Advanced Card Systems Ltd.
Card & Reader Technologies

ACR1283L Standalone Contactless Reader

Application Programming Interface V1.00

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

Table of Contents

List of Figures

List of Tables

1.0. Introduction

The ACR1283L Standalone Contactless Reader is a device that is used for accessing contactless cards. Its contactless interface is used to access ISO 14443 Types A and B cards and Mifare series. ACR1283L also has Secure Access Module (SAM) interface that ensures a high level of security in contactless smart card applications.

ACR1283L can operate in both PC-linked and Standalone mode. In standalone mode, the ACR1283L can perform contactless card operations without the commands coming from a host PC. This document provides a detailed guide in implementing a contactless application in standalone mode.

2.0. Features

- Dual Operation Modes:
  - PC-linked
  - Standalone
- PC-linked Operation:
  - USB 2.0 Full Speed Interface
  - CCID Compliance
  - Supports PC/SC
  - Supports CT-API (through wrapper on top of PC/SC)
- Standalone Operation:
  - Support for third-party application programming
  - Over 400 KB memory space for third-party application
  - Over 500 KB memory space for data storage
  - Supported development platform:
    - IAR Embedded Workbench, Version 5.50 or above
    - CoIDE(GCC), Version 1.3.0 or above
- Smart Card Reader:
  - Read/Write speed of up to 848 kbps
  - Built-in antenna for contactless tag access, with card reading distance of up to 50 mm (depending on tag type)
  - Support for ISO 14443 Part 4 Type A and B and Mifare series
  - Built-in anti-collision feature (only one tag is accessed at any time)
  - Four ISO 7816 compliant SAM slots
- Built-in Peripherals:
  - Two-line graphic LCD
  - Four user-controllable LEDs
  - User-controllable buzzer
  - Twelve-key capacitive touch keypad
- Real-time clock (RTC) with independent back up battery
- In-device AES (128 and 256), DES and 3DES encryption
- Supports Android™ OS 3.1 and above
- USB Firmware Upgradability

- Compliant with the following standards:
  - ISO 14443
  - CE
  - FCC
  - PC/SC
  - CCID
  - Microsoft® WHQL
  - RoHS
  - USB 2.0 Full Speed Support

3.0. Symbols and Abbreviations

Abbreviation	Meaning
APDU	Application Protocol Data Unit
APDU Command	Four-byte sequence that begins with APDU (e.g., CLA INS P1 P2)
Header	(ISO/IEC 7816-4 § 5.3.1)
ATR	Answer To Reset
CCID	Integrated Circuit(s) Cards Interface Device conforming to this specification
API	Application Programming Interface
DES	Date Encryption Standard (FIPS Pub 46)
3DES	Triple DES (which applies the DES cipher algorithm three times to each data block)
AES	Advanced Encryption Standard
PnP	Plug-and-Play
FRAM	Ferroelectric non-volatile RAM

Table 1: Symbols and Abbreviations

4.0. USB Communication Protocol

Communication data frame package follows CCID_Rev110 standard. Please see the reference note and sample description below for more information.

Note: *Does not support extended APDU.*

4.1. Data Package Format

Data package includes 10-byte header and follows with user data. The 10-byte header includes the following message:

Offset	Field	Size
0	*bMessageType*	1
1	*dwLength*	4
5	*bSlot*	1
6	*bSeq*	1
7	*bBWI*	1
8	*wLevelParameter*	2
10	*adData*	Byte array

bSlot 0 = PICC card slot
1 = SAM1 card slot
2 = SAM2 card slot
3 = SAM3 card slot
4 = SAM4 card slot

Notes:

1. *Data length does not include the 10-byte header.*
2. *LSB transmits first, MSB the last.*

4.2. Sample Code

Command: 6F 05 00 00 00 011100 00 00 80 84 00 00 08

Explanation:

6F	05 00 00 00	01	11	00 00 00	80 84 00 00 08

Information Type: 0x6Fh
Note: *For PC_to_RDR_XfrBlock, please refer to CCID_Rev110.*

