Advanced Card Systems Ltd.
Card & Reader Technologies

ACR890
All-in-One Mobile Smart Card Terminal

Reference Manual V1.01

Subject to change without prior notice

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) calls developed specifically for the ACR890 terminal. Application software developers can make use of these APIs to develop their smart-card related applications.

2.0. 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

3.0. Keypad APIs

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

3.1. Open keypad file descriptor

This function is used to open 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

3.2. Close Keypad file descriptor

This function is used to close a keypad file descriptor.

```
int kpd_close()
```

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

3.3. Get current keypad state

This function is used to return 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;
};
```

[out] keycode	Key code of the key pressed.
[in] timeout	Waiting time to get the valid key code of the pressed key (in ms).

Return Values

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

3.4. 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
};
```

[in] nMode	The mode value to Input;

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

3.5. Get Power Button Working Mode

This function is used to obtain the current power button working mode.

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

Parameters

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

[out] pMode	pointer to store the mode value.

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

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;
}
```

4.0. Backlight Control APIs

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

4.1. Get current backlight level

This function is used to retrieve 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
} ;
```

[out] stat The 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;
}
```

4.2. Set backlight level

This function is used to set 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
};
```

[in] level 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;
}
```

5.0. Battery and Charger APIs

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

5.1. Get battery and charger state

This function is used to retrieve 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 [%]
   };
```

[out] stat The returned battery state information.

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(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;
}
```

6.0. LED Control APIs

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

6.1. Set LED state

This function is used to set the individual LED state to either 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
};
```

[in] led The ID number of the specified LED.

[in] stat 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

6.2. Get LED’s blinking status

This function gets a specified LED’s current state.

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

Parameters

[in] led The individual LED’s ID number.

[out] stat The 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;
}
```

7.0. GPRS Module Power Management APIs

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

7.1. Power on GPRS

This function is used to power on the GPRS module.

```
int gprs_power_on(void)
```

Parameter

None.

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

7.2. Power off GPRS

This function is used to power 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

7.3. Set pppd connect parameter

This function is to set ppp 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

[in] telephone	The telephone number for dial-up networking (e.g., *99***1#).
[in] local_ip	Local IP address if known (dynamic = 0.0.0.0.)
[in] remote_ip	Remote IP address if desired (normally 0.0.0.0).
[in] netmask	The 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

7.4. Set pppd dialer parameter

This function is used to set the dialer parameters, such as protocol and login_point.

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

Parameters

[in] protocol	The protocol for communication (e.g., ip).
[in] login_point	The APN of 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

7.5. Start up pppd process

This function is used to start up 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

7.6. Turn off pppd process

This function is used to turn 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;
}
```

7.7. Transmit one AT command

This function is used to send one AT command to 3G module, and get a response string code.

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

Parameters

[in] AtCmd	The AT Command for send to 3G module.
[out] RecvBuffer	The buffer for store responsed string code of 3G module.
[in] RecvLength	The length of RecvBuffer.

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

7.8. Get IMEI Serial Number

This function is to get IMEI Serial Number information of 3G module.

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

Parameters

[out] IMEI_SN	The buffer for store IMEI serial number information
[in] IMEILength	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;
}
```

8.0. Audio (ALSA) APIs

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

8.1. Get system audio volume

This function is used to retrieve 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
};
```

[out] stat The 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;
}
```

8.2. Set system audio volume

This function is used to set sound volume level.

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

Parameters

[in] volume	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

8.3. Sound playback

This function is used to play back a wave format sound file.

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

Parameters

[in] file_path	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;
}
```

8.4. Speaker Sound Control

This function is used to turn the speaker sound on/off.

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

Parameters

[in] onoff	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;
}
```

9.0. Firmware APIs

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

9.1. Get firmware version

This function is used to get 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;

};
```

[out] version	Data pointer of device version(major, minor, revision)

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

10.0.Thermal Printer APIs

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

10.1. Open the printer port

This function is used to open the printer port.

```
    int printer_open(void)
```

Parameters

None.

Return Values

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

10.2. Close the printer port

This function is used to close 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

10.3. Reset the printer

This function is used to clear 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 Values

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

10.4. Feed paper to printer

This function is used to feed the paper to the printer.

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

Parameters

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

Return Values

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

10.5. Set line space in Standard Mode

This function is used to set the line space in Standard Mode.

