PM500 Payment SDK Guide

Pointmobile Co., Ltd.
S/W R&D Team

COMPANY CONFIDENTIAL
Only to be disclosed to Point Mobile employees concerned with the project.
DO NOT MAKE COPIES

PM500 Payment SDK Guide

Revision History

Revision	Date	Change Description	Author
V01	2020.08.19	Initial draft	Emma Jung
V02	2020.09.24	Added Lib_PiccCheckEmv (Class PICC)	Emma Jung
V03	2020.11.18	Added MCR version (Class MCR) & Parameter of Lib_McrOpen Modified values of Set gray parameter	Emma Jung

Table of Contents

1. Abstract

Introduction

This document describes the APIs in the PM500 SDK for application development based on the secure payment module.

The SDK supports below features.

- Control the system/device - Class System
- Control the Card reader module - Class ICC (For ICCR),
  Class MCR (For MSR),
  Class PICC (For contactless card module)
- Control the Thermal Printer - Class Printer
- Control the secure key management - Class PCI

Development Environment

System Configuration	RAM	64M or more
	Free hard disk space	200M or more
	Operating System	Windows 2000 or later
Development Tools	Eclipse	
Library	PaymentAPI.jar libAndroid.so	

2. Application Download

Connect the USB cable, and download and install application through ADB.

3. Application API Class Description

The package name of API is vpos.apipackage

Class System

- **Constructor:** int Sys();
- **Methods**
  - Initialization
  - Beep
  - Get Version
  - Get PCI Version
  - Get Device Info
  - Get Chip SN

1) Initialization

Initialize / close communication with the master app

Note

*** Init**

```
import vpos.messenger.MessengerClient;

MessengerClient mClient = null;
mClient =
MessengerClient.getInstance(getApplicationContext());
mClient.init();

new Thread(){
    public void run() {
        if(!mClient.isConnect()) {
            sleepMs(500);
            if(!mClient.isConnect()) {
                sleepMs(1000);
                if(!mClient.isConnect()) {
                    SendMsg("Connect Master failed!", 4);
                    return;
                }
            }
        }

        SendMsg("Connect Master Success!", 4);
    };
}.start();
```

*** Close**

```
mClient.close();
```

2) Beep

Sound beep in fixed frequency.

Function	`int Lib_Beep();`
Parameter	void
Return Value	0: Success Other: Failed

3) Get Version

Get the version information of secure module and App.

Function	`int Lib_GetVersion(byte[] buf);`
Parameter	buf (byte[]) [out] – The size of buffer should be larger than 6 to save version info. buf[0~2]: secure module's version (1.0.0) buf[3~5]: App's version (1.0.0)
Return Value	0: Success Other: Failed

4) Get PCI Version

Get the PCI version information of secure module and App.

Function	int Lib_GetPciVersion(byte[] buf);
Parameter	buf (byte[]) [out] – The size of buffer should be larger than 6 to save version info. Buf[0~2]: secure module's version (1.0.0) Buf[3~5]: App's version (1.0.0)
Return Value	0: Success Other: Failed

5) Get Device Info

Get the device information.

Function	`int Lib_GetDeviceInfo(byte[] icn,`
	`byte[] icv,`
	`byte[] msrn,`
	`byte[] msrv,`
	`byte[] piccn,`
	`byte[] piccv);`

Parameter	icn (byte[]) [out] – ICC module name
	icv (byte[]) [out] – ICC module version
	msrn (byte[]) [out] – MSR module name
	msrv (byte[]) [out] – MSR module version
	piccn (byte[]) [out] – PICC module name
	piccv (byte[]) [out] – PICC module version
Return Value	0: Success Other: Failed

6) Get Chip SN

Get the sn information of secure module.

Function	`int Lib_GetSN(byte[] sn);`
Parameter	sn (byte[]) [out] – The size of buffer should be larger than 13 to save version info.
Return Value	0: Success Other: Failed

Class ICC

- **Constructor**: Icc();
- **Methods**
  - Open
  - Close
  - Command Exchange
  - Check

1) Open

Reset card and get the reset response.

