



**Advanced Card Systems Ltd.**  
Card & Reader Technologies

# ACR890

## All-in-One Mobile Smart Card Terminal

Reference Manual V1.03





## Table of Contents

<b>1.0.</b>	<b>Introduction .....</b>	<b>5</b>
<b>2.0.</b>	<b>Features .....</b>	<b>6</b>
<b>3.0.</b>	<b>File and Directory Structure .....</b>	<b>7</b>
<b>4.0.</b>	<b>Pseudo APDU commands for Contactless Interface .....</b>	<b>8</b>
4.1.	Get Data.....	8
4.2.	PICC Commands (T=CL Emulation) for MIFARE® Classic 1K/4K Memory Cards .....	9
4.2.1.	Load Authentication Keys .....	9
4.2.2.	Authentication for MIFARE Classic 1K/4K.....	10
4.2.3.	Read Binary Blocks.....	13
4.2.4.	Update Binary Blocks.....	14
4.2.5.	Value Block Operation (Increment, Decrement, Store) .....	15
4.2.6.	Read Value Block.....	16
4.2.7.	Copy Value Block.....	17
<b>5.0.</b>	<b>Keypad APIs .....</b>	<b>18</b>
5.1.	Open keypad file descriptor .....	18
5.2.	Close Keypad file descriptor .....	18
5.3.	Get current keypad state .....	19
5.4.	Set Power Button Working Mode .....	19
5.5.	Get Power Button Working Mode .....	20
<b>6.0.</b>	<b>Backlight Control APIs.....</b>	<b>22</b>
6.1.	Get current backlight level .....	22
6.2.	Set backlight level.....	23
<b>7.0.</b>	<b>Battery and Charger APIs .....</b>	<b>24</b>
7.1.	Get battery and charger state .....	24
<b>8.0.</b>	<b>LED Control APIs.....</b>	<b>25</b>
8.1.	Set LED state.....	25
8.2.	Get LED's blinking status.....	26
<b>9.0.</b>	<b>GPRS Module Power Management APIs.....</b>	<b>27</b>
9.1.	Power on GPRS .....	27
9.2.	Power off GPRS .....	28
9.3.	Set pppd connect parameter .....	29
9.4.	Set pppd dialer parameter .....	30
9.5.	Start up pppd process.....	31
9.6.	Turn off pppd process.....	32
9.7.	Transmit one AT command .....	34
9.8.	Get IMEI Serial Number.....	35
<b>10.0.</b>	<b>Audio (ALSA) APIs .....</b>	<b>36</b>
10.1.	Get system audio volume .....	36
10.2.	Set system audio volume.....	37
10.3.	Sound playback .....	38
10.4.	Speaker Sound Control .....	39
<b>11.0.</b>	<b>Firmware APIs .....</b>	<b>40</b>
11.1.	Get firmware version.....	40
<b>12.0.</b>	<b>Thermal Printer APIs .....</b>	<b>41</b>
12.1.	Open the printer port.....	41
12.2.	Close the printer port .....	42



12.3.	Reset the printer .....	43
12.4.	Feed paper to printer .....	44
12.5.	Set line space in Standard Mode.....	45
12.6.	Print string in Standard Mode .....	46
12.7.	Print string in Page Mode .....	47
12.8.	Print data array in Standard Mode.....	49
12.9.	Print data array in Page Mode .....	50
12.10.	Print an image.....	51
12.11.	Get status of the printer .....	52
12.12.	Initialize freetype library .....	54
12.13.	Set the character size .....	55
12.14.	Get glyph image and printing.....	56
12.15.	Release freetype library.....	57
<b>13.0.</b>	<b>Wireless LAN Module Control APIs.....</b>	<b>58</b>
13.1.	Power on wireless LAN module.....	58
13.2.	Power off wireless LAN module.....	59
<b>14.0.</b>	<b>Contact Smart Card Reader APIs .....</b>	<b>60</b>
14.1.	Open the contact card reader module .....	60
14.2.	Close the contact card reader module.....	61
14.3.	Check if a contact card is present .....	62
14.4.	Power on a contact smart card.....	63
14.5.	Power off a contact smart card.....	64
14.6.	Send PPS to contact smart card .....	65
14.7.	Contact smart card APDU transfer .....	66
<b>15.0.</b>	<b>Contactless Reader APIs .....</b>	<b>68</b>
15.1.	Open the contactless reader module.....	68
15.2.	Close the contactless reader module .....	69
15.3.	Read a contactless card .....	70
15.4.	Activate the contactless card.....	71
15.5.	Deactivate the contactless card.....	72
15.6.	Contactless card data transfer.....	73
15.7.	FeliCa card data transfer .....	74
15.8.	Contactless card reader antenna control.....	75
<b>16.0.</b>	<b>Magnetic Stripes Card APIs.....</b>	<b>79</b>
16.1.	Get track data from a magnetic stripe card .....	79
<b>17.0.</b>	<b>Error code description APIs .....</b>	<b>81</b>
17.1.	Get the error description by a given error code.....	81
<b>18.0.</b>	<b>INI file parser APIs.....</b>	<b>82</b>
18.1.	Get an ini keyword value .....	82
18.2.	Set a ini keyword value.....	83
18.3.	Add a ini keyword .....	84
18.4.	Set hardware value according to all keywords in /etc/config.ini .....	85
18.5.	Set hardware value according to the specified keyword .....	86
<b>19.0.</b>	<b>Power Management APIs.....</b>	<b>87</b>
19.1.	Set system sleep timeout.....	87
19.2.	Get system sleep time .....	87
19.3.	Enable or disable system auto sleep.....	88



