Advanced Card Systems Ltd.
Card & Reader Technologies

ACR890
一体化移动终端

参考手册 V2.00

info@acs.com.hk
www.acs.com.hk

目录

表目录

1.0. 简介

ACR890 是集智能卡、磁条以及非接触技术为一体的新一代高性能移动终端，其高分辨率触摸屏能够带来市场上最具交互性的用户界面和功能体验，非常适合客户使用。这款先进的产品提供更快的处理速度和更大的内存容量及便携性。

本参考手册将介绍 ACR890 终端专用的 API（应用程序编程接口）函数。软件开发人员可使用这些 API 开发自己的智能卡应用。

2.0. 特性

- 32 位 A8 处理器，运行嵌入式 Linux®操作系统
- 4 GB 闪存和 256 MB DDR RAM
- 支持 Micro SD 卡扩展（1GB - 16GB）
- 连接方式：
  - Wi-Fi
  - 4 频段 GPRS/GSM(850 MHz、900 MHz、1800 MHz、1900 MHz)
  - 3G（900 MHz/2100 MHz 或 850 MHz/1900 MHz）
  - USB 客户端（高速，Micro-B 型接头）
  - RS-232 串口（Mini-B 型接头)
- 接触式界面：
  - 1 个全尺寸接触卡卡槽（下落式卡座）
- 非接触界面：
  - 集成式非接触智能卡接口
- 支持磁条卡
- SAM 接口：
  - 2 个 SAM 卡卡槽（摩擦式卡座）
- SIM 接口：
  - 1 个标准 SIM 卡卡槽（用于 GPRS 功能）
- 条码扫描器（可选）
- 固件升级
- 内置外设：
  - 易于阅读的高分辨率彩色显示屏
  - 3.5 英寸电阻式触摸屏
  - 高度耐用且抗化学品的 20 键键盘
  - 热敏打印机
  - 带独立备用电池的实时时钟（RTC）
  - 4 个 LED 状态指示灯
  - 内置扬声器
- 符合下列标准：
  - ISO 7816
  - ISO 14443
  - ISO 7811
  - USB Full Speed
  - RoHS 2

3.0. ACR890 阶段 1 和阶段 2 命令对比

ACR890 阶段 1 和阶段 2 设备的命令对比见下表。

***注：**此列表并未列出所有的可用命令。如果某命令未在表中列出，则该命令既适用于 ACR890 阶段 1 设备，也适用于阶段 2 设备。*

命令	ACR890 阶段 1	ACR890 阶段 2
GPRS 模块		
开启 GPRS 模块	`int gprs_power_on(void)`	`int gprs_power_on(unsigned char timeout)`
打印机模块		
重启打印机	适用	不适用
向打印机送纸	`int printer_page_feed(unsigned char nr_len)`	`int printer_page_feed(char cControl, char cLine)`
准备打印机以打印数据	不适用	适用
设置标准模式行距	适用	不适用
开始打印数据	不适用	适用
打印 bmp 文件	不适用	适用
在页面模式下打印字符串	适用	不适用
在标准模式下打印字符串	`int printer_printStrSM(const char *str)`	`int printer_printStrSM(const char *pString, const int nLen, unsigned int Mode)`
在标准模式下打印数据数组	适用	不适用
在页面模式下打印数据数组	适用	不适用
打印图像。	`int printer_print_img(const unsigned char *bitmap, unsigned short width, unsigned short height, unsigned char mode);`	不适用
智能卡操作		
查看是否插入接触式卡	适用	**修改为用于智能卡操作的 PC/SC API**
打开接触式智能卡	适用	**修改为用于智能卡操作的 PC/SC API**
关闭接触式智能卡	适用	**修改为用于智能卡操作的 PC/SC API**
向接触式智能卡发送 PPS	适用	**修改为用于智能卡操作的 PC/SC API**
向接触式智能卡发送 APDU 命令	适用	**修改为用于智能卡操作的 PC/SC API**

命令	ACR890 阶段 1	ACR890 阶段 2
开启非接触式读写器模块	适用	修改为用于智能卡操作的 PC/SC API
关闭非接触式读写器模块	适用	修改为用于智能卡操作的 PC/SC API
读取非接触式卡	适用	修改为用于智能卡操作的 PC/SC API
激活非接触式卡	适用	修改为用于智能卡操作的 PC/SC API
取消激活非接触式卡	适用	修改为用于智能卡操作的 PC/SC API
传输非接触卡数据	适用	修改为用于智能卡操作的 PC/SC API
传输 FeliCa 卡数据	适用	修改为用于智能卡操作的 PC/SC API
开启/关闭非接触天线	适用	修改为用于智能卡操作的 PC/SC API
打开管理模块电源		
获取当前系统休眠时间	不适用	适用
启用或禁用系统自动休眠。	适用	不适用

表1： ACR890 阶段 1 和阶段 2 命令对比

如需了解更多仅适用于 ACR890 阶段 1 设备的命令信息，请参考 **ACR890 阶段 1 命令**。

3.1. 识别 ACR890 设备

您可查看产品背部标签来识别 ACR890 设备的阶段版本。下表可帮您确定 ACR890 的所属阶段。

ACR890	序列号	固件版本号
阶段 1	以 RR312 开头	1XXX
阶段 2	以 RR400 开头	2XXX

表2： ACR890 阶段 1 和阶段 2 属性对比

4.0. 文件及目录结构

文件名称	文件位置	描述
acs_api.h	主机	API 头文件
acs_errno.h	主机	API 返回的错误编号的定义
libacs_api.so	目标机	API 共享库

5.0. 键盘的 API

本节介绍用于配置设备键盘的 API 函数。

5.1. 打开键盘文件说明符

此函数用于打开键盘文件说明符。

```
int kpd_open()
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

5.2. 关闭键盘文件说明符

此函数用于关闭键盘文件说明符。

```
int kpd_close()
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

5.3. 获取当前键盘状态

此函数用于返回按压状态以及按下按键时的键码值。

```
int kpd_state_get(struct kPoint *keycode, unsigned int timeout)
```

参数

```
struct kPoint {
      unsigned short type;
      unsigned short code;
};
```

keycode [out]　　所按下按键的键码。

timeout [in]　　等待时间，用于获取所按下按键的键码（以毫秒为单位）。

返回值

如果成功，返回值为 0。

如果失败或超时，返回值为-2。

其他情况下，返回值为-1。

要求

头文件　　在 acs_api.h 中声明

库文件　　使用 libacs_api.so

5.4. 设置电源按钮工作模式

此函数用于设置电源按钮的工作模式。

```
int pwrbtn_set_mode(enum pwrbtnMode nMode)
```

参数

```
enum pwrbtnMode {
      CMD_TESTMODE=0,
      CMD_ONOFFMODE,
      CMD_FAIL
};
```

nMode [in]　　要输入的模式值。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

5.5. 获取电源按钮工作模式

此函数用于获取电源按钮的当前工作模式。

```
int pwrbtn_get_mode (enum pwrbtnMode *pMode)
```

参数

```
enum pwrbtnMode {
      CMD_TESTMODE=0,
      CMD_ONOFFMODE,
      CMD_FAIL
};
```

pMode [out] 指向存储模式值的指针。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   struct kPoint key_Point;

   enum pwrbtnMode mode = CMD_TESTMODE;
   enum pwrbtnMode m;

   ret = kpd_open();


   pwrbtn_get_mode(&m);       //obtain current powerkey working mode
   printf("m1 = %d\n",(int)m);

   pwrbtn_set_mode(mode);  //set current powerkey working mode to Test
   Mode

   pwrbtn_get_mode(&m); //obtain current powerkey working mode
   printf("m2 = %d\n",(int)m);

   ret = kpd_state_get(&key_Point,5000);  //read key press within 5s
```

```
    printf("Type:%d, Code:%d\n", key_Point.type, key_Point.code);

    mode = CMD_ONOFFMODE;
    pwrbtn_set_mode(mode);  //set  current  powerkey  working  mode  to
    PowerKey Mode

    pwrbtn_get_mode(&m); //obtain current powerkey working mode
    printf("m3 = %d\n",(int)m);

    ret = kpd_close();
    printf("ret = %d\n", ret);

    return 0;
}
```

6.0. 背光控制的 API

本节将介绍用于背光配置的 API 函数。

6.1. 获取当前背光级别

此函数用于返回当前背光状态。

```
int backlight_get(struct bl_state *stat)
```

参数

```
struct bl_state {
      int brightness;   //current user requested brightness level(0 -
      max_brightness)
      int max_brightness;// maximal brightness level
      int fb_power;  //current fb power mode (0: full on, 1..3: power
      saving; 4: full off)
      int actual_brightness;// actual brightness level
} ;
```

stat [out]　　指向返回的背光状态的指针。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1 或-2。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   struct bl_state state;

   ret = backlight_get(&state); //call api to get backlight state
   if(0 == ret)
   {//show out the backlight state you get just now.
      printf("brightness=%d.max_brightness=%d,fb_power=%d,actual_brightn
   ess=%d",
   state.brightness,state.max_brightness,state.fb_power,state.actual_bri
   ghtness);
   }

   return ret;
}
```