Function	`int Lib_IccOpen(byte slot,`
	`byte vccMode,`
	`byte[] ATR);`

Parameter	slot (byte) [in] – Slot No. 0: EMV IC Card 1: PSAM1 2: PSAM2 3: PSAM3 4: PSAM4
	vccMode (byte) [in] – Card power supply voltage specified 1: 5V 2: 3V 3: 1.8V (reserve)
	ATR (byte[]) [out] – Card reset response. (At least 32 +1 bytes of space) The content length (1 byte) + reset response content
Return Value	0: Open successful (-2403): Channel number error (-2405): Pull out the card or no card (-2404): Protocol error (-2500): IC card reset voltage mode error (-2503): Communication failure Refer to Note row to find more error code.
Note	• ATR information is different as different card. Please refer to IC card manual to supply the buffer in enough length. (Maximum length is no more than 33 bytes). • The card can only be operated after being reset successfully. • Details of the IC card command return code

	0	Success
	(-2400)	Data length Error

		(-2401)	Parity Error
		(-2402)	Parameter cannot be void
		(-2403)	Card slot error
		(-2404)	Protocol error
		(-2405)	No card
		(-2406)	Card hasn't been initialized
		(-2500)	IC card reset voltage mode error
		(-2503)	Communication failed
	Card Reset Error Code Details	(-2100)	TS error
		(-2101)	TCK error
		(-2102)	ATR response timeout
		(-2103)	TA1 error
		(-2104)	TA2 error
		(-2105)	TA3 error
		(-2106)	TB1 error
		(-2107)	TB2 error
		(-2108)	TB3 error
		(-2109)	TC1 error
		(-2110)	TC2 error
		(-2111)	TC3 error
		(-2112)	TD1 error
		(-2113)	TD2 error
		(-2114)	ATR length error
	T=0 Card Communication Error Code Details	(-2200)	Card response timeout
		(-2201)	Resend error
		(-2202)	Receive error
		(-2203)	Character parity error
		(-2204)	State byte error
	T=1 Card Communication Error Code Details	(-2300)	BWT error
		(-2301)	CWT error
		(-2302)	ABORT communication error
		(-2303)	EDC error
		(-2304)	Synchronous communication error
		(-2305)	EGT error
		(-2306)	BGT error
		(-2307)	NAD error
		(-2308)	PCB error
		(-2309)	LEN error
		(-2310)	IFSC error
		(-2311)	IFSD error
		(-2312)	Too many times wrong
		(-2313)	Character parity error
		(-2314)	Invalid characters group

2) Close

Disable the card.

Function	`int Lib_IccClose(byte slot);`
Parameter	slot (byte) [in] – Slot No. 0: EMV IC Card 1: PSAM1 2: PSAM2 3: PSAM3 4: PSAM4
Return Value	0: Success Other: Failed
Note	No action other illegal value

3) Command Exchange

IC card operation function, which supports general ICC interface protocol (T = 0 and T = 1).

Function	`int Lib_IccCommand(byte slot,`
	`byte[] apduSend,`
	`byte[] apduResp);`

Parameter	Slot (byte) [in] – Slot number 0: EMV IC Card 1: PSAM1 2: PSAM2 3: PSAM3 4: PSAM4
	apduSend (byte[]) [in] – According to the define of “APDU_SEND” structure in “Comments”, send the bytearray in order. (Lc max is 512, data length max is 512 bytes.)
	apduResp (byte[]) [out] – According to the define of “APDU_RESP” structure in “Comments”, recv the bytearray in order.
Return Value	0: Success (-2400): The length of send data is too long (-2401): Parity error (-2403): The slot No is error (-2404): The protocol error (not T = 0 and T = 1) (-2405): Card moved out (-2406): Card reset error (-2503): Time out
Note	**APDU_SEND structure:** Struct { byte Command[4]; short Lc; byte DataIn[512];

	short Le;
	};
	
	Command[] = {CLA, INS, P1, P2}
	Lc = the length of DataIn
	DataIn = the data to be sent to ICC
	Le = the respect length of return data
	
	Case 1: Lc = 0; Le = 0 (No data send, and no data return)
	Case 2: Lc = 0; Le > 0 (No data send, but expect data return. If the length of return data is unsure, Le = 256.)
	Case 3: Lc > 0; Le = 0 (Data sent but no data return)
	Case 4: Lc > 0; Le > 0 (Data send and expect data return. If the length of return data is unsure, Le = 256.)
	
	APDU_RESP structure:
	Struct {
	short LenOut;
	byte DataOut[512];
	byte SWA;
	byte SWB;
	};
	
	LenOut = the length of return data from ICC
	DataOut = the return data from ICC
	SWA = Status Byte 1
	SWB = Status Byte 2

4) Check

Check there is a card in the slot.