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

Parameters

[in] nr_len	Paper space to be fed, range from 0 to 255, space is equal to nr_len * 0.125 (in mm)

Return Values

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

10.6. Print string in Standard Mode

This function is used to print 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

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

Return Values

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

10.7. Print string in Page Mode

This function is used to print 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.*

[in] param Printing area to be printed.

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

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

Return Values

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

10.8. Print data array in Standard Mode

This function is used to print 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

[in] data	Pointer to the array of characters to be printed.
[in] size	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

10.9. Print data array in Page Mode

This function is used to print 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

[in] param	Printing area.
[in] data	Pointer to the array of characters to be printed.
[in] size	Size of the array of characters to be printed (range from 1 to 490 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

10.10. Print an image

This function is used to print 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

[in] bitmap	Data pointer of Image to be printed.
[in] width	Width of image.
[in] height	Height of image.
[in] mode	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 Values

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

10.11. Get status of the printer

This function is used to return 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 Values

Any value of enum `printer_state`.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

Example Code

```
int main(int argc, char *argv[])
{
   int nRet=0,i=0,j=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);
   PRT_PAGE_MODE_PARAM param;
   memset(&param,0,sizeof(param));
   //Page mode printing
   param.HorizontalOrigin_X = 50;
   param.VerticalOrigin_Y   = 50;
   param.PrintWidth_X     = 100;
   param.PrintHeight_Y    = 150;
   param.ucLineSpace  = 25;
   nRet = printer_status_get();
   if(nRet !=PRINTER_READY)
   {
      printf("printer_printStrPM nRet = [%x]\n",nRet);
      return -1;
   }
   printer_printStrPM(&param,"Logyi 1233745771234567456813457");
   //Bitmap mode
   char szImage[14] = {0};
   szImage[0] = 0x01;
   szImage[1] = 0x02;
   szImage[2] = 0x04;
   szImage[3] = 0x08;
   szImage[4] = 0x10;
   szImage[5] = 0x20;
   szImage[6] = 0x40;
   szImage[7] = 0x80;
   szImage[8] = 0x40;
   szImage[9] = 0x20;
   szImage[10] = 0x10;
   szImage[11] = 0x08;
   szImage[12] = 0x04;
   szImage[13] = 0x02;
   nRet = printer_status_get();
   if(nRet !=PRINTER_READY)
   {
      printf("ACR890_Printer_PrintImagenRet = [%x]\n",nRet);
      return -1;
   }

   printer_printImage(szImage,14,0,false);
   //Set line spaces
   printer_setLineSpace(50);
   //Printing in standard mode
   printer_printStrSM(szcom);
   nRet = printer_satus_get();
   if(nRet !=PRINTER_READY)
   {
      printf("ACR890_Printer_PrintStrSM nRet = [%x]\n",nRet);
      return -1;
   }
   printer_printStrSM("ABCDEFGFH");
   //printing and change new line
   printer_page_feed(100);
   printer_printStrSM("BBBBBB");
   printer_pintStrSM("CCCCC");
   nRet = pinter_status_get();
   if(nRet !=PRINTER_READY)
   {
      printf("ACR890_Printer_PrintStrSM nRet = [%x]\n",nRet);
      return -1;
   }
//Reset and cache flush
   ACR890_Printer_close();
   i++;
   sleep(1);
}
```

11.0.Wireless LAN Module Control APIs

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

11.1. Power on wireless LAN module

This function turns on the wireless LAN module.

```
int wifi_pwr_on(void)
```

Parameter

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

11.2. Power off wireless LAN module

This function is used turn off the wireless LAN module.

```
int wifi_pwr_off(void)
```

Parameter

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

11.3. Bluetooth Module Control APIs

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

11.4. Power on Bluetooth module

This function is used to turn on the Bluetooth module.

```
int bluetooth_pwr_on(void)
```

Parameter

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

11.5. Power off bluetooth module

This function is used to turn off the bluetooth module.

```
    int bluetooth_pwr_off(void)
```

Parameter

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

12.0.Contact Smart Card Reader APIs

This section describes the API functions for contact smart card reader module.