Data Length: 0x00000005 >> 5-bytes user data

Card Slot: 0x01h >> Data send to SAM1

Serial Number: 0x11h

Special Usage: 0x00h 0x00h 0x00h

User Data: 0x80h 0x84h 0x00h 0x00h 0x08h

5.0. Third-party Programming Architecture

5.1. Architecture

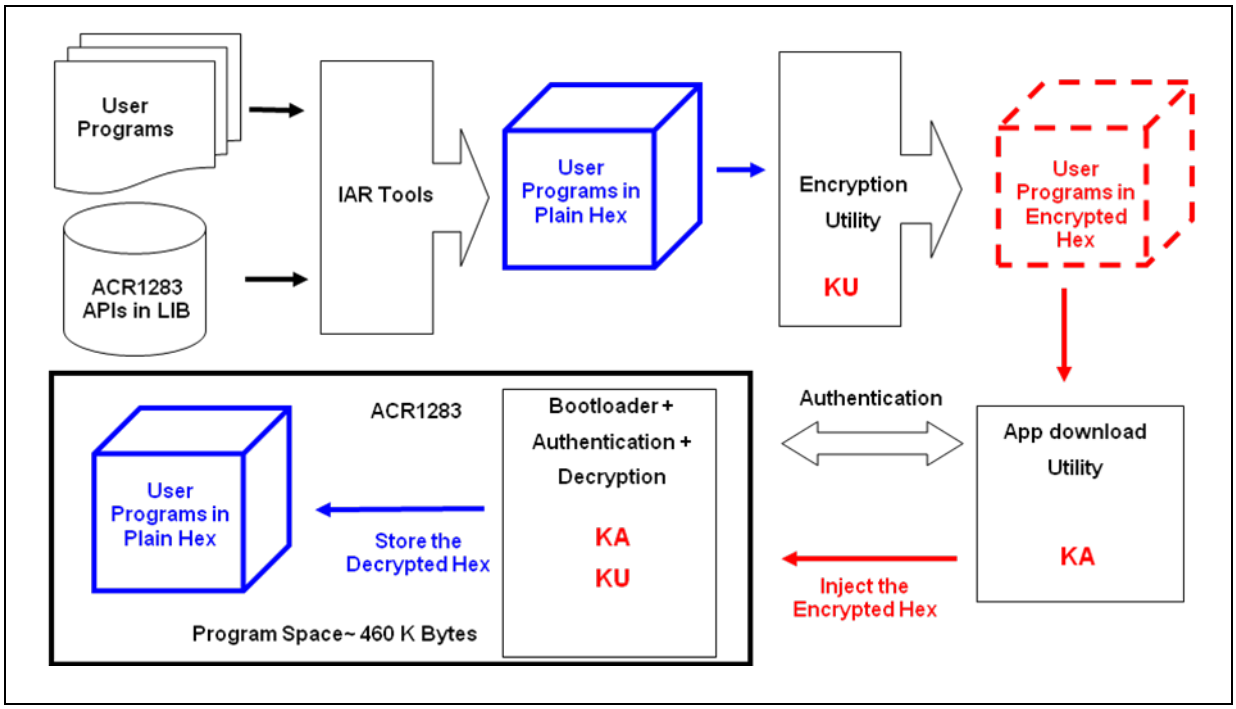

Figure 1: Third-party Programming Architecture

User Programs	The program that is written by third party, which performs the application that the user defined.
ACR1283 APIs in LIB	The internal program for controlling all the peripherals (which is not an open source), and provides the API for User Program Use.
IAR Tools	Compiled Tools. For compiling APP, and supporting IAR and GCC.
Encrypted Hex	Encrypted by AES to protest the IP issue.
App Download Utility	Embeds the authentication process and download process.

5.2. Entry Point

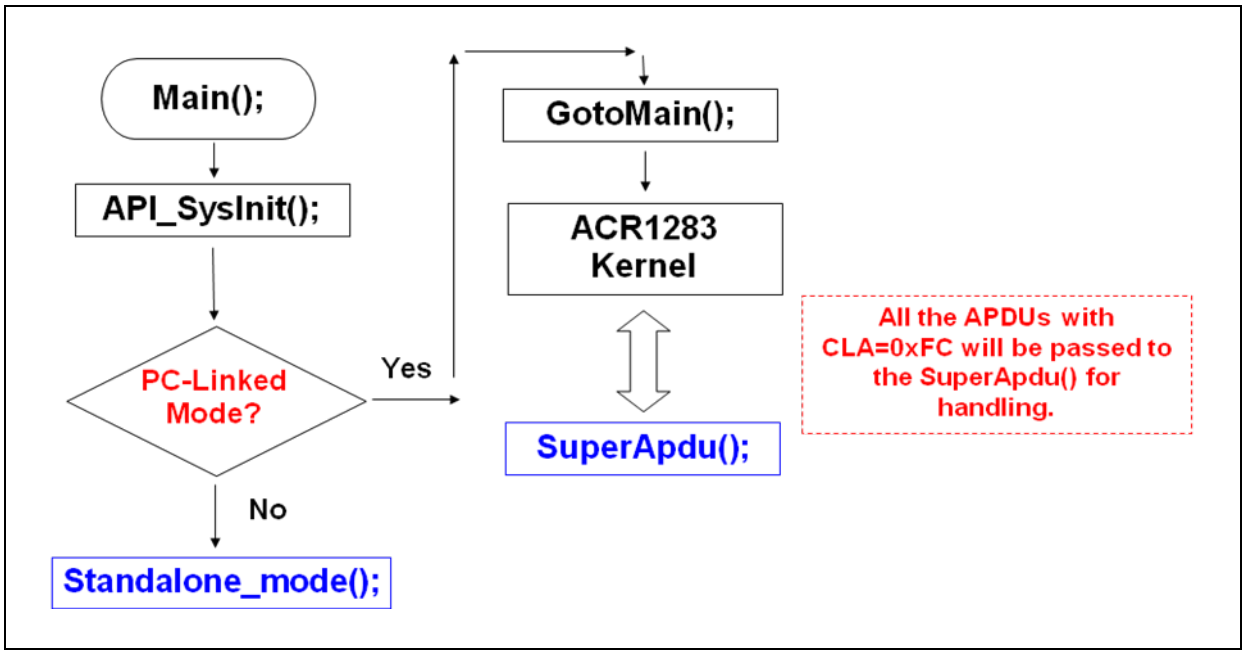

Figure 2: Entry Point Flowchart

6.0. Application Programming Interface

This section introduces the function, required parameters and return values of each API. Simple source code is provided and aimed for illustrating purpose only. Users have to modify these based on their needs.

6.1. Data Type

- **INT8** 8 bits signed integer
- **UINT8** 8 bits unsigned integer
- **INT16** 16 bits signed integer
- **UINT16** 16 bits unsigned integer
- **INT32** 32 bits signed integer
- **UINT32** 32 bits unsigned integer

6.2. Card Operation API Functions

This section lists out the API functions that are related to card operation. For these commands, the card interface to be used must be set through a parameter. The table below shows the arrangement.

ucSlot	Card Slot
0x01h	PICC
0x02h	SAM1
0x03h	SAM2
0x04h	SAM3
0x05h	SAM4

6.2.1. Data Exchange

This function is used to access APDU and pseudo APDU, and then receive the return message:

```
UINT8 API_ExAPDU(UINT8 ucSlot,  UINT8 *pucApdu,  UINT16 uiApduLen,
UINT8 *pucRsp,  UINT16 *puiRspLen, UINT8 *pucSw)
```

Parameters

ucSlot	Card slot index
pucApdu	Address of the buffer where the data is stored; to send
uiApduLen	Length of the data (16 bits)
pucRsp	Address of the buffer for storing the return message
puiRspLen	Address of the buffer for storing the length of the return message (16 bits)
pucSw	Address of the buffer for storing the return status (SW0 SW1)

Note: *Return status length is 2 bytes.*

Return Parameters

The return value indicates the status of command execution. Please refer to **Appendix A** for the description of the status codes.

Sample Code

```
void GetRandomNbr( void )
{
UINT8  status, slot, response_array[8], sw[2];
UINT16 senddatalen, recdatalen;
static UINT8 apdu_array[] = { 0x80, 0x84, 0x00, 0x00, 0x08 };

slot = 0x01;
senddatalen = 0x05;

status              =              API_ExAPDU(slot,apdu_array,senddatalen,
response_array,&recdatalen, sw);
/*status = 00, mean success, if failed, please refer to appendix 1 for
the status code
Content of response_array:8 bytes random number;
recdatalen = 0x0A; 8 bytes random number + SW1 SW2
*/
......

}
```

Meaning

Get Challenge is sent to SAM1 in order to get 8 bytes random number.