Function	`int Lib_IccCheck(byte slot);`
Parameter	slot (byte) [in] – Slot No. 0: EMV IC Card 1: PSAM1 2: PSAM2 3: PSAM3 4: PSAM4
Return Value	0: Success Other: Failed
Note	No action other illegal value

- **Error Code**

ICC_T0_TIMEOUT	(-2200)
ICC_T0_MORESENDERR	(-2201)
ICC_T0_MORERECEERR	(-2202)
ICC_T0_PARERR	(-2203)
ICC_T0_INVALIDSW	(-2204)
ICC_DATA_LENTHERR	(-2400)
ICC_PARERR	(-2401)
ICC_PARAMETERERR	(-2402)
ICC_SLOTERR	(-2403)
ICC_PROTOCALERR	(-2404)
ICC_CARD_OUT	(-2405)
ICC_NO_INITERR	(-2406)
ICC_ICCMESSOVERTIME	(-2407)
ICC_PPSERR	(-2408)
ICC_ATR_TSERR	(-2100)
ICC_ATR_TCKERR	(-2101)
ICC_ATR_TIMEOUT	(-2102)
ICC_TS_TIMEOUT	(-2115)
ICC_ATR_TA1ERR	(-2103)
ICC_ATR_TA2ERR	(-2104)
ICC_ATR_TA3ERR	(-2105)
ICC_ATR_TB1ERR	(-2106)
ICC_ATR_TB2ERR	(-2107)
ICC_ATR_TB3ERR	(-2108)
ICC_ATR_TC1ERR	(-2109)
ICC_ATR_TC2ERR	(-2110)
ICC_ATR_TC3ERR	(-2111)

ICC_ATR_TD1ERR	(-2112)
ICC_ATR_TD2ERR	(-2113)
ICC_ATR_LENGTHERR	(-2114)
ICC_T1_BWTERR	(-2300)
ICC_T1_CWTERR	(-2301)
ICC_T1_ABORTERR	(-2302)
ICC_T1_EDCERR	(-2303)
ICC_T1_SYNCHERR	(-2304)
ICC_T1_EGTERR	(-2305)
ICC_T1_BGTERR	(-2306)
ICC_T1_NADERR	(-2307)
ICC_T1_PCBERR	(-2308)
ICC_T1_LENGTHERR	(-2309)
ICC_T1_IFSCERR	(-2310)
ICC_T1_IFSDERR	(-2311)
ICC_T1_MOREERR	(-2312)
ICC_T1_PARITYERR	(-2313)
ICC_T1_INVALIDBLOCK	(-2314)

Class MCR

- **Constructor** – Mcr();
- **Methods**
  - Open
  - Close
  - Check
  - Read

1) Open

Open the card reader.

Function	`int Lib_McrOpen(int mode);`
Parameter	mode (int) [in] 1 – encrypt mode 0 – unencrypt mode
Return Value	0: Success Other: Failed
Note	Read the magnetic card data by using interrupt way, once open the card reader, the magnetic head can read data as long as there is swipe, even the read function not called. Therefore, the magnetic card reader will be better closed when there no need to use the magnetic card reader.

2) Close

Close the card reader.

Function	`int Lib_McrClose();`
Parameter	void
Return Value	0: Success Other: Failed

3) Check

Check there is a swipe or not.

Function	`int lib_McrCheck();`
Parameter	void
Return Value	0: Swiped card Other: No card swiped
Note	This function will return immediately no matter there is a swipe or not.

4) Read

Read the 1, 2, 3 track data in the buffer.

Function	`int Lib_McrRead(byte[] track1,`
	`byte[] track2,`
	`byte[] track3);`

Parameter	track1 (byte[]) [out] – The buffer of track1 data
	track2 (byte[]) [out] – The buffer of track2 data
	track3 (byte[]) [out] – The buffer of track3 data
Return Value	0: Brush card error >0: * bit0 = 1: read track1 data ok * bit1 = 1: read track2 data ok * bit2 = 1: read track3 data ok * bit4 = 1: parity error of track1 data * bit5 = 1: parity error of track2 data * bit6 = 1: parity error of track3 data
Note	Coordinate with Lib_McrCheck function. If no need some track data, the corresponding track buffer pointer can be set to NULL, then the track data will not be output. Generally, the magnetic card's data is according to the ISO7811 standard: Track1 – 79 bytes Track2 – 40 bytes Track3 – 107 bytes This function also supports the magnetic card not conform to the ISO7811 standard.

5) MCR Version

Get the MCR version.