6.2. 设置背光级别

此函数用于设置背光的亮度。

```
int backlight_set(enum bl_level level)
```

参数

```
enum bl_level {
      BACKLIGHT_LEVEL_0 = 0,/* Turn off */
      BACKLIGHT_LEVEL_1,
      BACKLIGHT_LEVEL_2,
      BACKLIGHT_LEVEL_3,
      BACKLIGHT_LEVEL_4,
      BACKLEGHT_LEVEL_5,
      BACKLEGHT_LEVEL_6,
      BACKLEGHT_LEVEL_7,
      BACKLEGHT_LEVEL_8,
      BACKLEGHT_LEVEL_9,
      BACKLIGHT_LEVEL_MAX
};
```

level [in]　　指定的背光亮度级别。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1 或-2。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   enum bl_level level = BACKLIGHT_LEVEL_4;

   ret = backlight_set(level); //call api to set the level of backlight
   brightness.

   return ret;
}
```

7.0. 电池和充电器的 API

本节将介绍用于配置电池和充电器的 API 函数。

7.1. 获取电池及充电器状态

此函数用于返回当前电池状态。若电源管理 IC 没有处于工作状态，则检测不到电池。

```
int battery_state_get(struct battery_state *stat)
```

参数

```
struct battery_state {
   int ifdc;//if have dc power   [0/1 = dc power absent/present]
   int ifbattery;//if have battery power  [0/1 = battery absent/present]
   int chargerstate;//charger state
      [0/1/2/3=discharging/charging/full]
   unsigned int batt_voltage; //battery voltage[uV]
   unsigned int batt_voltage_max; //battery max voltage[uV]
   unsigned int batt_voltage_min; //battery min voltage[uV]
   unsigned int batt_volpercent; //battery capacity [%]
   };
```

stat [out]　返回的电池状态信息。

返回值

如果成功，返回值为 0。

如果失败，返回值小于 0。

要求

头文件　在 in acs_api.h 中声明

库文件　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   struct battery_state stat;
   ret = bat_get_charger_state(&state); //call api to get battery and
   charger state
   if(ret == 0)
   { //print the battery state you get just now.
      printf("ifdc = %d, ifbattery = %d, chargerstate = %d, batt_voltage
   = %d,       batt_voltage_max    =    %d,    batt_voltage_min    =    %d,
      batt_volpercent          =          %d\n",          state.ifdc,
      state.ifbattery,state.chargerstate,          state.batt_voltage,
      state.batt_voltage_max,                  state.batt_voltage_min,
      state.batt_volpercent);
   }
   return ret;
}
```

8.0. LED 控制的 API

本节介绍用于配置 LED 指示灯的 API 函数。

8.1. 设置 LED 状态

该函数将单个 LED 的状态设置为开启、关闭或闪烁。

```
int led_set_state(enum led_id led, struct led_state stat)
```

参数

```
enum led_id {
   LED_ID_BLUE = 0,
   LED_ID_YELLOW,
   LED_ID_GREEN,
   LED_ID_RED,
   LED_ID_MAX,
};
enum led_blink_state {
   LED_STATE_SOLID_OFF = 0,
   LED_STATE_SOLID_ON,
   LED_STATE_BLINK,
   LED_STATE_MAX,
};
struct led_state {
enum led_blink_state bs;//led blink state
unsigned int on_time;   //led blink state on period time in ms
unsigned int off_time;  //led blink state off period time in ms
};
```

led [in]　　指定的 LED 的 ID 号。

stat [in]　　指定的 LED 的状态。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1 或-2。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

8.2. 获取 LED 的闪烁状态

此函数用于返回指定的 LED 的当前状态。

```
int led_get_state(enum led_id led, struct led_state *stat)
```

参数

led [in]　　LED 的 ID 号。

stat [out]　　指向返回的 LED 状态的指针。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   enum led_id led = LED_ID_BLUE; //get 0-blue led state
   struct led_state stat;
   int ret;

   memset(&stat, 0x00, sizeof(struct led_state));

   ret = led_get_state(led, &state); //call API to get led state
   if(0 == ret)
   {
      printf("led-d%,state=d%,ontime=%d,offtime=%d.\n",
      stat.bs, stat.on_time, stat.off_time);
   }
   else
   {
      printf(“Fail to get current led state, ret=%d\n”, ret);
   }
   stat.bs = LED_STATE_BLINK;
   stat.on_time = 100;
   stat.off_time = 900;
   //call API to set blue led blink on for 100ms and blink off for 900ms
   periodically.
   ret = led_set_state(led, stat);
   if(0 != ret)
   {
      printf( " Set led blink state failed !, ret = %d\n", ret);
   }

   return ret;
}
```

9.0. GPRS 模块电源管理的 API

本节介绍用于配置 GPRS 模块的 API 函数。

9.1. 开启 GPRS 模块

此函数用于开启 GPRS 模块。

```
int gprs_power_on(unsigned char timeout)
```

注：如果使用的是 ***ACR890 阶段 1*** *设备，命令格式为：*

```
int gprs_power_on(void)
```

参数

timeout [in] 检测*/dev/ttyUSB2* 的等待时间。若操作超时且*/dev/ttyUSB2* 不存在，会返回 *EGPRS_POWER_ON_TIMEOUT*。若找到/dev/ttyUSB2，会返回 *EGPRS_SUCCEEDED*。

若超时时间值等于 0 且检测不到 */dev/ttyUSB2*，会立刻返回。

若超时时间值不等于 0，超时时间可设为≥5 并且 ≤ 9。API 会检测*/dev/ttyUSB2*。

返回值

如果成功，返回值为 *EGPRS_SUCCEEDED*。

如果失败，返回值为 *ENODEV* 或 *EGPRS_POWER_ON_FAILED*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

9.2. 关闭 GPRS 模块

此函数用于关闭 GPRS 模块。

```
int gprs_power_off(void)
```

参数

无。

返回值

如果成功，返回值为 *EGPRS_SUCCEEDED*。

如果失败，返回值为 *ENODEV 或 EGPRS_POWER_OFF_FAILED*。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

9.3. 设置 pppd 连接参数

此函数用于设置 *pppd* 参数，例如电话、本地 IP、远程 IP 以及网络掩码。

```
int set_ppp_param(char *telephone, char *local_ip, char *remote_ip, char
*netmask).
```

参数

telephone [in] 用于网络拨号的电话号码（例如*99***1#）。

local_ip [in] 本地 IP 地址（如果已知；动态 = 0.0.0.0.）。

remote_ip [in] 远程 IP 地址（如果必要；通常为 0.0.0.0）。

netmask [in] 正确的网络掩码（如果需要）。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

9.4. 设置 pppd 拨号器参数

此函数用于设置拨号器的参数，例如协议和登录_点。

```
int set_dialer_param(char *protocol , char *login_point).
```

参数

protocol [in] 通信协议（例如 IP）。

login_point [in] 移动网络运营商支持的 APN。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

9.5. 开始 pppd 过程

此函数用于开始 *pppd* 拨号过程。

```
void ppp_on(void).
```

参数

无。

返回值

无。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

9.6. 关闭 pppd 过程

此函数用于关闭 *pppd* 拨号过程。

```
void ppp_off(void ).
```

参数

无。

返回值

无。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
/* Tips :After you finish using ppp_on() connected the Internet, please
execute ppp_off() to disconnect Internet, and finally execute
gprs_power_off() to turnoff 3g modules;*/

int main(int argc, char *argv[])
{
   int ret=0;
   int count = 0;

   ret = gprs_power_on();
   if(ret != EGPRS_SUCCEEDED)
   {
      printf("gprs power on failed, ret = %d\n”,ret);
      return -1;
```

```
    }

    /* notice:After poweron 3g module, must wait for 9s, and then check
if '/dev/ttyUSB2' exist */
    sleep(9);
    if(access("/dev/ttyUSB2",0) != 0)
    {
        printf("no exist /dev/ttyUSB2\n");
        return -1;
    }

    ret     =     set_ppp_param("*99***1#",     "0.0.0.0",     "0.0.0.0",
"255.255.255.0");
    if(ret != 0)
    {
        printf("set ppp param Failed!\n);
        gprs_power_off();
        return -1;
    }

    ret = set_dialer_param("IP", "3gnet");
    if(ret != 0)
    {
        printf("set dialer param Failed!\n");
        return -1;
    }
    ppp_on();

    while(1)
    {
        printf("count = %d\n",count);
        ret = system(" ifconfig | grep 'ppp0' ");
        if(ret == 0)
        {
            system("cp /etc/ppp/resolv.conf /etc/resolv.conf");
            break;
        }
        sleep(1);
        count++;
        if(count > 15)
        {
            printf("Timeout!!!\n");
            break;
        }
    }
    return 0;
}
```

9.7. 设置传输超时时间

此函数用于为 TransmitATCmd API 函数设置超时时间，以毫秒为单位。

int SetTransmitTimeoutMs(int tMs).