APDU = “0x80h 0x84h 0x00h 0x00h 0x08h”

Response = 8 bytes random number

SW1 SW2 = “0x90h 0x00h”

6.2.2. Activating a Card

This function activates the card and receives ATR:

```
UINT8 API CardPowerOn(UINT8 ucSlot, UINT8 *pcuATR, UINT8 *pucAtrLen)
```

Parameters

ucSlot	Card slot index
pcuATR	Address of the buffer for storing the returned ATR from card
pucAtrLen	Address of the buffer for storing the length of *pcuATR*

Return Parameters

The return value indicates the result of card activation. Please refer to **Appendix A** for the description of the status codes.

Sample Code

```
void PowerOn( void )
 {
UINT8  status,  artlen, atr_array[64];
status = API_CardPowerOn ( 0x02, atr_array , &artlen ),

/* status = 00, mean success, if failed, please refer to appendix 1 for
the status code contend of atr_array, for example:0x3b 0x8f 0x80 0x01
```

```
0x80 0x4f 0x0c 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00
0x00 0x6a
artlen = 0x14 */
......
 }
```

Meaning

Activates SAM2.

Example of ATR: "0x3B 0x8F 0x80 0x01 0x80 0x4F 0x0c 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00 0x00 0x6Ah"

6.2.3. Deactivating a Card

This function deactivates the card due to complete operating the card:

```
UINT8 API_CardPowerOff(UINT8 ucSlot)
```

Parameters

ucSlot Card slot index

Return Parameters

Success API_OK

Fail API_ERR_INDEX/API_ERR_CMDFAIL

6.2.4. Reactivating a Card

This function activates the card and returns the card slot status:

```
UINT8 API_InterfaceRefresh(UINT8 ucSlot, UINT8 *pucStatus);
```

Note: *This function does not return ATR.*

Parameters

ucSlot Card slot index

pucStatus Card slot's current status

pucStatus Bit definition:

Bit	Card Slot
Bit-0	PICC
Bit-1	SAM1
Bit-2	SAM2
Bit-3	SAM3
Bit-4	SAM4

Note: *All 0 means no card, or it fails to activate the slot.*

Return Parameters

The return value indicates the result of card activation. Please refer to **Appendix A** for the meaning of

the status codes.

6.2.5. Setting PICC Communication Speed

This function sets the PICC communication speed. If the speed set by the user is larger than the maximum speed of the card, the maximum speed of the card is used.

```
Void API_SetPPSPicc(UINT8 ucPar);
```

Parameters

ucPar Communication speed set by user

- 0x00h = 106k bps
- 0x01h = 212k bps
- 0x02h = 424k bps (default setting)
- 0x03h = 848k bps

Return Parameter

None.

6.2.6. Setting SAM1~SAM4 Communication Speed

FiDi set ICC communication speed:

```
UINT8 API_SetPPSIcc(UINT8 ucSlot,UINT8 pucFiDi,UINT8 ucPar);
```

***Fi* and *f* (max.)**

Bits 8 to 5	0000	0001	0010	0011	0100	0101	0110	0111
Fi	372	372	558	744	1116	1488	1860	RFU
f (max.) MHz	4	5	6	8	12	16	20	—

Bits 8 to 5	1000	1001	1010	1011	1100	1101	1110	1111
Fi	RFU	512	768	1024	1536	2048	RFU	RFU
f (max.) MHz	—	5	7.5	10	15	20	—	—

Di

Bits 4 to 1	0000	0001	0010	0011	0100	0101	0110	0111
Di	RFU	1	2	4	8	16	32	64

Bits 4 to 1	1000	1001	1010	1011	1100	1101	1110	1111
Di	12	20	RFU	RFU	RFU	RFU	RFU	RFU

Parameters

ucSlot Card slot index

pucFiDi Communication speed between card and reader

- bit7 to bit4 for Fi
- bit3 to bit0 for Di

ucPar Speed type

- 0 for data communication speed
- 1 for getting ATR speed

Setting ATR should be done before activation by:

```
UINT8 API_CardPowerOn(UINT8 ucSlot,UINT8 *pcuATR,UINT8 *pucAtrLen)
```

If user wants to set PPS, API, `UINT8 API_AutoPPSSet(UINT8 ucSlot,UINT8 ucPar)`, should be used first, and set PPS has to be done before activation.

Return Parameters

Success API_OK

Fail API_ERR_INDEX/API_ERR_CMDFAIL

6.2.7. Setting Auto PPS

This function enables/disables Auto PPS:

```
UINT8 API_AutoPPSSet(UINT8 ucSlot,UINT8 ucPar);
```

Parameters

ucSlot Card slot index

ucPar Enables/disables parameter

- 0 = disable auto PPS
- 1 = enable auto PPS

Return Value

Success 0x00h

Fail 0x01h

6.2.8. Getting ATR

This function returns ATR, but does not activate the card slot:

```
UINT8 API_GetATR(UINT8 ucSlot, UINT8 *pucATR, UINT8 *pucATRLen)
```

Parameters

ucSlot Card slot index

pucATR Address of the buffer for storing current card ATR

pucATRLen Address of the buffer for storing ATR length

Note: *Returns 0x00h if no card or other error.*

Return Value

Return status. Please refer to **Appendix A** for the status code.

Sample Code

```
void Get_ATR( void )
 {
UINT8  status,  artlen, atr_array[64];
status = GetATR ( 0x02, atr_array , &artlen ),
/* status = 00, mean success, if failed, please refer to appendix 1 for
the status code contend of atr_array, for example:0x3b 0x8f 0x80 0x01
0x80 0x4f 0x0c 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00
0x00 0x6a
artlen = 0x14 */
......
}
```

Meaning

Activates SAM2.