Function	`int Lib_McrFWVersion(byte[] version);`
Parameter	version (byte[]) [out] – 2 bytes New MCR – 0x32 0x40 Old MCR – 0x32 0x38, 0x32 0x39
Return Value	0: success Other: Failed
Note	

Class PCI

- **Constructor** : PinPad();
- **Methods**
  - Enter Loading Mode
  - Get Random
  - Write Main Key
  - Write Work Key
  - Write Dukpt Key
  - MAC encryption
  - Offline cipher PIN verification
  - Offline plain PIN Verification
  - Get Encrypted PAN
  - Write NVRAM
  - Read NVRAM
  - Get Encrypted Mac by DUKPT
  - Get PINBLOCK by DUKPT
  - Get ISO9564 format 0 PINBLOCK
  - Get ISO9564 format 3 PINBLOCK
  - Get ISO9564 format 4 PINBLOCK
  - Connect OpenSSL
  - OpenSSL Send
  - OpenSSL Recv
  - Disconnect OpenSSL
  - Load Openssl Key / Certificate

1) Enter Loading Mode

Enter the load master key mode.

Function	`int Lib_EnterLoad();`
Parameter	void
Return Value	0: Success Other: Administrator password error

Note	When entering the download mode, you need to enter the administrator password, and then you have 15 minutes to download master keys. If you exceed the time, you need to call the interface to re-enter the password. If the administrator password is entered incorrectly five times in a row, it will enter the trigger mode. Please refer to the PCI documentation for trigger recovery and administrator password download.

2) Get Random

Get 8 bytes random numbers from secure chip.

Function	`int Lib_GetRand(byte[] rand);`
Parameter	rand (byte[]) [out] – 8 bytes random
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

3) Write Main Key

Write PIN Main Key.

Function	`int Lib_LoadKeyPinPad(byte keyType,`
	`byte mode,`
	`byte keyNo,`
	`byte keyLen,`
	`byte[] key);`

Parameter	keyType (byte) [in] #define AUTHSKMACK_TYPE – 0x00 #define AUTHPINMKEY_TYPE – 0x01 #define AUTHMACMKEY_TYPE – 0x02 #define AUTHAESMK_TYPE – 0x03 #define AUTHFKEY_TYPE – 0x04 #define AUTHPANMK_TYPE – 0x05
	mode (byte) [in] – (reserve)
	keyNo (byte) [in] – Main key number (0~9)
	keyLen (byte) [in] – Length of key (16 or 24 bytes)
	key (byte[]) [in] – A pointer to main key data to write
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)
Note	Need to download the master key in download mode, refer to the “Lib_EnterLoad” interface.

4) Write Work Key

Write work key.

Function	`int Lib_LoadEncryptWorkKeyPinPad(byte keyType,`
	`byte MainKeyNo,`
	`byte WorkKeyNo,`
	`byte KeyLen,`
	`byte[] Key,`
	`byte[] Key_Crc);`

Parameter	keyType (byte) [in] #define KEYCRC_SESSION_TYPE_PIN – 0x06 #define KEYCRC_SESSION_TYPE_MAC – 0x07 #define KEYCRC_SESSION_TYPE_AES – 0x08 #define KEYCRC_SESSION_TYPE_PAN – 0x09
	MainKeyNo (byte) [in] – Main key number (0~9)
	WorkKeyNo (byte) [in] – Work key number (0~9)
	KeyLen (byte) [in] – Length of key (8, 16 or 24 bytes)
	Key (byte[]) [in] – Work key
	Key_Crc (byte[]) [in] – Key mac value encrypted with SK_MACK
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)
Note	• Length of the Mkey must be same as key length or bigger. • The master key used to encrypt the work keys must be loaded through App-TRSM before calling this function. • Use the sk mac key to calculate the following data mac: Keytype (1 byte) + MainKeyNo (1 byte) + keyLen (16 / 24) + SKMacKeyNo (1 byte) + 0x00000000 (4 bytes) + key (16 / 24 / 32)

5) Write Dukpt Key

Get encrypt PIN – BLOCK by specified PIN key.

Function	`int Lib_LoadDukptKey(byte KeyNo,`
	`byte KsnLen,`
	`byte BdkLen,`
	`byte[] Ksn,`
	`byte[] Bdk);`

Parameter	KeyNo (byte) [in] – Key number (0~9)
	KsnLen (byte) [in] – Length for KSN (10)

	BdkLen (byte) [in] – Length for BDK (16)
	Ksn (byte) [in] – Ksn data
	Bdk (byte) [in] – Bdk data
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)
Note	Need in download mode.