参数

tMs [in]　　设置 transmit_timeout 为 tMs 毫秒。

返回值

无

要求

头文件　　在 acs_api.h 中声明

库文件　　使用 libacs_api.so

9.8. 传输一个 AT 命令

此函数向 3G 模块发送一个 AT 命令，然后获取一个响应字符串编码。

```
int TransmitATCmd(char *AtCmd, char *RecvBuffer, int RecvLength).
```

参数

AtCmd [in]　　要发送给 3G 模块的 AT 命令。

RecvBuffer [out]　　存放 3G 模块响应字符串编码的缓冲区

RecvLength [in]　　RecvBuffer 的长度。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 acs_api.h 中声明

库文件　　使用 libacs_api.so

9.9. 获取 IMEI 序列号

此函数返回 3G 模块的 IMEI（国际移动设备识别码）序列号信息。

```
int Get_IMEI_SN(char *IMEI_SN, int IMEILength).
```

参数

IMEI_SN [out]　　存放 IMEI 序列号信息的缓冲区

IMEILength [in]　　*IMEI_SN* 缓冲区的长度。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(int argc, char *argv[])
{
   int ret=0;
   char IMEI_Buf[64];

   ret = gprs_power_on();
   if(ret != EGPRS_SUCCEEDED)
   {  printf("gprs power on failed, ret = %d\n”,ret);
      return -1;
   }
   /* notice:After poweron 3g module, must wait for 9s, and then check
if ‘/dev/ttyUSB2’ exist */
   sleep(9);
   if(access(“/dev/ttyUSB2”,0) != 0)
   {
      Printf(“no exist /dev/ttyUSB2\n”);
      return -1;
   }
   ret = Get_IMEI_SN(IMEI_Buf, sizeof(IMEI_Buf));
   if(ret != 0)
   {  printf("Get IMEI_SN Failed, ret = %d\n”,ret);
      gprs_power_off();
      return -1;
   }
   printf(“IMEI Serial Number = %s\n”, IMEI_Buf);

   ret = gprs_power_off();
   if(ret != EGPRS_SUCCEEDED)
   {  printf(“gprs power off Failed!\n”);
      return -1;
   }
   return 0;
}
```

10.0.音频（ALSA）的 API

本节介绍用于进行设备音频设置的 API 函数。

10.1. 获取系统音频音量

此函数返回系统音频音量。

```
int audio_volume_get(struct volume_state *stat)
```

参数

```
struct volume_state {
   unsigned int min_vol; //the minimal level of volume
   unsigned int max_vol; //the maximal level of volume
   unsigned int current_vol;  //the left current volume
};
```

stat [out]　　指向所返回音量状态的指针。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   struct volume_state stat;

   memset(&stat, 0x00, sizeof(struct volume_state));

   ret = volume_get((&state); //call API to Obtain volume level
   if(0 == ret)
   {//print the volume state you get just now.
      printf("max volume is %ld\nmin volume is %ld\n, left
      volume is %ld\nright volume is %ld\n", stat.max_vol,
      state.min_vol, stat.left_vol, stat.right_vol);
   }

   return ret;
}
```

10.2. 设置系统音频音量

此函数用于设置音量。

```
int audio_volume_set(unsigned int volume)
```

参数

volume [in]　　要设定的音量级别（范围从 0 到 18；高于 18 的音量级将按 18 级处理）。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

10.3. 声音文件重放

此函数用于重放 WAVE 格式声音文件。

```
int sound_play(char *file_path)
```

参数

file_path [in]　　声音文件的完整路径。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;

   ret = sound_play ("./Niose.wav");//call api to play a specified audio
   file

   return ret;
}
```

10.4. 扬声器声音控制

此函数用于开启/关闭扬声器声音。

```
int speaker_onoff(int onoff)
```

参数

onoff [in]　　1 = 开启声音；0 = 关闭声音

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;

   ret = speaker_onoff(1);//call api to sound on speaker.

   return ret;
}
```

11.0.固件的 API

本节介绍用于配置设备固件的 API 函数。

11.1. 获取固件版本号

此函数返回设备的固件版本号（主版本号、次版本号、修订版本号）。

```
int get_acr890_version(struct acr890_version *version)
```

参数

```
struct acr890_version{
   unsigned int major;
   unsigned int minor;
   unsigned int revision;

};
```

version [out]　　固件版本号（主版本号、次版本号和修订版本号）的数据指针

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   struct acr890_version version;
   ret = get_acr890_version(&version);
   if(ret == 0) {
      printf("Major = %d\n", version.major);
      printf("Minor = %d\n", version.minor);
      printf("Revision = %d\n", version.revision);

   }
   return ret;
}
```

12.0.热敏打印机的 API

本节介绍用于配置设备热敏打印机的 API 函数。

12.1. 开启打印机端口

此函数用于开启打印机端口。

```
int printer_open(void)
```

参数

无

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_OPEN_ERR*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

12.2. 关闭打印机端口

此函数用于关闭打印机端口。

```
int printer_close(void)
```

参数

无

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_CLOSE_ERR*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

12.3. 获取打印机状态

此函数返回打印机状态。

```
int printer_status_get(void)
```

参数

```
enum printer_state {
      PRINT_STAT_UNKNOWN = 0, /* unknown state */
      PRINT_STAT_INT  =  0x53, /* print suspend (no paper) */
      PRINT_STAT_IDLE = 0x90, /* Idle state */
      PRINT_STAT_BFULL = 0x65, /* full of buffer */
      PRINT_STAT_NOPAPER = 0x68, /* out of paper */
      PRINT_STAT_BFNP = 0x63, /* full of buffer and out of paper */
      PRINT_STAT_MAX
};
```

返回值

任何 *enum printer_state* 值。

要求

头文件 在 acs_api.h 中声明。

库文件 使用 libacs_api.so。

12.4. 向打印机送纸

此函数用于向打印机送纸。

```
int printer_page_feed(char cControl, char cLine)
```

***注：**如果使用的是 **ACR890 阶段 1** 设备，命令格式为：*

```
int printer_page_feed(unsigned char nr_len)
```

参数

cControl [in] 控制打印机轮向前或向后的代码。

0 – 向前进纸

1 – 向后进纸

nr_len [in] 进纸的距离（范围是 0 - 255。距离等于 *nr_len* x 0.125，以毫米为单位）

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_FPAPER_ERR*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

12.5. 准备打印机以打印数据

此函数用于准备打印机的打印数据。

```
int printer_prepare_data(char *pstr, const int nlen)
```

参数

pstr [in] 给打印机的输入字符串数据。

nlen [in] 输入字符串数据的长度。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_DATA_ERR*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

12.6. 开始打印数据

此函数开始打印刚刚准备好的字符串数据。

```
int printer_printing(void)
```

参数

无

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_STRPRINT_ERR*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

12.7. 设置打印机

此函数用于设置打印间距和字体像素大小。

```
int printer_setPrinter(unsigned int uiFormat,unsigned int iFormatValue)
```

参数

uiFormat [in]　　设置行间距和列间距

uiFormatValue [in]　设置字体像素大小，有效值如下：

EM_prn_ASCIIPRN1X1
EM_prn_HZPRN1X1
EM_prn_ASCIIPRN2X1
EM_prn_HZPRN2X1
EM_prn_ASCIIPRN1X2
EM_prn_HZPRN1X2
EM_prn_ASCIIPRN2X2
EM_prn_HZPRN2X2
EM_prn_ASCIIPRN1X3
EM_prn_HZPRN1X3
EM_prn_ASCIIPRN3X1
EM_prn_HZPRN3X1
EM_prn_ASCIIPRN3X2
EM_prn_HZPRN3X2
EM_prn_ASCIIPRN2X3
EM_prn_HZPRN2X3
EM_prn_ASCIIPRN3X3
EM_prn_HZPRN3X3

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_STRPRINT_ERR*。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

12.8. 在标准模式下打印字符串

此函数在标准模式下打印字符串。打印数据的长度不得超过 65535 个字节。可以使用控制字符'\n'。

```
int printer_printStrSM(const char *pString, cons tint nLen, unsigned int Mode)
```

参数

pString [in]　　待打印的以空字符结尾的字符串。

nLen [in]　　输入的 *pString* 的长度。

Mode [in]　　打印时的对齐方式。

09h - 右对齐

0Ah - 中间对齐

08h 或 0Bh - 左对齐

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_STRPRINT_SM*。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

12.9. 打印位图文件

此函数打印位图（.bmp）文件。

```
int printer_print_bmp(const unsigned char *pBmpName, unsigned int uiStartPosition);
```

参数

pBmpName [in]　　输入的位图文件的完整路径名。位图文件支持的颜色深度为：1-比特、8-比特、24-比特和 32-比特。

uiStartPosition [in]　　打印时的起始位置。

返回值

如果成功，返回值为 *SUCCESS*。

如果成功，返回值为 *ETHP_IMAGEPRINT*。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

示例代码