Example of ATR: “0x3B 0x8F 0x80 0x01 0x80 0x4F 0x0C 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00 0x00 0x6Ah”

6.2.9. Getting Card Slot Status

This function gets the status of the card slot:

```
UINT8 API_GetSlotStatus(UINT8 ucSlot, UINT8 *pucStatus);
```

Parameters

ucSlot Card slot index

pucStatus Card slot status

Bit	Card Slot
Bit-0	PICC
Bit-1	SAM1
Bit-2	SAM2
Bit-3	SAM3
Bit-4	SAM4

Note: *All 0 means no card or it fails to read the slot.*

Return Value

Return status. Please refer to **Appendix A** for the status code.

6.2.10. Escape Function

This function is used to handle self-define message (e.g., reader setting, self-define command, etc.)

```
UINT8 API_EscapeCmd(UINT8 ucSlot, UINT8 *pucCmd, UINT16 uiCmdLen, UINT8
*pucRsp, UINT16 *puiRspLen);
```

Parameters

ucSlot Card slot index

- 08 = Reader command

pucCmd Data address

uiCmdLen Data length

pucRsp Return data address

puiRspLen Return data address length

Return Value

Return status. Please refer to **Appendix A** for the status code.

6.2.11. Getting UID

This function gets the card IUD:

```
void API_GetUID(UINT8 * pucUIDBuf, UINT8 * pucUIDLength);
```

Parameters

pucUIDBuf Buffer for storing card UID

pucUIDLength Card UID length

Return Value

None.

6.2.12. Getting Volatile Key

This function gets the volatile key of the card:

```
void API_GetMFVolatileKey(UINT8 * pucKeyBuf);
```

Parameters

pucKeyBuf Buffer for storing 6-byte volatile key

Return Value

None.

6.2.13. Saving Volatile Key

This function saves the card's volatile key:

```
void API_StoreMFVolatileKey(UINT8 * pucKeyBuf);
```

Parameters

pucKeyBuf Buffer for storing 6-byte volatile key

Return Value

None.

6.2.14. Polling a Card

This function is used to poll a card:

```
UINT8 API_PollPICC(UINT8 ucTagType, UINT8 ucPollRetry, UINT8
ucPollInterval, UINT8 * pucUIDBuf, UINT8 * pucUIDLength);
```

Parameters

ucTagType Card type to be polled:

- 0 = TypeA
- 1 = TypeB

ucPollRetry Number of times of polling a card

- 1 = Poll once
- 2 = Poll twice, etc. (if *ucPollRetry* = 0, it will poll 256 times)

ucPollInterval Interval of polling card, unit is 10ms, 0 for polling card continuously

pucUIDBuf Buffer for storing card UID

pucUIDLength Card UID length

Return Value

Return status. Please refer to **Appendix A** for the description of status codes.

6.2.15. Setting Mifare Authentication

This function sets authentication for a Mifare card:

```
void API_MFAuthNeed(UINT8 ucCmd);
```

Parameters

ucCmd 0 = Does not require Mifare card authentication

1 = Requires Mifare card authentication

Return Value

None.

6.2.16. Setting Parameter for Polling a Card

This function sets the parameter for polling a card:

```
void API_PICCPollingSet(UINT8 ucAutoPolling, UINT8 ucCardTypes, UINT8
ucPart4Enter, UINT8 ucPollingInterval);
```

Parameters

ucAutoPolling 0 = Turn off auto polling card

1 = Turn on auto polling card

ucCardTypes Card type to be polled

- 0x01h = Type A
- 0x02h = Type B

- 0x03h = Type A + Type B

ucPart4Enter Enter OSP/IEC 14443 part 4,

- 0 = Do not enter part 4
- 1 = Enter part 4

ucPollingInterval Interval for polling card, unit is 125 ms

- 0 = for polling card continuously

Return Value

None.

6.2.17. Halting a Mifare Card

This function sets the selected Mifare card to halt mode:

```
UINT8 API_HaltMF(void);
```

Parameters

No parameters.

Return Value

Return status. Please refer to **Appendix A** for the description of status codes.

6.2.18. Waking up a Mifare card

This function is used to wake up a halted Mifare card:

```
UINT8 API_WupMF(void)
```

Parameters

No parameters.

Return Value

Return status. Please refer to **Appendix A** for the description of status codes.

6.2.19. Checking SAM File Status

This function checks the SAM card key file if it is opened or closed (used for testing):

```
UINT8 API_GetSAMFileStatus(UINT8 ucSlot);
```

Parameters

ucSlot Card slot index

Return Value

Return status. Please refer to **Appendix A** for the description of status codes.

6.2.20. Setting SAM File Status

This function sets the SAM card key file to open or close status:

```
UINT8 API_SAMFileStatusConfigure(UINT8 ucSlot, UINT8 ucCmd);
```

Parameters

ucSlot Card slot index

ucCmd 0 = key file is closed

1 = key file is opened

Return Value

Success API_OK

Fail API_ERR_INDEX

6.3. Reader API Functions

This part introduces API functions for setting the reader and attaining reader status.

6.3.1. Checking the Firmware Version

This function is used to read the firmware version:

```
void API_GetFWVersion(UINT8 *pucBuffer, UINT8 *pucLen);
```

Parameters

pucBuffer Buffer for store firmware version

17 bytes of firmware version length, the format as follow:

ACR1283U Vx.x.x.x; where x represents version number.

Note: *Each character of the bytes is using ASCII (e.g. Y = 0x59, S = 0x53, T = 0x54)*

pucLen Firmware version length

Return Value

None.

6.3.2. Resetting the Reader

This function resets the reader:

```
void API_SystemReset(void);
```

Parameters

No parameters.

Return Value

None.

6.3.3. Entering Firmware Upgrade Mode

This function is used to enter the reader's firmware upgrade mode:

```
UINT8 API_EnterDfuMode(void);
```

Parameters

No parameters.

Return Value

Success No return

Fail Return API_ERR_CMDFAIL

6.3.4. Controlling the Buzzer

This function sets the buzzer operation mode:

```
void API_BuzzerCtrl(UINT8 ucOnTime,UINT8 ucOffTime,UINT8 ucCnt,UINT8
ucMode);
```

Parameters

ucOnTime Buzzer on time

ucOnTime Setting range is 0x01h to 0xFFh; unit is 10 ms

(e.g., if *ucOnTime* equal to 0x64h, buzzer on time is 100 x 10 ms = 1000 ms)

ucOffTime Buzzer off time

- *ucOffTime* setting range is 0x01h to 0xFFh; unit is 10 ms

ucCnt Repeat time

- 0x00h = Buzzer off
- 0x01h~0xFEh = The number of repeat tuning buzzer on
- 0xFFh = Buzzer on continuously

ucMode Mode setting

- 0 = blocking mode (the software will not run until buzzer on is finished)
- 1 = unblocking mode

Return Value

None.

