Advanced Card Systems Ltd.
Card & Reader Technologies

ACR890
All-in-One Mobile Smart Card Terminal

Reference Manual V2.00

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

Table of Contents

List of Tables

1.0. Introduction

ACR890 is the next generation, high-performance mobile smart card terminal that combines smart card, magnetic stripe and contactless technologies. With its high-resolution touch screen, it is suitable for customers who want to experience the most interactive interface and features available in the market. This state-of-art product offers faster processing speed, large memory and portability.

This reference manual describes the API (Application Programming Interface) functions developed specifically for the ACR890 terminal. Software developers can make use of these APIs to develop their own smart-card related applications.

2.0. Features

- 32-bit A8 Processor running Embedded Linux®
- 4 GB Flash and 256 MB DDR RAM
- Expandable Micro SD Card support with memory of 1 GB up to 16 GB
- Connectivity Support:
  - Wi-Fi
  - GPRS/GSM quad band (850 MHz, 900 MHz, 1800 MHz, 1900 MHz)
  - 3G connectivity support (900 MHz/2100 MHz or 850 MHz/1900 MHz)
  - USB Client High Speed (Micro-B Type Connector)
  - Serial RS-232 (Mini-B Type Connector)
- Contact Interface:
  - One Full-sized Contact Card Slot (Landing Connector)
- Contactless Interface:
  - Integrated Contactless Smart Card Interface
- Magnetic Stripe Card Support
- SAM Interface:
  - Two SAM Slots (Contact Connector)
- SIM Interface:
  - One Standard SIM Card Slot (GPRS function)
- Barcode Scanner (optional)
- Firmware Upgradeability
- Built-in Peripherals:
  - Easy-to-Read, High Resolution Colored LCD
  - 3.5-inch Resistive Touch Screen LCD
  - Highly Durable Chemical Resistant 20-button Keypad
  - Thermal Printer
  - Real-time Clock (RTC) with independent backup battery
  - 4 LED Status Indicators
  - Built-in Speaker
- Compliant with the following standards:
  - ISO 7816
  - ISO 14443
  - ISO 7811
  - USB Full Speed
  - RoHS 2

3.0. Comparison of ACR890 Phase 1 and Phase 2 commands

For comparison of some commands of the ACR890 Phase 1 and Phase 2 devices, please see the table below.

Note: *This is not a list of all available commands. If the command is not listed in the table, the command is both applicable to ACR890 Phase 1 and Phase 2 devices.*

Commands	ACR890 Phase 1	ACR890 Phase 2
GPRS Module		
Turn on GPRS module	`int gprs_power_on(void)`	`int gprs_power_on(unsigned char timeout)`
Printer Module		
Reset printer	APPLICABLE	NOT APPLICABLE
Feed paper to printer	`int printer_page_feed(unsigned char nr_len)`	`int printer_page_feed(char cControl, char cLine)`
Prepare printer for printing data	NOT APPLICABLE	APPLICABLE
Set line space in standard mode	APPLICABLE	NOT APPLICABLE
Start printing data	NOT APPLICABLE	APPLICABLE
Print bmp file	NOT APPLICABLE	APPLICABLE
Print string in page mode	APPLICABLE	NOT APPLICABLE
Print string in standard mode	`int printer_printStrSM(const char *str)`	`int printer_printStrSM(const char *pString, const int nLen, unsigned int Mode)`
Print data array in standard mode	APPLICABLE	NOT APPLICABLE
Print data array in page mode	APPLICABLE	NOT APPLICABLE
Print an image	`int printer_print_img(const unsigned char *bitmap, unsigned short width, unsigned short height, unsigned char mode);`	NOT APPLICABLE
Smart Card Operations		
Check if contact card is present	APPLICABLE	**Modified to PCSC API for smart card operations**
Turn on contact smart card	APPLICABLE	**Modified to PCSC API for smart card operations**
Turn off contact smart card	APPLICABLE	**Modified to PCSC API for smart card operations**
Send PPS to contact smart card	APPLICABLE	**Modified to PCSC API for smart card operations**
Send APDU command to contact smart card	APPLICABLE	**Modified to PCSC API for smart card operations**

Commands	ACR890 Phase 1	ACR890 Phase 2
Open contactless reader module	APPLICABLE	**Modified to PCSC API for smart card operations**
Close contactless reader module	APPLICABLE	**Modified to PCSC API for smart card operations**
Read contactless card	APPLICABLE	**Modified to PCSC API for smart card operations**
Activate contactless card	APPLICABLE	**Modified to PCSC API for smart card operations**
Deactivate contactless card	APPLICABLE	**Modified to PCSC API for smart card operations**
Transfer contactless card data	APPLICABLE	**Modified to PCSC API for smart card operations**
Transfer FeliCa card data	APPLICABLE	**Modified to PCSC API for smart card operations**
Turn on/off contactless antenna	APPLICABLE	**Modified to PCSC API for smart card operations**
Power Management Module		
Get current system sleep mode	NOT APPLICABLE	APPLICABLE
Enable or disable system auto sleep	APPLICABLE	NOT APPLICABLE

Table 1: Comparison of ACR890 Phase 1 and Phase 2 Commands

For commands that are only applicable to ACR890 Phase 1 devices, see **ACR890 Phase 1 Commands**.

