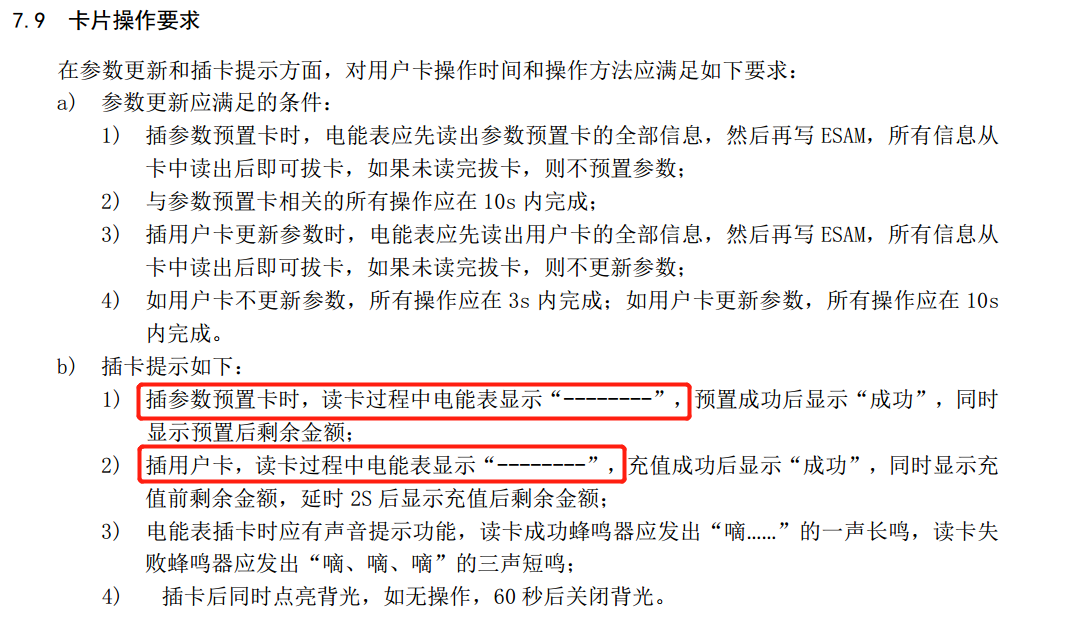
# 去掉插卡循环显示“-”到“--------”功能

* 1. 功能要求



* 1. 修改设计方案：

程序原来做法：

插入参数预置卡，一直显示“--------”、插入用户卡，动态循环显示“-”到“--------”

程序修改后做法：

按照规范程序修改后：插入参数预置卡和用户卡，都一直显示“--------” ，去掉循环显示功能。

修改方案：

* **Card.C文件：**

1：去掉void Card\_DispDynamicSign(void)函数，不需要循环显示“-”到“--------”功能；

用户卡所有调用该函数的地方都删除；

2：uchar8 Card\_Esam\_Proc(StrCardParse \*pV\_CardParse)

用户卡处理函数中，插卡增加显示“--------”

3：删除全局变量GV\_ReadPresetCardFlag ：判断是参数预置卡还是用户卡的标志，该全局变量可以删除；

4：不需要判断当前状态是否是全显模式，显示模块可以去掉判断是否处于全显模式函数SF\_DispAllScreenMode() （人机交互等级最高）

5：去掉接口函数：定时器中断函数读取当前插入的卡是参数预置卡还是用户卡的功能：

uchar8 Card\_ReadPresetCardFlag(void)

6：去掉接口函数：初始化参数预置卡还是用户卡状态

void InF\_CardParam(void)

* **Timer.C文件**

去掉ETIM4定时器的相关处理函数：

1：中断函数：void BackUpBaseTimer\_IRQHandler(void)

2：启动定时器函数：void InF\_BackUpBaseTimerStart(void)