```
int main(int argc, char *argv[])
{
   int nRet=0;
   char szcom[64]={"ABCDEF1234567890ABCDEFIJG12345QWERT"};
   nRet=printer_open();
   if(nRet<0)
   {
      printf("printer_open erro %d\n",nRet);
   }
    nRet = printer_status_get();
   if(nRet != PRINTER_READY)
   {
      printf("printf error %d\n", nRet);
      return -1;
   }
   //standard mode printing
   printer_printStrSM(szcom, strlen(szcom), 0x08);

   nRet = printer_status_get();
   if(nRet !=PRINTER_READY)
   {
      printf("printer_printStrPM nRet = [%x]\n",nRet);
      return -1;
   }

   //printing and change new line
    printer_page_feed(0, 3);


  //Reset and cache flush
   printer_close();
}
```

12.10. 初始化 FreeType 库

此函数用于初始化 freetype 库，专门用于非英语字体的打印。

注：*只有固件版本 v1.1.5 及更高版本支持该函数。*

```
int freetype_init(const char* fontname)
```

参数

fontname [in]　　字体文件的路径。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_INITFREE_ERR*。

要求

头文件　　在 lib_freetype.h 中声明

库文件　　使用 libacs_printer.so

12.11. 设置字符大小

此函数用于设置字符的大小，专门用于非英语字体的打印。

注：只有固件版本 v1.1.5 及更高版本支持该函数。

```
int freetype_setsize(unsigned int width, unsigned int height)
```

参数

width [in]　　字符宽度（以整像素为单位）。有效范围是 11-16。

height [in]　　字符高度（以整像素为单位）。有效范围是 11-16。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_SETSIZE_ERR*。

要求

头文件　　在 lib_freetype.h 中声明

库文件　　使用 libacs_printer.so

12.12. 获取字形图像及打印

此函数用于获取字形图像并打印，专门用于非英语字体的打印。

注：*只有固件版本 v1.1.5 及更高版本支持该函数。*

```
int freetype_printString(char *string, int nlen)
```

参数

string [in]　　指向待打印的字符数组的指针。

nlen [in]　　待打印的字符数组的大小，以字节为单位。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_IMAGEPRINT*。

要求

头文件　　在 lib_freetype.h 中声明

库文件　　使用 libacs_printer.so

12.13. 释放 freetype 库

此函数用于释放 freetype 库，专门用于非英语字体的打印。

注：*只有固件版本 v1.1.5 及更高版本支持该函数。*

```
void freetype_release(void)
```

参数

无。

返回值

无。

要求

头文件 在 lib_freetype.h 中声明

库文件 使用 libacs_printer.so

13.0.无线 LAN 模块的 API

本节介绍用于配置无线 LAN 模块的 API 函数。

13.1. 开启无线 LAN 模块

此函数用于开启无线 LAN 模块。

```
int wifi_pwr_on(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值不等于-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

13.2. 关闭无线 LAN 模块

此函数用于关闭无线 LAN 模块。

```
int wifi_pwr_off(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

14.0.蓝牙®模块的 API

本节介绍用于配置蓝牙模块的 API 函数。

14.1. 开启蓝牙模块。

此函数用于开启蓝牙模块。

```
int bluetooth_pwr_on(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

14.2. 关闭蓝牙模块

此函数用于关闭蓝牙模块。

```
int bluetooth_pwr_off(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

15.0.智能卡读写器的 API

本节将介绍设备的接触式和非接触式智能卡读写器模块的 API 函数。

15.1. 开启智能卡读写器模块

此函数用于开启设备的智能卡读写器模块。在执行 PC/SC 命令之前，会有两秒左右的延迟。

```
int smartcard_open(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值 != 0。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

15.2. 关闭智能卡读写器模块

此函数用于关闭设备的智能卡读写器模块。

```
int smartcard_close(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值 ≠ 0。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

15.3. 用于智能卡操作的 PC/SC API

关于 PC/SC API 函数的列表和说明，请参考以下链接：

https://pcsclite.alioth.debian.org/api/modules.html

示例代码

```
int main(int argc,char *argv[])
{
SCARDCONTEXT hcontext;
char readerName[8][32];

lone rv;
lone len;
int readerNum = 0;
char *pReader;
char pmszReaders[256];

int ret;

/***Open pcscd daemon process***/
ret = smartcard_open();
if(ret != 0)
{
   return -1;
}

/***List All Reader***/
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hcontext);
if(rv != SCARD_S_SUCCESS)
{
   printf("rv = %s\n", pcsc_stringify_error(rv));
}
len = sizeof(pmszReaders);
rv = SCardListReaders(hcontext, NULL, pmszReaders, &len);
if(rv != SCARD_S_SUCCESS)
{
   printf("rv = %s\n", pcsc_stringify_error(rv));
   return rv;
}
pReader = pmszReaders;
while(*pReader != '\0')
{
   strcpy(readerName[readerNum], pReader);
   pReader = pReader + strlen(pReader) + 1;
   readerNum++;
}

/***Transmit Apdu Data***/
DWORD dwAP;
DWORD dwLen;
DWORD dwAtrLen;
unsigned char pAtr[64];
DWORD dwState;
DWORD dwProtocol;
int retval;
int I;
SCARD_IO_REQUEST ioSendPci;
SCARD_IO_REQUEST ioRecvPci;
```

```
unsigned char pTx[] = {0x80, 0x84, 0x00, 0x00, 0x08};
unsigned int txLen;
unsigned char pRx[64];
unsigned char rxLen = sizeof(pRx);

rv    =    SCardConnect(hcontext,    readerName[0],    SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1, &hcard, &dwAP);
if(rv != SCARD_S_SUCCESS)
{
   printf("rv = %s\n", pcsc_stringify_error(rv));
   return rv;
}

ioSendPci.dwProtocol = dwAP;
ioSendPci.cbPciLength = sizeof(SCARD_IO_REQUEST);

rv  =  SCardTransmit(hcard,  &ioSendPci,  pTx,  (DWORD)txLen,  &ioRecvPci,
pRx, (DWROD*)&rxLen);
if(rv != SCARD_S_SUCCESS)
{
   printf("rv = %s\n", pcsc_stringify_error(rv));
   return rv;
}

/***Close Smart Card Reader***/
SCardDisconnect(hcard, SCARD_LEAVE_CARD);
SCARDReleaseContext(hcontext);

smartcard_close();

}
```

16.0.磁条接口的 API

本节介绍用于配置磁条模块的 API 函数。

16.1. 从磁条卡获取磁道数据

此函数用于在指定期间内从磁条卡读取磁道数据。

```
int msr_trackdata_get(struct msr_data *data, unsigned int time)
```

参数

```
struct msr_data {/* Each track data end with character '\0' */
   char track_data1[80]; /* track #1 data */
   char track_data2[41]; /* track #2 data */
   char track_data3[108]; /* track #3 data */
   unsigned int track_state;
      };
```

data [out]　　包含磁道数据和状态。

time [in]　　刷卡等待时间（以秒为单位）刷卡通常应在 5-30 秒内完成。

位	说明
Bit 31:27	保留
Bit 26	磁道#3 存在数据
Bit 25	磁道#2 存在数据
Bit 24	磁道#1 存在数据
Bit 23:20	保留
Bit 19	磁道#3 数据 LRC 错误
Bit 18	磁道#3 数据结束字节错误
Bit 17	磁道#3 数据奇偶校验错误
Bit 16	磁道#3 数据起始字节错误
Bit 15:12	保留
Bit 11	磁道#2 数据 LRC 错误
Bit 10	磁道#2 数据结束字节错误
Bit 9	磁道#2 数据奇偶校验错误
Bit 8	磁道#2 数据起始字节错误
Bit 7:4	保留
Bit 3	磁道#1 数据 LRC 错误
Bit 2	磁道#1 数据结束字节错误
Bit 1	磁道#1 数据奇偶校验错误
Bit 0	磁道#1 数据起始字节错误

表3： 磁道数据状态位

返回值

如果所有磁道都 OK，返回值为 0。

否则返回值不等于 0。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(int argc, char *argv[])
{
      struct msr_data tMSRData;
      int iTimeout =30;
      int ret;
      int i;

      if(2 == argc)
      {
         iTimeout = atoi(argv[1]);/* Get how much time need to wait */
      }
      memset(&tMSRData, 0x00, sizeof(struct msr_data));
      ret = msr_track_data_get(&tMSRData, iTimeout);
      if (tMSRData.track_state & BIT(24))/* Got track #1 data */
      {
         printf("track #1 data :\n");
         for(i = 0; i < sizeof(tMSRData. track_data1); i++)
         {
      printf("0x%02X ", tMSRData. track_data1[i]);
         }
         printf("\n");
      }
      else
      {
         printf("track 1 error :");
         PRINT_MSR_ERROR(tMSRData.track_state);
         printf("\n");
      }

      return ret;
}
```