6) MAC Encryption

Get MAC data of input data by specified MAC key.

Function	`int Lib_GenerateMAC(byte dmode,`
	`byte KeyNo,`
	`byte DataLen,`
	`byte[] Datain,`
	`byte[] DataOut);`

Parameter	dmode (byte) [in] 0 – algorithm 1 1 – algorithm 2 2 – algorithm 3
	KeyNo (byte) [in] – The key index of MAC key (0~9)
	DataLen (byte) [in] – Length of data to encrypt (Maximum length must be less than 256 bytes)
	Datain (byte[]) [in] – Pointer to data to encrypt.
	DataOut (byte[]) [out] – Pointer to MAC data result.
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

Note	• If the message has a length that is not a multiple of 8 bytes, add the smallest number of 0x00 bytes to the right, such that the length of resulting message is a multiple of 8 bytes. Message is then divided into 8-bytes blocks X1, X2, …, Xk. • Algorithm selection: 0: Algorithm 1 1: Algorithm 2 2: Algorithm 3 • **arithmetic 1:** Result 0 = “\x00\x00\x00\x00\x00\x00\x00\x00\” XOR1= Result0⊕X1; Result1 = DES/3DES(MAC Key, XOR1); XOR2= Result1⊕X2; Result2 = DES/3DES(MAC Key, XOR2); …… XORk= Result1⊕Xk; Resultk = DES/3DES(MAC Key, XORk); Final result = Resultk. • **arithmetic 2:** Result 0 = “\x00\x00\x00\x00\x00\x00\x00\x00” Result1 = Result0⊕X1; Result2 = Result1⊕X2; …… Resultk = Resultk-1⊕Xk; Final result = DES/3DES(MAC Key, Resultk). • **arithmetic 3 :** Result 0 = “\x00\x00\x00\x00\x00\x00\x00\x00” XOR1= Result0⊕X1; Result1 = DES(MAC KeyA, XOR1); XOR2= Result1⊕X2; Result2 = DES(MAC KeyA, XOR2); …… XORk= Result1⊕Xk; Resultk = DES(MAC KeyA, XORk); Final result = TDES(MAC KeyB, Resultk). • **PS:** Length(MAC key) = 8 ► KeyA = KeyB = MAC KEY; Length(MAC key) = 16► KeyA = Left 8 bytes of MAC KEY; KeyB = Right 8 bytes of MAC KEY.

7) Offline Cipher PIN Verification

Offline Encrypt PIN verification functions. Application obtains an encrypted PIN asymmetric structure public key from the card through the card reader command, and obtain the corresponding random number from the card, combined with the application random numbers with the input PIN on terminal to do the RSA encryption and return the encrypted PIN BLOCK, then send the result to the IC card to verify.

Function	`int Lib_PciOfflineEncPin(byte Slot,`
	`byte Min_Len,`
	`byte Max_Len,`
	`byte Waittime_sec,`
	`PUBLIC_KEY pk,`
	`byte AmountLen,`
	`byte[] Amount);`

Parameter	Slot (byte) [in] – Icc slot number, 0
	Min_Len (byte) [in] – The minimum length of PIN, range: 4~12
	Max_Len (byte) [in] – The maximum length of PIN, range: 4~12
	Waittime_sec (byte) [in] – Time to wait. Unit is second. (0~60) 0: default 60s >= 60: 60s
	pk (PUBLIC_KEY) [in] – Encrypt key. PUBLIC_KEY: class for vpos.apipackage.PUBLIC_KEY.java
	AmountLen (byte) [in] – The length of amount
	Amount (byte[]) [in] – Transaction amount
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

8) Offline Plain PIN Verification

Offline plain PIN verification functions. Input PIN on terminal, and in accordance with the card command format and card channel number what the application provides, sent the plaintext PINBLOCK to card to verify.

Function	`int Lib_PciOfflinePlainPin(byte Slot,`
	`byte Min_Len,`
	`byte Max_Len,`
	`byte Waittime_sec,`
	`byte AmountLen,`
	`byte[] Amount);`

Parameter	slot (byte) [in] – Icc slot number, 0
	Min_Len (byte) [in] – The minimum length of PIN, range: 4~12

	Max_Len (byte) [in] – The maximum length of PIN, range: 4~12
	Waittime_sec (byte) [in] – Time to wait. (0~60 sec). 0: default 60s >= 60: 60s
	AmountLen (byte) [in] – The length of amount
	Amount (byte[]) [in] – Transaction amount
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

9) Get Encrypted PAN

Get encrypt pan.