3.1. Identify your ACR890 device

You can identify the phase release of your ACR890 device by looking at the product label at the back. Use the table below to help you determine which ACR890 phase you have.

ACR890	Serial Number	Firmware Version
Phase 1	Starts with RR312	1XXX
Phase 2	Starts with RR400	2XXX

Table 2: ACR890 Phase 1 and Phase 2 Property Comparison

4.0. File and Directory Structure

File Name	Functional System	Description
acs_api.h	Host	API header
acs_errno.h	Host	API returned error number defines
libacs_api.so	Target	API shared library

5.0. Keypad APIs

This section describes the API functions in configuring the keypad of the device.

5.1. Open keypad file descriptor

This function opens a keypad file descriptor.

```
int kpd_open()
```

Parameters

None.

Return Values

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

5.2. Close keypad file descriptor

This function closes a keypad file descriptor.

```
int kpd_close()
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

5.3. Get current keypad state

This function returns the pressing state and the key code value whenever a key is pressed.

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

Parameters

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

keycode [out] A key code of the key pressed.

timeout [in] The waiting time to get the valid key code of the pressed key (in ms).

Return Value

If successful, the return value is 0.

If failed for timeout, the return value is -2.

Otherwise, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

5.4. Set power button working mode

This function is used to set the power button working mode.

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

Parameters

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

nMode [in] The mode value to input.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

5.5. Get power button working mode

This function obtains the current power button working mode.

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

Parameters

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

pMode [out] A pointer to store the mode value.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Backlight Control APIs

This section describes the API functions in configuring the backlight of the device.

6.1. Get current backlight level

This function returns the current state of the backlight.

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

Parameters

```
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] A pointer of the returned backlight state.

Return Value

If successful, the return value is 0.

If failed, the return value is -1 or -2.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Set backlight level

This function sets the brightness level of the backlight.

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

Parameters

```
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] The specified level of backlight brightness.

Return Value

If successful, the return value is 0.

If failed, the return value is -1 or -2.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Battery and Charger APIs

This section describes the API functions in configuring the battery and charger of the device.

7.1. Get battery and charger state

This function returns the current battery state. If the power management IC is out of work, the battery is not detected.

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

Parameters

```
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] The returned battery state information.

Return Value

If successful, the return value is 0.

If failed, the return value is less than 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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 Control APIs

This section describes the API functions in configuring the LEDs of the device.

8.1. Set LED state

This function sets the individual LED state to ON, OFF or blinking state.

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

Parameters

```
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] The ID number of the specified LED.

stat [in] The state of the specified LED.

Return value

If successful, the return value is 0.

If failed, the return value is -1 or -2.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

8.2. Get LED's blinking status

This function returns the current state of the specified LED.

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

Parameters

led [in] An ID number of an LED.

stat [out] A pointer of the returned LED state.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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 Module Power Management APIs

This section describes the API functions in configuring the GPRS module of the device.

9.1. Turn on GPRS module

This function turns on the GPRS module.

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

Note: *If you are using an **ACR890 Phase 1** device, the format for this command is:*

```
int gprs_power_on(void)
```

Parameter

timeout [in] The Waiting time to detect */dev/ttyUSB2*. If it has timed out but */dev/ttyUSB2* does not exist, it will return *EGPRS_POWER_ON_TIMEOUT*. If /dev/ttyUSB2 is found, it will return *EGPRS_SUCCEEDED*.

If the value of timeout is equal to 0 and cannot detect */dev/ttyUSB2*, it will return immediately.

If the value of timeout is not equal to 0, the timeout can be set to ≥5 and timeout ≤ 9. The API will detect */dev/ttyUSB2*.

Return Value

If successful, the return value is *EGPRS_SUCCEEDED*.

If failed, the return value is *ENODEV or EGPRS_POWER_ON_FAILED*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

9.2. Turn off GPRS module

This function turns off the GPRS module.

```
int gprs_power_off(void)
```

Parameters

None.

Return Value

If successful, the return value is *EGPRS_SUCCEEDED*.

If failed, the return value is *ENODEV or EGPRS_POWER_OFF_FAILED*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

9.3. Set pppd connect parameter

This function sets the *pppd* parameters such as telephone, local IP, remote IP, and netmask.

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

Parameters

telephone [in]	A telephone number for dial-up network (e.g., *99***1#).
local_ip [in]	A local IP address if known (dynamic = 0.0.0.0.).
remote_ip [in]	A remote IP address if desired (normally 0.0.0.0).
netmask [in]	A proper netmask if needed.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

9.4. Set pppd dialer parameter

This function sets the dialer parameters, such as protocol and login_point.

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

Parameters

protocol [in]	A protocol for communication (e.g., IP).
login_point [in]	An APN of a mobile network operator support.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

9.5. Start up pppd process

This function starts the *pppd* dial process.

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

Parameters

None.

Return Value

None.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

9.6. Turn off pppd process

This function turns off the *pppd* dial process.

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

Parameters

None.

Return Value

None.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
/* 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. Set Transmit Timeout

This function sets timeout in millisecond for TransmitATCmd API function

```
int SetTransmitTimeoutMs(int tMs).
```

Parameters

tMs [in] Set transmit_timeout to tMs millisecond.

Return Value

none

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

9.8. Transmit one AT command

This function sends one AT command to the 3G module, and then gets a response string code.

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

Parameters

AtCmd [in] An AT command to be sent to a 3G module.

RecvBuffer [out] A buffer for storing response string code of 3G module.

RecvLength [in] The length of *RecvBuffer*.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

9.9. Get IMEI serial number

This function returns the IMEI (International Mobile Equipment Identity) serial number information of the 3G module.

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

Parameters

IMEI_SN [out]	A buffer for storing IMEI serial number information.
IMEILength [in]	The length of *IMEI_SN* buffer.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Example Code

```
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. Audio (ALSA) APIs

This section describes the API functions in configuring the audio settings of the device.

