

1. HS300 读写器 .....	6
1.1 设备使用说明 .....	6
1.1.1 HS300 设备使用说明 .....	6
1.2 概要说明 .....	9
1.2.1 读卡通用函数 .....	10
1.2.1.1 hs_init .....	15
1.2.1.2 hs_exit.....	16
1.2.1.3 hs_card.....	17
1.2.1.4 hs_card_hex .....	18
1.2.1.5 hs_request .....	19
1.2.1.6 hs_anticoll.....	21
1.2.1.7 hs_select.....	22
1.2.1.8 hs_load_key .....	25
1.2.1.9 hs_authentication .....	26
1.2.1.10 hs_read.....	28
1.2.1.11 hs_write.....	29
1.2.1.12 hs_halt.....	31
1.2.1.13 hs_des .....	32
1.2.1.14 hs_changeb3 .....	33
1.2.1.15 hs_authentication_passaddr .....	36
1.2.1.16 hs_initval.....	38
1.2.1.17 hs_increment.....	39

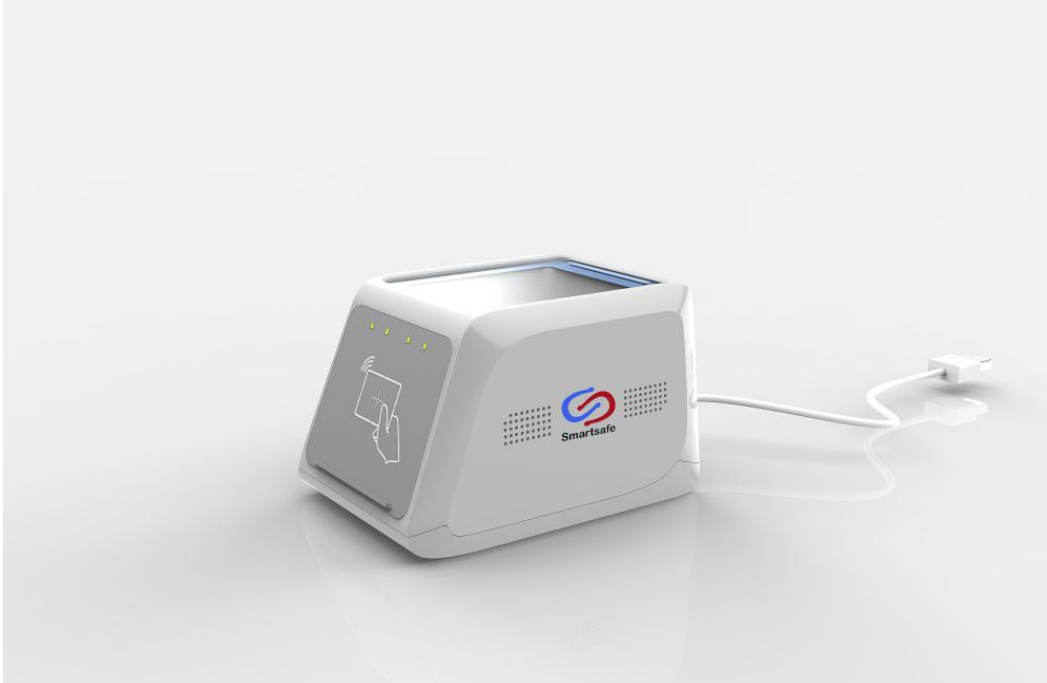
1.2.1.18	hs_readval .....	40
1.2.1.19	hs_decrement .....	41
1.2.1.20	hs_HL_authentication .....	42
1.2.1.21	hs_HL_read .....	44
1.2.1.22	hs_HL_write .....	46
1.2.1.23	hs_restore .....	48
1.2.1.24	hs_transfer .....	49
1.2.1.25	hs_authentication_2 .....	50
1.2.1.26	hs_authentication_pass .....	52
1.2.1.27	hs_card_double .....	54
1.2.1.28	hs_CheckCard .....	56
1.2.2	设备函数 .....	57
1.2.2.1	hs_beep .....	57
1.2.2.2	hs_getver .....	57
1.2.2.3	hs_srd_eeprom .....	58
1.2.2.4	hs_swr_eeprom .....	60
1.2.2.5	hs_a_hex .....	61
1.2.2.6	hs_hex_a .....	62
1.2.2.7	hs_pro_reset .....	64
1.2.2.8	hs_pro_command .....	64
1.2.3	ISO14443-4 卡专用函数 .....	65
1.2.3.1	hs_config_card .....	70

1.2.3.2	hs_pro_command.....	71
1.2.3.3	hs_pro_commandsource .....	74
1.2.3.4	hs_pro_commandlink .....	76
1.2.3.5	hs_pro_reset(TYPE A 专用) .....	78
1.2.3.6	hs_pro_halt (TYPE A 专用) .....	79
1.2.4	CPU(SAM)专用函数 .....	80
1.2.4.1	hs_setcpu.....	82
1.2.4.2	hs_setcpupara.....	83
1.2.4.3	hs_cpureset .....	84
1.2.4.4	hs_cpuapdu .....	85
1.2.4.5	hs_cpuapdusource.....	87
1.2.5	FM11RF005 专用函数.....	88
1.2.5.1	hs_read_fm11rf005.....	93
1.2.5.2	hs_write_fm11rf005 .....	94
1.2.5.3	hs_getsnr_fm11rf005.....	95
1.2.6	24C 系列卡专用函数.....	96
1.2.6.1	hs_read_24c .....	98
1.2.6.2	hs_write_24c.....	99
1.2.6.3	hs_read_24c64 .....	101
1.2.6.4	hs_write_24c64.....	102
1.2.7	4442 卡专用函数 .....	103
1.2.7.1	hs_verifypin_4442 .....	107

1.2.7.2	hs_changepin_4442 .....	108
1.2.7.3	hs_readpincount_4442 .....	110
1.2.7.4	hs_readpin_4442 .....	111
1.2.7.5	hs_read_4442 .....	113
1.2.7.6	hs_write_4442 .....	114
1.2.7.7	hs_Check_4442 .....	116
1.2.8	4428 卡专用函数 .....	117
1.2.8.1	hs_verifypin_4428 .....	120
1.2.8.2	hs_changepin_4428 .....	121
1.2.8.3	hs_readpincount_4428 .....	122
1.2.8.4	hs_readpin_4428 .....	123
1.2.8.5	hs_read_4428 .....	125
1.2.8.6	hs_write_4428 .....	126
1.2.8.7	hs_Check_4428 .....	128
1.2.9	身份证（港澳台居住证）操作函数 .....	128
1.2.9.1	hs_GetIDInfo .....	128
1.2.9.2	hs_ReadIDInfor .....	129
1.2.9.3	hs_getuid_i_d .....	131
1.2.9.4	hs_ParsePhotoInfo .....	131
1.2.9.5	hs_ReadIDInforHKMOTW .....	132
1.2.10	社保卡操作函数 .....	134
1.2.10.1	hs_ReadMFEF05 .....	134

1.2.10.2	hs_ReadMFEF06 .....	136
1.2.11	扫码接口函数 .....	138
1.2.11.1	hs_GetQrcodetimeout .....	138
1.2.11.2	hs_GetQrcode .....	138
1.2.11.3	hs_OpenQrcode .....	139
1.2.11.4	hs_CloseQrcode .....	140
1.3	附录 .....	140
1.3.1	函数错误代码 .....	140
1.3.2	判断卡型的特征值 .....	143
1.3.3	DEMO 操作说明 .....	144

# 1. HS300 读写器



## 1.1 设备使用说明

### 1.1.1 HS300 设备使用说明

#### HS300 技术性能说明

HS300是一款集成智能卡读写功能扫码平台，影像式CMOS，具有75mm\*60mm超大读窗口，适读能力强大，支持常见的一维码和二维码扫描，并支持10种以上扫码场景交易语音提示，同时可以选配智能卡读写模块及第二代居民身份证模块，可支持符合ISO7816标准的接触式IC卡和ISO14443 TypeA的非接触式IC卡，以及中华人民共和国第二代居民身份证。适用于：小微商户，银联，支付宝/微信等三方公司手机支付（可支持O2O业务），快递物流，商超，公共交通，图书馆/药店，餐饮娱乐等应用领域。

HS300 参数表

产品名称		性能说明
图像传感器		30 万像素 CMOS 传感芯片；
最大分辨率		640*480
视读 码制	二维码	DM Code, QR Code, Chinese-Sensible code PDF417, MIRCO QR code, MIRCO PDF417, GM Code, etc.
	一维码	Code 11, Code 39, Code 93, Code 128, etc.
读取方向		360°
读取速度		80ms（平均），支持连续读取
读取距离		至窗口镜面 10cm
光源		LED
提示方式		LED 灯提示，蜂鸣提示，语音提示
状态显示		4 个 LED 指示灯，指示电源、通讯、读卡、交易 等状态
语音提示		内置语音芯片，包含 10 种不同的语音提示（可 预定特殊语音）
操作系统		支持 Windows 98、Me、2K、XP、win7， win10, Linux, Android 下的部分用户二次开发
接触式卡支持		主卡座支持 1 个 ISO7816 标准卡尺寸，采用下 降式卡座，可使用 20 万次。支持的卡型是符合

	ISO7816 的异步卡如：T=0、T=1 的 CPU 卡，同步卡如常用的存储卡 AT24 系列,4442,4428 等；
非接触式卡支持	支持 ISO14443TypeB 的中华人民共和国第二代身份证卡； 支持 ISO14443TypeA 的非接触卡型支持 PHILIPS 公司的 MF1ICS50、MF1ICL10、MF1ICS70 、Mifare Pro、Mifare Ultralight、Mifare Desfire； SIEMENS 公司的 SLE44R35、SLE44R31 等
核心平台	采用 ARM 技术的开发平台
PSAM 卡接口	同时可附加 2 个符合 GSM 11.11 的 Sim 的卡尺寸 SAM 卡座
状态显示	4 个 LED 指示灯，指示电源、通讯、读卡、交易等状态
存储支持（可选）	提供 4MBIT 的可读写空间用于用户存储数据用，根据用户需求可定制存储空间空量
与 PC 通讯类型	HIDPOS,可定制 4G、GPRS 或 wifi
电源	5V,具有过压保护
外形规格	106*144*90mm
重量	420 克
外观颜色	通用如图，亦可根据用户需求定制
温度适用范围	-20 到+60° C



湿度	95%
所遵循的标准	ISO 7816、ISO14443、GSM11.11、FCC、 ROHS、CE、CCC
其他特性	提供通用接口函数库，可支持多种操作系统和 语言开发平台，支持在线升级。

产品主要应用领域：

公安、社保、市民卡、城市通卡、会员卡系统等

## 1.2 概要说明

(1) 读卡器先要用通讯口初始函数 `hs_init`，其返回值为设备标符，它  
将作为其它函数的调用参数。

(2) 调用扫码接口时，程序退出之前要执行 `hs_exit (HANDLE icdev)`  
函数，关闭串口句柄 `icdev` 否则出错。

(3) 函数调用错误类型，请参照函数错误类型代码。所有数的错误代  
码均以负数形式返回；

(4) 动态库的应该声明在相应目录中或缺省的目录当中，否则会有无  
法寻找到动态库的错。

(5) 函数的十六进制 HEX 方式调用中，入和读出的字符数组是以十进  
制字符串的方进行的，其余参数调用方式尽在详细说明中不再此列

举。

。

### 扫码相关配置：

(1) 因为硬件连接都是通过一根 USB 实现，所以扫码和读卡功能不能同步执行，需要分步实现相关功能，扫码功能只需调用

hs\_GetQrcodetimeout 即可接收扫码结果，想要读卡时再调用 hs\_init 等相关读卡器函数即可。

。

## 1.2.1 读卡通用函数

M1 卡操作调函数流程

### 1. 设备初始化

hs\_init

### 2. 寻卡

hs\_card(hs\_card\_hex)或者 hs\_card\_double(hs\_card\_double\_hex)

### 3. 装载设备密钥

hs\_loadkey

### 4. 验证密钥

hs\_authentication

### 5. 读卡

hs\_read\_hex

## 6. 写卡

hs\_write\_hex

示 例:

```
icdev=hs_init(g_comnum,115200);  
if(icdev<0)  
{  
    m_strRecvData += "init error\r\n";  
    UpdateData(FALSE);  
    CEdit *pEdit=NULL;  
    pEdit = ((CEdit*)GetDlgItem(IDC_EDIT_RECV));  
    pEdit->LineScroll(pEdit->GetLineCount()-1,0);  
  
    //dev err  
    return;  
}  
  
st=hs_card_double_hex(icdev,0x01,Snr);  
if(st==0)  
{  
    strcard.Format("Card sn is %s\r\n", Snr);  
    m_strRecvData += strcard;
```

```

    }

else

{

    m_strRecvData += "Please put the card on the right position.\r\n";

    hs_exit(icdev);

    UpdateData(FALSE);

//    OpenComPort();

    CEdit *pEdit=NULL;

    pEdit = ((CEdit*)GetDlgItem(IDC_EDIT_RECV));

    pEdit->LineScroll(pEdit->GetLineCount()-1,0);


    return;

}

memcpy(sbuff,"fffffffffff",12);

st = hs_load_key_hex(icdev,0,1,sbuff);

if(st != 0)

{

    m_strRecvData += "load key err\r\n";

    hs_exit(icdev);

    UpdateData(FALSE);

//    OpenComPort();

    CEdit *pEdit=NULL;

```

```

pEdit = ((CEdit*)GetDlgItem(IDC_EDIT_RECV));

pEdit->LineScroll(pEdit->GetLineCount()-1,0);


    return;

}

st = hs_authentication(icdev,0,1);

if(st != 0)

{

    m_strRecvData += "dc authentication error\r\n";

    UpdateData(FALSE);

    hs_exit(icdev);

//  OpenComPort();

    CEdit *pEdit=NULL;

pEdit = ((CEdit*)GetDlgItem(IDC_EDIT_RECV));

pEdit->LineScroll(pEdit->GetLineCount()-1,0);


    return;

}

//  m_strRecvData += "ss authentication ok\r\n";


//  m_strRecvData += "read info from block 4 of sector 1...";

st = hs_read_hex(icdev,4,rbuff);

```

```

if(st!=0)
{
    m_strRecvData += "read info error\r\n";
    UpdateData(FALSE);
    hs_exit(icdev);
//    OpenComPort();
    CEdit *pEdit=NULL;
    pEdit = ((CEdit*)GetDlgItem(IDC_EDIT_RECV));
    pEdit->LineScroll(pEdit->GetLineCount()-1,0);

    return;
}

//    m_strRecvData += "read info ok\r\n";

temp.Format ("Read the block4 : %s\r\n",(char *)rbuff);
m_strRecvData += temp;
//    m_strRecvData += "write '11223344556677889900aabbccddeeff'
into block 4 of sector 1\r\n";
memcpy(rbuff,"11223344556677889900aabbccddeeff",32);
st = hs_write_hex(icdev,5,rbuff);

```

### 1.2.1.1 hs\_init

HANDLE USER\_API hs\_init(short port, long baud);

功 能:

初始化通讯口

参 数:

**【in】** short port:取值为 0~19 时，表示串口 1~20；为 100 时，表示 USB 口通讯，此时波特率无效。取值 100 以上（含 100），是 usb 通信方式，接一台 usb 设备 port 为 100，接第 2 台 usb 设备 port 为 101，以此类推.....