Function	`int Lib_PciGetEncPAN(byte dmode,`
	`byte KeyNo,`
	`byte DataLen,`
	`byte[] Datain ,`
	`byte[] DataOut);`

Parameter	dmode (byte) [in] – (reserve)
	KeyNo (byte) [in] – The Key index of PAN key (0~9)
	DataLen (byte) [in] – Plaintext PAN
	Datain (byte[]) [in] – Length of plaintext PAN
	DataOut (byte[]) [out] – Ciphertext PAN
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

10) Write NVRAM

Write NVRAM.

Function	`int Lib_WriteNVRam(byte[] data);`
Parameter	data (byte[]) [in] – Limit 64 bytes
Return Value	0: Success Other: Failed

11) Read NVRAM

Read NVRAM.

Function	`int Lib_ReadNVRam(byte[] data);`
Parameter	data (byte[]) [out] – Limit 64 bytes
Return Value	0: Success Other: Failed

12) Get Encrypted Mac by DUKPT

Calculate the MAC value of message by using DUKPT.

Function	`int Lib_PciGetMacDukpt(byte dmode,`
	`byte KeyNo,`
	`byte DataLen,`
	`byte[] Datain,`
	`byte[] DataOut,`
	`byte[] OutKsn);`

Parameter	dmode (byte) [in] 0: algorithm 1 1: algorithm 2 2: algorithm 3
	KeyNo (byte) [in] – The key index of DUKPT (0~9)
	DataLen (byte) [in] – The length of mac message to be encryted, must less than 256 bytes
	Datain (byte[]) [in] – The mac message
	DataOut (byte[]) [out] – The encrypted result
	OutKsn (byte[]) [out] – Ksn
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

13) Get PINBLOCK by DUKPT

Get encrypted PIN-BLOCK by using DUKPT.

Function	`int Lib_PciGetPinDukpt(byte Key_No,`
	`byte Min_Len,`
	`byte Max_Len,`
	`byte Card_Len,`
	`byte[] CardNo,`
	`byte Waittime_sec,`
	`byte[] PinBlock,`
	`byte[] OutKsn);`

Parameter	Key_No (byte) [in] – The key index of DUKPT
	Min_Len (byte) [in] – The minimum length of PIN, range: 4~12
	Max_Len (byte) [in] – The maximum length of PIN, range: 4~12
	Card_Len (byte) [in] – The length of card No.
	CardNo (byte[]) [in] – The card No.
	Waittime_sec (byte) [in] – Time to wait (0~60 sec) 0: Default 60s >= 60: 60s
	PinBlock (byte[]) [out] – PIN block result (8 bytes)
	OutKsn (byte[]) [out] - Ksn
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

14) Get ISO9564 Format 0 PINBLOCK

Get encrypted PIN-BLOCK by using PIN key.

Function	`int Lib_PciGetPin(byte PinKey_No,`
	`byte Min_Len,`
	`byte Max_Len,`
	`byte Card_Len,`
	`byte[] CardNo,`
	`byte Waittime_sec,`
	`byte[] PinBlock,`
	`byte AmountLen,`
	`byte[] Amount);`

Parameter	PinKey_No (byte) [in] – The key index of PIN Key
	Min_Len (byte) [in] – The minimum length of PIN, range: 4~12
	Max_Len (byte) [in] – The maximum length of PIN, range: 4~12

	Card_Len (byte) [in] – The length of card No.
	CardNo (byte[]) [in] – The card No.
	Waittime_sec (byte) [in] – Time to wait (0~60 sec) 0: Default 60s >= 60: 60s
	PinBlock (byte[]) [out] – PIN block result (8 bytes)
	AmountLen (byte) [in] – Transaction amount length
	Amount (byte[]) [in] – Transaction amount
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

15) Get ISO9564 Format 3 PINBLOCK

Get encrypted PIN-BLOCK by fix key.

Function	`int Lib_PciGetPinFixK(byte FixKey_No,`
	`byte Min_Len,`
	`byte Max_Len,`
	`byte Card_Len,`
	`byte[] CardNo,`
	`byte Waittime_sec,`
	`byte[] PinBlock);`

Parameter	FixKey_No (byte) [in] – The key index of Fix Key
	Min_Len (byte) [in] – The minimum length of PIN, range: 4~12
	Max_Len (byte) [in] – The maximum length of PIN, range: 4~12
	Card_Len (byte) [in] – The length of card No.
	CardNo (byte[]) [in] – The card No.
	Waittime_sec (byte) [in] – Time to wait (0~60 sec) 0: Default 60s >= 60: 60s
	PinBlock (byte[]) [out] – Output encrypt PIN block (8 bytes)
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

16) Get ISO9564 Format 4 PINBLOCK

Get encrypted PIN-BLOCK by AES key.