10.1. Get system audio volume

This function returns the system audio volume.

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

Parameters

```
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] A pointer of the returned volume state.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Set system audio volume

This function sets the sound volume level.

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

Parameters

volume [in] The number of volume level to be set (Ranges from 0 to 18. All levels higher than 18 are treated as level 18).

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

10.3. Playback a sound file

This function plays back a wave format sound file.

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

Parameters

file_path [in] A full path of the sound file.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

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

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

   return ret;
}
```

10.4. Control speaker sound

This function turns on/off the speaker sound.

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

Parameters

onoff [in] 1 = Sound on; 0 = Sound off

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

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

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

   return ret;
}
```

11.0. Firmware APIs

This section describes the API function in configuring the firmware of the device.

11.1. Get firmware version

This function returns the firmware version (major, minor, revision) of the device.

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

Parameters

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

};
```

version [out] A data pointer of the firmware version (major, minor, revision).

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Thermal Printer APIs

This section describes the API functions in configuring the thermal printer of the device.

12.1. Open the printer port

This function opens the printer port.

```
int printer_open(void)
```

Parameters

None

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_OPEN_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.2. Close the printer port

This function closes the printer port.

```
int printer_close(void)
```

Parameter

None

Return Values

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_CLOSE_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.3. Get status of the printer

This function returns the printer status.

```
int printer_status_get(void)
```

Parameters

```
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
};
```

Return Value

Any value of *enum printer_state*.

Requirements

Header Declared in acs_api.h.

Library Use libacs_api.so.

12.4. Feed paper to printer

This function feeds the paper to the printer.

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

Note: *If you are using an **ACR890 Phase 1** device, the format for this command is:*

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

Parameters

cControl [in] The code to control the printer wheel to either forward or backward direction.

0 – feed page forward

1 – feed page backward

nr_len [in] The paper space to be fed (Ranges from 0 to 255. The space is equal to *nr_len* x 0.125, in millimeters)

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_FPAPER_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.5. Prepare printer for printing data

This function prepares the printing data to the printer.

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

Parameters

pstr [in] An input *str* data to printer.

nlen [in] The length of input *str* data.

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_DATA_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.6. Start printing data

This function starts printing the last prepared string data.

```
int printer_printing(void)
```

Parameters

None

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_STRPRINT_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.7. Set Printer

This function sets the printing space and font pixel size.

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

Parameters

uiFormat [in] Set the row or column space.

uiFormatValue [in] Set the font pixel size with valid value below:

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

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_STRPRINT_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.8. Print string in standard mode

This function prints a string in standard mode. The printing data size should be less than or equal to 65535 bytes. The control character \n can be used.

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

Parameters

pString [in] A null terminated string of characters to be printed.

nLen [in] The length of input *pString*.

Mode [in] The alignment mode of printing.

09h - Right align

0Ah - Centre align

08h or 0Bh - Left align

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_STRPRINT_SM*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

12.9. Print bitmap file

This function prints a bitmap (.bmp) file.

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

Parameters

pBmpName [in] A full path name of the input bitmap file. The supported color depths of bitmap files are: 1-bit, 8-bit, 24-bit and 32-bit.

uiStartPosition [in] The starting position of printing.

Return Value

If successful, the return value is *SUCCESS*.

If successful, the return value is *ETHP_IMAGEPRINT*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Initialize freetype library

This function initializes the freetype library. This function is designed to print non-English fonts.

Note: *Supported in firmware v1.1.5 and later only.*

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

Parameters

fontname [in] A path to the font file.

Return Values

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_INITFREE_ERR*.

Requirements

Header Declared in lib_freetype.h

Library Use libacs_printer.so

12.11. Set character size

This function sets the character size. This function is designed to print non-English fonts.

Note: *Supported in firmware v1.1.5 and later only.*

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

Parameters

width [in] The character width in integer pixels. Valid range is 11 to 16.

height [in] The character height in integer pixels. Valid range is 11 to 16.

Return Value

If successful, the return value is *SUCCESS*.

If failed the return value is *ETHP_SETSIZE_ERR*.

Requirements

Header Declared in lib_freetype.h

Library Use libacs_printer.so

12.12. Get glyph image and printing

This function gets the glyph image and printing. This function is designed to print non-English fonts.

Note: *Supported in firmware v1.1.5 and later only.*

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

Parameters

string [in] A pointer to the array of characters to be printed.

nlen [in] The size of the array of characters to be printed in bytes.

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_IMAGEPRINT*.