17.0. 错误代码说明的 API

本节介绍关于错误代码说明的 API 函数。

17.1. 获取错误代码说明

进行调试时，可以使用此 API 从指定的错误代码获取更多信息。

```
char *acs_err(const int errno_code)
```

参数

errno_code [in]　　待解析的错误编码。

返回值

错误编号解析字符串。

要求

头文件　　在 acs_api.h 中声明

库文件　　使用 libacs_api.so

18.0. 电源管理的 API

本节介绍用于设备的系统电源管理的 API 函数。

18.1. 设置系统休眠超时

此函数用于设置设备空闲时间。空闲时间到期后，系统将切换到休眠模式。然而任何输入事件都会使空闲计时器复位。

```
int pm_sleep_timeout_set(unsigned long seconds)
```

参数

seconds[in]　　进入休眠模式前的最大时间。取值范围是 30-1800。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 acs_api.h 中声明。

库文件　　使用 libacs_api.so。

18.2. 获取当前系统休眠模式

此函数用于设置设备当前的空闲模式

```
unsigned int pm_get_sleep_mode(enum sleep_mode *mode);
```

参数

mode [out]　　存储当前的系统休眠模式值。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件　　在 acs_api.h 中声明。

库文件　　使用 libacs_api.so。

18.3. 获取系统休眠时间

此函数用于获取设备空闲时间。

```
unsigned int pm_sleep_timeout_get(void)
```

返回值

设备的空闲时间值。

要求

头文件 在 acs_api.h 中声明。

库文件 使用 libacs_api.so。

19.0.条码扫描器的 API

本节介绍用于配置条码扫描器模块的 API 函数。

要求

头文件 在 acs_api.h 中声明。

库文件 使用 libacs_api.so。

19.1. 开启扫描器模块

```
int scanner_open(void)
```

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_OPEN_ERR (1056)。

19.2. 关闭扫描器模块

```
int scanner_close(void)
```

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_CLOSE_ERR (1057)。

19.3. 连接扫描器

```
int scanner_Connect(void)
```

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_CONNECT_ERR (1058)。

19.4. 扫描数据

```
int scanner_Scan(unsigned char pScanBuf, unsigned int pScanLen,unsigned
int TimeOutSec);
```

参数：

[out]- pScanBuf = 扫描数据缓冲区

[out]- pScanLen = 扫描数据长度

[in] - timeoutSec = 扫描超时时间（以秒为单位）

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_SCAN_ERR (1059)。

19.5. 获取扫描器的读取过程

```
int scanner_ReadingSession(int status)
```

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_READSESSION_PM (1062)。

19.6. 获取扫描器固件版本

```
int scanner_GetFirmwareVersion(unsigned char pBuf, unsigned int pLen); -
- get Scanner's firmware version
```

参数：

[out] - pBuf = 固件版本数据缓冲区

[out] pLen = 返回的缓冲区的长度

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_GETVERSION_SM (1061)。

19.7. 断开扫描器

```
`int scanner_Disconnect(void)
```

返回值

如果成功，返回值为 SUCCESS (0)。

如果失败，返回值为 ESCAN_DISCONNECT (1063)。

附录A ACR890 阶段 1 命令

本节介绍一些仅适用于 ACR890 阶段 1 设备的命令。

附录A.1. 非接触接口的私有 APDU 指令

本节介绍仅适用于 ACR890 阶段 1 设备并与非接触接口相关的命令。

附录A.1.1. 获取数据（Get Data）

此命令用于返回“已建立连接的 PICC”的序列号或 ATS。

GET UID 的 APDU 结构（5 个字节）

命令	CLA	INS	P1	P2	Le
Get Data	FFh	CAh	00h 01h	00h	00h (全长)

若 P1 = 00h，响应格式为获取 UID（UID + 2 个字节）

响应	响应数据域					
结果	UID (LSB)			UID (MSB)	SW1	SW2

若 P1 = 01h，则获取 ISO 14443 A 类卡的 ATS（ATS + 2 个字节）

响应	响应数据域		
结果	ATS	SW1	SW2

Get Data 的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
警告	62 82h	UID/ATS 的末尾先于 Le 字节到达（Le 大于 UID 的长度）
错误	6C XXh	长度错误（错误的 Le：‘XX’表示确切的数字），如果 Le 小于 UID 的长度
错误	63 00h	操作失败
错误	6A 81h	不支持此功能

例 1：

获取“已经建立连接的 PICC”的序列号

```
UINT8 GET_UID[5]={FFh, CAh, 00h, 00h, 00h}
```

例 2：获取“已经建立连接的 ISO 14443-A PICC”的 ATS

```
UINT8 GET_ATS[5]={FFh, CAh, 01h, 00h, 00h};
```

附录A.2. MIFARE Classic® 1K/4K 存储卡的 PICC 命令（T=CL 模拟）

附录A.2.1. 加载认证密钥（Load authentication keys）

此命令用于向读写器加载认证密钥。认证密钥用于验证 Mifare Classic 1K/4K 存储卡的特定扇区。读写器提供了两种认证密钥位置：易失密钥位置和非易失密钥位置。

Load Authentication Keys 的 APDU 结构（11 个字节）

命令	CLA	INS	P1	P2	Lc	命令数据域
Load Authentication Keys	FFh	82h	密钥结构	密钥号	06h	密钥（6 字节）

其中：

密钥结构 1 字节

00h = 秘钥被载入读写器的易失存储器

其它 = 保留

密钥号 1 字节

00h – 01h = 密钥位置。一旦读写器与电脑断开连接，密钥会被删除。

密钥 6 字节。载入读写器的密钥值。

例如：{FF FF FF FF FF FFh}。

Load Authentication Keys 的响应结构（2 字节）

响应	响应数据域	
结果	SW1	SW2

Load Authentication Keys 的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

例：

向密钥位置 00h 加载密钥{FF FF FF FF FF FFh}。

APDU = {FF 82 00 00 06 FF FF FF FF FF FFh}

附录A.2.2. MIFARE Classic 1K/4K 卡认证（Authenticate MIFARE Classic 1K/4K）

此命令使用存储在读写器内的密钥来验证 MIFARE Classic 1K/4K 卡（PICC），其中会用到两种认证密钥：TYPE_A 和 TYPE_B。

Load Authentication Keys 的 APDU 结构（6 个字节）

命令	CLA	INS	P1	P2	P3	命令数据域
Authentication	FFh	88h	00h	块号	密钥类型	密钥号

Load Authentication Keys 的 APDU 结构（10 个字节）

命令	CLA	INS	P1	P2	Lc	命令数据域
Authentication	FFh	86h	00h	00h	05h	认证数据字节

认证数据字节（5 字节）

字节 1	字节 2	字节 3	字节 4	字节 5
版本号 01h	00h	块号	密钥类型	密钥号

其中：

块号 1 个字节。指出待验证的存储块。

密钥类型 1 字节

60h = 该密钥被用作 TYPE A 密钥进行验证

61h = 该密钥被用作 TYPE B 密钥进行验证

密钥号 1 字节

00h ~ 01h = 密钥位置

注：*MIFARE Classic 1K 卡的内存分为 16 个扇区，每个扇区包含 4 个连续的块。例如：扇区 00 包含块{00h、01h、02h 和 03h}；扇区 01h 包含块{04h、05h、06h 和 07h}；最后一个扇区 0Fh 包含块{3Ch、3Dh、3Eh 和 3Fh}。*

验证通过后，读取同一扇区内的其他块不需要再次进行验证。详情请参考 MIFARE Classic 1K/4K 卡标准。

Load Authentication Keys 的响应结构（2 字节）

响应	响应数据域	
结果	SW1	SW2

Load Authentication Keys 的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

扇区 (共 16 个扇区，每个扇区包含 4 个连续的块)	数据块 (3 个块，每块 16 个字节)	尾部块 (1 个块，16 字节)
扇区 0	00h – 02h	03h
扇区 1	04h – 06h	07h
..	..	..
..	..	..
扇区 14	38h – 0Ah	3Bh
扇区 15	3Ch – 3Eh	3Fh

1 KB

表4： MIFARE 1K 卡的内存结构

扇区 (共 32 个扇区，每个扇区包含 4 个连续的块)	数据块 (3 个块，每块 16 个字节)	尾部块 (1 个块，16 字节)
扇区 0	00h – 02h	03h
扇区 1	04h – 06h	07h
...	...	...
...	...	...
扇区 30	78h – 7Ah	7Bh
扇区 31	7Ch – 7Eh	7Fh

2 KB

扇区 (共 8 个扇区，每个扇区包含 16 个连续的块)	数据块 (15 个块，每块 16 个字节)	尾部块 (1 个块，16 字节)
扇区 32	80h – 8Eh	8Fh
扇区 33	90h – 9Eh	9Fh
...	...	...
...	...	...
扇区 38	E0h – EEh	EFh
扇区 39	F0h – FEh	FFh