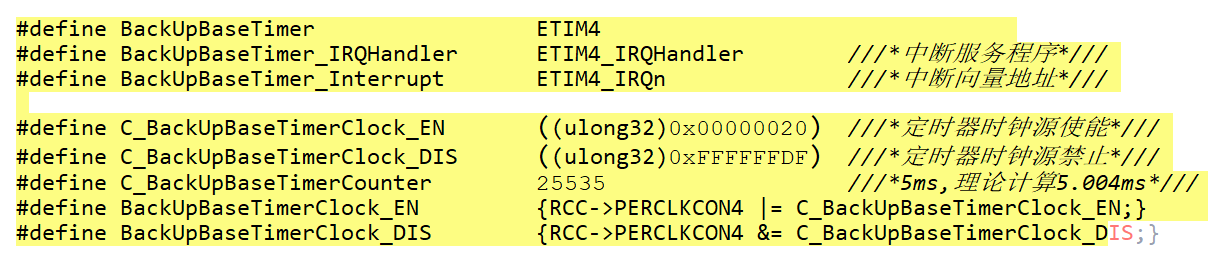
3：关闭定时器函数：void InF\_BackUpBaseTimerStop(void)

4; 返回具体应显示哪个“-”：uchar8 InF\_CountDynamicDispNum(void)

5:全局变量：去掉SV\_CountDynamicDispNum：用于计算当前应该显示几个“-”的标志

* **Timer.h文件**

去掉定时器配置相关宏定义：



* **Display.c文件**

去掉判断当前是否是全显模式的接口函数SF\_DispAllScreenMode()。

* 1. 评审结果

通过

# 异常插卡事件记录优化代码

* 1. 功能要求

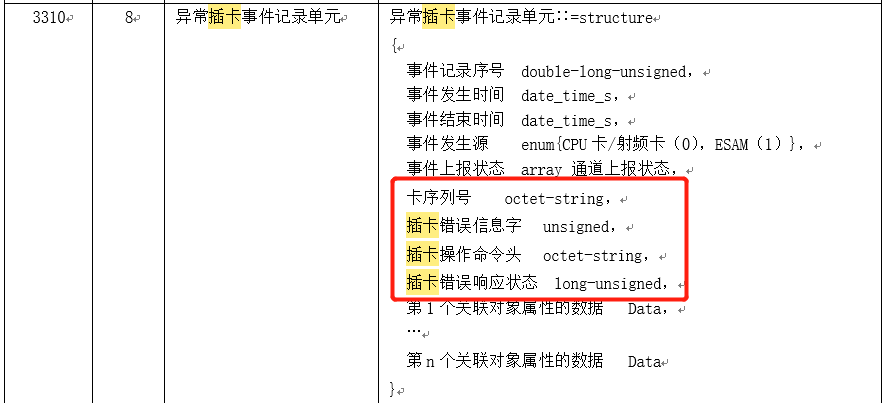
记录异常插卡事件记录，插卡前初始化异常插卡事件记录缓存结构体。

* 1. 修改设计方案：
* 记录异常插卡事件记录结构体的长度可以调整

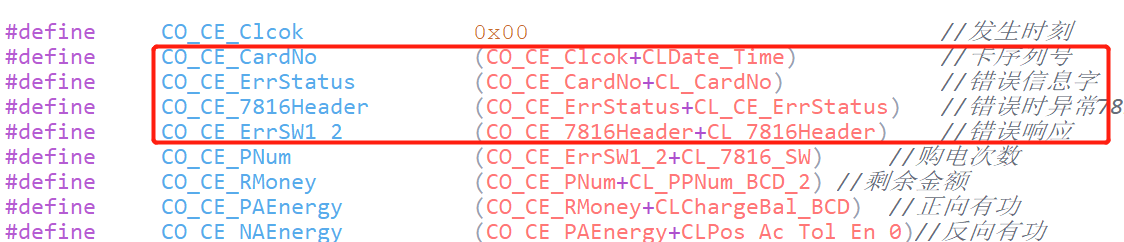


C:\Users\yblx\AppData\Local\Temp\1630975683(1).png

之前按照645格式定义的事件记录长度，修改为只保存 （卡序列号+错误信息字+操作命令头+错误响应状态）



SecurityCommunication.h文件的事件记录头偏移宏定义需要修改：



* 初始化该事件记录

只需要初始化：卡序列号+错误信息字+操作命令头+错误响应状态，之前还初始化购电总次数和剩余金额，现在不需要。

void Init\_IC\_ErrEvent(StrCardParse \*pV\_CardParse)