Requirements

Header Declared in lib_freetype.h

Library Use libacs_printer.so

12.13. Release freetype library

This function releases the freetype library. This function is designed to print non-English fonts.

Note: *Supported in firmware v1.1.5 and later only.*

```
void freetype_release(void)
```

Parameters

None.

Return Value

None.

Requirements

Header Declared in lib_freetype.h

Library Use libacs_printer.so

13.0. Wireless LAN Module APIs

This section describes the API functions in configuring the wireless LAN module of the device.

13.1. Turn on wireless LAN module

This function turns on the wireless LAN module.

```
int wifi_pwr_on(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is not equal to -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

13.2. Turn off wireless LAN module

This function turns off the wireless LAN module.

```
int wifi_pwr_off(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

14.0.Bluetooth® Module APIs

This section describes the API functions in configuring the Bluetooth module of the device.

14.1. Turn on Bluetooth module

This function turns on the Bluetooth module.

```
int bluetooth_pwr_on(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

14.2. Turn off bluetooth module

This function turns off the Bluetooth module.

```
int bluetooth_pwr_off(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

15.0.Smart Card Reader APIs

This section describes the API functions for contact and contactless smart card reader module of the device.

15.1. Open the smart card reader module

This function opens the smart card reader module. It needs to have a delay of about two seconds before proceeding with PC/SC commands.

```
int smartcard_open(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is != 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

15.2. Close the smart card reader module

This function closes the smart card reader module of the device.

```
int smartcard_close(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is ≠ 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

15.3. PC/SC API for smart card operations

For the list and description of the PC/SC API functions, please refer to this link:

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

Example Code

```
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. Magnetic Stripes Interface APIs

This section describes the API functions in configuring the magnetic stripe module.

16.1. Get track data from magnetic stripe card

This function reads the track data from the magnetic stripe card within a given period.

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

Parameters

```
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] This contains the track data and state.

time [in] The waiting time for a swiping event (in seconds). Typically a swipe event should be done within range from 5 to 30 seconds.

Bits	Description
Bit 31:27	Reserved
Bit 26	Track #3 data present
Bit 25	Track #2 data present
Bit 24	Track #1 data present
Bit 23:20	Reserved
Bit 19	Track #3 data LRC Error
Bit 18	Track #3 data End Byte Error
Bit 17	Track #3 data Parity Error
Bit 16	Track #3 data Start Byte Error
Bit 15:12	Reserved
Bit 11	Track #2 data LRC Error
Bit 10	Track #2 data End Byte Error
Bit 9	Track #2 data Parity Error
Bit 8	Track #2 data Start Byte Error
Bit 7:4	Reserved
Bit 3	Track #1 data LRC Error
Bit 2	Track #1 data End Byte Error
Bit 1	Track #1 data Parity Error
Bit 0	Track #1 data Start Byte Error

Table 3: Track Data State Bits Table

Return Value

If all tracks are OK, the return value is 0.

Otherwise, the return value is not equal to 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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. Error Code Description APIs

This section describes the API functions for error code description.

17.1. Get error code description

For debugging purposes, use this API to get more details from a given error code.

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

Parameters

errno_code [in] An error number to parse.

Return Value

A parsed string for an error number.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

18.0. Power Management APIs

This section describes the API functions of the system power management of the device.

18.1. Set system sleep timeout

This function sets the device idle time. The system will switch to sleep mode when the idle timer expires. However, any input event will cause the idle timer to be reset.

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

Parameters

seconds[in] Maximum time to enter the sleep mode. Value range is 30 to 1800.

Return Values

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h.

Library Use libacs_api.so.

18.2. Get current system sleep mode

This function gets the device current idle mode.

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

Parameters

mode [out] Store current system sleep mode value.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h.

Library Use libacs_api.so.

18.3. Get system sleep time

This function gets the device idle time.

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

Return Value

The value of the device's idle time.

Requirements

Header Declared in acs_api.h.

Library Use libacs_api.so.

19.0.Barcode Scanner APIs

This section describes the API functions in configuring the barcode scanner module.

Requirements

Header Declared in acs_api.h.

Library Use libacs_api.so.

19.1. Open Scanner Module

```
int scanner_open(void)
```

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_OPEN_ERR (1056).

19.2. Close scanner module

```
int scanner_close(void)
```

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_CLOSE_ERR (1057).

19.3. Connect to scanner

```
int scanner_Connect(void)
```

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_CONNECT_ERR (1058).

19.4. Scan Data

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

Parameters:

[out]- pScanBuf = Buffer for scanned data

[out]- pScanLen = length of scannedData

[in] - timeoutSec = Scan timeout in seconds

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_SCAN_ERR (1059).

19.5. Get Scanner's Reading Session

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

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_READSESSION_PM (1062).

19.6. Get Scanner's Firmware Version

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

Parameters:

[out] - pBuf = Buffer for firmware version data

[out] pLen = length of returned buffer

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_GETVERSION_SM (1061).

19.7. Disconnect the scanner

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

Return Value

If successful, the return value is SUCCESS (0).

If failed, the return value is ESCAN_DISCONNECT (1063).

Appendix A. ACR890 Phase 1 Commands

This section provides some of the commands that are only applicable to the ACR890 Phase 1 device.