**【in】** long baud:为通讯波特率 9600~115200

返 回:

成功则返回串口标识符>0;

失败返回负值，见错误代码表

示 例:

```
HANDLE icdev = (HANDLE)-1;
```

```
icdev=hs_init(0,9600);//初始化串口 1，波特率 9600
```

```
if((int)icdev < 0)
```

```
{
```

```

    MessageBox("打开端口失败");

    return ;

}

MessageBox("打开端口成功");

return ;

```

### 1.2.1.2 hs\_exit

short USER\_API hs\_exit(HANDLE icdev);

功 能： 关闭端口

参 数：

**【in】** HANDLE icdev:hs\_init 返回的设备描述符

返 回：

成功： 返回 0

失败： 返回非 0

示 例：

```

short st= -1;

st = hs_exit(icdev);

if(st)

{

    MessageBox("关闭端口失败");

    return;

}

```



```
MessageBox("关闭端口成功");
```

```
return;
```

注：在 WIN32 环境下 icdev 为端口的设备句柄，必须释放后才可以再次连接。

### 1.2.1.3 hs\_card

```
short USER_API hs_card(HANDLE icdev, unsigned char _Mode,  
unsigned long *_Snr);
```

功 能：寻卡，能返回在工作区域内某张卡的序列号(该函数包含了  
hs\_request,hs\_anticoll,hs\_select 的整体功能)

参 数：

**【in】HANDLE icdev：**hs\_init 返回的设备描述符

**【in】unsigned char \_Mode：**寻卡模式分三种情况：IDLE 模式、ALL  
模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

1——表示 ALL 模式，一次可对多张卡操作；

2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

mode\_card

**【out】unsigned long \*\_Snr：**返回的卡序列号

返 回：成功则返回 0

示 例：

```

short st = -1;

unsigned long snr = 0;

st=hs_card(icdev,0,&snr);

if(st)

{

    MessageBox("寻卡失败");

    return;

}

MessageBox("寻卡成功");

return;

```

注：选择 IDLE 模式，在对卡进行读写操作，执行 [hs\\_halt\(\)](#) `hs_halt` 指令中止卡操作后，只有当该卡离开并再次进入操作区时，读写器才能够再次对它进行操作。

#### 1.2.1.4 hs\_card\_hex

```

short USER_API hs_card_hex(HANDLE icdev,unsigned char
_Mode,unsigned char *snrstr)

```

功 能：寻卡，能返回在工作区域内某张卡的序列号(该函数包含了 `hs_request`,`hs_anticoll`,`hs_select` 的整体功能)

参 数：

**【in】** HANDLE icdev:`hs_init` 返回的设备描述符

【in】 unsigned char \_Mode:寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

1——表示 ALL 模式，一次可对多张卡操作；

2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）  
【out】 unsigned char \*snrstr:返回的卡序列号

返回：成功则返回 0

示例：

```
short st = -1;

unsigned long snr = 0;

st = hs_card_hex(icdev,0,&snr);

if(st)

{

    MessageBox("寻卡失败");

    return;

}

MessageBox("寻卡成功");

return;
```

注：选择 IDLE 模式，在对卡进行读写操作，执行 [hs\\_halt\(\)](#) hs\_halt 指令中止卡操作后，只有当该卡离开并再次进入操作区时，读写器才能够再次对它进行操作。

### 1.2.1.5 hs\_request

```
short USER_API hs_request(HANDLE icdev,unsigned char  
_Mode,unsigned short *TagType);
```

功 能：寻卡请求

参 数：

【in】HANDLE icdev:hs\_init 返回的设备描述符

【in】unsigned char \_Mode:寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

1——表示 ALL 模式，一次可对多张卡操作；

2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

【out】unsigned short \*TagType:卡类型值

卡型	特征值 1	特征值 2
MIFARE 1	4	136
FM11RF32	4	83
S70	2	24
ULTRA LIGHT	68	4
SHC1102	13056	
MIFARE LIGHT	16	129

MIFARE PRO	8	32
DESFIRE	836	36
FM11RF005	5	
MIFARE PLUS	180	

在执行 `hs_request` 函数时使用的 `TagType` 值是特征值 1，执行 `hs_select` 函数时使用的 `size` 值是特征值 2。

返 回：成功则返回 0

示 例：

```
short st = -1;

unsigned int tagtype;

st=hs_request(icdev,0,&tagtype);

if(st)
{
    MessageBox("寻卡失败");
    return;
}

MessageBox("寻卡成功");

return;
```

### 1.2.1.6 `hs_anticoll`

```
short USER_API hs_anticoll(HANDLE icdev, unsigned char _Bcnt,
unsigned long *_Snr);
```

功 能： 防卡冲突，返回卡的序列号

参 数：

**【in】** HANDLE icdev:hs\_init 返回的设备描述符

**【in】** unsigned char \_Bcnt:设为 0

**【out】** unsigned long \*\_Snr:返回的卡序列号地址

返 回： 成功则返回 0

示 例：

```
short st = -1;

unsigned long snr = 0;

st=hs_anticoll(icdev,0,&snr);

if(st)

{

    MessageBox("防卡冲突失败");

    return;

}

MessageBox("防卡冲突成功");

return;
```

注： request 指令之后应立即调用 anticoll，除非卡的序列号已知。

### 1.2.1.7 hs\_select

```
short USER_API hs_select(HANDLE icdev, unsigned long _Snr, unsigned
char *_Size);
```

功 能：从多个卡中选取一个给定序列号的卡

参 数：

【in】HANDLE icdev:hs\_init 返回的设备描述符

【in】 unsigned long \_Snr:卡序列号

【out】 unsigned char \*\_Size:指向返回的卡容量的数据

卡型	特征值 1	特征值 2
MIFARE 1	4	136
FM11RF32	4	83
S70	2	24
ULTRA LIGHT	68	4
SHC1102	13056	
MIFARE LIGHT	16	129
MIFARE PRO	8	32
DESFIRE	836	36
FM11RF005	5	
MIFARE PLUS	180	

在执行 hs\_request 函数时使用的 TagType 值是特征值 1，执行 hs\_select 函数时使用的 size 值是特征值 2。

返 回：成功则返回 0

示 例：

```
short st = -1;
```

```

short type = -1;

unsigned char size = 0;

unsigned long snr = 0;

st=hs_request(icdev,0,&type);

if(st)
{
    MessageBox("寻卡失败");

    return;
}

MessageBox("寻卡成功");

st=hs_antcoll(icdev,0,&snr);

if(st)
{
    MessageBox("防卡冲突失败");

    return;
}

MessageBox("防卡冲突成功");

st = hs_select(icdev,snr,&size);

if(st)
{
    MessageBox("选卡失败");

    return;
}

```



```
}  
  
MessageBox("选卡成功");
```

### 1.2.1.8 hs\_load\_key

```
short USER_API hs_load_key(HANDLE icdev, unsigned char _Mode,  
unsigned char _SecNr, unsigned char *_NKey);
```

功 能： 将密码装入读写模块 RAM 中

参 数：

**【in】** HANDLE icdev:hs\_init 返回的设备描述符

**【in】** unsigned char \_Mode:装入密码模式对于 M1 卡的每个扇区，在读写器中均对应有三套密码（KEYSET0、KEYSET1、KEYSET2），每套密码包括 A 密码（KEYA）和 B 密码（KEYB），共六个密码，用 0~2、4~6 来表示这六个密码：

0——KEYSET0 的 KEYA`

1——KEYSET1 的 KEYA

2——KEYSET2 的 KEYA

4——KEYSET0 的 KEYB

5——KEYSET1 的 KEYB

6——KEYSET2 的 KEYB

**【in】** unsigned char \_SecNr: 扇区号（M1 卡：0~15； ML 卡：0）

**【in】** unsigned char \*\_NKey: 写入读写器中的卡密码

返 回： 成功则返回 0

示 例：//装入 1 扇区的 0 套 A 密码

```
unsigned char password[7]={0xa0,0xa1,0xa2,0xa3,0xa4,0xa5};  
if((hs_load_key(icdev,0,1,password))!=0)  
{  
    printf("Load key error!");  
    hs_exit(icdev);  
}
```

相关 HEX 函数：

```
short USER_API hs_load_key_hex(HANDLE icdev,unsigned char  
_Mode,unsigned char _SecNr,unsigned char *_NKey);
```

示 例：//装入 1 扇区的 0 套 A 密码

```
unsigned char password[13]="\0";  
memcpy(password,"FFFFFFFFFFFF",12);  
if((hs_load_key_hex(icdev,0,1,password))!=0)  
{  
    printf("Load key error!");  
    hs_exit(icdev);  
}
```

### 1.2.1.9 hs\_authentication

```
short USER_API hs_authentication(HANDLE icdev, unsigned char  
_Mode, unsigned char _SecNr);
```

功 能：核对密码函数

参 数：

【in】HANDLE icdev: hs\_init 返回的设备描述符

【in】unsigned char \_Mode: 密码验证模式

对于 M1 卡的每个扇区，在读写器中均对应有三套密码（KEYSET0、KEYSET1、KEYSET2），每套密码包括 A 密码（KEYA）和 B 密码（KEYB），共六个密码，用 0~2、4~6 来表示这六个密码：

0——KEYSET0 的 KEYA`

1——KEYSET1 的 KEYA

2——KEYSET2 的 KEYA

4——KEYSET0 的 KEYB

5——KEYSET1 的 KEYB

6——KEYSET2 的 KEYB

【in】unsigned char \_SecNr: 要验证密码的扇区号

返 回：成功返回 0

示 例：

```
short st = -1;

st = hs_authentication(icdev,0,1);

if(st)
{
    MessageBox("核对密码失败");
    return;
```

```

    }

    MessageBox("核对密码成功");

```

### 1.2.1.10 hs\_read

```

short USER_API hs_read(HANDLE icdev, unsigned char _Adr, unsigned
char *_Data);

```

功 能:读卡中数据,

对于 M1 卡，一次一个块的据，为 16 个字节；

对于 ML 卡，一次读相同属性的两页（0 和 1，2 和 3，...），为 8 个字节

参 数:

**【in】** HANDLE icdev:hs\_init 返回的设备描述

**【in】** unsigned char \_Adr:

M1 卡——块地址（0-63）,MS70(0-255),ML 卡——页地址（0-

11） **【out】** unsigned char \*\_Data:读出数据

返 回：成功则返回 0

示 例:

```

short st = -1;

unsigned char data[6] = "\0";

st=hs_read(icdev,4,data); //读 1 卡块 4 的数据

if(st)
{

```

```

    MessageBox("读卡失败");

    return;
}

```

```

MessageBox("读卡成功")

```

相关 HX 函数：

```

short USER_API hs_read_hex(HANDLE icdev, unsigned char _Adr, char
*_Data);

```

### 1.2.1.11 hs\_write

```

short USER_API hs_write(HANDLE icdev, unsigned char _Adr, unsigned
char *_Data);

```

功 能：向卡中写数据

对于 M1 卡，一次必须写个块，为 1 字节；

于 ML 卡，一次必须写一页，为 1 个字节

参 数：

**【in】** HANDLE icdev:hs\_init 返回的设描述符

**【in】** unsigned char \_Adr: M1 卡——块地址（1~63）,M1S70 卡——块地址（1-255）；卡一页地址（2~11）

**【in】** unsigned char \*\_Data: 要写入的数据

返 回：成功则返回 0

示 例：

```

short st = -1;

```

```

unsigned char data[17]="\0";

memcpy(data,"\x11\x22\x33\x44\x55\x66\x77\x88\x99\x00\xaa\xbb\xcc\xdd\xee\xff",16);

st=hs_write(icdev,4,data);//写第四块

if(st)

{

    MessageBox("写卡失败);

    return;

}

MessageBox("成功");

```

关 HEX 函数:

```

short USER_API hs_write_hex(HANDLE icdev,unsigned char _Adr,char
*Data);

```

示 例:

```

int st;

char data[33]=" \0" ;

memcpy(data,"11223344556677889900abbcdddeeff",32);

st=hs_write_hex(icdev,4,data);//写第四块

if(st)

{

    MessageBox(写卡失败");

    return;

```

```
}
```

```
MessageBox("写卡成功");
```

### 1.2.1.12 hs\_halt

```
short USER_API hs_halt(HANDLE icdev);
```

功 能:中止对该卡操作

参 数:

【in】HANDLE icdev:hs\_init 返回的设备描述符

返 回: 成功则返 0

示 例:

```
short st = -1;
```

```
st=hs_halt(icdev);
```

```
if(st)
```

```
{
```

```
    MessageBox("中止对该操作败");
```

```
    return;
```

```
}
```

```
MessageBox("中止该卡操作成功");
```

说明使用 hs\_card()函数时有个\_Mode 参，如果\_Mode=0 则在对卡进行操作完毕后，执行 hs\_halt();则该卡进入 HALT 模式，必须把卡移开感应区再进来才能寻得张卡。

### 1.2.1.13 hs\_des

```
short USER_API hs_des(unsigned char *key,unsigned char *sour,unsigned  
char *dest,short m);
```

功 能： DES 算法加解函数

参 数：

**【in】** unsigned char \*key:密钥

**【in】** unsigned char \*sour:加解密数据

**【out】** unsigned char \*dest:加解密后的数据

**【in】** short m:加解密模式 m=1 时， 为加密； m=0 时， 为解密过程

返 回： 成功返回 0

示 例：

```
short st = -1;  
  
unsigned char key[9]="\0";  
  
unsigned char sour[9]="\0";  
  
unsigned char dest[9]="\0";  
  
memcpy(key,"\x11\x22\x33\x44\x55\x66\x77\x88");  
memcpy(sour,"\x1a\x2b\x3c\x4d\x5e\x6f\x17\x88");  
  
st = dcodes(key,sour,dest,1);  
  
if(st)  
{  
    MessageBox("des 加密失败");  
    return;  
}
```



```
}
```

```
MessageBox("des 密成功");
```

相关 hex 函数

```
__int6 USER_API hs_des_hex(unsigned char *key,unsigned char  
*sour,unsigned char *dest, short m);
```

示 例:

```
__int6 st = -1
```

```
unsigned char key[7]="\0";
```

```
unsigned char sour[17]="\0";
```

```
unsigned char dest[17]="\0";
```

```
memcpy(key,"1122334455667788",16);
```

```
memcpy(sour,"a2b34de6f7182",16);
```

```
st = hs_des_hex(key,sour,dest,1);
```

```
if(st)
```

```
{
```

```
    MessageBox("des 加密失败")
```

```
    return;
```

```
}
```

```
MessageBox("des 加密成功");
```

### 1.2.1.14 hs\_changeb3

```
short USER_API hs_changeb3(HANDLE icdev, unsigned char _SecNr,
```

unsigned char \*\_KeyA, unsigned char \_B0, unsigned char \_B1, unsigned char \_B2, unsigned char \_B3, unsigned char \_Bk, unsigned char \*\_KeyB);

功 能：修改块 3 的数(当为 M1S70 卡时，当扇区号大于 1 时，修改块 15 的数据(即一个扇区的最后块))

参 数：

【in】HANDLE icdev:hs\_init 返回的设备描述符

【in】unsigned char \_SecNr:扇区号（M1:0~15,M1S70:0-39）

【in】unsigned char \*\_KeyA:密码 A

【in】unsigned char \_B0:块 0 控制(当一扇区有 16 块时，对应块 0-4 的控制字)，3 位（D2D1D）对应 C10、C20、C3

【in】unsigned char \_B1:块控制字（当一扇区有 16 块时，对应为块 5-9 的控制），低 3 位（D2D10）对应 C11、C21、C31

【in】unsigned char \_B2:块 2 控制（当一扇区有 16 块时，对应为块 0-14 的控制字），3 位（D2D1D0）对应 C12、C22、C32

【in】unsigned char \_B3:块 3 控制字（当一扇区有 16 块时，对应块 15 的控制字），低 3 位（D21D0）应 C13、C23、C33

【in】unsigned char \_Bk:保留参数，取值为 0

【in】unsigned char \*\_KeyB:密码 B

返 回：成功则返回 0

示 例：

```
short st = -1;
```

```
unsigned char keya[7]="\0";
```

```

unsigned char keyb[7]="\0";

memset(keya,0xff,6);

memset(keyb,0xff,6);

st=hs_changeb3(icdev,keya,0x00,00,0x00,x01,0,keyb);

if(st)

{

    MessageBox("修改块的数失败");

    return;

}