* 事件记录存储格式（上报状态）调整：

增加事件上报状态 array 通道上报状态的存储，存储一个字节的null；

void InF\_WRPre1ErrCardRecord(StrCardParse \*pV\_CardParse)

* 事件记录存储格式（记录序号）调整：

之前记录事件序号存储hex格式的数据，将事件记录次数读出来不进行处理（之前是进行BCD加一）

void InF\_WRPre1ErrCardRecord(StrCardParse \*pV\_CardParse)

* 非法插卡次数调整

之前存储的是BCD格式，读出来BCD+1，修改后存储hex格式，读出来直接+1。

void Write\_IC\_ErrEvent(StrCardParse \*pV\_CardParse)

* 1. 评审结果

通过

# 判断电压合法性函数合并

* 1. 功能要求

插用户卡和参数预置卡，判断三相电压都小于跳闸下限并且不在厂内，插卡失败，异常插卡事件记录记录表计电压过低。

* 1. 修改设计方案：
* Card\_Parse（）函数中：原调用判断三相电压是否都小于跳闸下限函数，调用费控模块提供的接口函数。

uchar8 INF\_JudgeVolage\_Relay(void)

* 去掉原卡文件中判断三相电压是否都小于跳闸下限函数。

uchar8 IsVolage\_Ok(void)

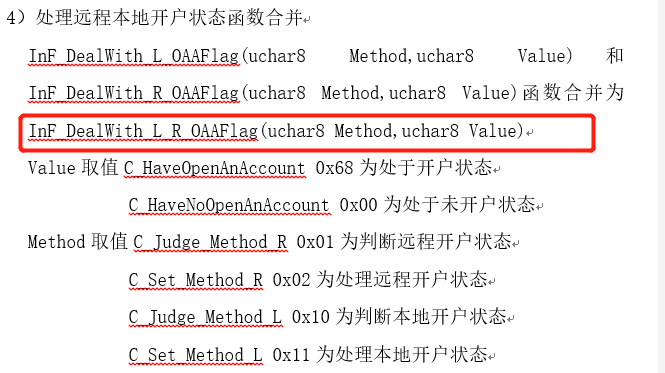
* 1. 评审结果

通过

# 本地、远程开户状态修改

* 1. 功能要求

插参数预置卡，清本地和远程开户状态。



* 1. 修改设计方案：
* uchar8 PreSetCard\_Esam\_Proc(StrCardParse \*pV\_CardParse)函数中：清远程、本地状态函数调用接口函数修改。
* uchar8 Judge\_OpenStutusAndCustNumCardNo(StrCardParse \*pV\_CardParse)函数中：判断远程和本地开户状态调用接口函数修改
* void Finish\_L\_Open(StrCardParse \*pV\_CardParse) 完成本地开户动作:保存卡号，且写本地开户状态函数中：置本地开户状态调用接口函数修改。
  1. 评审结果

通过

# 数据颠倒接口函数合并

* 1. 功能要求

插入卡片，参数更新时，由于卡内保存的是大端模式，需要对参数进行数据项颠倒再写进数据层。

* 1. 修改设计方案：
* void InF\_BackupData\_Reversal(uchar8 Type,uchar8 \*pData)函数中：调用数据颠倒的地方，统一调用公共函数PF\_ OneDataReversed(uchar8 \*pSBuf, uchar8 \*pDBuf,uchar8 Len)
* void OneDataReversed(uchar8 \*pSBuf, uchar8 \*pDBuf,uchar8 Len)函数删除
  1. 评审结果