Appendix A.1. Pseudo APDU commands for Contactless Interface

This section describes the contactless interface-related commands that are only applicable to the ACR890 Phase 1 device.

Appendix A.1.1. Get data

This command returns the serial number or ATS of the connected PICC.

Get UID APDU Format (5 bytes)

Command	Class	INS	P1	P2	Le
Get Data	FFh	CAh	00h 01h	00h	00h (Full Length)

Get UID Response Format (UID + 2 bytes) if P1 = 00h

Response	Data Out					
Result	UID (LSB)			UID (MSB)	SW1	SW2

Get ATS of an ISO 14443 A card (ATS + 2 bytes) if P1 = 01h

Response	Data Out		
Result	ATS	SW1	SW2

Get Data Response Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Warning	62 82h	End of UID/ATS reached before Le bytes (Le is greater than UID Length)
Error	6C XXh	Wrong length (wrong number Le: 'XX' encodes the exact number) if Le is less than the available UID length
Error	63 00h	The operation has failed
Error	6A 81h	Function not supported

Example 1:

To get the serial number of the connected PICC:

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

Example 2: To get the ATS of the connected ISO 14443 A PICC

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

Appendix A.2. PICC Commands (T=CL Emulation) for MIFARE Classic® 1K/4K Memory Cards

Appendix A.2.1. Load authentication keys

This command loads the authentication keys into the reader. The authentication keys are used to authenticate the particular sector of the MIFARE Classic 1K/4K memory card. Two kinds of locations for authentication keys are provided, volatile and non-volatile.

Load Authentication Keys APDU Format (11 bytes)

Command	Class	INS	P1	P2	Lc	Data In
Load Authentication Keys	FFh	82h	Key Structure	Key Number	06h	Key (6 bytes)

Where:

Key Structure 1 byte

00h = Key is loaded into the reader's volatile memory

Other = Reserved

Key Number 1 byte

00h – 01h = Key Location. The keys will be removed once the reader is disconnected from the PC.

Key 6 bytes. The key value loaded into the reader.

E.g. {FF FF FF FF FF FFh}.

Load Authentication Keys Response Format (2 bytes)

Response	Data Out	
Result	SW1	SW2

Load Authentication Keys Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Example:

Load a key {FF FF FF FF FF FFh} into the key location 00h.

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

Appendix A.2.2. Authenticate MIFARE Classic 1K/4K

This command uses the keys stored in the reader to do authentication with the MIFARE Classic 1K/4K card (PICC). Two types of authentication keys used: TYPE_A and TYPE_B.

Load Authentication Keys APDU Format (6 bytes)

Command	Class	INS	P1	P2	P3	Data In
Authentication	FFh	88h	00h	Block Number	Key Type	Key Number

Load Authentication Keys APDU Format (10 bytes)

Command	Class	INS	P1	P2	Lc	Data In
Authentication	FFh	86h	00h	00h	05h	Authenticate Data Bytes

Authenticate Data Bytes (5 bytes)

Byte1	Byte 2	Byte 3	Byte 4	Byte 5
Version 01h	00h	Block Number	Key Type	Key Number

Where:

Block Number 1 byte. This is the memory block to be authenticated.

Key Type 1 byte

60h = Key is used as a TYPE A key for authentication

61h = Key is used as a TYPE B key for authentication

Key Number 1 byte

00h ~ 01h = Key Location

Note: *For MIFARE Classic 1K Card, it has 16 sectors and each sector consists of 4 consecutive blocks. Ex. Sector 00 consists of Blocks {00h, 01h, 02h and 03h}; Sector 01h consists of Blocks {04h, 05h, 06h and 07h}; the last sector 0Fh consists of Blocks {3Ch, 3Dh, 3Eh and 3Fh}.*

Once the authentication is successful, there is no need to do the authentication again if the blocks to be accessed belong to the same sector. Please refer to the MIFARE Classic 1K/4K specification for more details.

Load Authentication Keys Response Format (2 bytes)

Response	Data Out	
Result	SW1	SW2

Load Authentication Keys Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Sectors (Total 16 sectors. Each sector consists of 4 consecutive blocks)	Data Blocks (3 blocks, 16 bytes per block)	Trailer Block (1 block, 16 bytes)
Sector 0	00h – 02h	03h
Sector 1	04h – 06h	07h
..	..	..
..	..	..
Sector 14	38h – 0Ah	3Bh
Sector 15	3Ch – 3Eh	3Fh

1 KB

Table 4: MIFARE 1K Memory Map

Sectors (Total of 32 sectors. Each sector consists of 4 consecutive blocks)	Data Blocks (3 blocks, 16 bytes per block)	Trailer Block (1 block, 16 bytes)
Sector 0	00h – 02h	03h
Sector 1	04h – 06h	07h
...	...	...
...	...	...
Sector 30	78h – 7Ah	7Bh
Sector 31	7Ch – 7Eh	7Fh

2 KB

Sectors (Total of 8 sectors. Each sector consists of 16 consecutive blocks)	Data Blocks (15 blocks, 16 bytes per block)	Trailer Block (1 block, 16 bytes)
Sector 32	80h – 8Eh	8Fh
Sector 33	90h – 9Eh	9Fh
...	...	...
...	...	...
Sector 38	E0h – EEh	EFh
Sector 39	F0h – FEh	FFh

2 KB

Table 5: MIFARE 4K Memory Map

Example 1:

To authenticate Block 04h with the following characteristics: TYPE A, non-volatile, key number 00h, from PC/SC V2.01 (Obsolete).