2 KB

表5： MIFARE 4K 卡的内存结构

例 1：

通过下列特征验证块 04h：A 类, 非易失, 密钥号 00h, 来自 PC/SC V2.01 (作废)。

APDU = {FF 88 00 04 60 00h};

例 2：

类似于前面的例子，通过下列特征验证块 04h：A 类, 非易失, 密钥号 00h, 来自 PC/SC V2.07。

APDU = {FF 86 00 00 05 01 00 04 60 00h}

注：MIFARE Ultralight®的用户数据区域可以自由读写，所以不需要验证。

字节号	0	1	2	3	页
序列号	SN0	SN1	SN2	BCC0	0
序列号	SN3	SN4	SN5	SN6	1
内部/锁	BCC1	Internal	Lock0	Lock1	2
OTP	OPT0	OPT1	OTP2	OTP3	3
数据读/写	Data0	Data1	Data2	Data3	4
数据读/写	Data4	Data5	Data6	Data7	5
数据读/写	Data8	Data9	Data10	Data11	6
数据读/写	Data12	Data13	Data14	Data15	7
数据读/写	Data16	Data17	Data18	Data19	8
数据读/写	Data20	Data21	Data22	Data23	9
数据读/写	Data24	Data25	Data26	Data27	10
数据读/写	Data28	Data29	Data30	Data31	11
数据读/写	Data32	Data33	Data34	Data35	12
数据读/写	Data36	Data37	Data38	Data39	13
数据读/写	Data40	Data41	Data42	Data43	14
数据读/写	Data44	Data45	Data46	Data47	15

512 位
或
64 字节

表6： MIFARE Ultralight® 卡的内存结构

附录A.2.3. 读二进制块（Read Binary Blocks）

此命令用于从 PICC 卡中取回多个数据块。执行该命令前必须先对数据块/尾部块进行验证。

Read Binary 的 APDU 结构（5 字节）

命令	CLA	INS	P1	P2	Le
Read Binary Blocks	FFh	B0h	00h	块号	待读取的字节数

其中：

块号 1 个字节。待访问的块。

待读取的字节数 1 个字节。最大值为 16 个字节。

Read Binary Block 的响应结构（N + 2 个字节）

响应	响应数据域		
结果	0 <= N <= 16	SW1	SW2

Read Binary Block 命令的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

例 1： 从二进制块 04h 中读取 16 个字节（MIFARE 1K 或 4K）

APDU = {FF B0 00 04 10}

例 2： 从二进制页 04h 中读取 4 个字节（MIFARE Ultralight）

APDU = {FF B0 00 04 04}

例 3： 从二进制页 04h 开始读取 16 个字节（MIFARE Ultralight）（读取页 4, 5, 6 和 7）

APDU = {FF B0 00 04 10}

附录A.2.4. 更新二进制块（Update Binary Blocks）

此命令用于向 PICC 写入多个数据块。执行该命令前必须先对数据块/尾部块进行验证。

Update Binary 的 APDU 结构（4 或 16 + 5 个字节）

命令	CLA	INS	P1	P2	Lc	命令数据域
Update Binary Blocks	FFh	D6h	00h	块号	待更新的字节数	块数据 MIFARE Ultralight：4 个字节 或 MIFARE 1K/4K：16 个字节

其中：

块号 1 个字节。待更新的起始块。

待更新的字节数 1 个字节。

MIFARE 1K/4K 的待更新字节数为 16 个字节

MIFARE Ultralight 的待更新字节数为 4 个字节

块数据 4 或 16 个字节

待写入二进制块的数据。

Update Binary Block 的响应状态码（2 字节）

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

例 1：将 MIFARE 1K/4K 卡中的二进制块 04h 的数据更新为{00 01 ...0Fh}

APDU = {FF D6 00 04 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0Fh}

例 2：将 MIFARE Ultralight 卡中的二进制块 04h 的数据更新为{00 01 02 03h}

APDU = {FF D6 00 04 04 00 01 02 03h}

附录A.2.5. 值块操作（Value block operation）（Increment, Decrement, Store）

此命令用于处理基于数值的交易（例如：增加值块的值等）。

Value Block Operation 的 APDU 结构（10 字节）

命令	CLA	INS	P1	P2	Lc	命令数据域	
Value Block Operation	FFh	D7h	00h	块号	05h	VB_OP	VB_Value（4 字节）{MSB ..LSB}

其中：

块号 1 个字节。待操作的值块。

VB_OP 1 个字节

00h = 将 VB_Value 存入该块，然后该块变为值块。

01h = 使值块的值增加 VB_Value，仅用于对值块的操作。

02h = 使值块的值减少 VB_Value，仅用于对值块的操作。

VB_Value 4 个字节。该数据的值是一个有符号长整数（4 个字节），用于数值操作。

例 1：Decimal - 4 = {FFh, FFh, FFh, FCh}

VB_Value			
MSB			LSB
FFh	FFh	FF	FCh

例 2：Decimal 1 = {00h, 00h, 00h, 01h}

VB_Value			
MSB			LSB
00h	00h	00h	01h

Value Block Operation 的响应结构（2 字节）

响应	响应数据域	
结果	SW1	SW2

Value Block Operation 的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

附录A.2.6. 读值块（Read Value Block）

此命令用于返回值块中的数值，仅适用于对值块的操作。

Read Value Block 的 APDU 结构（5 个字节）

命令	CLA	INS	P1	P2	Le
Read Value Block	FFh	B1h	00h	块号	04h

其中：

块号 1 字节。待读取的值块。

Read Value Block 的响应结构（4 + 2 字节）

响应	响应数据域		
结果	值 {MSB ..LSB}	SW1	SW2

其中：

值 4 字节。卡片返回的值。是一个有符号长整数（4 字节）。

例 1：Decimal - 4 = {FFh, FFh, FFh, FCh}

值			
MSB			LSB
FFh	FFh	FFh	FC

例 2：Decimal 1 = {00h, 00h, 00h, 01h}

值			
MSB			LSB
00h	00h	00h	01h

Read Value Block 命令的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

附录A.2.7. 复制值块（Copy Value Block）

此命令用于将一个值块中的数值复制到另外一个值块。

Copy Value Block 的 APDU 结构（7 字节）

命令	CLA	INS	P1	P2	Lc	命令数据域	
Copy Value Block Operation	FFh	D7h	00h	源块号	02h	03h	目标块号

其中：

源块号 1 字节。源值块中的值会被复制到目标值块。

目标块号 1 字节。待恢复的值块。源值块和目标值块必须位于同一个扇区。

Copy Value Block 的响应结构（2 字节）

响应	响应数据域	
结果	SW1	SW2

Copy Value Block 命令的响应状态码

结果	SW1 SW2	含义
成功	90 00h	操作成功完成。
错误	63 00h	操作失败

例 1：将数值“1”存入块 05h

APDU = {FF D7 00 05 05 00 00 00 00 01h}

例 2：读取值块 05h

APDU = {FF B1 00 05 00h}

例 3：将值块 05h 的值复制到值块 06h

APDU = {FF D7 00 05 02 03 06h}

例 4：使值块 05h 的值增加“5”

APDU = {FF D7 00 05 05 01 00 00 00 05h}

应答：90 00h [$9000]

附录A.3. 热敏打印机的 API

本节介绍一些仅用于 ACR890 阶段 1 设备并与热敏打印机相关的命令。

附录A.3.1. 重启打印机

此函数用于清除接收缓冲区和打印缓冲区中存储的全部数据。同时还会重启打印机，并将所有用户设置恢复为默认值。

```
void printer_reset(void)
```

参数

无。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_RESET_ERR*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.3.2. 设置标准模式行距

此函数用于设置标准模式下的行距。

```
int printer_setLineSpaceSM(unsigned char nr_step)
```

参数

nr_len [in] 送纸长度，范围从 0 至 255，距离等于 nr_len * 0.125（单位为毫米）

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_SETLINE_SPACE*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.3.3. 在标准模式下打印字符串

此函数在标准模式下打印字符串。打印数据的长度不得超过 65535 个字节。可以使用控制字符'\n'。

```
int printer_printStrSM(const char *str)
```

参数

str [in] 待打印的以空字符结尾的字符串。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_STRPRINT_SM*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.3.4. 在页面模式下打印字符串

此函数用于在页面模式下打印字符串。打印数据的长度不得超过 490 个字节。如果数据长度超过 490 个字节，超出的数据会被弃用。另外可以使用控制字符'\n'。

```
int printer_printStrPM(const struct print_page_mode *param,
const char *data, unsigned short size)
```

参数

```
typedef struct print_page_mode {
   unsigned short HorizontalOrigin_X;
   unsigned short VerticalOrigin_Y;
   unsigned short PrintWidth_X;
   unsigned short PrintHeight_Y;
   unsigned short ucLineSpace;
}PRT_PAGE_MODE_PARAM;
```

