字节 | 按发送顺序从0 开始循环累加 |

设备ID的字节格式：



## 消息体格式

### 设备心跳 （设备 --> PC）

消息ID : 0x0001

消息体: 无

### 事件信息 (设备 --> PC)

消息ID ; 0x0002

消息体采用标准JSON格式表达，每条事件包括 时间，模块名，状态信息。其中”event”可为object和list：

* object下，为单一事件。
* list下，每一个list的item为一个事件的object

例子:

{“event”:

{"date": "20190730105242",

"mod": "MONITOR\_MOD\_MCU\_CAN1",

"info": "OK"}

}

{“event”: [

{"date": "20190730105242",

"mod": "MONITOR\_MOD\_MCU\_CAN1",

"error": "Data Abort"

},

{"date": "20190730105242",

"mod": "MONITOR\_MOD\_MCU\_CAN2",

"error": "Data Abort"

}

]

}

### Setting (PC ---> 设备)

消息ID：0x04

消息体采用标准JSON格式表达，根据不同指令内容不同。基础结构如下：

{“Setting”:

{“MOD\_RTC”:

{

“year”: YYYY

“month”: YY

“day”: YY

“hour”: YY

“min”: YY

“sec”: YY

}

}

}

后续可以根据需要添加需要设置参数的MOD。

MOD\_RTC: YYYY/YY Int 类型

### 上报信息状态 （设备 --> PC）

消息ID : 0x0100

消息体用JSON格式表达，用来获取系统信息中的一项或者几项

例子:

{“system”:

{“version”:

{“sw”: “1.0”,

“hw”: “2.0”

},

“temperature”: “37”,

“cpu”:

{“cpu0”: “78”,

“cpu1”: “80”

},

“memory”:

{“total”: “2048M”,

“used”: “1024M”

}

}

}

## 校验码

校验码指从消息头开始，直到校验码前一个字节，进行CRC16的校验值，占用2个字节。Python参考实现代码如下：

def crc16(bytes\_buffer):

""" crc16的校准 """

crc = 0xffff

for byte in bytes\_buffer:

crc = ((crc >> 8) | (crc << 8)) & 0xFFFF

crc ^= byte

crc ^= ((crc & 0xff) % 256) >> 4

crc ^= ((crc << 8) << 4) & 0xFFFF

crc ^= ((crc & 0xff) << 4) << 1

return crc