Sample Code

```
void BuzzerTest(void)
{
//100ms on, 100ms off, repeat 10 times in blocking mode
API_BuzzerCtrl(10,10,10,0);
// 100ms on, 200ms off, repeat 5 times in blocking mode
API_BuzzerCtrl(10,20,5,0);
// 100ms on, 500ms off, repeat 5 times in unblocking mode
API_BuzzerCtrl(10,50,5,1);
}
```

6.3.5. Setting LED Status

This function sets the status of LEDs 1 to 4:

```
void API_LedCtrl(UINT8 ucStatus,UINT8 ucNbr);
```

Parameters

ucStatus	Turn on/off LED 1 to 4
ucNbr	0 = All D1 to D4
	1 = D1
	2 = D2
	3 = D3
	4 = D4

If *ucNbr* is equal to 1 to 4, setting the less significant bit of *ucStatus* to 1 will turn on the related LED and setting it to 0 will turn it off.

If *ucNbr* is equal to 0, the four LEDs can be controlled at the same time. Setting the low four bits b0, b1, b2, b3 can control D1, D2, D3, D4 respectively. If the bit is 1/0, the related LED will turn on/off.

Return Value

None.

Sample Code

```
void DelayLedTest(void)
{
API_LedCtrl(0x00,0); // all off

API_LedCtrl(0x01,1); // on D1
API_LedCtrl(0x01,2); // on D2
 API_LedCtrl(0x01,3); // on D3
 API_LedCtrl(0x01,4); // on D4

 API_LedCtrl(0x0f,0); // all on

 API_LedCtrl(0x00,1); // off D1
 API_LedCtrl(0x00,2); // off D2
 API_LedCtrl(0x00,3); // off D3
 API_LedCtrl(0x00,4); // off D4
}
```

6.3.6. Reading LED Current Status

This function is used to read the LED's current status:

```
UINT8 API_LedStatus(void);
```

Parameters

No parameters.

Return Value

Return the current LED status.

- bit0 <-------> LED1 (PCB print : D1)
- bit1 <-------> LED2 (PCB print : D2)
- bit2 <-------> LED3 (PCB print : D3)
- bit3 <-------> LED4 (PCB print : D4)
- “1” = LED is turned on
- “0” = LED is turned off

6.4. LCD API Functions

This part introduces the LCD-related APIs.

6.4.1. Setting LCD Backlight

This function sets the LCD’s backlight status:

```
void APT_LcdBackLightCtrl(UINT8 ucCtrlCode) ;
```

Parameters

ucCtrlCode 0 = turns backlight off

1 = turns backlight on

Return Value

None.

6.4.2. Displaying Characters in LCD

This function is used to display characters in the LCD according to the character table:

```
UINT8 API_LcdDisplay(UINT8 Table, UINT8 *pLCDBuf, UINT8 r len, unsigned
char X, UINT8 Y)
```

Parameters

Table Table = 0X for ASCII Table1

Table = 1X for ACSII Table2

Table = 2X for ACSII Table3

Table = 4X for GB

Table = 0xX1 bold character

Table = 0xX0 normal character

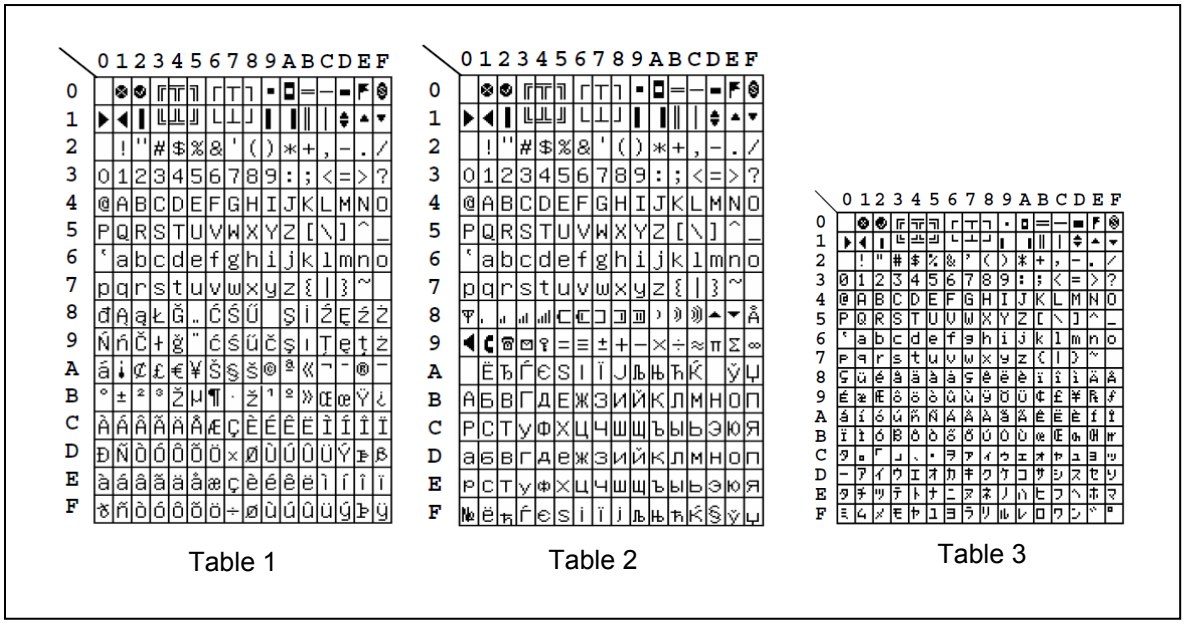

Figure 3: Character Display Table

pLCDBuf Show the address of the message

len Show the length of the message (maximum 16 bytes)

- X: x coordinate (0 to15)
- Y: y coordinate (0 or 1)