```

MessageBox("修改块 3 的数据成功");

相关 Hex 函数

```

short USER_API hs_changeb3_hex(HANDLE icdev,unsigned char

_SecNr, const char *_KeyA,unsigned char _B0,unsigned char

_B1,unsigned char _B2,unsigned char _B3,unsigned char _Bk, const char

*_KeyB);

```

示 例:

```

short st = -1;

unsigned char keya[13]="\0";

unsigned char keyb[13]="\0";

memset(keya,'F',12);

memset(keyb,'F',12);

st=hs_changeb3_hex(icdev,keya,0x00,00,0x00,x01,0,keyb);

```

```

if(st)
{
    MessageBox("修改块的数失败");
    return;
}
MessageBox("修改块 3 的数据成功");

```

### 1.2.1.15 hs\_authentication\_passaddr

short USER\_API hs\_authentication\_passaddr(HANDLE icdev, unsigned char \_Mode, unsigned char \_Addr, unsigned char \*passbuff);

功 能：核对密码函数，用此函数时，可以不用执行 dc\_load\_key() 函数

参 数：

**【in】** HANDLE icdev: hs\_init 返回的备描述符

**【in】** unsigned char \_Mode: 对 M1 的每个扇区，在读写器中均对应三套码（KEYSET0、EYST1、KEYSET2），每套密码包括 A 密码（KEYA）和 B 密码（KEYB），共六个密码，用 0~2、4~6 来表示这六个密码：

0——KEYSET0 的 KEYA

1——KEYSET1 的 KEYA

2——EYSET2 的 KEYA

4——KEYSET0 的 KEYB

5—KESET1 的 KEYB

6——KEYSET2 的 KEYB

**【in】** unsigned char \_Addr: 要验证密码的块地址号

**【in】** unsigned char \*passbuff:密码字符串

返 回: 成功返回 0

示 例:

```
short st = -1;
```

```
unsigned char passbuff[7] = "\0";
```

```
passbuff[0] = 0xFF;
```

```
passbuff[1] = 0xFF;
```

```
passbuff[2] = 0xFF;
```

```
passbuff[3] = 0xFF;
```

```
passbuff[4] = 0xFF;
```

```
passbuff[5] = 0xFF;
```

```
st = hs_authentication_passaddr(icdev,0,1,passbuff);
```

```
if(st)
```

```
{
```

```
    MessageBox("校密码失败);
```

```
    return;
```

```
}
```

```
MessageBox("验密码成功");
```

```
return;
```

相关 HEX 函数:

```
short USER_API hs_authentication_passaddr_hex(HANDLE  
icdev,unsigned char _Mode,unsigned char _Addr,unsigned char  
*passbuff);
```

示 例:

```
short st = -1;  
  
unsigned char passbuff[13]= "\0";  
  
memcpy(passbuff,"FFFFFFFFFFFF",12);  
  
st = hs_authentication_passaddr_hex(icdev,0,1,passbuff);  
  
if(st)  
{  
    MessaeBox("校验密码失败");  
    return;  
}  
  
MessgeBox("校验密码成功");  
  
return;
```

### 1.2.1.16 hs\_initval

```
short USER_API hs_initval(HANDLE icdev, unsigned char _Adr,  
unsigned long _Value);
```

功 能: 初始化块值

参 数:

【in】HANDLE icdev: hs\_init 返回的备描述符

【in】unsigned char \_Adr: 块地址

【in】unsigned long \_Value: 初始值

返 回: 成功则返 0

示 例://将块 1 的值初始化为 1000

```
short st = -1;

unsigned long value = 0;

value=1000;

st=hs_initval(icdev,1,value);

if(st)

{

    MessageBox("初始块值失败");

    return;

}

MessageBox("初始化块值成功");

return;
```

注: 在进行值操作时必须先执行初始化值函数, 然后才可以读、减、的操作。

### 1.2.1.17 hs\_increment

```
short USER_API hs_increment(HANDLE icdev, unsigned char _Adr,
```

unsigned long \_Value);

功 能：块加值

参 数：

【in】HANDLE icdev: hs\_init 返回的设备描述符

【in】unsigned char \_Adr: 块地址

【in】unsigned long \_Value: 要增加的值

返 回：成功则返回 0;

示 例://将块的增加 value

```
short st = -1;
```

```
unsigned long value = 0;
```

```
value=10;
```

```
st=hs_increment(icdev,1,value);
```

```
if(st)
```

```
{
```

```
    MessageBox("块加值失败");
```

```
    return;
```

```
}
```

```
MessageBox("块加值成功");
```

```
return;
```

### 1.2.1.18 hs\_readval

```
short USER_API hs_readval(HANDLE icdev,unsigned char
```



`_Adr,unsigned long *_Value);`

功 能： 读值

参 数：

**【in】** HANDLE icdev: `hs_init` 返回的设备描述符

**【in】** unsigned char `_Adr`:块地址

**【in】** unsigned long `*_Value`:读出值的地址

返 回： 成功则返回 0

示 例： //读出块 1 的值， 放 value

```
short st = -1;
```

```
unsigned long value = 0;
```

```
st=hs_readval(icdev,1,&value);
```

```
if(st)
```

```
{
```

```
    MessageBox("读块值失败");
```

```
    return;
```

```
}
```

```
MessageBox("读块值成功");
```

```
return;
```

### 1.2.1.19 `hs_decrement`

```
short USER_API hs_decrement(HANDLE icdev, unsigned char _Adr,
```

```
unsigned long _Value);
```

功 能：块减值

参 数：

【in】HANDLE icdev: hs\_init 返回的备描述符

【in】unsigned char \_Adr: 块地址

【in】unsigned long \_Value: 要减的值

返 回:成功则返回 0

示 例：//将块 1 的值减少 value

```
short st = -1;
```

```
unsigned long value = 0;
```

```
value=10;
```

```
st=hs_decrement(icdev,1,value);
```

```
if(st)
```

```
{
```

```
    MessageBox("块减值失败");
```

```
    return;
```

```
}
```

```
MessageBox("块值成功");
```

```
return;
```

### 1.2.1.20 hs\_HL\_authentication

```
short USER_API hs_HL_authentication(HANDLE icdev, unsigned char
```

```
reqmode, unsigned long snr, unsigned char authmode, unsigned char
```

secnr);功 能：高级证（无需调用寻卡函数）

参 数：

**【in】HANDLE icdev:** hs\_init 返回的设备描述符

**【in】unsigned char reqmode:** 寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

1——表示 ALL 模式，一次可对多张卡操作；

2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

**【in】unsigned long snr:** 卡序列号（在寻卡模式为 2 时使用）

**【in】unsigned char authmode:** 对于 M1 卡的每个扇区，在读写器中均对应有三套密码（KEYSET0、KEYSET1、KEYSET2），每套密码包括 A 密码（KEYA）和 B 密码（KEYB），共六个密码，用 0~2、4~6 来表示这六个密码：

0——KEYSET0 的 KEYA`

1——KEYSET1 的 KEYA

2——KEYSET2 的 KEYA

4——KEYSET0 的 KEYB

5——KEYSET1 的 KEYB

6——KEYSET2 的 KEYB

**【in】unsigned char secnr:** 扇区号

返 回:成功则返 0

示 例：

```
short st = -1;

st = hs_HL_authentication(icdev,0,snr,0,1);

if(st)

{

    MessageBox("密验证败");

    return;

}

MessageBox("密码成功");

return;
```

### 1.2.1.21 hs\_HL\_read

```
short USER_API hs_HL_read(HANDLE icdev, unsigned char _Mode,
unsigned char _Adr, unsigned long _Snr, unsigned char *_Data, unsigned
long *_NSnr);
```

功 能：高级读函数

参 数：

**【in】HANDLE icdev：**hs\_init 返回的设备描述符

**【in】unsigned char \_Mode：**寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

- 1——表示 ALL 模式，一次可对多张卡操作；  
2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

**【in】** unsigned char \_Adr: 块地址

**【in】** unsigned long \_Snr: 卡的序列号

**【out】** unsigned char \*\_Data: 读出的数据

**【out】** unsigned long \*\_NSnr: 返回卡的序列号

返 回：成功则返回 0

示 例：

```
unsigned char _Mode = 1;
unsigned char _Adr = 5;
unsigned long snr;
unsigned char HLdata[50] = "\0";
unsigned long Rsnr = 0;
if((hs_HL_read(icdev, _Mode, _Adr, snr, HLdata, &Rsnr))!=0)
{
    MessageBox("read HL Rvalue wrong");
}
```

相关 HEX 函数：

```
short USER_API hs_HL_readhex(HANDLE icdev, unsigned char _Mode,
unsigned char _Adr, unsigned long _Snr, unsigned char *_Data, unsigned
long *_NSnr);
```

```
short USER_API hs_HL_read_hex(HANDLE icdev, unsigned char  
_Mode, unsigned char _Adr, unsigned long _Snr, unsigned char *_Data,  
unsigned long *_NSnr);
```

### 1.2.1.22 hs\_HL\_write

```
short USER_API hs_HL_write(HANDLE icdev, unsigned char _Mode,  
unsigned char _Adr, unsigned long *_Snr, unsigned char *_Data);
```

功 能：高级写函数

参 数：

**【in】HANDLE icdev：**hs\_init 返回的设备描述符

**【in】unsigned char \_Mode：**寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

1——表示 ALL 模式，一次可对多张卡操作；

2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

**【in】unsigned char \_Adr：**块地址

**【in】unsigned long \*\_Snr：**卡的序列号地址

**【in】unsigned char \*\_Data：**写入的数据

返 回：成功则返回 0

示 例：

```
unsigned char _Mode = 1;
```

```

unsigned char _Adr = 5;

unsigned long _Snr = 0;

unsigned char HLdata[50] = "\0";

memcpy(HLdata,"\x11\x22\x33\x44\x55\x66\x77\x88\x99\xaa\xbb\xcc\xdd\xee\xff",16);

if((hs_HL_write(icdev,1,5,&Snr,HLdata))!=0)
{
    printf("hs_HL_write wrong");
}

```

相关 HEX 函数:

```

short USER_API hs_HL_writehex(HANDLE icdev, unsigned char _Mode,
unsigned char _Adr, unsigned long *_Snr, unsigned char *_Data);

short USER_API hs_HL_write_hex(HANDLE icdev, unsigned char
_Mode, unsigned char _Adr, unsigned long *_Snr, unsigned char *_Data);

```

示 例:

```

unsigned char _Mode = 1;

unsigned char _Adr = 5;

unsigned long _Snr = 0;

unsigned char HLdata[50] = "\0";

memcpy(HLdata,"112233445566778899aabbccddeeff",32);

if((hs_HL_write_hex(icdev,1,5,&Snr,HLdata))!=0)
{

```

```

    printf("hs_HL_write_hex wrong");
}

返 回： 成功返回 0

示 例：

unsigned char _Mode = 1;

unsigned har _Adr = 5;

unsigned lng _Snr = 0;

unsigned char HLdata[50] = "\0";

Memcpy(HLdta,"\x11\x22\x33\x44\x55\x67\x77\x88\x99\xaa\xbb\xcc\xdd\xee\xff",16);

if((d_HL_write(icdev,1,5,&Snr,HLdata))!=0)
{
    printf("hs_HL_write wrong");
}

```

### 1.2.1.23 hs\_restore

short USER\_API hs\_restore(HANDLE icdev, unsigned char \_Adr);

功 能： 回传函数，将 EEPROM 中的容传卡的内部寄存器

参 数：

**【in】** HANDLE icdev: hs\_init 返回设备描述符

**【in】** unsigned char \_Adr: 要进行回传的块地址

返 回： 成功返回 0



示 例：

```
short st= -1;
```

```
st = hs_restore(icdev,1);
```

```
if(st)
```

```
{
```

```
    MessageBox("回传失败");
```

```
    return;
```

```
}
```

```
MessageBox("回传成功");
```

```
return;
```

```
st = hs_transfer(icdev,2);
```

```
if(st)
```

```
{
```

```
    MessageBox("传送败");
```

```
    return;
```

```
}
```

```
MessageBox("传送成功");
```

注：用此函数某一块中的数值传入内寄存器，然用 `hs_transfer()` 函数将寄存器数再送到另一块中去，现块与块之间数传。函数只用于值块。

### 1.2.1.24 hs\_transfer

```
short USER_API hs_transfer(HANDLE icdev, unsigned char _Adr);
```

功 能：传送，将寄存器的内容传送到 EEPROM 中

参 数：

**【in】** HANDLE icdev: hs\_init 返回的备描述

**【in】** unsigned char \_Adr: 要送的块地址

返 回：成功返 0

示 例：//将块 1 的内容传到块 2

```
short st= -1;
```

```
st = hs_restore(icdev,1);
```

```
if(st)
```

```
{
```

```
    MessageBox("回传失败");
```

```
    return;
```

```
}
```

```
MessageBox("回传成功");
```

```
return;
```

```
st = hs_transfer(icdev,2);
```

```
if(st)
```

```
{
```

```
    MessageBox("传送失败");
```

```
    return;
```

```
}
```

```
MessageBox("传送成功");
```

## 1.2.1.25 hs\_authentication\_2

short USER\_API hs\_authentication\_2(HANDLE icdev, unsigned char  
\_Mode, unsigned char KeyNr, unsigned char Adr);

功 能：验证密码

参 数：

**【in】** HANDLE icdev: hs\_init 返回的设备描述符

**【in】** unsigned char \_Mode: 对于 M1 卡的每个扇区，在读写器中均  
对应有套密码 KEYSET0、KEYSET1、KEYSET2），套密码包括密码  
（KE）和 B 密码（KYB），共六个密码，用 0~2、4~6 来示这个密  
码：

0——KEYSET0 的 KEYA

1——KEYSET1 的 KEYA

2——KEYSET2 的 KEYA

4——KEYSET0 的 KEYB

5——KEYSET1 的 KEYB

6——KEYSET2 的 KEYB

**【in】** unsigned char KeyNr: 扇地址

**【in】** unsigned char Adr: 块地址

返 回：成功则返回 0

示 例：

```
short st = -1;
```

```
st=hs_authentication_2(icdev,0,3,3);
```

```

if(st)
{
    MessageBox("卡验证密码失败");

    return;
}
MessageBox("卡验证密码成功");

```

### 1.2.1.26 hs\_authentication\_pass

short USER\_API hs\_authentication\_pass(HANDLE icdev, unsigned char \_Mode, unsigned char \_Addr, unsigned char \*passbuff);

功 能:核对密码函数，用此函数时，可不用执行 hs\_load\_key() 函数  
(只支持 M1 和 S70 卡)

参 数:

**【in】** HANDLE icdev: hs\_init 返回的设备描述符

**【in】** unsigned char \_Mode:对于 M1 卡每个扇区，在读写器中均对应有三套密码（KEYSE、KEYSET1、KEYSE2，每套密码包括 A 密（EYA）和 B 密码（KEYB），共六个密码，用 0~2、4~6 来示这六个密码:

0——KEYSET0 的 KEYA

1——KEYSE1 的 KEYA

2——EYSET2 的 KEYA

4——KEYSET0 的 KEB

5——KEYSET1 的 KEYB

6——KYSET2 的 KEYB

【in】 unsigned char \_Addr: 要验证密码的扇区号

【in】 unsigned char \*passbuff:密码字符串

返 回: 成返回 0

示 例:

```
short st = -1;
```

```
unsigned char password[7]={0xff,0xff0xff,0xff,0xff,xff};
```

```
st = hs_authentication_pass(icdev,,0,psswr);
```

```
if(st)
```

```
{
```

```
    MessageBox("核对密码失败");
```

```
    return;
```

```
}
```

```
MessageBox("核对密码成功");
```

```
return;
```