Function	`int Lib_PciGetPinAes(byte AesKey_No,`
	`byte Min_Len,`
	`byte Max_Len,`
	`byte Card_Len,`
	`byte[] CardNo,`
	`byte Waittime_sec,`
	`byte[] PinBlock);`

Parameter	AesKey_No (byte) [in] – The key index of AES Key
	Min_Len (byte) [in] – The minimum length of PIN, range: 4~12
	Max_Len (byte) [in] – The maximum length of PIN, range: 4~12
	Card_Len (byte) [in] – The length of card No.
	CardNo (byte[]) [in] – The card No.
	Waittime_sec (byte) [in] – Time to wait (0~60 sec) 0: Default 60s >= 60: 60s
	PinBlock (byte[]) [out] – Output encrypt PIN block (16 bytes)
Return Value	0: Success Other: Failed (Refer to PCI_ERROR_CODE)

17) Connect OpenSSL

Connect openSSL.

Function	`int Lib_OpensslConnect(byte keyno,`
	`int port,`
	`byte[] host);`

Parameter	keyno (byte) [in] – Type for openssl certificate (1~4) 1: User for remote update the system OTA 2: User to remote load master key 3: User to remote load application 4: User to TMS
	port (int) [in] – port
	host (byte[]) [in] – IP address
Return Value	0: Success Other: Failed

18) OpenSSL Send

OpenSSL send data.

Function	`int Lib_OpensslSend(byte[] send,` `int inlen);`
Parameter	send (byte[]) [in] – Send data
	inlen (int) [in] – Data length (Less than 2048)
Return Value	0: Success Other: Failed

19) OpenSSL Recv

Openssl recv data.

Function	`int Lib_OpensslRecv(byte[] recv,`
	`int[] outLen,`
	`int timeout,`
	`int maxlen);`

Parameter	recv (byte[]) [out] – Data receive
	outLen (int[]) [out] – Data length received
	timeout (int) [in] – Time to wait. Unit is millisecond. (0~60 * 1000)
	maxlen (int) [in] – Max length to recv (0~2048)
Return Value	0: Success Other: Failed

20) Disconnect OpenSSL

Disconnect openssl.

Function	`int Lib_OpensslDisConnect();`
Parameter	void
Return Value	0: Success Other: Failed

21) Load Openssl Key / Certificate

Function	`int Lib_LoadTLSCert(byte keyno,`
	`byte type,`
	`int dLen,`
	`byte[] data);`

Parameter	keyno (byte) [in] – Type for openssl certificate (1~4) 1: User for remote update the system OTA 2: User to remote load master key 3: User to remote load application 4: User to TMS
	type (byte) [in] – Key / Certificate for Openssl (1~4) 1: ca.crt 2: client.crt 3: client.key 4: server.crt
	dLen (int) [in] – Key / Certificate length
	data (byte[]) [in] – Key / Certificate data buffer
Return Value	0: Success Other: Failed

- **PCI_ERROR_CODE**

PCI_KeyType_Err	(1)	
PCI_KeyLrc_Err	(2)	
PCI_KeyNo_Err	(3)	
PCI_KeyLen_Err	(4)	
PCI_KeyMode_Err	(5)	
PCI_InputLen_Err	(6)	
PCI_InputCancel_Err	(7)	
PCI_InputTimeOut_Err	(8)	
PCI_NoKey_Err	(9)	
PCI_WriteKey_Err	(10)	
PCI_ReadKey_Err	(11)	
PCI_DataLen_Err	(12)	
PCI_NoInput_Err	(13)	
PCI_ReadMMK_Err	(14)	
PCI_WriteMMK_Err	(15)	
PCI_EnDecrypt_Err	(16)	
PCI_KeyBoard_Err	(17)	
PCI_Counter_Err	(18)	120 transactions possible per 1 hour
PCI_ReadDukpt_Err	(19)	
PCI_WriteDukpt_Err	(20)	
PCI_IccChallenge_Err	(21)	
PCI_IccVerify_Err	(22)	
PCI_KeySame_Err	(23)	
PCI_Pwd_Err	(24)	
PCI_Param_Err	(-7001)	

Class PICC

- **Constructor**: Picc();
- **Methods**
  - Open
  - Close
  - Check
  - Command Exchange
  - M1 Card Authority
  - M1 Card Read Block
  - M1 Card Write Block
  - ReadT1T
  - WriteT1T
  - Read 15693
  - Write 15693
  - Read Felica
  - Write Felica
  - Init Felica State
  - Get Felica State
  - Read T2T
  - Write T2T
  - Init T2T State
  - Get T2T State
  - Check Emv

1) Open

Power on and reset contactless card module, check the status of module.