通过

# 插卡参数更新 给通信的参数数据调整

* 1. 功能要求

插卡更新参数到数据层。将卡的数据长度、顺序和数据格式进行调整，调用通信的独立事件编程函数和写数据层。

* 1. 修改设计方案：

IC\_UpdateParm（）函数中：。

* 更新费率电价调整

12个费率电价转成hex格式写数据层，

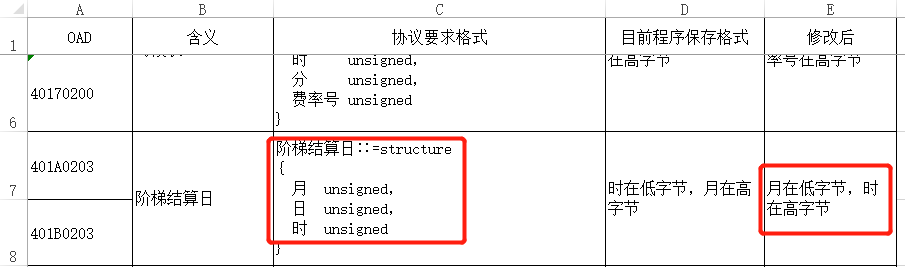
* 更新备用套费率电价调整

12个费率电价转成hex格式写数据层，两套费率切换时间转成data\_time\_s格式

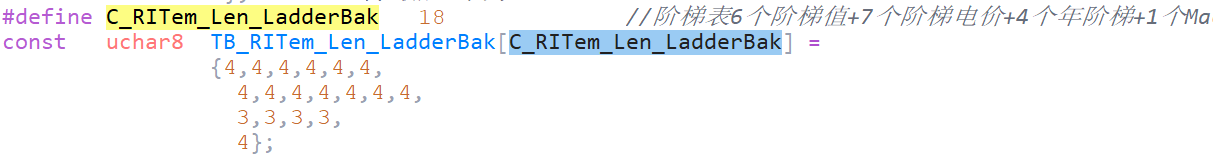
* 更新当前套阶梯电价

6个阶梯值转hex、7个阶梯电价转hex、阶梯结算日的月、日、年单个字节分别转换成hex格式，并且数据顺序是年月日，与之前645协议不一致。

通信修改如下：



SecurityModule.h中数据颠倒表格和宏定义需要改动：



* 更新备用套阶梯电价

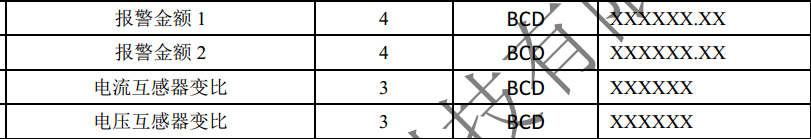
备用套阶梯电价跟当前套阶梯电价一致（如上）

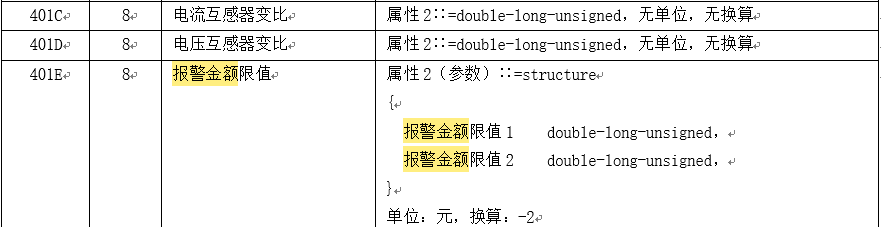
两套阶梯电价切换时间转成data\_time\_s格式

* 更新其他参数

报警金额1、2，4字节的BCD转hex

电压电流互感器变比3字节BCD转hex，补成4个字节





* 1. 评审结果

通过

# 参数预置卡和用户卡读卡文件

* 1. 功能要求

插入参数预置卡，需要读出钱包文件里面的预置金额；

插入用户卡，需要读出钱包文件里面的购电金额和购电次数；

* 1. 修改设计方案：
* uchar8 UserCard\_Esam\_Proc(StrCardParse \*pV\_CardParse)用户卡处理函数中：

读钱包文件调用接口函数修改为：

V\_ReturnData = InF\_SC\_Get\_MFile\_Card(pV\_CardParse);

* uchar8 PreSetCard\_Esam\_Proc(StrCardParse \*pV\_CardParse)参数预置卡处理函数中：

/\*读钱包文件中预置金额,698参数预制卡的钱包文件中没有购电次数\*/

读钱包文件调用接口函数修改为：

V\_ReturnData = InF\_SC\_Get\_MFile\_Card(pV\_CardParse);