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

Example 2:

Similar to the previous example, if we authenticate Block 04h with the following characteristics: TYPE A, non-volatile, key number 00h, from PC/SC V2.07

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

Note: *MIFARE Ultralight® does not need authentication since it provides free access to the user data area.*

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

512 bits or 64 bytes

Table 6: MIFARE Ultralight® Memory Map

Appendix A.2.3. Read binary blocks

This command is used for retrieving multiple data blocks from the PICC. The data block/trailer block must be authenticated first.

Read Binary APDU Format (5 bytes)

Command	Class	INS	P1	P2	Le
Read Binary Blocks	FFh	B0h	00h	Block Number	Number of Bytes to Read

Where:

Block Number 1 byte. The block to be accessed.

Number of Bytes to Read 1 byte. Maximum 16 bytes.

Read Binary Block Response Format (N + 2 bytes)

Response	Data Out		
Result	0 <= N <= 16	SW1	SW2

Read Binary Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Example 1: Read 16 bytes from the binary block 04h (MIFARE 1K or 4K)

APDU = {FF B0 00 04 10}

Example 2: Read 4 bytes from binary Page 04h (MIFARE Ultralight)

APDU = {FF B0 00 04 04}

Example 3: Read 16 bytes from binary Page 04h (MIFARE Ultralight) (Pages 4, 5, 6 and 7 will be read)

APDU = {FF B0 00 04 10}

Appendix A.2.4. Update binary blocks

This command writes multiple data blocks into the PICC. The data block/trailer block must be authenticated first.

Update Binary APDU Format (4 or 16 + 5 bytes)

Command	Class	INS	P1	P2	Lc	Data In
Update Binary Blocks	FFh	D6h	00h	Block Number	Number of Bytes to Update	Block Data 4 Bytes for MIFARE Ultralight or 16 Bytes for MIFARE 1K/4K

Where:

Block Number	1 byte. This is the starting block to be updated.
Number of Bytes to Update	1 byte
	16 bytes for MIFARE 1K/4K
	4 bytes for MIFARE Ultralight
Block Data	4 or 16 bytes
	The data to be written in to binary block/blocks

Update Binary Block Response Codes (2 Bytes)

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Example 1: Update the binary block 04h of MIFARE 1K/4K with Data {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}

Example 2: Update the binary block 04h of MIFARE Ultralight with Data {00 01 02 03h}

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

Appendix A.2.5. Value block operation (Increment, Decrement, Store)

This command handles value-based transactions (e.g., increment a value of the value block, etc.).

Value Block Operation APDU Format (10 bytes)

Command	Class	INS	P1	P2	Lc	Data In	
Value Block Operation	FFh	D7h	00h	Block Number	05h	VB_OP	VB_Value (4 bytes) {MSB .. LSB}

Where:

Block Number 1 byte. The value block to be manipulated.

VB_OP 1 byte

00h = Store the VB_Value into the block. The block will then be converted to a value block.

01h = Increment the value of the value block by the VB_Value. This command is only valid for value block.

02h = Decrement the value of the value block by the VB_Value. This command is only valid for value block.

VB_Value 4 bytes. The value of this data, which is a signed long integer (4 bytes), is used for value manipulation.

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

VB_Value			
MSB			LSB
FFh	FFh	FF	FCh

Example 2: Decimal 1 = {00h, 00h, 00h, 01h}

VB_Value			
MSB			LSB
00h	00h	00h	01h

Value Block Operation Response Format (2 bytes)

Response	Data Out	
Result	SW1	SW2

Value Block Operation Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Appendix A.2.6. Read value block

This command returns the value from the value block. This command is only valid for value blocks.

Read Value Block APDU Format (5 bytes)

Command	Class	INS	P1	P2	Le
Read Value Block	FFh	B1h	00h	Block Number	04h

Where:

Block Number 1 byte. The value block to be accessed.

Read Value Block Response Format (4 + 2 bytes)

Response	Data Out		
Result	Value {MSB .. LSB}	SW1	SW2

Where:

Value 4 bytes. This is the value returned from the card. The value is a signed long integer (4 bytes).

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

Value			
MSB			LSB
FFh	FFh	FFh	FC

Example 2: Decimal 1 = {00h, 00h, 00h, 01h}

Value			
MSB			LSB
00h	00h	00h	01h

Read Value Block Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Appendix A.2.7. Copy value block

This command copies a value from a value block to another value block.

Copy Value Block APDU Format (7 bytes)

Command	Class	INS	P1	P2	Lc	Data In	
Copy Value Block Operation	FFh	D7h	00h	Source Block Number	02h	03h	Target Block Number

Where:

Source Block Number 1 byte. The value of the source value block will be copied to the target value block.

Target Block Number 1 byte. This is the value block to be restored. The source and target value blocks must be in the same sector.