相关 HEX 数:

```
short USER_API hs_authentication_pass_hex(HANDLE icdev,unsigned
```

```
char _Mode,unsigned char _Addr,unsigned char *passbuff);
```

示 例:

```
short st = -1;
```

```
unsigned char password[13]="\0";
```

```

emcy(password,"FFFFFFFF",12);

st = hs_authentication_pass_hex(icdev,0,0,password);

if(st)

{

    MessageBox("对密码失败");

    return;

}

MessageBox("核对密码成功");

return;

```

### 1.2.1.27 hs\_card\_double

```

short USER_API hs_card_double(HANDLE icdev, unsigned char _Mode,
unsigned char *_Snr);

```

功 能：寻和 hs\_card 差别为多调了 d\_anticoll2，hs\_select2

参 数：

**【in】HANDLE icdev：** hs\_init 返回的备描述符

**【in】unsigned har \_Mode：** 寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。

0——表示 IDLE 模式，一次只对一张卡操作；

1——表示 ALL 模式，一次可对多张卡操作；

2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

**【out】** unsigned char \_Snr: 返回的卡序列(8 节)

返 回: 成功则返回 0

示 例:

```
short st = -1;

unsigned char snr[8] = "\0";

st=hs_card_double(icdev,0,snr);

if(st)

{

    MessageBox("寻失败");

    return;

}

MessaeBox("寻卡成功");

return;
```

注:选择 IDLE 模式,在对卡进行读写操作,执行 hs\_halt()hs\_halt 指令中止卡操作后,只有当该卡离开再次进入操作区时,读写器才能再次对它进行作。

关 HEX 函数:

```
short USER_API hs_card_double_hex(HANDLE icdev, unsigned char

_Mode, unsigned char *_Snr);
```

示 例:

```
short st = -1;

unsigned char snr[17] = \0";
```

```
st=hs_card_double_hex(icdev,0,snr);

if(st)

{

    MessageBox("寻卡失败");

    return;

}

MessageBox("寻卡成功");

return;
```

1.2.1.28 hs\_CheckCard

```
short USER_API hs_CheckCard(HANDLE icdev);
```

功 能： 判断接触式卡型

参 数：

【in】HANDLE icdev： hs\_init 通讯设备标符

返回：

返回值	卡类型
8	4442 卡
9	4428 卡
30	T=0 的 CPU 卡
31	T=1 的 CPU
21	24C01 卡
22	24C02 卡



23	24C04 卡
24	24C08 卡
25	24C16 卡
26	24C64 卡

示 例:

```
short j = hs_CheckCard(m_icdev);

return j;
```

## 1.2.2 设备函数

### 1.2.2.1 hs\_beep

```
short USER_API hs_beep(HANDLE icdev, unsigned short _Msec);
```

功 能：蜂鸣

参 数：

**【in】** HANDLE icdev: hs\_init 返回的设备描述符

**【in】** unsigned short \_Msec: 蜂鸣时间，位是 10 毫秒

返 回：成功则返回 0

示 例：//鸣叫 100 毫秒

```
short st = -1;

st=hs_beep(icdev,10);

if(st)

{

    MessageBox("蜂鸣失败);
```

```

        return;
    }
    MessageBox("蜂鸣成")
    return;

```

### 1.2.2.2 hs\_getver

```
short USER_API hs_getver(HANDLE icdev, unsigned char *sver);
```

功 能：读取硬件版本

参 数：

**【in】** HANDLE icdev:hs\_init 返回的设备描述符

**【out】** unsigned char \*sver: 存放版本号的缓冲区，

返 回：成功则回 0

示 例：

```

short st = -1;

unsigned char buf[5] = \0";

st = hs_getver(icdev,buf);

if(st)
{
    MessgeBox("读取件版本号失败");

    return;
}

MessageBox("读取件版本号成功");

```

return;

### 1.2.2.3 hs\_srd\_eeprom

short USER\_API hs\_srd\_eeprom(HANDLE icdev, short offset, short lenth,  
unsigned char \*rec\_buffer);

功 能：读取读写器备注信息

参 数：

【in】HANDLE icdev:hs\_init 返回的设备描述符

【in】short offset:偏移地址（0~178）

【in】short lenth:读取信息长度（1~1279）

【out】unsigned char\* receive\_buffer:读取到的信息

返 回：功则返回 0

示 例：

```
short st = -1;
```

```
unsigned char buffer[10] = "\0";
```

```
st=hs_srd_eeprom(icdev,0,10,buffer);
```

```
if(st)
```

```
{
```

```
    MessageBox("读读写器备信息失");
```

```
    return;
```

```
}
```

```
MesaeBox("读读写器备注信息成功");
```

return;

相关 HEX 函数

```
short USER_API hs_srd_eepromhex(HANDLE icdev, short offset, short  
lenth, unsigned char* receive_buffer);
```

```
short USER_API hs_srd_eeprom_hex(HANDLE icdev, short  
offset, short lenth, unsigned char *rec_buffer);
```

### 1.2.2.4 hs\_swr\_eeprom

```
short USER_API hs_swr_eeprom(HANDLE icdev, short offset, short  
lenth, unsigned char *send_buffer);
```

功 能： 向写器备注区中写入信息

参 数：

**【in】** HANDLE icdev: hs\_init 返回设备描述符

**【in】** short offset: 偏移地址（0～178）

**【in】** short lenth: 写入信息度（1～1279）

**【in】** unsigned char\* send\_buffer: 要写入的息

返 回： 成功则返回 0

示 例：

```
short st = -1;
```

```
unsigned char buffer[100] = \0";
```

```
memcpy(buffer, "x31\x32\x33\x34\x35\x36"6);
```

```
st=hs_swr_eeprom(icdev,0,6,buff);
```

```

if(st)
{
    MessageBox("向读写器备注区中写入信息失败");

    return;
}

```

```

MessgeBox("向读写器备注区中写入信息成功;

return;

```

相关 HEX 函数:

```

short USER_API hs_swr_eepromhex(HANDLE icdev,short offset,short
lenth,unsigned char* send_buffer);

```

```

short USER_API hs_swr_eeprom_hex(HANDLE icdev,short
offset,short lenth,unsigned char* send_buffer);

```

示 例:

```

short st = -1;

unsigned char buffer[10] = "\0";

memcpy(buffer,"31323343536",12);

st=hs_swr_eeprom_hex(icdev,0,6,buffer);

```

```

if(st)
{
    MessageBox("向读写器备注区中入信息失败");

    return;
}

```

```
MessageBox("向读写器备注区中写入信息成功");
```

```
return;
```

### 1.2.2.5 hs\_a\_hex

short USER\_API hs\_a\_hex( unsigned char \*a, unsigned char \*hex, short len);功 能： 字符串转换数，十进字符转换成通字符(长转短)。

参 数：

**【in】** unsigned char \*a : 要转换的字符

**【out】** unsigned char \*hex: 转换后的字符

**【in】** short len: 字符 a 长度

返 回： 成功则返回 0

示 例：

把"31323343536"12 字节转换成 “x31\x32\x3\x34\x35\x36” 6 个字节

```
short st = -1;
```

```
unsigned char a[13] = "\0";
```

```
memcpy(a,"313233343536",12);
```

```
unsigned char hex[7] = "\0";
```

```
st = a_hex(a,hex,12);
```

```
if(st)
```

```
{
```

```
    MessageBox("字符串转换失败");
```

```
    return;
```

```

}

MessageBox("字符串转换成功");

return;

```

### 1.2.2.6 hs\_hex\_a

```

short USER_API hs_hex_a (unsigned char *hex, unsigned char *a, short
length);

```

功 能：字符串转函数，普通字转换成十六进制字符(短转长)。

参 数：

**【in】** unsigned char \*hex: 要转换的字符

**【out】** unsigned char \*a: 转换的字符

**【in】** short length: 字符 hex 的长度

返 回：功则返回 0

示 例：

把 “x1\x32\x33\x34\x35\x36” 6 个字节转成"3132343536"12 个字节

```

short st = -1;

unsigned char hex[7] = "\0"

memcpy(hex, "\x31\x32\x33\x34\x35\x36", 6);

unsigned char a[13] = "\0";

st = a_hex(ex, a, 6);

if(st)
{

```

```

    MessageBox("字符串转换失败");

    return;
}

MessageBox("字符串转换成功");

return;

```

### 1.2.2.7 hs\_pro\_reset

- \* @brief 非接触式 CPU 卡复位。
- \* @par 说明：
- \* 对感应区 CPU 卡进行复位操作。
- \* @param[in] icdev 设备标识符。
- \* @param[out] rlen 返回复位信息的长度。
- \* @param[out] receive\_data 返回的复位信息，请至少分配 128 个字节。
- \* @return <0 表示失败，==0 表示成功。
- \*/

```

short USER_API hs_pro_reset(HANDLE icdev, unsigned char *rlen,
unsigned char *receive_data);

```

### 1.2.2.8 hs\_pro\_command

```

short USER_API hs_pro_command(HANDLE icdev, unsigned char slen,
unsigned char *sendbuffer, unsigned char *rlen, unsigned char *databuffer,

```



```
unsigned char timeout);
```

参 数:

对感应区 CPU 卡进行指令交互操作，注意此接口已封装卡协议部分。

\* @param[in] icdev 设备标识符。

\* @param[in] slen 发送数据的长度。

\* @param[in] sendbuffer 发送数据。

\* @param[out] rlen 返回数据的长度。

\* @param[out] databuffer 返回的数据。

\* @param[in] timeout 超时值，此值只在部分设备的底层使用，单位为 250 毫秒，一般调用建议值为 7。

返 回： <0 错误。其绝对值为错误

=0 成功

示 例:

```
short st = -1;
```

```
unsigned char buf[128];
```

```
st = hs_cpureset (icdev,buf);
```

## 1.2.3 ISO14443-4 卡专用函数

TypeA 卡操作

1.设置卡型

```
hs_config_card
```

## 2.寻卡

hs\_card 或者 hs\_card\_double\_hex

## 3.卡片复位

hs\_pro\_reset

## 4.卡片发指令

hs\_pro\_commandlink

示 例:

```
HANDLE icdev = (HANDLE)-1;
```

```
icdev=hs_init(100,115200);
```

```
if((int)icdev<0)
```

```
{
```

```
    MessageBox("打开端口失败");
```

```
    return;
```

```
}
```

```
short st = -1;
```

```
char tmp[256],recdata[512];
```

```
unsigned long _Snr=0;
```

```
unsigned char slen,rlen,sdata[100],rdata[512];
```

```
memset(tmp,0,sizeof(tmp));
```

```
memset(recdata,0,sizeof(recdata));
```

```
memset(sdata,0,sizeof(sdata));
```

```
memset(rdata,0,sizeof(rdata));
```

```

st=hs_config_card(icdev,'A');

if(st)

{

    MessageBox("选卡失败");

    return ;

}

st=hs_card(icdev,0,&_Snr); //一次一卡模式下寻卡;

if(st!=0)

{

    MessageBox("寻卡失败");

    return ;

}

st=hs_pro_resethex(icdev,&rln,recdata); //上电复位，返回信息长度，
和存放。

if(st!=0)

{

    MessageBox("卡片复位失败");

    return ;

}

recdata[2*rln]='\0'; //16 进制函数，复位信息存放数组要求 2 倍；同
时字符串末尾放空字节。

st=hs_pro_commandlink(icdev,5,(unsigned char

```

```
*)"x00\x84\x00\x00\x08",&rlen,(unsigned char *)rdata,7,56);
```

```
if(st)
```

```
{
```

```
    MessageBox("CPU 卡片发送指令失败");
```

```
    return ;
```

```
}
```

TypeB 卡操作

1.设置卡型

```
hs_config_card
```

2.寻卡

```
hs_request_b
```

3.卡复位

```
hs_attrib
```

4.卡发送指令

```
hs_pro_commandlink
```

示 例：

```
HANDLE icdev = (HANDLE)-1;
```

```
icdev=hs_init(100,115200);
```

```
if((int)icdev<0)
```

```
{
```

```
    MessageBox("打开端口失败");
```

```
    return;
```

```

}

short st = -1;

char tmp[256],recdata[512];

unsigned char slen,rlen,sdata[100],rdata[512];

st=hs_reset(icdev,10);

st=hs_config_card(icdev,'B');

if(st)

{

    MessageBox("选 B 卡失败");

    return;

}

st = hs_request_b(icdev,0,0,0,(unsigned char *)tmp);

if (st)

{

    MessageBox("寻 B 卡失败");

    return;

}

rlen = 13;

hex_a((unsigned char *)tmp,(unsigned char *)recdata,rlen);

st = hs_attrb(icdev,(unsigned char *)&tmp[1],0);

if (st)

{

```

```

    MessageBox("B 卡复位失败");

    return;

}

st=hs_pro_commandlink(icdev,5,(unsigned char
*)"x00\x84\x00\x00\x08",&rln,(unsigned char *)rdata,7,56);

if(st)
{
    MessageBox("B 卡发指令失败");

    return;

}

```

### 1.2.3.1 hs\_config\_card

```

short USER_API hs_config_card(HANDLE icdev, unsigned char
cardtype);

```

功 能： 设置读写器将要对哪一种卡操作，读写器上电缺省的时候是对 TYPEA 操作。

参 数：

**【in】** HANDLE icdev: hs\_init 返回的设备描述符；

**【in】** unsigned char cardtype: 当为'A'的时候表示设置读写器对 TYPE A 操作，'B'表示对 TYPE B 操作

返 回： 成功则返回 0；

示 例：

```

short st = -1;

st= hs_config_card ( icdev,'A');//设置对 TYPE A 卡操作

if (st)

{

    AfxMessageBox("设置 TYPE A 卡错误");

    return;

}

AfxMessageBox("设置 TYPE A 卡成功");

short st = -1;

st= hs_config_card ( icdev,'B');//设置对 TYPE B 卡操作

if (st)

{

    AfxMessageBox("设置 TYPE B 卡错误");

    return;

}

AfxMessageBox("设置 TYPE B 卡成功");

```

### 1.2.3.2 hs\_pro\_command

```

short USER_API hs_pro_command(HANDLE icdev, unsigned char slen,
unsigned char *sendbuffer, unsigned char *rlen, unsigned char *databuffer,
unsigned char timeout);

```

功 能：应用协议数据单元信息交换函数。该函数已封装 T=CL 操作

参 数:

【in】HANDLE icdev : hs\_init 函数返回的端口标识符

【in】unsigned char slen : 发送的信息长度

【in】unsigned char \* sendbuffer : 存放要发送的信息

【out】unsigned char \*rlen : 返回信息的长度

【out】unsigned char \* databuffer : 存放返回的信息

【in】unsigned char timeout : 延迟时间, 单位为: 10ms

返 回: <0 错误。其绝对值为错误号

=0 成功。

示 例:

```
short st = -1;

unsigned char slen,rlen,snddata[100], recdata[100];

slen=5;

snddata[0]=0x00;snddata[1]=0x84;snddata[2]=0x00;

snddata[3]=0x00;snddata[4]=0x04;

st= hs_pro_command ( icdev,slen,snddata,&rlen,recdata,7);

if (st)

{

    AfxMessageBox("信息交换错误");

    return;
```



```
}
```

```
AfxMessageBox("信息交换成功");
```

对卡发取随机数命令

相关 HEX 函数:

```
short USER_API hs_pro_commandhex(HANDLE idComDev,unsigned
```

```
char slen, char * sendbuffer,unsigned char *rlen, char *
```

```
databuffer,unsigned char timeout) ;
```

示 例:

```
short st = -1;
```

```
unsigned char slen,rlen,senddata[100], recdata[100];
```

```
slen=5;
```

```
memcpy(senddata,"0084000008",10);
```

```
st= hs_pro_commandhex( icdev,slen,senddata,&rlen,recdata,7);
```

```
if (st)
```

```
{
```

```
    AfxMessageBox("信息交换错误");
```

```
    return;
```

```
}
```

```
AfxMessageBox("信息交换成功");
```