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

Sample Code

```
API_LcdDispay(0x10,"ACR1283",7,5,0); //display the message ”ACR1283”
```

6.4.3. Displaying Graphs in LCD

This function displays graph in the LCD:

```
UINT8 API_LcdDraw(UINT8 line, UINT8 len, UINT8 * pMessage);
```

Parameters

Line Line index (refer to the table below)

Len Length of pixel data

pMessage Buffer for storing the message

	Byte 0x00h (X = 0x00h)								Byte 0x01h (X = 0x01h)								…	Byte 0x0Fh (X = 0x0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
0x00h																									
0x01h																									
0x02h																									
0x03h																									
0x04h																									
0x05h																									
0x06h																									
0x07h																									
0x08h																									
0x09h																									
…	…																								
0x1Fh																									

6.4.4. Controlling the LCD Scroll

This function is used to control the scrolling of LCD display:

```
UINT8 API_LcdScroll(UINT8 ScrollDirect, UINT8  ScrollSpeed,
                UINT8 X_Start, UINT8 Y_Start,
                UINT8 X_end, UINT8 Y_end);
```

Parameters

ScrollDirect Direction of scroll

Bit1	Bit0	Scrolling Direction
0	0	From Left to Right
0	1	From Right to Left
1	0	From Top to Bottom
1	1	From Bottom to Top

ScrollSpeed Speed of scroll

Bit0~Bit3 – number of pixels pre-scrolling

Bit7	Bit6	Bit5	Bit4	Scrolling period
0	0	0	0	1 Unit
0	0	0	1	3 Units
0	0	1	0	5 Units
0	0	1	1	7 Units
0	1	0	0	17 Units
0	1	0	1	19 Units

Bit7	Bit6	Bit5	Bit4	Scrolling period
0	1	1	0	21 Units
0	1	1	1	23 Units
1	0	0	0	129 Units
1	0	0	1	131 Units
1	0	1	0	133 Units
1	0	1	1	135 Units
1	1	0	0	145 Units
1	1	0	1	147 Units
1	1	1	0	149 Units
1	1	1	1	151 Units

X_Start start at X coordination

Y_Start start at Y coordination

X_end end of X coordination

Y_end end of Y coordination

LCD Display Position (Total LCD Size: 128x32)

	Byte 0x00h (X = 0x00h)								Byte 0x01h (X = 0x01h)								…	Byte 0x0Fh (X = 0x0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
0x00h																									
0x01h																									
0x02h																									
0x03h																									
0x04h																									
0x05h																									
0x06h																									
0x07h																									
0x08h																									
0x09h																									
…	…																								
0x1Fh																									

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

6.4.5. Clearing the LCD

This function is used to clear the LCD display:

```
UINT8 API_LcdCLear(void) ;
```

Parameters

No parameters.

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

6.4.6. Pausing the LCD Scroll

This function is used to pause the scrolling of the LCD display:

```
UINT8 API_LcdPauseScroll(void) ;
```

Parameters

No parameters.

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

6.4.7. Stopping the LCD Scroll

This function stops the scrolling of the LCD and returns to its initial position:

```
UINT8 API_LcdStopScroll(void);
```

Parameters

No parameters.

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

6.5. Keypad API Functions

This part introduces the keypad-related API.

6.5.1. Resetting Keypad Flash

This function is used to reset keypad parameters in flash:

```
UINT8 API_ResetKeypadFlash(void)
```

Parameters

No parameters.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.2. Configuring Keypad Flash Parameters

This function is used to configure keypad parameters in flash:

```
UINT8 API_ConfigKeypadFlashParam(UINT8* pfbuffer)
```

Parameters

pfbuffer Keypad parameters

Note: *Keypad parameters include:*

- *11 baseline control parameters (MHDR, NHDR, NCLR, FDLR, MHDF, NHDF, NCLF, FDLF, NHDT, NCLT, FDLT)*
- *24 threshold parameters (E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH, E10TTH, E10RTH, E11TTH, E11RTH)*
- *2 debounce parameters (Debounce Touch, Debounce Release)*
- *24 charge/discharge current and charge/discharge time parameters (CDC0, CDC1, CDC2, CDC3, CDC4, CDC5, CDC6, CDC7, CDC8, CDC9, CDC10, CDC11, CDT0, CDT1, CDT2, CDT3, CDT4, CDT5, CDT6, CDT7, CDT8, CDT9, CDT10, CDT11)*

The length of parameters must be equal to 61, or the API execution will fail.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.3. Configuring Keypad Parameters

This function is used to configure keypad parameters:

```
UINT8 API_ConfigKeypadParam(UINT8* pebuffer)
```

Parameters

pebuffer Keypad parameters

Note: *Keypad parameters include:*

- *11 baseline control parameters (MHDR, NHDR, NCLR, FDLR, MHDF, NHDF, NCLF, FDLF, NHDT, NCLT, FDLT)*
- *24 threshold parameters (E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH, E10TTH, E10RTH, E11TTH, E11RTH)*
- *2 debounce parameters (Debounce Touch, Debounce Release)*
- *24 charge/discharge current and charge/discharge time parameters (CDC0, CDC1,*

CDC2, CDC3, CDC4, CDC5, CDC6, CDC7, CDC8, CDC9, CDC10, CDC11, CDT0, CDT1, CDT2, CDT3, CDT4, CDT5, CDT6, CDT7, CDT8, CDT9, CDT10, CDT11)

The length of parameters must be equal to 61, or the API execution will fail.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.4. Resetting the Keypad

This function resets the keypad:

```
UINT8 API_ResetKeypad(void)
```

Parameters

No parameters.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.5. Configuring Baseline Control Parameters

This function is used to configure baseline control parameters:

```
UINT8 API_ConfigBaselineConParam(UINT8* pbbuffer)
```

Parameters

pbbuffer Baseline control parameters

Note: *Baseline control parameters include:*

- *MHDR, NHDR, NCLR, FDLR, MHDF, NHDF, NCLF, FDLF, NHDT, NCLT, FDLT*

The length of parameters must be equal to 11 or the API execution will fail.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.6. Configuring Threshold Parameters

This function is used to configure threshold parameters:

```
UINT8 API_ConfigThresholdParam(UINT8* psbuffer)
```