用于在页面模式下设置打印区域。

数据成员	值（包含）	说明
HorizontalOrigin_X	0 至 383	X 轴起点
VerticalOrigin_Y	0 至 882	Y 轴起点
PrintWidth_X	1 至 384	打印区域的宽度
PrintHeight_Y	1 至 883	打印区域的高度
ucLineSpace	24 至 255	行距

注：

- *HorizontalOrigin_X + PrintMidth_X 应小于等于 384*
- *VerticalOrigin_Y + PrintHeight_Y 应小于等于 883*
- *水平方向起点与绝对原点之间的距离为 HorizontalOrigin_X*0.125 毫米*
- *垂直方向起点与绝对原点之间的距离为 VerticalOrigin_Y*0.125 毫米*
- *实际打印宽度 = PrintWidth_X*0.125 毫米*
- *实际打印高度 = PrintHeight_Y*0.125 毫米*
- *实际打印行距 = ucLineSpace*0.125 毫米*
- *绝对原点指可打印区域的左上角，打印宽度和打印高度都不能设置为 0*
- *行距包含字体的高度*

param [in]　　待打印的区域。

data [in]　　指向待打印的字符数组的指针。

size [in]　　待打印的字符数组的大小（范围从 1 至 490 字节）。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_STRPRINT_PM*。

要求

头文件　　在 in acs_api.h 中声明

库文件　　使用 libacs_api.so

附录A.3.5.　　在标准模式下打印数据数组

此函数用于在标准模式下打印字符数组，可以使用控制字符'\n'。

```
int printer_printDataSM(const unsigned char *data, unsigned short size)
```

参数

data [in]　　指向待打印的字符数组的指针。

size [in]　　待打印的字符数组的大小，以字节为单位。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_DATAPRINT_PM*。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.3.6. 在页面模式下打印数据数组

此函数用于在“页面模式”下打印数据数组，打印数据的长度不得超过 490 个字节。另外可以使用控制字符‘\n’。

```
int printer_printDataPM(const struct print_page_mode *param,
const unsigned char *data, unsigned short size);
```

参数

param [in] 打印区域。

data [in] 指向待打印的字符数组的指针。

size [in] 待打印的字符数组的大小（范围从 1 至 490 字节）

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_DATAPRINT_PM*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.3.7. 打印图像

此函数用于打印图像。每个字节代表在水平方向上打印的八个点。图像数据按照从左至右和从上至下的方式按字节在纸上打印出来。

```
int printer_print_img(const unsigned char *bitmap, unsigned short width,
unsigned short height, unsigned char mode);
```

参数

bitmap [in] 要打印的图像的数据指针。

width [in] 图像的宽度

height [in] 图像的高度

mode [in] 图像打印模式如果选择单一模式并且 Width 参数介于 1-192（含）之间，则输入“FALSE”。如果选择双模式并且 Width 参数介于 1-384（含）之间，则输入“TRUE”。

返回值

如果成功，返回值为 *SUCCESS*。

如果失败，返回值为 *ETHP_IMAGEPRINT*。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4. 接触式接口（ICC）的 API

本节介绍仅适用于 ACR890 阶段 1 设备并与接触式接口相关的一些命令。

附录A.4.1. 开启接触式接口模块

此函数用于开启 ICC 模块。

```
int icc_open(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-ENODEV。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4.2. 关闭接触式接口模块

此函数用于关闭 ICC 模块。

```
int icc_close(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-ENODEV。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4.3. 查看是否插入接触式卡

此函数用于查看指定的接触式卡槽的状态。

```
int icc_slot_check(enum icc_slot idx)
```

参数

```
enum icc_slot {
ICC_SLOT_ID_0,
SAM_SLOT_ID_1,
SAM_SLOT_ID_2,
ICC_SLOT_MAX
};
```

idx [in] 指定卡槽的索引（IFD）。

返回值

如果成功，返回值 = 0（插入卡片）。

如果失败，返回值 ≠ 0（没有插入卡片）。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4.4. 接触式智能卡上电

此函数用于打开接触式智能卡。

```
int icc_power_on(enum icc_slot ifd, unsigned char *atr, unsigned int
*atrLen)
```

参数

idx [in] 指定卡槽的索引（IFD）。

atr [out] 返回的 ATR 数据的缓冲区。

patrLen [out] 返回的 ATR 长度。

返回值

如果成功，返回值为 0。

如果失败，返回值 ≠ 0。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4.5. 接触式智能卡下电

此函数用于关闭接触式智能卡。

```
int icc_power_off(enum icc_slot idx)
```

参数

idx [in] 指定卡槽的索引 ID（IFD）。

返回值

如果成功，返回值为 0。

如果失败，返回值 ≠ 0。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4.6. 向接触式智能卡发送 PPS

此命令用于发送 PPS 请求给接触式智能卡。

```
int icc_pps_set(enum icc_slot idx, unsigned char fidi)
```

参数

idx [in] 指定卡槽的索引 ID（IFD）

fidi [in] *fi* 和 *di* 值。

返回值

如果成功，返回值为 0。

如果失败，返回值 ≠ 0。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.4.7. 接触式智能卡 APDU 传输

此函数用于发送 APDU 命令给接触式智能卡。

```
int icc_apdu_transmit(enum icc_slot idx, unsigned char *cmd,
unsigned long cmdLen, unsigned char *resp, unsigned long *respLen)
```

参数

idx [in]	指定卡槽的索引 ID（IFD）。
cmd [in]	待发送的 APDU 命令的缓冲区。
cmdLen [in]	待发送的 APDU 命令的长度。
resp [out]	指向响应数据的指针。
respLen [out]	响应数据的长度。

返回值

如果成功，返回值为 0。

如果失败，返回值 ≠ 0。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(int argc,char *argv[])
{
   int ret = -EINVAL;
   enum icc_slot idx = ICC_SLOT_ID_0;
   unsigned int i = 0;
   unsigned char atr[33];
   unsigned long mlen = 0;
   unsigned char mfidi = 0x95;
   unsigned char txcmd[5] = {0x80,0x84,0x00,0x00,0x08};
   unsigned char rxcmd[256];
   unsigned long rxlen = 0;

   ret = icc_open();
   if(0 == ret)
   {
      ret = icc_slot_check(idx);
      if(0 == ret)
      {
         printf("Found card in slot %d!\n", idx);
         ret = icc_power_on(idx, atr, &mlen);
         if(0 == ret)
         {
            printf("ATR ");
            for(i = 0; i < mlen; i++)
            {
               printf("%02X ",atr[i]);
```

```
                }
                printf("\n");

            ret =icc_pps_set(idx, mfidi);
            if(0 == ret)
            {
                printf("Set PPS succeed!\n");
            }

            ret  =  icc_apdu_transmit(idx,  txcmd,  sizeof(txcmd),  rxcmd,
        &rxlen);
            if(0 == ret)
            {
                printf("RES ");
                for(I = 0; I < rxlen; i++)
                {
                   printf("%02X ",rxcmd[i]);
                }
                printf("\n");
            }
            ret = icc_power_off(idx);
        }
        else
        {
            printf(“Power on card %d failed!\n”, idx);
        }
    }
    else
    {
        printf("No card in slot %d!\n", idx);
    }
}
icc_close();

return ret;
}
```

附录A. 5 非接触接口（PICC）的 API

本节介绍仅适用于 ACR890 阶段 1 设备并与非接触卡相关的命令。

附录A.5.1. 开启非接触接口模块

此函数用于开启 PICC 模块。

```
int picc_open(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.5.2. 关闭非接触式接口模块

此函数用于关闭 PICC 模块。

```
int picc_close(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.5.3. 读取非接触式卡

此函数用于轮询及返回卡片状态。

```
int picc_poll_card(struct picc_card *card)
```

参数

```
enum picc_card_type {
PICC_TYPE_UNKNOWN = 0,
PICC_TYPE_A = 0x01,
PICC_TYPE_B = 0x02,
PICC_TYPE_FELICA212 = 0x04,
PICC_TYPE_FELICA424 = 0x08,
PICC_TYPE_END
   };
struct picc_card {
enum picc_card_type type;/* Card's type */
unsigned char uid[16];/* Card's UID */
unsigned char uidlength;/* Length of the card UID */
};
```

card [out] 指向返回的数据的指针，指明返回的卡片类型以及 UID。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

***注：**此函数不适用于 FeliCa 卡。如需获取 FeliCa 卡片信息，可使用 FeliCa 轮询命令（参考**开启I关闭非接触天线**中例 2 内容）。*

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.5.4. 激活非接触式卡

此函数用于激活非接触卡并获取 ATR。