对卡发取随机数命令

```
short USER_API hs_pro_command_hex(HANDLE idComDev,unsigned
```

```
char slen, char * sendbuffer,unsigned char *rlen, char *
```

```
databuffer,unsigned char timeout);
```

示 例:

```
short st = -1;
```

```
unsigned char slen,rlen,senddata[100], recdata[100];
```

```
slen=5;
```

```
memcpy(senddata,"0084000008",10);
```

```
st= hs_pro_command_hex( icdev,slen,senddata,&rlen,recdata,7);
```

```
if (st)
```

```
{
```

```
    AfxMessageBox("信息交换错误");
```

```
    return;
```

```
}
```

```
AfxMessageBox("信息交换成功");
```

对卡发取随机数命令

### 1.2.3.3 hs\_pro\_commandsource

```
short USER_API hs_pro_commandsource(HANDLE icdev, unsigned char  
slen, unsigned char *sendbuffer, unsigned char *rlen, unsigned char  
*databuffer, unsigned char timeout);
```

功 能：应用协议数据单元信息交换函数。该函数不封装，用户需自行组织数据发送

参 数：

**【in】** HANDLE icdev:hs\_init 函数返回的串口标识符

**【in】** unsigned char slen:发送的信息长度

**【in】** unsigned char \* sbuff :存放要发送的信息

**【out】** unsigned char \*rlen:返回信息的长度

**【out】** unsigned char \* rbuff :存放返回的信息

**【in】** unsigned char timeout :延迟时间，单位为：10ms

返 回：

<0 错误。其绝对值为错误号

=0 成功。

示 例：

```
short st = -1;
```

```
unsigned char slen,rlen,snddata[100], recdata[100];
```

```
slen=5;
```

```
senddata[0]=nad;senddata[1]=pcb;senddata[2]=5,
```

```
senddata[3]=0x00;senddata[4]=0x84;senddata[5]=0x00;
```

```
senddata[6]=0x00;senddata[7]=0x08;
```

```
for(st=0;st<8;st++)
```

```
senddata[8]^=senddata[st]; //计算异或和
```

```
st=hs_pro_commandsource( icdev,slen,senddata,&rlen,recdata,7);
```

```
if (st)
```

```
{
```

```
    AfxMessageBox("信息交换错误");
```

```

        return;
    }

    AfxMessageBox("信息交换成功");

    相关 HEX 函数:

    short USER_API hs_pro_commandsourcehex(HANDLE ICDev,unsigned
    char slen,unsigned char * sbuff,unsigned char *rlen,unsigned char *
    rbuff,unsigned char timeout);

    short USER_API hs_pro_commandsource_hex(HANDLE
    ICDev,unsigned char slen,unsigned char * sbuff,unsigned char
    *rlen,unsigned char * rbuff,unsigned char timeout);

```

### 1.2.3.4 hs\_pro\_commandlink

```

short USER_API hs_pro_commandlink(HANDLE icdev, unsigned char
slen, unsigned char *sendbuffer, unsigned char *rlen, unsigned char
*databuffer, unsigned char timeout, unsigned char FG);

```

功 能：应用协议数据单元信息交换函数。该函数已封装 T=CL 操作

参 数：

**【in】** HANDLE idComDev:hs\_init 函数返回的端口标识符

**【in】** unsigned char slen:发送的信息长度

**【in】** unsigned char \* sendbuffer:存放要发送的信息

**【out】** unsigned char \*rlen:返回信息的长度

**【out】** unsigned char \* databuffer:存放返回的信息

**【in】** unsigned char timeout:延迟时间，单位为：10ms

**【in】** unsigned char FG:分割长度。建议此值小于 64

返 回：

<0 错误。其绝对值为错误号

=0 成功。

示 例：

```
short st = -1;
```

```
unsigned char slen,rlen,snddata[100], recdata[100];
```

```
slen=5;
```

```
snddata[0]=0x00;snddata[1]=0x84;snddata[2]=0x00;
```

```
snddata[3]=0x00;snddata[4]=0x04;
```

```
st= hs_pro_commandlink ( icdev,slen,snddata,&rlen,recdata,7, 56) // 对
```

卡发取随机数命令

```
if (st)
```

```
{
```

```
    AfxMessageBox("信息交换错误");
```

```
    return;
```

```
}
```

```
AfxMessageBox("信息交换成功");
```

相关 HEX 函数：

```
short USER_API hs_pro_commandlink_hex(HANDLE ICDev, unsigned
```

```
char slen,unsigned char *buff, unsigned char *rlen,unsigned char *
```

```
rbuff,unsigned char tt,unsigned char FG)
```

示 例:

```
short st = -1;
```

```
unsigned char slen,rlen,senddata[100], recdata[100];
```

```
slen=5;
```

```
memcpy(senddata,"0084000008",10);
```

```
st= hs_pro_commandlink_hex( icdev,slen,senddata,&rlen,recdata,7,
```

```
56) // 对卡发取随机数命令
```

```
if (st)
```

```
{
```

```
    AfxMessageBox("信息交换错误");
```

```
    return;
```

```
}
```

```
AfxMessageBox("信息交换成功");
```

### 1.2.3.5 hs\_pro\_reset(TYPE A 专用)

```
short USER_API hs_pro_reset(HANDLE icdev, unsigned char *rlen,
```

```
unsigned char *receive_data);
```

功 能：卡上电复位函数,仅针对于 TYPE A 卡

参 数:

**【in】** HANDLE icdev:hs\_init 函数返回的端口标识符

**【out】** unsigned char \*rlen:返回复位信息的长度

**【out】** unsigned char \*receive\_data:存放返回的复位信息

返 回:

<0 错误。其绝对值为错误号

=0 成功。

示例:

```
short st = -1;

st=hs_pro_reset(icdev,rlen,DataBuffer);

if (st)
{
    AfxMessageBox("信息交换错误");
    return;
}

AfxMessageBox("信息交换成功");
```

相关 HEX 函数:

```
short USER_API hs_pro_reset_hex(HANDLE icdev,unsigned char *rlen,
char *receive_data);

short USER_API hs_pro_resethex(HANDLE icdev,unsigned char *rlen,
char *receive_data);
```

### 1.2.3.6 hs\_pro\_halt (TYPE A 专用)

short USER\_API hs\_pro\_halt(HANDLE icdev);

功 能：中止对该卡操作，仅针对于 TYPE A 卡

参 数：

【in】HANDLE icdev:hs\_init 函数返回的端口标识符

返 回：成功则返回 0

示 例：

```
short st = -1;
```

```
st=hs_pro_halt(icdev);
```

```
if (st)
```

```
{
```

```
    AfxMessageBox("中止对该卡操作错误");
```

```
    return;
```

```
}
```

```
AfxMessageBox("中止对该卡操作成功");
```

说明：使用 hs\_card()函数时，有个\_Mode 参数，如果\_Mode=0 则在对卡进行操作完毕后，执行 hs\_pro\_halt();则该卡进入 HALT 模式，则必须把卡移开感应区再进来才能寻得这张卡。

### 1.2.4 CPU(SAM)专用函数

设置卡座



hs\_setcpu

2.设置卡片波特率，波特率为 9600 的卡不需要这一步

hs\_setcpupara

卡片复位

hs\_cpureset\_hex

发送命令

hs\_cpuapdu

示 例:

```
unsigned char rlen=0;
```

```
char recdata[512],tmp[256]={0};
```

```
unsigned long _Snr=0;
```

```
memset(recdata,0,512);
```

```
memset(tmp,0,256);
```

```
HANDLE icdev = (HANDLE)-1;
```

```
icdev=hs_init(100,115200);
```

```
if((int)icdev<0)
```

```
{
```

```
    MessageBox("打开端口失败");
```

```
    return;
```

```
}
```

```
st=hs_setcpu(icdev,0x0c);
```

```
st=hs_cpureset_hex(icdev,&rlen,recdata);
```

```

if(st!=0)
{
    sprintf((char *)recdata,"ERR_CODE=%d",st);
    MessageBox((char *)recdata,"错误提示",MB_OK);
    return;
}

st=hs_cpuapdu_hex(icdev,5,(unsigned char
*)"0084000008",&rlen,recdata);

if(st!=0)
{
    sprintf((char *)recdata,"ERR_CODE=%d",st);
    MessageBox((char *)recdata,"错误提示",MB_OK);
    return;
}

```

### 1.2.4.1 hs\_setcpu

short USER\_API hs\_setcpu(HANDLE icdev, unsigned char \_Byte);

功 能： 设置要操作的 SAM 卡座

参 数：

**【in】** HANDLE icdev:hs\_init 函数返回的端口标识符

**【in】** unsigned char \_Byte:设置要操作的卡座号

0x0c:附卡座

0x0d:SAM1

0x0e:SAM2

0x0f:SAM3

0x11:SAM4

返 回:

<0 错误。其绝对值为错误号

=0 成功

示 例:

```
short st = -1;
```

```
st = hs_setcpu(icdev,0x0c); //2005-2-25
```

```
if(st!=0)
```

```
{
```

```
    MessageBox("设置 SAM 卡座失败");
```

```
    return;
```

```
}
```

```
    MessageBox("设置 SAM 卡座成功");
```

## 1.2.4.2 hs\_setcpupara

short USER\_API hs\_setcpupara(HANDLE icdev, unsigned char cputype,

unsigned char cpupro, unsigned char cpuetu);

功 能：设置 CPU 卡的参数,上电后的默认参数是 cpupro=0(T=0 协议)cpuetu=92(波特率 9600)

参 数：

【in】HANDLE icdev:hs\_init 函数返回的端口标识符

【in】unsigned char cputype:卡座类型，12=主卡座 13=SAM1 卡座 14=SAM2 卡座 15=SAM3 卡座（十进制）

【in】unsigned char cpupro:卡的协议类型 =0 表示 T=0 协议 =1 表示 T=1 协议

【in】unsigned char cpuetu:卡操作时候的延时数据（十进制）不同波特率的卡，此参数的值不同，对于 9600 波特率的卡 cpuetu 设置成 92 对于 38400 波特率的卡 cpuetu 设置成 20

返 回：

<0 错误。其绝对值为错误号

=0 成功。

示 例：

```
short st = -1;

st = hs_setcpupara(icdev,0x0C,0,92);

if (st)
{
    MessageBox("设置 cpu 卡参数错误");
    return ;
}
```

```
MessageBox("设置 cpu 卡参数成功");
```

### 1.2.4.3 hs\_cpureset

```
short USER_API hs_cpureset(HANDLE icdev, unsigned char *rlen,  
unsigned char *databuffer);
```

功 能：CPU 卡上电复位函数，复位后自动判断卡片协议

参 数：

**【in】** HANDLE icdev:hs\_init 函数返回的端口标识符

**【out】** unsigned char \*rlen:返回复位信息的长度

**【out】** unsigned char \*databuffer:存放返回的复位信息

返 回：

<0 错误。其绝对值为错误号

=0 成功。

示 例：

```
short st = -1;  
  
unsigned char rlen = 0;  
  
char recdata[512] = "\0";  
  
st=hs_cpureset(icdev,&rlen,recdata);  
  
if(st!=0)  
{  
  
    MessageBox("CPU 卡上电复位失败");  
  
    return;  
}
```

```
}
```

```
MessageBox("CPU 卡上电复位成功");
```

相关 HEX 函数:

```
short USER_API hs_cpureset_hex(HANDLE icdev,unsigned char *rlen,  
char *databuffer);
```

#### 1.2.4.4 hs\_cpuapdu

```
short USER_API hs_cpuapdu(HANDLE icdev, unsigned char slen,  
unsigned char *sendbuffer, unsigned char *rlen, unsigned char  
*databuffer);
```

功 能: CPU 卡 APDU (应用协议数据单元) 信息交换函数。该函数封装了 T=0 和 T=1 操作

参 数:

**【in】** HANDLE icdev:hs\_init 函数返回的端口标识符

**【in】** unsigned char \*slen:发送的信息长度

**【in】** unsigned char \* sendbuffer:存放要发送的信息

**【out】** unsigned char \*rlen:返回信息的长度

**【out】** unsigned char \* databuffer:存放返回的信息

返 回:

<0 错误。其绝对值为错误号

=0 成功。

示 例:

```

short st = -1;

unsigned char slen,rlen,snddata[100], recdata[100];

slen=5;

snddata[0]=0x00;snddata[1]=0x84;snddata[2]=0x00;

snddata[3]=0x00;snddata[4]=0x04;

st= hs_cpuapdu ( icdev,slen,snddata,&rlen,recdata)

```

对卡发取随机数命令

相关 HEX 函数：

```

short USER_API hs_cpuapdu_hex(HANDLE icdev,unsigned char slen,
char * sendbuffer,unsigned char *rlen,unsigned char * rbuff);

```

示 例：

```

short st = -1;

unsigned char slen,rlen,snddata[100], recdata[100];

slen=5;

memcpy(snddata,"0084000008",10);

st= hs_cpuapdu_hex( icdev,slen,snddata,&rlen,recdata);

if(st)

{

    MessageBox("发送指令失败");

    return;

}

```

对卡发取随机数命令

## 1.2.4.5 hs\_cpuapdusource

```
short USER_API hs_cpuapdusource(HANDLE icdev, unsigned char slen,  
unsigned char *sendbuffer, unsigned char *rlen, unsigned char  
*databuffer);
```

功 能：CPU 卡 APDU（应用协议数据单元）信息交换函数。该函数不封装，用户需自行判断协议类型并组织数据发送

参 数：

**【in】** HANDLE icdev:hs\_init 函数返回的串口标识符

**【in】** unsigned char slen:发送的信息长度

**【in】** unsigned char \* sendbuffer:存放要发送的信息

**【out】** unsigned char \*rlen:返回信息的长度

**【out】** unsigned char \* databuffer:存放返回的信息

返 回：

<0 错误。其绝对值为错误号

=0 成功。

示 例：//T=1 协议卡的操作

```
short st = -1;  
  
unsigned char slen,rlen,snddata[100], recdata[100];  
  
slen=5;  
  
snddata[0]=nad;  
  
snddata[1]=pcb;  
  
snddata[2]=5,
```



```
senddata[3]=0x00;senddata[4]=0x84;senddata[5]=0x00;
```

```
senddata[6]=0x00;senddata[7]=0x08;
```

```
for(st=0;st<8;st++)
```

```
    senddata[8]^=senddata[st]; //计算异或和
```

```
st=hs_cpuapdusource( icdev,slen,senddata,&rlen,recdata);
```

相关 HEX 函数:

```
short USER_API hs_cpuapdusource_hex(HANDLE icdev,unsigned char
```

```
slen, char * sendbuffer,unsigned char *rlen,unsigned char * rbuff);
```

## 1.2.5 FM11RF005 专用函数

### 1.装载密码

```
hs_load_key
```

### 2.寻卡

```
hs_card_fm11rf005
```

### 3.校验密码

```
hs_authentication
```

### 4.读卡

```
hs_read_fm11rf005
```

### 5.写卡

```
hs_write_fm11rf005
```

示 例:

```
HANDLE icdev = (HANDLE)-1;
```

```

char key[20] = "\0";

icdev=hs_init(100,115200);

if((int)icdev<0)

{

    MessageBox("打开端口失败");

    return;

}

memcpy(key,"112233445566",12);

st=hs_load_key_hex(icdev,0,0,key);

if(st)

{

    AfxMessageBox("装载密码错误！ ");

    return;

}

st=hs_authentication(icdev,0,0);

if(st)

{

    AfxMessageBox("校验密码失败！ ");

    return;

}

unsigned short TagType;

unsigned char buff[20];

```

```

st= hs_card_fm11rf005(icdev,0,&TagType);

if(st==1)

{

    AfxMessageBox("无卡! ");

    return;

}

if(st!=0)

{

    AfxMessageBox("寻卡请求失败! ");

    return;

}

st= hs_getsnr_fm11rf005_hex( icdev, buff);

if(st)

{

    AfxMessageBox("寻卡失败");

    return;

}

buff[8]='\0';

char databuff[100] = "\0";

unsigned char m_addr = 1;

if(m_addr<0||m_addr>15)

{