Parameters

psbuffer Threshold parameters

Note: *Threshold parameters include:*

- *E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH,*

E10TTH, E10RTH, E11TTH, E11RTH

The length of parameters must be equal to 24 or the API execution will fail.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.7. Configuring CDC Parameters

This function is used to configure charge/discharge current parameters:

```
UINT8 API_ConfigCDCParam(UINT8* pcbuffer)
```

Parameters

pcbuffer Charge/discharge current parameters

Note: *Charge/discharge current parameters include:*

- *CDC0, CDC1, CDC2, CDC3, CDC4, CDC5, CDC6, CDC7, CDC8, CDC9, CDC10, CDC11*

The length of parameters must be equal to 12 or the API execution will fail.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.8. Configuring CDT Parameters

This function is used to configure charge/discharge time parameters:

```
UINT8 API_ConfigCDTParam(UINT8* ptbuffer)
```

Parameters

ptbuffer Charge/discharge time parameters

Note: *Charge/discharge time parameters include*

- *CDT0, CDT1, CDT2, CDT3, CDT4, CDT5, CDT6, CDT7, CDT8, CDT9, CDT10, CDT11*

The length of parameters must be equal to 12 or the API execution will fail.

Return Parameters

Success API_OK

Fail API_ERR_CMDFAIL

6.5.9. Reading Key Status

This function is used to read the status of the keys:

```
void API_ReadKeyStatus(void)
```

Parameters

No parameters.

Return Parameters

Status of the keys.

6.5.10. Reading Keys

This function is used to read sequence of keys and save the keys touched:

```
void API_ReadKeys(UINT8 mode, UINT8 X, UINT8 Y, UINT8* ptkey, UINT8
keylength, UINT8 leavecondition, UINT8 waittime)
```

Parameters

Mode	0x0X: NOMAL, save the keys touched only 0x1X: DISPLAY, display and save the keys touched 0x2X: PASSWARD, display '*' and save the keys touched 0x4X: Beep after Key Press 0xX0: the keys 1, 2 …9, *, 0, # 0xX1: the keys A, B…I, *, _, # X, Y: Ref to "LCD character display"
ptkey	Point of the keys touched
keylength	Length of keys
Leavecondition	Complete sign • 0x01: Key-10 • 0x02: Key-11 • 0x04: Key-12
waittime	Time for user to touch keys. The range is 0~254s and 0xFFh is infinite.

Return Parameters

API_OK

6.6. Flash Memory API Functions

This part introduces the flash memory access API.

6.6.1. Writing to Flash Memory

This function is used to write data to the flash memory:

```
UINT8 API_FlashMemoryWrite(UINT8* pBuffer, UINT32 WriteAddr, UINT16
NumByteToWrite)
```

Parameters

pBuffer	Get the data buffer address
ReadAddr	Write the data address (Please refer to M25P40 datasheet)

NumByteToWrite Write the data length

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

Sample Code

```
//Write 6 bytes data from Cmdbuffer to 0x000000fd
API_FlashMemoryWrite(Cmdbuffer, 0x000000fd, 0x06)
```

6.6.2. Reading Flash Memory

This function is used to read data from the flash memory:

```
UINT8 API_FlashMemoryRead (UINT8* pBuffer, UINT32 ReadAddr, UINT16
NumByteToRead)
```

Parameters

pBuffer Save the data buffer address

ReadAddr Read the data address (Please refer to M25P40 datasheet)

NumByteToRead Read the data length

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

Sample Code

```
//Read 6 bytes data from Resbuffer to 0x000000fd
API_ FlashMemoryRead(Resbuffer, 0x000000fd, 0x06)
```

6.6.3. Erasing Flash Memory

This function erases the flash memory:

```
UINT8 API_FlashMemoryErase(UINT8 sector)
```

Parameters

sector sector = 0 Erase the whole flash memory

sector = 1 Erase the 1st sector

sector = 2 Erase the 2nd sector

sector = 3 Erase the 3rd sector

sector = 4 Erase the 4th sector

sector = 5 Erase the 5th sector

sector = 6 Erase the 6th sector

sector = 7 Erase the 7th sector

sector = 8 Erase the 8th sector

sector > 8 neglect

Note: *Please refer to M25P40 datasheet.*

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

Sample Code

```
API_FlashMemoryErase(0x04) // erase sector 4
```

6.7. Backup Registry API Functions

ACR1283L has an 84 bytes backup registry where it can be used to store important data since battery is present and the tamper switch is closed. Therefore, the data is still in the area once the system or power is rebooted. However, this data is deleted if the casing is forced to open (e.g., the tamper switch is opened).

6.7.1. Writing Data to Backup Registry

This function is used to store data to backup registry:

```
UINT8 API_BkpStoreData(UINT8 Addr,UINT8 *pData,UINT8 Len)
```

Parameters

Addr Address for the data (0-83)

pData Pointer address of the data to be written

Len Data length

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

Sample Code

```
// store 2 bytes data from Cmdbuffer to backup registry address 0
API_BkpStoreData(0, Cmdbuffer, 2)
```

6.7.2. Reading Data from the Backup Registry

This function is used to read data from the backup registry:

```
UINT8 API_BkpReadData(UINT8 Addr,UINT8 *pData,UINT8 Len)
```

Parameters

Addr Address of the data (0-83)

pData Pointer address of the data to be read

Len Data length

Return Value

Success	API_OK
Fail	API_ERR_CMDFAIL

Sample Code

```
// Read 2 bytes data from backup registry address 0 to Resbuffer
API_BkpReadData(0, Resbuffer, 2)
```

6.8. Encryption API Functions

This part introduces DEC, 3DEC and AES functions.

6.8.1. Encrypting DES

This function is used for DES encryption or decryption of data:

```
void API_DES(UINT8 *pucData, UINT32 ulDataLen, UINT8 *pucKey, UINT8
*pucInitialVector, UINT8 ucMode);
```

Parameters

pucData	Save address of encrypted/decrypted data or operation result
ulDataLen	Data length of the encrypted/decrypted data
pucKey	Save address of the encryption key (64 bits)
pucInitialVector	Save address of initial vector value
ucMode	Mode of operation: • mode = 0x0Eh (encryption) • mode = 0x0Dh (decryption) • mode = 0xE0h (EBC) • mode = 0xD0h (CBC)

Return Value

None.