```
int picc_power_on(unsigned char *atr, unsigned char *atr_len)
```

参数

atr [out] 返回非接触卡的 ATR 字符串。

atrLen [out] 返回的 ATR 的长度。

注：

- ATR 的最大长度是 32 个字节。因此 ATR 字符串容器的存储大小必须等于 32 字节。
- 此函数不适用于 FeliCa 卡。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明。

库文件 使用 libacs_api.so。

附录A.5.5. 取消激活非接触式卡

此函数用于取消激活非接触卡。

```
int picc_power_off(void)
```

参数

无。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.5.6. 传输非接触卡数据

此函数用于传输 APDU 命令。

```
int picc_transmit(unsigned char *cmd, unsigned long cmdLen,
unsigned char *resp, unsigned long *respLen)
```

参数

cmd [in] 待发送的 APDU 命令

cmdLen [in] 待发送的 APDU 命令的长度。

resp [out] 指向响应数据的指针

respLen [out] 响应数据的长度

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

注：*响应缓冲的最大长度为 272 字节。因此*resp *字符串容器的存储容量必须大于 272 字节。*

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.5.7. 传输 FeliCa 卡数据

此函数用于传输 FeliCa 命令，仅用于 FeliCa 卡。

注：*只有固件版本 v1.1.0 及更高版本支持该函数。*

```
int picc_felica_transmit(unsigned char *cmd, unsigned long cmdLen,
unsigned char *resp, unsigned long *respLen)
```

参数

cmd [in] 待发送的 FeliCa 命令

cmdLen [in] 待发送的 FeliCa 命令的长度

resp [out] 指向响应数据的指针

respLen [out] 响应数据的长度

返回值

如果函数执行成功，返回值为 0。

如果函数执行失败，返回值为-1。

注：*此函数仅适用于 FeliCa 卡。响应缓冲的最大长度为 272 字节。因此，*resp *字符串容器的存储容量必须大于 272 字节（参考**开启/关闭非接触天线**中例 2 的内容。）*

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

附录A.5.8. 开启/关闭非接触天线

此函数用于开启/关闭磁场（13.56 MHz）。

```
int picc_field_ctrl(enum field_ctrl mode)
```

参数

mode [in] 开启/关闭模式

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

1. 用于 TypeA 和 TypeB

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <sys/ioctl.h>
#include "acs_api.h"

int main(int argc, char* argv[])
{
   int rv = 0, i = 0;
   unsigned char txbuf[5] = {0x00,0x84,0x00,0x00,0x08};
   unsigned int txlen = 5;
   unsigned char rxbuf[272];
   unsigned int rxlen = 0;
   unsigned char buf[36];
   unsigned int len = 0;
   struct picc_card info;

   picc_open();
   picc_field_ctrl(1);

   rv = picc_poll_card(&info);
   if(!rv)
   {
      printf("Card uid[%d]::",info.uidlength);
      for(i=0;i<info.uidlength;i++)
         printf("%02x ",info.uid[i]);
      printf("\n");
   }
   else
```

```
    {
        printf("poll card fail rv = %d\n",rv);
        picc_field_ctrl(0);
        picc_close();
        return 0;
    }

    len = 36;
    rv = picc_power_on(buf,&len);
    if(!rv)
    {
        printf("Card ATR[%d]::",len);
        for(i=0;i<len;i++)
            printf("%02x ",buf[i]);
        printf("\n");
    }
    else
    {
        printf("power on fail rv = %d\n",rv);
        picc_field_ctrl(0);
        picc_close();
        return 0;
    }

    rxlen = 272;
    rv = picc_transmit(txbuf,txlen,rxbuf,&rxlen);
    if(!rv)
    {
        printf("Command:");
        for(i=0;i<5;i++)
            printf("%02x ",tx1buf[i]);
        printf("\n");

        printf("Response:");
        for(i=0;i<rxlen;i++)
            printf("%02x ",rxbuf[i]);
        printf("\n\n");
    }
    else
    {
        printf("picc_transmit fail, return code = %d\n",rv);
    }

    picc_power_off();
    picc_field_ctrl(0);
    picc_close();

    return 0;
}
```

2. 用于 FeliCa

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <sys/ioctl.h>
#include "acs_api.h"

int main(int argc, char* argv[])
{
   int rv = 0, i = 0;

   unsigned char polling[6] = {0x06,0x00,0xff,0xff,0x00,0x00};
   unsigned char reqservice[17] =
{0x11,0x02,0x01,0x01,0x06,0x01,0x5F,0x03,0x34,0x03,0x03,0x48,0x39,0x8
8,0x39,0xC9,0x39};
   unsigned char rxbuf[272];
   unsigned long rxlen = 0;
   unsigned long txlen = 0;
   unsigned char uid[8];
   unsigned char uid_len = 0;

   picc_open();
   picc_field_ctrl(1);


   printf( "Send Polling\n");
   txlen = 6;
   rxlen = 272;
   rv = picc_felica_transmit(polling,txlen,rxbuf,&rxlen);
   if(!rv)
   {
      printf("Command:");
      for(i=0;i<txlen;i++)
         printf("%02x ",polling[i]);
      printf("\n");

      printf("Response:");
      for(i=0;i<rxlen;i++)
         printf("%02x ",rxbuf[i]);
      printf("\n\n");

      if(rxlen > 2){
         memcpy(uid,&rxbuf[2],8);
         uid_len = 8;
      }
   }
   else
   {
      printf("picc_felica_transmit fail, return code = %d\n",rv);
   }

   if(uid_len == 8)
   {
      printf( "Send Request Service\n");
      memcpy( &reqservice[2], uid, 8);
```

```
            txlen = 17;
            rxlen = 272;
            rv = picc_felica_transmit(reqservice,txlen,rxbuf,&rxlen);
            if(!rv)
            {
               printf("Command:");
               for(i=0;i<txlen;i++)
                  printf("%02x ",reqservice[i]);
               printf("\n");

               printf("Response:");
               for(i=0;i<rxlen;i++)
                  printf("%02x ",rxbuf[i]);
               printf("\n\n");
            }
            else
            {
               printf("picc_felica_transmit fail, return code = %d\n",rv);
            }
         }

         picc_field_ctrl(0);
         picc_close();

         return 0;
}
```

附录A.6. INI 文件解析器的 API

本节介绍用于解析初始化文件的 API 函数。

附录A.6.1. 获取 INI 关键字值

此函数用于从.ini 文件（/etc/config.ini）中获取关键字值。

```
Int get_a_ini_key_value(const char * module_name, const char * key_name,
char *key_value)
```

参数

module_name [in] .ini 文件的节名。

key_name [in] .ini 文件中的关键字名。

key_value [out] 缓冲区指针，该缓冲区保存返回的关键字的字符串值。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;
   char key_value[255];

   ret = get_a_ini_key_value (“lcd”, “brightness”, key_value); //call
api to get brightness value
   if(0 == ret)
   {//show out the brighness you get from ini file just now.
      printf("[lcd]\nbrightness=%s\n”, key_value);
   }

   return ret;
}
```

附录A.6.2. 设置 INI 关键字值

此函数用于设置.ini 文件（/etc/config.ini）中的关键字值。

```
int set_a_ini_key_value(const char * module_name, const char * key_name,
char *key_value)
```

参数

module_name [in] .ini 文件的节名。

key_name [in] .ini 文件中的关键字名。

key_value [in] 要在 ini 文件设置的关键字的字符串值。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;

   ret = set_a_ini_key_value (“lcd”, “brightness”, 3); //call api to set
brightness value to 3 in the ini file.

   return ret;
}
```

附录A.6.3. 添加 INI 关键字值

此函数用于在.ini 文件（/etc/config.ini）中添加关键字。

```
int add_a_ini_key_value(const char * module_name, const char * key_name,
char *key_value)
```

参数

module_name [in] 要在.ini 文件中添加的节名。

key_name [in] 要在.ini 文件中添加的关键字名。

key_value [in] 要添加到.ini 文件的关键字的字符串值。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(void)
{
   int ret;

   ret = add_a_ini_key_value (“lcd”, “brightness”, 3);

   return ret;
}
```

附录A.6.4. 根据/etc/config.ini 中的所有关键字设置硬件值

此函数用于根据.ini 文件（/etc/config.ini）中的全部关键字值设置所有硬件。

void ini_init_hw_all(void);

参数

无。

返回值

无。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(void)
{
   ini_init_hw_all();

   return 0;
}
```

附录A.6.5. 根据指定关键字设置硬件值

此函数用于根据.ini 文件（/etc/config.ini）中的指定关键值设置硬件。

```
int record_set_to_hw (const char * module_name, const char * key_name,
const char *key_value)
```

参数

module_name [in] .ini 文件的节名。

key_name [in] .ini 文件中的小节的关键字名。

key_value [in] 要在硬件中设置的值。

返回值

如果成功，返回值为 0。

如果失败，返回值为-1。

要求

头文件 在 in acs_api.h 中声明

库文件 使用 libacs_api.so

示例代码

```
int main(void)
{
    int ret;

   ret = record_set_to_hw (“lcd”, “brightness”, 3); //call api to set
brightness value to 3

    return ret;
}
```