```

```

    AfxMessageBox("请输入一个在 0 到 15 间的数! ");

    return;

}

st=hs_read_fm11rf005_hex(icdev,m_addr,databuff);

if(st)

{

    AfxMessageBox("读卡失败! ");

    return;

}

databuff[8]='\0';

char data[20];

if(m_addr<0||m_addr>15)

{

    AfxMessageBox("请输入一个在 0 到 15 间的数! ");

    return;

}

if(m_addr==0||m_addr==1)

{

    AfxMessageBox("此块为厂商数据块，不可写! ");

    return;

}

if(m_addr==8)

```

```

{
    if(MessageBox("所要写入的块是密码块,是否确定写入?","警告
",MB_OKCANCEL|MB_ICONWARNING)!=IDOK)

        return;
}

memcpy(data,"11223344",8);

st=hs_write_fm11rf005_hex(icdev,m_addr,data);

if(st)
{
    AfxMessageBox("写入失败！");

    return;
}

AfxMessageBox("写入成功！");

```

在对 FM11RF005 作时，密码 4 个字节，在装载设备密码和校密码时  
要将此 4 个字节后面补 2 个字节的 0 凑成 6 个字节来进行操作。

### 1.2.5.1 hs\_read\_fm11rf005

```
short USER_API hs_read_fm11rf005(HANDLE icdev, unsigned char _Adr, unsigned
char *_Data);
```

功 能：

读取 fm11rf005 卡中的数据

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** unsigned char \_Adr: 块地址（0~15）；

**【out】** unsigned char \*\_Data: 读出数据，长度为 4 字节

返 回：成功则返回 0

示 例：

```
short st = -1;
```

```
unsigned char data[5] = "\0";
```

```
st=hs_read_fm11rf005(icdev,4,data); //读 fm11rf005 卡块 4 的数据 if(st)
```

```
{
```

```
    MessageBox("读取 fm11rf005 卡中的数据失败");
```

```
    return;
```

```
}
```

```
MessageBox("读取 fm11rf005 卡中的数据成功");
```

```
return;
```

相关 HEX 函数：

```
short USER_API hs_read_fm11rf005_hex(HANDLE icdev,unsigned char  
_Adr,char *_Data);
```

## 1.2.5.2 hs\_write\_fm11rf005

```
short USER_API hs_write_fm11rf005(HANDLE icdev, unsigned char  
_Adr, unsigned char *_Data);
```

功 能：向 fm11rf005 卡中写入数据

参 数：

【in】HANDLE icdev: hs\_init 函数返回的端口标识符

【in】unsigned char \_Adr: 块地址（2~15）；

【in】unsigned char \*\_Data: 读出数据，长度为 4 字节

返 回：成功则返回 0

示 例：

```
short st = -1;
```

```
unsigned char data[5] = "\0";
```

```
st=hs_write_fm11rf005(icdev,4,data); //向 fm11rf005 卡第 4 块写入数
```

据

```
if(st)
```

```
{
```

```
    MessageBox("向 fm11rf005 卡中写入数据失败");
```

```
    return;
```

```
}
```

```
MessageBox("向 fm11rf005 卡中写入数据成功");
```

```
return;
```

相关 HEX 函数：

```
short USER_API hs_write_fm11rf005(HANDLE icdev,unsigned char
```

```
_Adr,unsigned char *_Data);
```

### 1.2.5.3 hs\_getsnr\_fm11rf005

```
short USER_API hs_getsnr_fm11rf005(HANDLE icdev, unsigned long
```

\*\_Snr);

功 能：读取 fm11rf005 卡的序列号

参 数：

【in】HANDLE icdev: hs\_init 函数返回的端口标识符

【out】unsigned long \*\_Snr: 返回的卡的序列号

返 回：成功则返回 0

示 例：

```
short st = -1;

unsigned long snr = 0;

st=hs_getsnr_fm11rf005(icdev,&snr);

if(st)

{

    MessageBox("读取 fm11rf005 卡的序列号失败");

    return;

}

MessageBox("读取 fm11rf005 卡的序列号成功");

return;
```

相关 HEX 函数：

```
short USER_API hs_getsnr_fm11rf005_hex(HANDLE icdev,unsigned char

*snrstr);
```



## 1.2.6 24C 系列卡专用函数

对于 24c01, 24c02, 24c04, 24c08, 24c16 卡读取数据

hs\_read\_24c

hs\_write\_24c

对于 24c64, 24c512, 24c1024hs\_read\_24c64

hs\_read\_24c64

hs\_write\_24c64

示 例:

```
short st = -1;
```

```
char recmess[1024] = "\0";
```

```
char mess[100] = "\0";
```

```
char sss1[100] = "\0";
```

```
st=hs_read_24c64_hex(icdev,0,10,(unsigned char*)recmess);
```

```
if(st!=0)
```

```
{
```

```
    strcpy(mess,"读卡出错，错误代码为：");
```

```
    sprintf(sss1,"%d",st);
```

```
    strcat(mess,sss1);
```

```
    AfxMessageBox(mess);
```

```
    return;
```

```
}
```

```
recmess[2*m_len]='\0';
```

```

st=hs_write_24c64_hex(icdev,0,10,(unsigned char
*)"1122334455667788990000");

if(st!=0)

{

    strcpy(mess,"命令出错， 错误代码为： ");

    sprintf(sss1,"%d",st);

    strcat(mess,sss1);

    AfxMessageBox(mess);

    return;

}

```

### 1.2.6.1 hs\_read\_24c

```

short USER_API hs_read_24c(HANDLE icdev, short offset, short lenth,
unsigned char *receive_buffer);

```

功 能： 读取 24c 卡中数据

对于 24c01， 24c02， 24c04， 24c08， 24c16 卡读取数据

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** short offset: 偏移量， 读卡的开始字节位数

**【in】** short lenth: 读出数据长度， 从 offset 函数开始读起

**【out】** unsigned char \* receive\_buffer: 读取的数据， 放在 receive\_buffer 数组里

返 回： 成功则返回 0

例 如：

```
short st = -1;

unsigned char recmess[1024]=" \0" ;

st=hs_read_24c(icdev,0,6,recmess);

if(st)

{

    MessageBox("读取 24c 卡中数据失败");

    return;

}

MessageBox("读取 24c 卡中数据成功");

return;
```

相关 HEX 函数：

```
short USER_API hs_read_24c_hex(HANDLE icdev,short offset,short

lenth,unsigned char * receive_buffer);
```

## 1.2.6.2 hs\_write\_24c

```
short USER_API hs_write_24c(HANDLE icdev, short offset, short lenth,

unsigned char *snd_buffer);
```

功 能： 写入 24c 卡中数据

对于 24c01， 24c02， 24c04， 24c08， 24c16 卡写入数据

参 数：

【in】HANDLE icdev: hs\_init 函数返回的端口标识符

【in】short offset: 偏移量，写卡的开始字节位数

【in】short lenth: 写入数据长度，从 offset 函数开始写起

【in】unsigned char \* snd\_buffer: 写入的数据内容，放在 snd\_buffer 数组里写入

返 回：成功则返回 0

示 例：

```
short st = -1;

unsigned char recmess[1024]=" 0x31,0x32,0x33,0x34,0x35,0x36" ;

st=hs_write_24c(icdev,0,6,recmess);

if(st)

{

    MessageBox("写 24c 卡中数据失败");

    return;

}

MessageBox("写 24c 卡中数据成功");
```

相关 HEX 函数：

```
short USER_API hs_write_24c_hex(HANDLE icdev,short offset,short lenth,unsigned char * snd_buffer);
```

示 例：

```
short st = -1;
```

```

unsigned char recmess[1024]=" \0" ;

memcpy(recmess,"313233343536",12);

st=hs_write_24c_hex(icdev,0,6,recmess);

if(st)

{

    MessageBox("写 24c 卡中数据失败");

    return;

}

MessageBox("写 24c 卡中数据成功");

```

### 1.2.6.3 hs\_read\_24c64

```

short USER_API hs_read_24c64(HANDLE icdev,short offset,short
lenth,unsigned char * receive_buffer);

```

功 能：读取 24c64 卡中数据

对于 24c64，24c512,24c1024

参 数：

**【in】HANDLE icdev：**hs\_init 函数返回的端口标识符

**【in】short offset：**偏移量，读卡的开始字节位数

**【in】short lenth：**读出数据长度，从 offset 函数开始读起

**【out】unsigned char \* receive\_buffer：**读取的数据，放在  
receive\_buffer 数组里

返 回：成功则返回 0

例 如：

```
short st = -1;

unsigned char recmess[1024]=" \0" ;

st=hs_read_24c64(icdev,0,6,recmess);

if(st)

{

    MessageBox("读取 24c 卡中数据失败");

    return;

}

MessageBox("读取 24c 卡中数据成功");

return;
```

相关 HEX 函数：

```
short USER_API hs_read_24c64_hex(HANDLE icdev,short offset,short  
lenth,unsigned char * receive_buffer);
```

#### 1.2.6.4 hs\_write\_24c64

```
short USER_API hs_write_24c64(HANDLE icdev,short offset,short  
lenth,unsigned char * snd_buffer);
```

功 能： 对于 24c64, 24c512,24c1024 写入数据

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** short offset: 偏移量，写卡的开始字节位数

**【in】 short lenth:** 写入数据长度, 从 offset 函数开始写起

**【in】 unsigned char \* snd\_buffer:** 写入的数据内容, 放在 snd\_buffer 数组里写入

返 回: 成功则返回 0

例 如:

```
short st = -1;
```

```
unsigned char recmess[1024]=" 0x31,0x32,0x33,0x34,0x35,0x36" ;
```

```
st=hs_write_24c64(icdev,0,6,recmess);
```

```
if(st)
```

```
{
```

```
    MessageBox("写 24c64 卡中数据失败");
```

```
    return;
```

```
}
```

```
MessageBox("写 24c64 卡中数据成功");
```

```
return;
```

相关 HEX 函数:

```
short USER_API hs_write_24c64_hex(HANDLE icdev,short offset,short
```

```
lenth,unsigned char * snd_buffer);
```

示 例:

```
short st = -1;
```

```
unsigned char recmess[1024]=" \0" ;
```

```
memcpy(recmess,"313233343536",12);
```

```
st=hs_write_24c_hex(icdev,0,6,recmess);  
  
if(st)  
{  
    MessageBox("写 24c64 卡中数据失败");  
    return;  
}  
MessageBox("写 24c64 卡中数据成功");
```

## 1.2.7 4442 卡专用函数

1.核对 4442 卡密码

hs\_verifypin\_4442

2.修改 4442 卡密码

hs\_changepin\_4442

3.4442 卡错误计数器

hs\_readpincount\_4442

4.读 4442 卡密码

hs\_readpin\_4442

5.读 4442 卡

hs\_read\_4442

6.写 4442 卡

hs\_write\_4442

示 例:



```

short st = -1;

char sendmess[1024],mess[100],sss1[100];

st=hs_verifypin_4442_hex(icdev,(unsigned char *)"ffffff");

if(st!=0)

{

    AfxMessageBox("核对密码失败!");

    return;

}

AfxMessageBox("核对密码失败!");

st=hs_changepin_4442_hex(icdev,(unsigned char *)"123456");

if(st!=0)

{

    AfxMessageBox("修改密码失败!");

    return;

}

AfxMessageBox("修改密码成功!");

st=hs_readpincount_4442(icdev);

if(st<0)

{

    AfxMessageBox("读错误计数器失败!");

    return;

}

```

```

else
{
    strcpy(mess,"错误计数器为: ");
    sprintf(sss1,"%d",st);
    strcat(mess,sss1);
    AfxMessageBox(mess);
    return;
}
st=hs_readpin_4442_hex(icdev,(unsigned char *)recmess);
if(st!=0)
{
    AfxMessageBox("读密码失败!");
    return;
}
recmess[6]='\0';
st=hs_read_4442_hex(icdev,20,10,(unsigned char*)recmess);
if(st!=0)
{
    AfxMessageBox("读 4442 卡失败!");
    return;
}
recmess[2*m_len]='\0';

```

```

st=hs_write_4442_hex(icdev,20,10,(unsigned char
*)"11223344556677889900");

if(st!=0)
{
    AfxMessageBox("写 4442 卡失败!");
    return;
}
AfxMessageBox("写 4442 卡成功!");

```

### 1.2.7.1 hs\_verifypin\_4442

```

short USER_API hs_verifypin_4442(HANDLE icdev,unsigned char
*passwd);

```

功 能： 对于 4442 校验密码（3 字节密钥，初始密钥一般为： 0xff, 0xff, 0xff）（核对 3 次错误，4442 卡做废）

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** unsigned char \*passwd: 用于校验的密码

返 回： 成功则返回 0

示 例：

```

short st = -1;

```

```
unsigned char sendmess[1024]=" 0xff,0xff,0xff" ;
```

```
st=hs_verifypin_4442(icdev,sendmess);
```

```
if(st)
```

```
{
```

```
    MessageBox("4442 校验密码失败");
```

```
    return;
```

```
}
```

```
MessageBox("4442 校验密码成功");
```

```
return;
```

相关 HEX 函数：

```
short USER_API hs_verifypin_4442_hex(HANDLE icdev,unsigned char  
*passwd);
```

示 例：

```
short st = -1;
```

```
unsigned char sendmess[1024]=" \0" ;
```

```
memcpy(sendmess,"ffffff",6);
```

```
st=hs_verifypin_4442_hex(icdev,sendmess);
```

```
if(st)
```

```
{
```

```
    MessageBox("4442 校验密码失败");
```

```
    return;
```

```
}
```

```
MessageBox("4442 校验密码成功");
```

```
return;
```

### 1.2.7.2 hs\_changepin\_4442

```
short USER_API hs_changepin_4442(HANDLE icdev,unsigned char  
*passwd);
```

功 能： 修改 4442 密码（3 字节密钥，初始密钥一般为： 0xff,  
0xff, 0xff）

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** unsigned char \*passwd: 修改后的密码

返 回： 成功则返回 0

示 例：

```
short st = -1;
```

```
unsigned char sendmess[1024]=" 0x1f,0x1f,0x1f" ;
```

```
st=hs_changepin_4442(icdev,sendmess);
```

```
if(st)
```

```
{
```

```
    MessageBox("修改 4442 密码失败");
```

```
    return;
```

```
}
```

```
MessageBox("修改 4442 密码成功");
```

```
return;
```

相关 HEX 函数:

```
short USER_API hs_changepin_4442_hex(HANDLE icdev,unsigned char  
*passwd);
```

```
short st = -1;
```

```
unsigned char sendmess[1024]=" \0" ;
```

```
memcpy(sendmess,"1f1f1f",6);
```

```
st=hs_changepin_4442_hex(icdev,sendmess);
```

```
if(st)
```

```
{
```

```
    MessageBox("修改 4442 密码失败");
```

```
    return;
```

```
}
```

```
MessageBox("修改 4442 密码成功");
```

```
return;
```

### 1.2.7.3 hs\_readpincount\_4442

```
short USER_API hs_readpincount_4442(HANDLE icdev);
```

功 能：读取 4442 卡错误计数器（1,2,3（3 次错误就 4442 卡废了））

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

返 回：成功则返回 4442 卡错误计数（1,2,3（3 次错误就 4442 卡废了））

错误返回负值

示 例：

```
short st = -1;

char mess[100],sss1[100];

st=hs_readpincount_4442(icdev);

if(st<0)
{
    strcpy(mess,"命令出错， 错误代码为： ");
    sprintf(sss1,"%d",st);
    strcat(mess,sss1);
    AfxMessageBox(mess);
    return;
}
else
{
    strcpy(mess,"错误计数器为： ");
    sprintf(sss1,"%d",st);
    strcat(mess,sss1);
    AfxMessageBox(mess);
    return;
}
```

```
}
```