Copy Value Block Response Format (2 bytes)

Response	Data Out	
Result	SW1	SW2

Copy Value Block Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully
Error	63 00h	The operation has failed

Example 1: Store a value "1" into block 05h

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

Example 2: Read the value block 05h

APDU = {FF B1 00 05 00h}

Example 3: Copy the value from value block 05h to value block 06h

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

Example 4: Increment the value block 05h by "5"

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

Answer: 90 00h [$9000]

Appendix A.3. Thermal Printer APIs

This section describes some of the commands for the thermal printer that are only applicable to the ACR890 Phase 1 device.

Appendix A.3.1. Reset the printer

This function clears all data stored in the receive buffer and the print buffer. This function also resets the printer and restores all user settings to default value.

```
void printer_reset(void)
```

Parameter

None.

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_RESET_ERR*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.3.2. Set line space in Standard Mode

This function sets the line space in standard mode.

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

Parameters

nr_len [in] The paper space to be fed, range from 0 to 255, space is equal to nr_len * 0.125 (in millimeters)

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_SETLINE_SPACE*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.3.3. Print string in standard mode

This function prints a string in standard mode. The printing data size should be less than or equal to 65535 bytes. The control character *\n* can be used.

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

Parameters

str [in] A null terminated string of characters to be printed.

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_STRPRINT_SM*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.3.4. Print string in page mode

This function prints a string in the “page mode”. The printing data size should be less than or equal to 490 bytes. If the data size is larger than 490 bytes, the exceeded data will be discarded. The control character ‘\n’ can be used.

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

Parameters

```
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;
```

Use to set the print area under “Page Mode”

Data Member	Value (inclusive)	Description
HorizontalOrigin_X	0 to 383	Starting point in x-axis
VerticalOrigin_Y	0 to 882	Starting point in y-axis
PrintWidth_X	1 to 384	Width of the printing area
PrintHeight_Y	1 to 883	Height of the printing area
ucLineSpace	24 to 255	Line space

Notes:

- *HorizontalOrigin_X + PrintMidth_X should be less than or equal to 384*
- *VerticalOrigin_Y + PrintHeight_Y should be less than or equal to 883*

- *Horizontal physical origin is equal to HorizontalOrigin_X*0.125 mm from the absolute origin*
- *Vertical physical origin is equal to VerticalOrigin_Y*0.125 mm from the absolute origin*
- *The actual width of printing = PrintWidth_X*0.125 mm*
- *The actual height of printing = PrintHeight_Y*0.125 mm*
- *The actual line space of printing = ucLineSpace*0.125 mm*
- *The absolute origin is the upper left of the printable area, and both print width and height cannot be set to 0*
- *The line spacing includes the height of the font*

param [in]	The area to be printed.
data [in]	A pointer to the array of characters to be printed.
size [in]	The size of the array of characters to be printed (range from 1 to 490 bytes).

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_STRPRINT_PM*.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Appendix A.3.5. Print data array in Standard Mode

This function prints an array of characters in the “Standard Mode”. Control character ‘\n’ can be used.

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

Parameters

data [in]	A pointer to the array of characters to be printed.
size [in]	The size of the array of characters to be printed in bytes.

Return Values

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_DATAPRINT_PM*.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Appendix A.3.6. Print data array in Page Mode

This function prints the array of data in the “page mode”. The printing data size should be less than or equal to 490 bytes. The control character ‘\n’ can be used.

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

Parameters

param [in] The printing area.

data [in] A pointer to the array of characters to be printed.

size [in] The size of the array of characters to be printed (range from 1 to 490 in bytes).

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_DATAPRINT_PM*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.3.7. Print an image

This function prints an image. Each byte represents eight points printed in horizontal direction. The image data is printed one byte by one byte, from left to right, and from top to bottom in the paper.

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

Parameters

bitmap [in] A data pointer of Image to be printed

width [in] The width of image

height [in] The height of image

mode [in] An image printing mode. Input “FALSE” if selecting single mode and the width range is between 1 and 192 (inclusive). Input “TRUE” if selecting double mode and the width range is between 1 and 384 (inclusive).

Return Value

If successful, the return value is *SUCCESS*.

If failed, the return value is *ETHP_IMAGEPRINT*.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4. Contact Interface (ICC) APIs

This section describes some contact interface–related commands that are only applicable to the ACR890 Phase 1 device.

Appendix A.4.1. Open the contact interface module

This function opens the ICC module.

```
int icc_open(void)
```

Parameter

None.

Return values

If successful, the return value is 0.

If failed, the return value is **-**ENODEV.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4.2. Close the contact interface module

This function closes the ICC module.

```
int icc_close(void)
```

Parameter

None.

Return values

If successful, the return value is 0.

If failed, the return value is **-**ENODEV.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4.3. Check if contact card is present

This function checks the state of a specified contact card slot.

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

Parameters

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

idx [in] The index of the specified slot (IFD).

Return Value

If successful, the return value is 0 (card is present).

If failed, the return value is ≠ 0 (card is not present).

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4.4. Power on contact smart card

This function turns on the contact smart card.

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

Parameters

idx [in] The index ID of specified slot (IFD).

atr [out] A buffer of the returned ATR data.

patrLen [out] The returned ATR length.

Return Value

If successful, the return value is 0.

If failed, the return value is ≠ 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4.5. Power off contact smart card

This function turns off the contact smart card.