## List of Tables

<b>Table 1</b> : MIFARE 1K Memory Map.....	11
<b>Table 2</b> : MIFARE 4K Memory Map.....	11
<b>Table 3</b> : MIFARE Ultralight Memory Map.....	12
<b>Table 4</b> : Track Data State Bits Table.....	79



## 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. Features

- 32-bit A8 Processor running Embedded Linux®
- 512 MB Flash and 512 MB LPDDR Memory
- Expandable Micro SD Card support with memory 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)
- 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. 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

## 4.0. Pseudo APDU commands for Contactless Interface

### 4.1. Get Data

The Get Data command will return 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};
```





## 4.2. PICC Commands (T=CL Emulation) for MIFARE® Classic 1K/4K Memory Cards

### 4.2.1. 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}



#### 4.2.2. Authentication for MIFARE Classic 1K/4K

This command uses the keys stored in the reader to do authentication with the MIFARE 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 done successfully, 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 1: 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 2: 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 3:** MIFARE Ultralight Memory Map



### 4.2.3. Read Binary Blocks

This command retrieves 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}



#### 4.2.4. 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}



#### 4.2.5. Value Block Operation (Increment, Decrement, Store)

This command manipulates 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.



#### 4.2.6. Read Value Block

This command retrieves 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.



### 4.2.7. 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]



## 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 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`



### 5.3. 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;  
};
```

- [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

### 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  
};
```

- [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



## 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  
};
```

**[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;
}
```



## 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 retrieves 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** 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  
};
```

**[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;  
}
```



## 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 retrieves 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;
}
```





## 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 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



## 8.2. Get LED's blinking status

This function gets the current state of the specified LED.

```
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;
}
```



## 9.0. GPRS Module Power Management APIs

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

### 9.1. Power on GPRS

This function turns 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



## 9.2. Power off GPRS

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 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



## 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

- [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



## 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. Transmit one AT command

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

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

### Parameters

- [in] AtCmd**            The AT Command to be sent to 3G module
- [out] RecvBuffer**    The buffer for store response 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 -1.

### Requirements

**Header**        Declared in `acs_api.h`

**Library**      Use `libacs_api.so`



## 9.8. Get IMEI Serial Number

This function gets IMEI (International Mobile Equipment Identity) 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;
}
```



## 10.0.Audio (ALSA) APIs

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

### 10.1. Get system audio volume

This function retrieves 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 %d\nmin volume is %d\n, left  
        volume is %d\nright volume is %d\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

**[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`



### 10.3. Sound playback

This function plays 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 (".\\Niiose.wav");//call api to play a specified audio
    file

    return ret;
}
```



## 10.4. Speaker Sound Control

This function turns 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;
}
```



## 11.0. Firmware APIs

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

### 11.1. Get firmware version

This function gets 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 firmware 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

#### 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 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



## 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. 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 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



## 12.4. Feed paper to printer

This function feeds 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



## 12.5. Set line space in Standard Mode

This function sets 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



## 12.6. 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



## 12.7. 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 + PrintWidth\_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**      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





## 12.8. 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

- [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



## 12.9. 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

- [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



## 12.10. 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

- [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



## 12.11. 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_BFNPP = 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]="ABCDEF1234567890ABCDEFGHIJG12345QWERT";  
  
    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("ABCDEFGGFH");
//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);
}
```



## 12.12. 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

**[in]** fontname      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.



## 12.13. Set the 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

**[in] width**      The character width, in integer pixels. Valid range is 11-16.

**[in] height**     The character height, in integer pixels. Valid range is 11-16.

### Return Values

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.



## 12.14. 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

- [in] string**      Pointer to the array of characters to be printed.  
**[in] nlen**        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\_IMAGEPRINT.

### Requirements

- Header**        Declared in lib\_freetype.h.  
**Library**      Use libacs\_printer.so.





## 12.15. 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 Values

None.