## 1.2.7.4 hs\_readpin\_4442

```
short USER_API hs_readpin_4442(HANDLE icdev,unsigned char  
*passwd);
```

功 能： 读取 4442 卡密码（3 字节密钥，初始密钥一般为： 0xff, 0xff, 0xff）

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【out】** unsigned char \*passwd: 读取密码

返 回： 成功则返回 0

示 例：

```
short st = -1;  
  
char mess[100],sss1[100];  
  
unsigned char recmess[1024];  
  
st=hs_readpin_4442(icdev,recmess);  
  
if(st!=0)  
{  
  
    strcpy(mess,"执行出错， 错误代码为: ");  
  
    sprintf(sss1,"%d",st);  
  
    strcat(mess,sss1);  
  
    AfxMessageBox(mess);  
}
```



```

        return;
    }

    recmess[3]='\0';

    m_recdatastr=recmess;

```

相关 HEX 函数:

```

short USER_API hs_readpin_4442_hex(HANDLE icdev,unsigned char
*passwd);

```

示 例:

```

short st = -1;

char mess[100],sss1[100];

unsigned char recmess[1024];

st=hs_readpin_4442(icdev,recmess);

if(st!=0)
{
    strcpy(mess,"执行出错， 错误代码为: ");
    sprintf(sss1,"%d",st);
    strcat(mess,sss1);
    AfxMessageBox(mess);

    return;
}

recmess[3]='\0';

m_recdatastr=recmess;

```

## 1.2.7.5 hs\_read\_4442

```
short USER_API hs_read_4442(HANDLE icdev,short offset,short  
lenth,unsigned char * buffer);
```

功 能： 读取 4442 卡中数据

参 数：

**【in】** HANDLE icdev： 通讯设备标识符

**【in】** short offset： 偏移量，读卡的开始字节位数

**【in】** short lenth： 读出数据长度，从 offset 函数开始读起

**【out】** unsigned char \* buffer： 读取的数据，放在 buffer 数组里

返 回： 成功则返回 0

示 例：

```
short st = -1;  
  
char mess[100],sss1[100];  
  
unsigned char recmess[1024];  
  
st=hs_read_4442(icdev,m_startaddr,m_len,recmess);  
  
if(st!=0)  
{  
  
    strcpy(mess,"读卡出错，错误代码为：");  
  
    sprintf(sss1,"%d",st);  
  
    strcat(mess,sss1);  
  
    AfxMessageBox(mess);  
  
    return;
```

```
}
```

相关 HEX 函数:

```
short USER_API hs_read_4442_hex(HANDLE icdev,short offset,short  
lenth,unsigned char* buffer);
```

### 1.2.7.6 hs\_write\_4442

```
short USER_API hs_write_4442(HANDLE icdev,short offset,short  
lenth,unsigned char * buffer);
```

功 能: 写入 4442 卡数据

参 数:

**【in】** HANDLE icdev: 通讯设备标识符

**【in】** short offset: 偏移量, 写卡的开始字节位数

**【in】** short lenth: 写入数据长度, 从 offset 函数开始写起

**【in】** unsigned char \* buffer: 写入的数据内容, 放在 snd\_buffer 数组里写入

返 回: 成功则返回 0

示 例:

```
short st = -1;
```

```
unsigned char buffer[1024]="0x31,0x32,0x33,0x34,0x35,0x36";
```

```
st=hs_write_4442(icdev,0,6,recmess);
```

```
if(st!=0)
```

```
{
```

```

strcpy(mess,"写卡出错， 错误代码为: ");

sprintf(sss1,"%d",st);

strcat(mess,sss1);

AfxMessageBox(mess);

return;

}

```

相关 HEX 函数:

```

short USER_API hs_write_4442_hex(HANDLE icdev,short offset,short
lenth,unsigned char* buffer);

```

示 例:

```

short st = -1;

unsigned char buffer[1024]="\0";

memcpy(buffer,"313233343536",12);

st=hs_write_4442_hex(icdev,0,6,recmess);

if(st!=0)

{

strcpy(mess,"写卡出错， 错误代码为: ");

sprintf(sss1,"%d",st);

strcat(mess,sss1);

AfxMessageBox(mess);

return;

}

```

### 1.2.7.7 hs\_Check\_4442

short USER\_API hs\_Check\_4442(HANDLE icdev);

功 能：判断此卡是否是 4442 卡

参 数：

**【in】** HANDLE icdev：通讯设备标识符

返 回：

    0：是 4442 卡

    非 0：不是 4442 卡

例 如：

```
short j = hs_Check_4442(m_icdev);
```

```
return j;
```

## 1.2.8 4428 卡专用函数

1.核对 4428 卡密码

hs\_verifypin\_4428

修改 4428 卡密码

hs\_changepin\_4428

3.4428 卡错误计数器

hs\_readpincount\_4428

读 4428 卡密码

hs\_readpin\_4428

读 4428 卡

hs\_read\_4428

6.写 4428 卡

hs\_write\_4428

示 例:

```
short st = -1;
```

```
char sendmess[1024],mess[100],sss1[100];
```

```
st=hs_verifypin_4428_hex(icdev,(unsigned char *)"ffff");
```

```
if(st!=0)
```

```
{
```

```
    AfxMessageBox("核对密码失败!");
```

```
    return;
```

```
}
```

```
AfxMessageBox("核对密码失败!");
```

```
st=hs_changepin_4428_hex(icdev,(unsigned char *)"1234");
```

```
if(st!=0)
```

```
{
```

```
    AfxMessageBox("修改密码失败!");
```

```
    return;
```

```
}
```

```
AfxMessageBox("修改密码成功!");
```

```
st=hs_readpincount_4428(icdev);
```

```
if(st<0)
```

```

{
    AfxMessageBox("读错误计数器失败!");
    return;
}
else
{
    strcpy(mess,"错误计数器为: ");
    sprintf(sss1,"%d",st);
    strcat(mess,sss1);
    AfxMessageBox(mess);
    return;
}
st=hs_readpin_4428_hex(icdev,(unsigned char *)recmess);
if(st!=0)
{
    AfxMessageBox("读密码失败!");
    return;
}
recmess[4]='\0';
st=hs_read_4428_hex(icdev,20,10,(unsigned char*)recmess);
if(st!=0)
{

```

```

    AfxMessageBox("读 4428 卡失败!");

    return;

}

recmess[2*m_len]='\0';

st=hs_write_4428_hex(icdev,20,10,(unsigned char
*)"11223344556677889900");

if(st!=0)

{

    AfxMessageBox("写 4428 卡失败!");

    return;

}

AfxMessageBox("写 4428 卡成功!");

```

### 1.2.8.1 hs\_verifypin\_4428

```

short USER_API hs_verifypin_4428(HANDLE icdev,unsigned char
*passwd);

```

功 能： 对于 4428 校验密码（2 字节密钥，初始密钥一般为： 0xff, 0xff）

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** unsigned char \*passwd: 用于校验的密码

返 回： 成功则返回 0



示 例：

```
short st = -1;

unsigned char sendmess[1024]="0xff,0xff";

st=hs_verifypin_4428(icdev,sendmess);

if(st!=0)

{

    strcpy(mess,"4428 校验密码出错，错误代码为：");

    sprintf(sss1,"%d",st);

    strcat(mess,sss1);

    AfxMessageBox(mess);

    return;

}
```

相关 HEX 函数：

```
short USER_API hs_verifypin_4428_hex(HANDLE icdev,unsigned char

*passwd);
```

示 例：

```
short st = -1;

unsigned char sendmess[1024]="\0";

memcpy(sendmess,"ffff",4);

st=hs_verifypin_4428_hex(icdev,sendmess);

if(st!=0)

{
```

```

strcpy(mess,"4428 校验密码出错，错误代码为：");

sprintf(sss1,"%d",st);

strcat(mess,sss1);

AfxMessageBox(mess);

return;

}

```

## 1.2.8.2 hs\_changepin\_4428

```

short USER_API hs_changepin_4428(HANDLE icdev,unsigned char
*passwd);

```

功 能： 修改 4428 密码（2 字节密钥，初始密钥一般为： 0xff, 0xff）

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【in】** unsigned char \*passwd: 修改后的密码

返 回： 成功则返回 0

例 如：

```

short st = -1;

unsigned char sendmess[1024]="0x1f,0x1f";

st=hs_changepin_4428(icdev,sendmess);

```

相关 HEX 函数：

```

short USER_API hs_changepin_4428_hex(HANDLE icdev,unsigned char

```

\*passwd);

### 1.2.8.3 hs\_readpincount\_4428

short USER\_API hs\_readpincount\_4428(HANDLE icdev);

功 能：读取 4428 卡错误计数器（1,2,3,4,5,6,7,8（8 次错误就 4428 卡废了））

参 数：

【in】HANDLE icdev：hs\_init 函数返回的端口标识符

返 回：成功则返回 4428 卡错误计数（1,2,3,4,5,6,7,8（8 次错误就 4428 卡废了））

错误返回负值

示 例：

```
short st = -1;
```

```
char mess[100],sss1[100];
```

```
st=hs_readpincount_4428(icdev);
```

```
if(st<0)
```

```
{
```

```
    strcpy(mess,"命令出错，错误代码为：");
```

```
    sprintf(sss1,"%d",st);
```

```
    strcat(mess,sss1);
```

```
    AfxMessageBox(mess);
```

```
    return;
```

```

}

else

{

    strcpy(mess,"错误计数器为: ");

    sprintf(sss1,"%d",st);

    strcat(mess,sss1);

    AfxMessageBox(mess);

    return;

}

```

#### 1.2.8.4 hs\_readpin\_4428

```

short USER_API hs_readpin_4428(HANDLE icdev,unsigned char
*passwd);

```

功 能： 读取 4428 卡密码（2 字节密钥，初始密钥一般为： 0xff, 0xff）

参 数：

**【in】** HANDLE icdev: hs\_init 函数返回的端口标识符

**【out】** unsigned char \*passwd: 读取密码

返 回： 成功则返回 0

示 例：

```

short st = -1;

```

```

char mess[100],sss1[100];

```

```

unsigned char recmess[1024];

st=hs_readpin_4428(icdev,recmess);

if(st!=0)

{

    strcpy(mess,"执行出错， 错误代码为： ");

    sprintf(sss1,"%d",st);

    strcat(mess,sss1);

    AfxMessageBox(mess);

    return;

}

recmess[3]='\0';

m_recdatastr=recmess;

```

相关 HEX 函数：

```

short USER_API hs_readpin_4428_hex(HANDLE icdev,unsigned char
*passwd);

```

### 1.2.8.5 hs\_read\_4428

```

short USER_API hs_read_4428(HANDLE icdev,short offset,short
lenth,unsigned char * buffer);

```

功 能： 读取 4428 卡中数据

参 数：

**【in】** HANDLE icdev： hs\_init 函数返回的端口标识符

**【in】 short offset:** 偏移量，读卡的开始字节位数

**【in】 short lenth:** 读出数据长度，从 offset 函数开始读起

**【out】 unsigned char \* buffer:** 读取的数据，放在 buffer 数组里

返 回：成功则返回 0

示 例：

```
short st = -1;

char mess[100],sss1[100];

unsigned char recmess[1024];

st=hs_read_4428(icdev,m_startaddr,m_len,recmess);

if(st!=0)

{

    strcpy(mess,"读卡出错，错误代码为：");

    sprintf(sss1,"%d",st);

    strcat(mess,sss1);

    AfxMessageBox(mess);

    return;

}
```

相关 HEX 函数：

```
short USER_API hs_read_4428_hex(HANDLE icdev,short offset,short
lenth,unsigned char* buffer);
```

## 1.2.8.6 hs\_write\_4428

```
short USER_API hs_write_4428(HANDLE icdev,short offset,short  
lenth,unsigned char * buffer);
```

功 能： 写数据入 4428 卡中数据

参 数：

【in】 HANDLE icdev: hs\_init 函数返回的端口标识符

【in】 short offset: 偏移量，写卡的开始字节位数

【in】 short lenth: 写入数据长度，从 offset 函数开始写起

【in】 unsigned char \* buffer: 写入的数据内容，放在 buffer 数组里写

入

返 回： 成功则返回 0

示 例：

```
short st = -1;  
  
unsigned char buffer[1024]="0x31,0x32,0x33,0x34,0x35,0x36";  
  
st=hs_write_4428(icdev,0,6,buffer);  
  
if(st!=0)  
{  
  
    strcpy(mess,"写卡出错，错误代码为：");  
  
    sprintf(sss1,"%d",st);  
  
    strcat(mess,sss1);  
  
    AfxMessageBox(mess);  
  
    return;  
}
```

```
}
```

相关 HEX 函数:

```
short USER_API hs_write_4428_hex(HANDLE icdev,short offset,short  
lenth,unsigned char* buffer);
```

示 例:

```
short st = -1;  
  
unsigned char buffer[1024]="\0";  
  
memcpy(buffer,"313233343536",12);  
  
st=hs_write_4428(icdev,0,6,buffer);  
  
if(st!=0)  
{  
    strcpy(mess,"写卡出错， 错误代码为: ");  
    sprintf(sss1,"%d",st);  
    strcat(mess,sss1);  
    AfxMessageBox(mess);  
    return;  
}
```

### 1.2.8.7 hs\_Check\_4428

```
short USER_API hs_Check_4428(HANDLE icdev);
```

功 能： 判断此卡是否是 4428 卡

参 数： **【in】** HANDLE icdev: 通讯设备标识符



返 回：

0： 是 4428 卡

非 0： 不是 4428 卡

示 例：

```
short j = hs_Check_4428(m_icdev);  
  
return j;
```

## 1.2.9 身份证（港澳台居住证）操作函数

### 1.2.9.1 hs\_GetIDInfo

```
short USER_API hs_GetIDInfo(HANDLE icdev, unsigned char *name,  
unsigned char *sex, unsigned char *nation, unsigned char *birth_day,  
unsigned char *address, unsigned char *id_number, unsigned char  
*department, unsigned char *expire_day, const char *bmp_name)
```

功 能：解析中国居民身份证文字信息，获取相应的条目，此接口不包含港澳台通行证号和签发次数的信息。

参 数： **【in】** HANDLE icdev： 通讯设备标识符

**【out】** name 姓名，请至少分配 64 个字节。

**【out】** sex 性别，请至少分配 8 个字节。

**【out】** nation 民族，请至少分配 12 个字节。

**【out】** birth\_day 出生日期，请至少分配 36 个字节。

**【out】** address 住址，请至少分配 144 个字节。

【out】 id\_number 公民身份号码，请至少分配 76 个字节。

【out】 department 签发机关，请至少分配 64 个字节。

【out】 expire\_day 证件签发日期，请至少分配 36 个字节。

【out】 bmp\_name 照片路径,请确保有写权限。

返 回：成功则返回 0

例 如：

```
st=hs_GetIDInfo(icdev, name, sex_buffer1, nation_buffer1, birth_day,  
address, id_number, department, expire_day, "d:/me.bmp" );
```

```
if (st!= 0)
```

```
{
```

```
    MessageBox("寻卡失败");
```

```
    return;
```

```
}
```

```
MessageBox("寻卡成功");
```

### 1.2.9.2 hs\_ReadIDInfor

```
short USER_API hs_ReadIDInfor(HANDLE icdev, unsigned char *name,  
unsigned char *sex, unsigned char *nation, unsigned char *birth_day,  
unsigned char *address, unsigned char *id_number, unsigned char  
*department, unsigned char *expire_start_day, unsigned char  
*expire_end_day, int *photo_len, unsigned char *photo, int  
*fingerprint_len, unsigned char *fingerprint, unsigned char *reserved);
```