12.1. Open the contact card reader module

This function is used to open 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

12.2. Close the contact card reader module

This function is used to close 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

12.3. Check if a contact card is present

This function is used to check the state of a specified 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
};
```

[in] idx	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

12.4. Power on a contact smart card

This function is used to turn on the contact smart card.

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

Parameters

[in] idx	The index ID of specified slot(IFD).
[out] atr	Buffer of the returned ATR data.
[out] patrLen	The returned ATR length.

Return values

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

12.5. Power off a contact smart card

This function is used to turn off the contact smart card.

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

Parameters

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

Return values

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

12.6. Send PPS to contact smart card

This function is used to send PPS request to contact smart card.

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

Parameters

[in] idx	The index ID of specified slot(IFD).
[in] fidi	The *fi* and *di* value.

Return Values

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

12.7. Contact smart card APDU transfer

This function is used to send 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

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

Return Values

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;
}
```

13.0.Contactless Reader APIs

This section describes the API functions for contactless card reader. Both MIFARE Classic and MIFARE DESFire cards are supported.

13.1. Open the contactless reader module

This function is used to open the PICC module.

```
int picc_open(void)
```

Parameter

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

13.2. Close the contactless reader module

This function is used to close the PICC module.

```
int picc_close(void)
```

Parameter

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

13.3. 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

[out] card The pointer of returned data, indicates the returned card type, and UID.

```
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 */
};
```

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

13.4. Power on the contactless card

This function is used to power on the contactless card and get the ATR from it.

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

Parameters

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

Note: *Maximum size of ATR is 32 bytes. Hence, the storage size of ATR string container MUST be equal to 32 bytes.*

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.

13.5. Power off the contactless card

This function is used to power off the contactless card.

```
    int picc_power_off(void)
```

Parameter

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

13.6. Contactless card data transfer

This function is used to transmit the APDU command.

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

Parameters

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

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

13.7. Contactless card reader antenna control

This function is used to turn the magnetic field (13.56 MHz) on/off.

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

Parameter

[in] mode	ON/OFF mode.

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

Example Code

```
int main(int argc,char *argv[])
{
   int ret;

   ret = picc_Open();
   if(ret)
   {
      printf("Open PICC fail\n");
   }
   else
   {
      printf("Open PICC succ\n");
   }
   picc_Close();

   return ret;
}
```

14.0.Magnetic Stripes Card APIs

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

14.1. Get track data from a magnetic stripes card

This function is used to read track data from magnetic stripes card in a given time period.

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

Parameters

[out] data Contains track data and state.

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

```
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;
      };
```

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 1: Track Data State Bits Table

Return values

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

Otherwise, 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[])
{
      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;
}
```

15.0. Error code description APIs

This section describes the API functions for contact smart card reader module.

15.1. Get the error description by a given error code

For debug purpose, use this API to get more details by a given error code.

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

Parameters

[in] errno_code	The error number to parse.

Return values

A parsed string for an error number.

Requirements

Header	Declared in acs_api.h
Library	Use libacs_api.so

16.0.INI file parser APIs

This section describes the API functions for parse initialization file.

16.1. Get a ini keyword value

This function is to obtain keyword value from ini file (/etc/config.ini).

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

Parameters

[in] module_name	The section name of ini file.
[in] key_name	The keyword name in ini file.
[out] key_value	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;
}
```

16.2. Set a ini keyword value

This function is to setting 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

[in] module_name	The section name of ini file.
[in] key_name	The keyword name in ini file.
[in] key_value	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;
}
```

16.3. Add a ini keyword

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

[in] module_name	The section name you want to add in ini file.
[in] key_name	The keyword name you want to add in ini file.
[in] key_value	The keyword string value will 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;
}
```

16.4. Sync setting hardware value according to all keyword in /etc/config.ini

This function is to setting 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;
}
```

16.5. Sync setting hardware value according to specified keyword

This function is used to setting hardware according to 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

[in] module_name	The section name in ini file.
[in] key_name	The keyword name of section in ini file.
[in] key_value	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;
}
```

17.0.Power Management APIs

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

17.1. Set system sleep timeout

This function is used to set 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

[in] seconds Maximum expires time to enter sleep mode,value range 30~1800

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.

17.2. Get system sleep time

This function is used to get the device idle time.

```
Unsigned int pm_sleep_timeout_get(void)
```

Return Values

The values of device idle time.

Requirements

Header	Declared in acs_api.h.
Library	Use libacs_api.so.

17.3. Enable or disable system auto sleep

This function is used to get the device idle time.

```
int pm_sleep_enable(unsigned char isEnable)
```

Parameters

[in] `isEnable`

```
isEnable = 1  enable device auto sleep
isEnable = 0  disable device auto sleep
```

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.