* 1. 评审结果

通过

# 读剩余金额改成读对象管理层接口函数

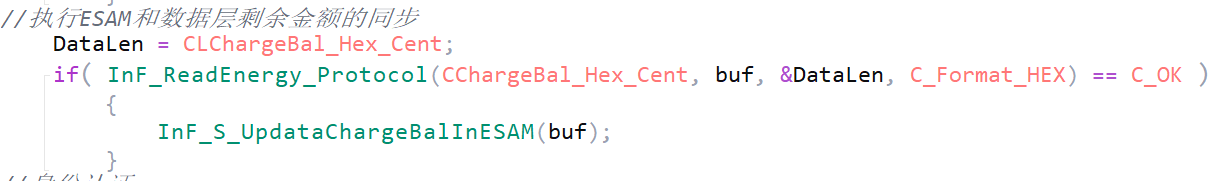
* 1. 功能要求

插入用户卡，首先先将esam内部的剩余金额和数据层的剩余金额同步：

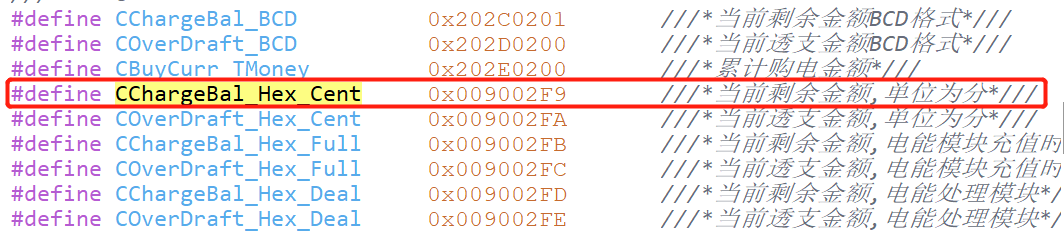
1：如果ESAM中的大于数据层，esam进行扣减

2：如果ESAM中的小于数据层则以ESAM为准，数据层剩余金额扣减

* 1. 修改设计方案：
* uchar8 UserCard\_Esam\_Proc(StrCardParse \*pV\_CardParse)函数中需要读取数据层的当前剩余金额，原来是调用电能的接口函数：



电能模块标识码宏定义



现在改成调用对象管理层接口函数：

void InF\_GetData\_ObjectManage（ulong32 V\_ulOAD， uchar8 \*pV\_ucDataBuf，ushort16 \*pV\_usDataLen,ushort16 \*P\_usDataEncode）；

返回的就是2位小数的hex格式的剩余金额，跟之前一致，程序不需要再转换。

* 1. 评审结果

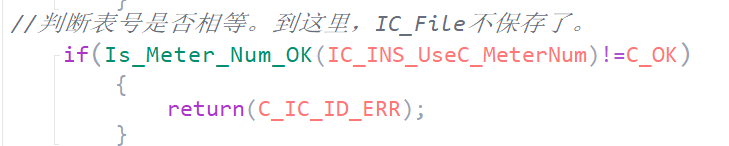
通过

# 判断表号是否相等函数

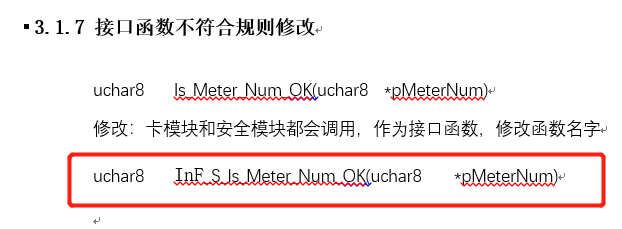
* 1. 功能要求

插入用户卡，需要判断卡的表号和电表数据层的表号是否一致。

* 1. 修改设计方案：
* uchar8 UserCard\_Esam\_Proc(StrCardParse \*pV\_CardParse)函数中调用判断表号是否相等函数：



此接口在安全模块中做了修改如下：



故调用做了修改。

* 1. 评审结果

通过

# 功能

* 1. 功能要求

插。

* 1. 修改设计方案：
* Card\_Parse（）函数中：。

uchar8 INF\_JudgeVolage\_Relay(void)

* 1. 评审结果

通过

# 功能

* 1. 功能要求

插。

* 1. 修改设计方案：
* Card\_Parse（）函数中：。

uchar8 INF\_JudgeVolage\_Relay(void)

* 1. 评审结果

通过