功 能： 开始读身份证卡操作，用于读取身份证卡信息，调用成功后，读出的身份证卡信息将保存在对应参数中，与上个函数的主要区别是此接口不读取身份证中的指纹信息，且照片数据以普通数据格式呈现需要配合 `hs_ParsePhotoInfo` 实现照片数据转换成文件的操作，详见 `hs_ParsePhotoInfo` 的说明

参 数： **【in】** `HANDLE icdev`： 通讯设备标识符

**【out】** `name` 姓名，请至少分配 64 个字节。

**【out】** `sex` 性别，请至少分配 8 个字节。

**【out】** `nation` 民族，请至少分配 12 个字节。

**【out】** `birth_day` 出生日期，请至少分配 36 个字节。

**【out】** `address` 住址，请至少分配 144 个字节。

**【out】** `id_number` 公民身份号码，请至少分配 76 个字节。

**【out】** `department` 签发机关，请至少分配 64 个字节。

**【out】** `expire_start_day` 证件签发日期，请至少分配 36 个字节。

**【out】** `expire_end_day` 证件终止日期，请至少分配 36 个字节。

**【out】** `photo` 照片信息，请至少分配 1024 个字节。

**【out】** `fingerprint` 照片信息，请至少分配 1024 个字节。

**【out】** `reserved` 预留项，请至少分配 76 个字节。

返 回： <0 表示失败，==0 表示成功。1 表示外国人居住证。

示 例：

```
st=hs_ReadIDInfor(icdev, name, sex_buffer1, nation_buffer1,
```

```
birth_day, address, id_number, department, expire_start_day,  
expire_end_day, &photo_len, photo, &fingerprint_len,  
fingerprint, reserved);
```

### 1.2.9.3 hs\_getuid\_i\_d

```
short USER_API hs_getuid_i_d(HANDLE icdev, unsigned char *uid);
```

功 能:

获取身份证 UID

参 数:

**【in】** HANDLE icdev: 通讯设备标识符

**【out】** uid 返回的 UID, 固定为 8 个字节。

返 回:

<0 表示失败, ==0 表示成功

例 如:

```
hs_getuid_i_d(icdev, uid);;
```

### 1.2.9.4 hs\_ParsePhotoInfo

```
short USER_API hs_ParsePhotoInfo(HANDLE icdev, int type, int  
info_len, const unsigned char *info, int *photo_len, unsigned char *photo);
```

功 能: 解析相片信息, 通过公安部相片解码库还原相片图像数据

参 数:

HANDLE icdev: 身份证卡信息标识符。

返 回:

如果=0 表示失败, 否则指向身份证卡图片信息。

例 如: `hs_ParsePhotoInfo(icdev, 0, photo_len, photo, &photo_len2, (unsigned char *)"C:\me.bmp");`

### 1.2.9.5 hs\_ReadIDInforHKMOTW

`short USER_API hs_ReadIDInforHKMOTW(HANDLE icdev, unsigned char *name, unsigned char *sex, unsigned char *reserved, unsigned char *birth_day, unsigned char *address, unsigned char *id_number, unsigned char *department, unsigned char *expire_start_day, unsigned char *expire_end_day, unsigned char *pass_number, unsigned char *sign_count, unsigned char *reserved2, unsigned char *type_sign, int *photo_len, unsigned char *photo, int *fingerprint_len, unsigned char *fingerprint, unsigned char *reserved3)`功 能: 读取身份证、港澳台通行证信息

参 数: **【in】** HANDLE icdev: 通讯设备标识符

**【out】** name 姓名, 请至少分配 64 个字节。

**【out】** sex 性别, 请至少分配 8 个字节。

**【out】** reserved 预留项, 请至少分配 12 个字节。

**【out】** birth\_day 出生日期, 请至少分配 36 个字节。

**【out】** address 住址, 请至少分配 144 个字节。

【out】 id\_number 公民身份号码，请至少分配 76 个字节。

【out】 department 签发机关，请至少分配 64 个字节。

【out】 expire\_start\_day 证件签发日期，请至少分配 36 个字节。

【out】 expire\_end\_day 证件终止日期，请至少分配 36 个字节。

【out】 pass\_number 通行证号码，请至少分配 40 个字节。

【out】 sign\_count 签发次数，请至少分配 12 个字节。

【out】 reserved2 预留项，请至少分配 16 个字节。

【out】 type\_sign 证件类型标识，请至少分配 8 个字节。

【out】 photo 照片信息，请至少分配 1024 个字节。

【out】 fingerprint 照片信息，请至少分配 1024 个字节。

【out】 reserved3 预留项，请至少分配 16 个字节。

功 能： 开始读港澳台通行证操作，用于读取港澳台通行证信息，调用成功后，读出的港澳台通行证信息将保存在对应参数中，照片指纹数据以普通数据格式呈现需要配合 hs\_ParsePhotoInfo 实现照片数据转换成文件的操作，详见 hs\_ParsePhotoInfo 的说明

返 回：

如果=0 表示失败，否则指向身份证卡图片信息。

例 如：

```
hs_ReadIDInforHKMOTW(icdev, name, sex, reserved, birth_day,  
address,id_number,department, expire_start_day, expire_end_day,  
pass_number, sign_count, reserved2, type_sign,
```

```
&photo_len,photo,&fingerprint_len, fingerprint,reserved3);
```

## 1.2.10 社保卡操作函数

### 1.2.10.1 hs\_ReadMFEF05

```
long USER_API hs_ReadMFEF05(HANDLE icdev,char *pincode,char  
*cardtype,char *version,char *orgnum,char *starttime,char *endtime,char  
*cardnum);
```

功 能： 开始读社保卡 005 文件操作，用于读社保表面信息，调用成功后，读出的社保卡信息将保存在对应参数中

参 数： **【in】** HANDLE icdev: 通讯设备标识符

**【out】** pincode 卡识别码，请至少分配 64 个字节。

**【out】** cardtype 卡类型，请至少分配 64 个字节。

**【out】** version 卡规范版本，请至少分配 64 个字节。

**【out】** orgnum 初始化机构编号，请至少分配 64 个字节。

**【out】** starttime 发卡日期，请至少分配 64 个字节。

**【out】** endtime 有效日期，请至少分配 64 个字节。

**【out】** cardnum 卡号，请至少分配 64 个字节。

返 回：

<0 表示失败，==0 表示成功

示 例：

```
char strOutput[64] = "\0";
```

```

char pincode[64]= "\0";

char cardtype[64]= "\0";

char version[64]= "\0";

char orgnum[64]= "\0";

char starttime[64]= "\0";

char endtime[64]= "\0";

char cardnum[64]= "\0";

CString  temp;

CloseComPort();

icdev=hs_init(g_comnum,115200);


int st=0;

if((int)icdev<0)

{

    m_strRecvData += "init error\r\n";

    UpdateData(FALSE);

//    OpenComPort();

    //dev err

    return;

}

st=hs_ReadMFEF05(icdev,pincode,cardtype,version,orgnum,starttime,
endtime,cardnum);

```



## 1.2.10.2 hs\_ReadMFEF06

```
long USER_API hs_ReadMFEF06(HANDLE icdev,char *IDnum,char  
*name,char *exname,char *sex,char *nation,char *addr,char *birthday);
```

功 能： 开始读社保卡 006 文件操作，用于读社保表面信息，调用成功后，读出的社保卡信息将保存在对应参数中

参 数： **【in】** HANDLE icdev: 通讯设备标识符

**【out】** IDnum 身份证号，请至少分配 64 个字节。

**【out】** name 姓名，请至少分配 64 个字节。

**【out】** exname 姓名扩展，请至少分配 64 个字节。

**【out】** sex 性别，请至少分配 64 个字节。

**【out】** nation 民族，请至少分配 64 个字节。

**【out】** addr 出生地，请至少分配 64 个字节。

**【out】** birthday 出生日期，请至少分配 64 个字节。

返 回：

<0 表示失败，==0 表示成功

示 例：

```
char IDnum[64]="\0";  
  
    char name[64]="\0";  
  
    char exname[64]="\0";  
  
    char sex[64]="\0";  
  
    char nation[64]="\0";  
  
    char birthday[64]="\0";
```

```

char addr[64]="\0";

int st = 0;

char strOutput[64] = "\0";


CString  temp;

CloseComPort();

icdev=hs_init(g_comnum,115200);


if((int)icdev<0)
{
    m_strRecvData += "init error\r\n";

    UpdateData(FALSE);

    //dev err
//    OpenComPort();

    return;

}

st=hs_ReadMFEF06(icdev,IDnum,name,exname,sex,nation,addr,birthd
ay);

```

## 1.2.11 扫码接口函数

### 1.2.11.1hs\_GetQrcodetimeout

功 能：获取扫码结果

参 数：

【in】int milliseconds。超时时间，1 表示 100 毫秒。时间到后扫不到码则函数退出

【out】unsigned char \*data：扫码结果

返 回：

<0 表示失败，==0 表示成功

```
result=hs_GetQrcodetimeout(buf,30);
```

### 1.2.11.2hs\_GetQrcode

功 能：获取扫码结果

说明：只有扫码成功才会退出，或者按 esc 热键退出，上册接口注意屏蔽 esc 的标准功能才能实现

参 数：

【in】int milliseconds。每次超时时间，无效参数，任意值。

【out】unsigned char \*data：扫码结果

返 回：

<0 表示失败，==0 表示成功

```
result=hs_GetQrcodetimeout(buf,30);
```

### 1.2.11.3 hs\_OpenQrcode

```
int USER_API hs_OpenQrcode(CALL_BACK_DATA pfnReadData);
```

功能： 打开扫码设备启动回调接口。

说明： 获取扫码结果前必须保证该端口资源是空闲的即如激活了读卡功能的话确保已经调用了 `hs_exit`，打开端口即占用资源需与 `hs_CloseQrcode` 配对使用以释放资源

参数：

**【in】** pfnReadData 扫码结果回通过此回调函数显示

返 回：

<0 表示失败，==0 表示成功

```
result =hs_OpenQrcode(ReadData);
```

```
void ReadData(unsigned char *data,unsigned int length,unsigned long  
userdata)
```

```
{
```

```
// PrintDataHex(data, length,0);
```

```
}
```

### 1.2.11.4 hs\_CloseQrcode

```
void USER_API hs_CloseQrcode(void)
```

功能：关闭扫码端口

说明：回调接口方式只有调用了此接口才能再调用读卡相关函数，和  
OpenQrcode 配对使用

```
hs_CloseQrcode();
```

## 1.3 附录

### 1.3.1 函数错误代码

32 位 Windows 库函数错误代码

返回值(负数)	错误类型
0x10(016)	通讯错误
0x1F(031)	定制版，设备 SDK 不匹配
0x16 (022)	端口参数错误
0x0A(010)	读卡模块异常，检查版本
0x11(017)	超时错误
0x20(032)	打开端口错误
0x21(033)	获得端口参数错误
0x23(034)	设置端口参数错误

0x23(035)	关闭端口出错
0x24(036)	端口被占用
0x30(048)	格错错误
0x31(049)	数据格式错误
0x32(050)	数据长度错误
0x40(064)	读错误
0x41(065)	写错误
0x42(066)	无接收错误
0x50(080)	不够减错误
0x51(081)	CPU 数据异或和错误
0x52(082)	485 通讯时地址号错误
0x73(115)	取版本号错误
0xc2(194)	CPU 卡响应错误
0xd3(211)	CPU 卡响应超时
0xd6(214)	CPU 卡校验错误
0xd7(215)	CPU 卡命令过程字错误

返回（正数）	错误类型
0x01(001)	未放置卡片或认证错误

0x02(002)	数据校验错误
0x03(003)	数值为空错误
0x04(004)	认证失败
0x05(005)	奇偶校验错误
0x06(006)	读写设备与卡片通讯错误
0x08(008)	读卡序列号错误
0x09(009)	密码类型错误
0x0a(010)	卡片尚未认证
0x0b(011)	读卡操作比特数错误
0x0c(012)	读卡操作字节数错误
0x0f(015)	写卡操作失败
0x10(016)	增值操作失败
0x11(017)	减值操作失败
0x12(018)	读卡操作失败
0x13(019)	传输缓冲区溢出
0x15(021)	传输帧错误
0x17(023)	未知的传输需求
0x18(024)	防冲突错误
0x19(025)	感应模块复位错误
0x1a(026)	非认证接口

0x1b(027)	模块通讯超时
0x3c(060)	非正常操作
0x64(100)	错误的参数值
0x7c(124)	错误的参数值

### 1.3.2 判断卡型的特征值

卡型	特征值 1	特征值 2
MIFARE 1	4	136
FM11RF32	4	83
S70	2	24
ULTRA LIGHT	68	4
SHC1102	13056	
MIFARE LIGHT	16	129
MIFARE PRO	8	32
DESFIRE	836	36
FM11RF005	5	
MIFARE PLUS	180	



## 1.3.3 DEMO 操作说明

HS300-hidpos 系列读写演示程序的说明:

1.在对任何类型的卡操作之前，先利用 DEMO 测试一下卡的类型是否正确。

2

M1 卡/S70 卡

对 M1 卡进行操作时:

1. 读写用户信息，是针对在读卡器里设备的数据进行读、写的操作。

2. 在“卡型操作”里的 MIFARE ONE/S70 卡的读写操作中，要先校验卡密码，才能进行卡的操作。同时，校验哪个块区的密码，下面的读写卡操作就是操作哪个块区。

3. 在“卡型操作”里的 MIFARE ONE/S70 卡的写操作中，以“ASCII 方式”写入数据时，若长度不够要以空格补齐，否则系统要有提示“写入长度不够”，不能进行卡的写入操作。

4、CPU 卡:

什么是 CPU 卡?

CPU 卡就相当于卡片中放了一台微型计算机，CPU 卡的物理结构组成是由硬件和软件两部分组成的，硬件包括：ROM、RAM、EEPROM、CPU，软件就是 COS，类似电脑上安装一个操作系统一样。只有硬件没有软件的裸机无法工作一样，只有安装了芯片卡操作系统的卡才能工作；

CPU 卡的核心在于其内部的 COS，就像是电脑的操作系统。可以建立目录、建立文件、读写文件、保存密钥、做加解密运算等操作。这些操作都要依据 COS 手册中的指令来实现，虽然各家 CPU 卡厂商的 COS 不尽相同，但一般都遵循 ISO7816 标准，最常用的就是 T=0、和 T=1 CPU 卡，对于我们读写器而言符合标准的 CPU 卡我们都支持。

关于 CPU 卡参数的设置说明

设置 CPU 卡的参数,上电后的默认参数是 `cpupro=0`(T=0 协议)`cpuetu=92`(波特率 9600)

卡操作时候的延时数据（十进制）不同波特率的卡，此参数的值不同，对于 9600 波特率的卡 `cpuetu` 设置成 92 对于 38400 波特率的卡 `cpuetu` 设置成 20。请根据参数需要设置！

关于读写器操作 CPU 卡的说明：

对于用读写器操作 CPU 卡而言一般用两个命令就可以实现：第一个是 CPU 卡复位，这个命令是对 CPU 卡上电并得到卡片的初始信息，复位成功后就可以进行指令操作。指令操作是发送 COS 手册上的命令到 CPU 卡并从 CPU 卡获得返回信息。对于读写器而言，就是建立一个卡片和用户的通道。在通道上传输数据要根据自己的卡片的 COS 手册、文件结构、密钥体系和要实现的功能来决定。

如何测试发送数据？

在命令数据空格输入：0084000008---取随机数指令（其他指令请参阅相应的 COS 手册）

或 00A40000023f00---取主文件指令，点击“发送命令”，返回相应数据。

FM11RF005M 卡：

在对 FM11RF005 操作时，密码是 4 个字节，但在装载设备密码和校验密码时要将此 4 个字节后面补 2 个字节的 0 即后 4 位补 0，凑成 6 个字节来进行操作。

在对 FM11RF005 操作时进行读写操作时，要首先寻卡和校验密码。