6.8.2. Encrypting 3DES

This function is used for 3DES encryption and decryption process:

```
void API_3DES(UINT8 *pucData, UINT32 ulDataLen, UINT8 *pucKey, UINT8
*pucInitialVector, UINT8 ucMode);
```

Parameters

pucData	Save address of encrypted/decrypted data or operation result
ulDataLen	Data length of the encrypted/decrypted data
pucKey	Save address of the encryption key (192 bits)
pucInitialVector	Save address of initial vector value
ucMode	Mode of operation:

- mode = 0x0Eh (encryption)
- mode = 0x0Dh (decryption)
- mode = 0xE0h (EBC)
- mode = 0xD0h (CBC)

Return Value

None.

6.8.3. Encrypting AES

This function is used for AES encryption and decryption process:

```
void API_AES(UINT8 *pucData, UINT32 ulDataLen, UINT8 *pucKey,  UINT16
uiKeyLen,UINT8 *pucInitialVector, UINT8 ucMode);
```

Parameters

pucData Save address of encrypted/decrypted data or operation result

ulDataLen Data length of the encrypted/decrypted data

pucKey Save address of the encryption key

uiKeyIen Length of encryption key (128 bits or 256 bits)

pucInitialVector Save address of initial vector value

ucMode Mode of operation:

- mode = 0x0Eh (encryption)
- mode = 0x0Dh (decryption)
- mode = 0xE0h (EBC)
- mode = 0xD0h (CBC)

Return Value

None.

6.9. Other API Functions

6.9.1. Delaying ms

This function is used to delay *N* ms. Incoming parameter determines the value of *N*.

```
void API_DelayNms(UINT16 uiVal,UINT8 ucMode);
```

Parameters

uiVal Value of the delay ms (e.g., 0x0Ah means delay 10 ms)

ucMode Delay mode:

- 0 = blocking mode (the software will not run until the delay time is passed)
- 1 = unblocking mode (the software can execute other part, UINT8 API_BlockingDelayStatus (void) will return whether the delay time is passed.)

Return Value

None.

6.9.2. Delay us

This function delays *N* us. Incoming parameter determines the value of *N*.

```
void API_DelayNus(UINT16 uiVal, UINT8 ucMode);
```

Parameters

uiVal Value of the delay us (e.g., 0x0Ah means delay 10 us)

ucMode Delay mode:

- 0 = blocking mode (the software will not run until the delay time is passed)
- 1 = unblocking mode (the software can execute other part, UINT8 API_BlockingDelayStatus(void) will return whether the delay time is passed.)

Return Value

None.

6.9.3. Unblocking Mode Delay Time Query

This function checks if the set delay time has arrived for the unblocking mode in the delay function:

```
UINT8 API_BlockingDelayStatus(void);
```

Parameters

No parameters.

Return Value

zero Delay time is reached

non-zero Delay time is not yet reached

6.9.4. Transferring Data through Serial Port

This function is used to print CCID *XfrBlock* data via the serial port and returns the message or user defined data. HEX or ASCII output format can be chosen.

```
UINT8 API_UsartTransmit(UINT8 Optl, UINT8 *pData, UINT32 Len)
```

Parameters

Optl 0xxx xxxxb for ACSII output

1xxx xxxxb for HEX output

x000 0000b for printing CCID *XfrBlock* data

x000 0001b for printing CCID *XfrBlock* return message

x000 0002b for printing user defined data

pData User data pointer

Len User data length

Return Value

Success API_OK

Fail API_ERR_CMDFAIL

6.9.5. Getting the Current Time

This function is used to get the current time from the real-time clock:

```
void API_GetTime(struct TIME *psTime);
```

Parameters

psTime Variable address which saves the time

```
struct TIME{
UINT8 Second;
UINT8 Minute;
UINT8 Hour;
UINT8 Day;
UINT8 Month;
 INT8 Weekday;
UINT16 Year;
UINT8 yDay;
};
```

Return Value

None.

Sample Code

```
TIME tCurentTime;
API_GetTime(&tCurentTime);
```

6.9.6. Setting the Time

This function sets the current time to the real-time clock:

```
void API_SetTime(struct TIME *psTime);
```

Parameters

psTime Variable address which saves the time

```
struct TIME{
UINT8 Second;
UINT8 Minute;
UINT8 Hour;
UINT8 Day;
UINT8 Month;
 INT8 Weekday;
UINT16 Year;
UINT8 yDay;
};
```

Return Value

Success	API_OK
Fail	API_ERR_CMDFAIL

Sample Code

```
TIME tCurentTime;
tCurentTime = {59; 59; 23; 31; 12; 6, 2011, 365};
API_ SetTime (&tCurentTime);
```

Appendix A. Status Codes

Status Code (Hex)	Description		Notes
FFh	API_ERR_CMDABORT	Command abort	The current command is stopped by reader
FEh	API_ERR_MUTE	No return	CCID timeout
FDh	API_ERR_PARITE	Parity error	Parity error during communicate with the card
FCh	API_ERR_LENGTH	Length error	Data length error message
FBh	API_ERR_HARDWARE	Hardware error	-
F8h	API_ERR_TS	TS error	-
F7h	API_ERR_TCK	TCK error	-
F6h	API_ERR_PROTOCOL	Protocol do not support	-
F5h	API_ERR_CLASS	Class do not support	-
E1h~F4h	RFU	Undefined	Future use
E0h	API_ERR_SLOT_BUSY	Card slot busy	Previous command is running and new command was received
DFh~87h	RFU	Undefined	Future use
86h	API_ERR_TIMEOUT	Timeout	Execute command timeout
85h	API_ERR_CMDFAIL	Command fail	-
84h	API_ERR_FRAM_LEN	Read/write FRAM length error	Start address plus length is greater than 0x1FFFh
83h	API_ERR_FRAM_ADDR	Read/write FRAM address error	Start address is greater than 0x1FFFh
82h	API_ERR_CMDNOTSUPPORT	Command not support	-
81h	API_OK	Success	Command execute success
80h	API_ERR_INDEX	Card slot index error	Slot_ Number > 0x05h except 0x08h
7Fh~00h	RFU	Undefined	Future use

Table 2: Status Codes