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

Parameters

idx [in] The index ID of specified slot (IFD).

Return Value

If successful, the return value is 0.

If failed, the return value is ≠ 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4.6. Send PPS to contact smart card

This function sends a PPS request to the contact smart card.

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

Parameters

idx [in] The index ID of specified slot(IFD).

fidi [in] The *fi* and *di* value.

Return Value

If successful, the return value is 0.

If failed, the return value is ≠ 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.4.7. Contact smart card APDU transfer

This function sends an APDU command to the contact smart card.

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

Parameters

idx [in]	The index ID of specified slot (IFD).
cmd [in]	A buffer of the APDU command to be sent.
cmdLen [in]	The length of the APDU command to be sent.
resp [out]	A pointer of response data.
respLen [out]	The length of the response data.

Return Value

If successful, the return value is 0.

If failed, the return value is ≠ 0.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

```
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;
}
```

Appendix A.5. Contactless Interface (PICC) APIs

This section describes contactless card–related commands that are only applicable to the ACR890 Phase 1 device.

Appendix A.5.1. Open the contactless interface module

This function opens the PICC module.

```
int picc_open(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.5.2. Close the contactless interface module

This function closes the PICC module.

```
int picc_close(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.5.3. Read a contactless card

This function is used to poll and return the status of the card.

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

Parameters

```
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] A pointer of returned data. This indicates the returned card type and UID.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Note: *This function does not apply to FeliCa cards. You can use the FeliCa polling command to get the card info (see Example 2 in* ***Turn on/off contactless antenna****).*

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.5.4. Activate a contactless card

This function activates the contactless card and gets the ATR.

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

Parameters

atr [out] The returned the ATR string of contactless card.

atrLen [out] The returned ATR length.

Notes:

- *Maximum size of ATR is 32 bytes. Hence, the storage size of ATR string container MUST be equal to 32 bytes.*
- *This function does not apply to FeliCa cards.*

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h.

Library Use libacs_api.so.

Appendix A.5.5. Deactivate a contactless card

This function deactivates the contactless card.

```
int picc_power_off(void)
```

Parameters

None.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.5.6. Transfer contactless card data

This function transmits the APDU command.

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

Parameters

cmd [in]	The APDU command to be sent.
cmdLen [in]	The length of the APDU command to be sent.
resp [out]	The pointer of response data.
respLen [out]	The length of the response data.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Note: *Maximum size of response buffer is 272 bytes. Hence, the storage size of *resp string container MUST be greater than 272 bytes.*

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.5.7. Transfer FeliCa card data

This function transmits FeliCa command and can only be used for FeliCa cards.

Note: *Supported in firmware v1.1.0 and later only.*

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

Parameters

cmd [in] The FeliCa command to be sent.

cmdLen [in] The length of the FeliCa command to be sent.

resp [out] The pointer of response data.

respLen [out] The length of the response data.

Return Value

If the function is successful, the return value is 0.

If the function is failed, the return value is -1.

Note: *This function is for FeliCa use only. Maximum size of response buffer is 272 bytes. Hence, the storage size of *resp string container MUST be greater than 272 bytes (see Example 2 in* ***Turn on/off contactless antenna****).*

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Appendix A.5.8. Turn on/off contactless antenna

This function turns on/off the magnetic field (13.56 MHz).

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

Parameters

mode [in] ON/OFF mode

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

1. For TypeA and 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. For 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;
}
```

Appendix A.6. INI File Parser APIs

This section describes the API functions for parsing initialization file.

Appendix A.6.1. Get INI keyword value

This function obtains the keyword value from an .ini file (/etc/config.ini).

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

Parameters

module_name [in]	The section name of .ini file.
key_name [in]	The keyword name in .ini file.
key_value [out]	The buffer pointer to store returned keyword string value.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Example Code

```
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;
}
```

Appendix A.6.2. Set INI keyword value

This function sets the keyword value to .ini file (/etc/config.ini).

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

Parameters

module_name [in]	The section name of .ini file.
key_name [in]	The keyword name in .ini file.
key_value [in]	The keyword string value to set to .ini file.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Example Code

```
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;
}
```

Appendix A.6.3. Add INI keyword value

This function is to add a keyword in .ini file (/etc/config.ini).

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

Parameters

module_name [in]	The section name to be added in .ini file.
key_name [in]	The keyword name to be added in .ini file.
key_value [in]	The keyword string value to be added to .ini file.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Example Code

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

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

   return ret;
}
```

Appendix A.6.4. Set hardware value according to all keywords in /etc/config.ini

This function sets all hardware according to all keyword value in .ini file (/etc/config.ini).

```
void ini_init_hw_all(void);
```

Parameters

None.

Return Value

None.

Requirements

Header Declared in acs_api.h

Library Use libacs_api.so

Example Code

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

   return 0;
}
```

Appendix A.6.5. Set hardware value according to specified keyword

This function sets the hardware according to the specified key value in .ini file (/etc/config.ini).

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

Parameters

module_name [in]	The section name in .ini file.
key_name [in]	The keyword name of section in .ini file.
key_value [in]	The value to be set in hardware.

Return Value

If successful, the return value is 0.

If failed, the return value is -1.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Example Code

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

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

    return ret;
}
```