### Requirements

**Header** Declared in lib\_freetype.h.

**Library** Use libacs\_printer.so.



## 13.0. Wireless LAN Module Control APIs

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

### 13.1. 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



## 13.2. Power off wireless LAN module

This function turns 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



## 14.0. Contact Smart Card Reader APIs

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

### 14.1. Open the contact card reader 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



## 14.2. Close the contact card reader 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



### 14.3. Check if a 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  
};
```

**[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  $\neq 0$  (card is not present).

#### Requirements

**Header**     Declared in `acs_api.h`

**Library**    Use `libacs_api.so`



## 14.4. Power on a 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

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

### Return values

If successful, the return value is 0.

If failed, the return value is  $\neq 0$ .

### Requirements

**Header**        Declared in `acs_api.h`

**Library**      Use `libacs_api.so`



## 14.5. Power off a contact smart card

This function turns 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  $\neq 0$ .

### Requirements

**Header** Declared in `acs_api.h`

**Library** Use `libacs_api.so`





## 14.6. 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

**[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  $\neq 0$ .

### Requirements

**Header** Declared in `acs_api.h`

**Library** Use `libacs_api.so`



## 14.7. 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

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

### Return Values

If successful, the return value is 0.

If failed, the return value is  $\neq 0$ .

### 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 mfid = 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, mfid);
    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;
}
```



## 15.0. Contactless Reader APIs

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

### 15.1. Open the contactless reader module

This function opens 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`



## 15.2. Close the contactless reader module

This function closes 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`



### 15.3. 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.

**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 **Contactless card reader antenna control**).

#### Requirements

Header      Declared in acs\_api.h

Library     Use libacs\_api.so



## 15.4. Activate the contactless card

This function activates the contactless card and gets the ATR.

```
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

### 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 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.



## 15.5. Deactivate the contactless card

This function deactivates 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





## 15.6. Contactless card data transfer

This function transmits the APDU command.

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

### Parameters

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

### Return Values

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

<b>Header</b>	Declared in acs_api.h
<b>Library</b>	Use libacs_api.so



## 15.7. FeliCa card data transfer

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

<b>[in] cmd</b>	The FeliCa command to be sent
<b>[in] cmdLen</b>	The length of the FeliCa command to be sent
<b>[out] resp</b>	The pointer of response data
<b>[out] respLen</b>	The length of the response data

### Return Values

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 **Contactless card reader antenna control**).

### Requirements

**Header** Declared in acs\_api.h

**Library** Use libacs\_api.so



## 15.8. Contactless card reader antenna control

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

```
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

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 ",txlbuf[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;
}
```



## 16.0. Magnetic Stripes Card APIs

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

### 16.1. Get track data from a 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

- [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 4:** Track Data State Bits Table



### Return values

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

Otherwise, the return value is  $\neq 0$ .

### 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 contact smart card reader module.

### 17.1. Get the error description by a given error code

For debugging purposes, 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



## 18.0.INI file parser APIs

This section describes the API functions for parse initialization file.

### 18.1. Get an 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

<b>[in] module_name</b>	The section name of .ini file
<b>[in] key_name</b>	The keyword name in .ini file
<b>[out] key_value</b>	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 brightness you get from ini file just now.  
        printf("[lcd]\nbrightness=%s\n", key_value);  
    }  
  
    return ret;  
}
```



## 18.2. Set a 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

- [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;  
}
```



### 18.3. 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 to be added in .ini file
- [in] key\_name**         The keyword name to be added in .ini file
- [in] key\_value**        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;  
}
```



## 18.4. 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;
}
```



## 18.5. Set hardware value according to the 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

**[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;  
}
```



## 19.0. Power Management APIs

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

### 19.1. 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

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

### 19.2. Get system sleep time

This function gets the device idle time.

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

#### Return Values

The value of the device's idle time

#### Requirements

**Header** Declared in acs\_api.h.

**Library** Use libacs\_api.so.



### 19.3. Enable or disable system auto sleep

This function enables/disables the device's auto sleep function.

There are two kinds of sleep mode: the deep sleep mode and the early sleep mode. The deep sleep mode freezes all the processes and makes the CPU go to sleep (see Example 1).

The early sleep mode sends the device to an early sleep when the Wi-Fi or GPRS is turned on. In this mode, the CPU is not in sleep mode, but the LCD and keypad are closed (see Example 2).

```
int pm_sleep_enable(unsigned char isEnabled)
```

#### Parameters

**[in]** isEnabled

isEnabled = 1 Enable device auto sleep

isEnabled = 0 Disable device auto sleep

#### Notes:

- To disable device to automatically go to sleep mode (see Example 3).
- To resume the device to normal working status, short press the power button.

#### 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 Codes

### 1. For deep sleep mode

```
#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[])
{
    /* if no touch screen and keypad operation within 30 seconds,
       device will going to deep sleep mode.
    */
    pm_sleep_timeout_set(30);
    pm_sleep_enable(1);

    return 0;
}
```

### 2. For early sleep mode

```
#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[])
{
    /* if WIFI or GPRS power on, you can short press power button
       to make device go to early sleep mode,
       and auto sleep is disable.
    */
    pm_sleep_timeout_set(30);
    pm_sleep_enable(1);

    wifi_pwr_on(); /* you can only short press power button
                   to make device go to early sleep mode */
    ...
    wifi_pwr_off(); /* when wifi power off the device will switch
                   to deep sleep mode, and auto sleep is reenable */
    return 0;
}
```



3. To disable device to automatically go to sleep mode

```
#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[])
{
    /* disable device auto go to sleep mode, and you can short press
       power button make device go to sleep mode*/

    pm_sleep_enable(0);

    return 0;
}
```