Function	`int Lib_PiccOpen();`
Parameter	void
Return Value	0: Success Other: Failed (Refer to Error code below)
Note	When device power on, the contactless module will be in close status default, before the transaction, this function will be called once. If this function not be called, other functions will be failed. This function will fail when the module is not installed or the module is fault.

2) Close

Close contactless card module, so that the module is turned off.

Function	`int Lib_PiccClose();`
Parameter	void
Return Value	0x00: Success Other: Failed (Refer to the error code below)
Note	After calling this function, contactless card module will be in a closed state, the module is no longer radiate the carrier And only the function Lib_PiccOpen () will be effective after call this function.

3) Check

Search the card according to the specified mode, select and active the card when searched.

Function	`int Lib_PiccCheck(byte mode,`
	`byte[] cardType,`
	`byte[] serialNo,`
	`byte[] ats);`

Parameter	mode (byte) [in] – 0
	cardType (byte[]) [out] – Card type byte buffer (2 bytes), cardType [0~1] “AC”: A-type card searched “BC”: B-type card searched “MC”: M-type card searched “A1”: T1T-type card searched “VC”: 15693-type card searched “A2”: T2T-type card searched “F2” / “F4”: Felica-type card searched
	serialNo (byte[]) [out] – Card serial number information. This information includes the length of the serial number and the content of serial number; serialNo[0]: length of the serial number serialNo[1~n]: To save the serial number (left justified)
	ats (byte[]) [out] ats[0]: length of the ats ats[1~n]: To save the ats (left justified)
Return Value	0: Success Other: Failed (Refer to error code below)

4) Command Exchange

Send APDU formatted data to the card, and receive a response from the card.

Function	`int Lib_PiccCommand(byte[] apduSend,` `byte[] apduResp);`
Parameter	apduSend (byte[]) [in] – Same with the ICC. (Lc max is 512, data length max is 512 bytes)
	apduResp (byte[]) [out] – Same with the ICC.
Return Value	0: Success Other: Failed (Refer to error code below)
Note	only Lib_PiccCheck() was called successfully, we can call this function; otherwise it will be failed.

5) M1 Card Authority

Verify the A or B password before access to the block.

Function	`int Lib_PiccM1Authority(byte type,`
	`byte blkNo,`
	`byte[] pwd,`
	`byte[] serialNo);`

Parameter	type (byte) [in] – Password type ‘A’ / ‘a’ / 0x0a: A password is submitted ‘B’ / ‘b’ / 0x0b: B password is submitted
	blkNo (byte) [in] – The block No. (0~63 for 1K capacity of M1 card)
	pwd (byte[]) [in] – Password
	serialNo (byte[]) [in] – The serial number was got from Lib_PiccCheck() return value: SerialNo + 1
Return Value	0: Success Other: Failed (Refer to error code below)
Note	Every four blocks compose to one sector, the last block of each sector is control block, which used to save A or B password and control information of each block; A or B password are all 6 bytes. The length of each block is 16 bytes, only after Lib_PiccCheck () was called successfully, this function can be called.

6) M1 Card Read Block

Read the contents of the specified block in M1 card (16 bytes).

Function	`int Lib_PiccM1ReadBlock(byte blkNo,` `byte[] blkValue);`
Parameter	blkNo (byte) [in] – The block No. (0~63 for 1K capacity of M1 card)
	blkValue (byte[]) [out] – The data in block. The size of the buffer should be 16 at least.
Return Value	0: Success Other: Failed (Refer to error code below)
Note	M1 card 's wallet is in a block consisting of a specific format, read the block to get the balance. The format is as follows: BALANCE [4] + ^ balance [4] + BALANCE [4] + BLK_NO + ^ blk_no + BLK_NO + ^ blk_no BALANCE [4] - 4-byte balance (low byte first), saved twice in the block ^ balance [4] –Anti-code of the balance. BLK_NO – the wallet's block number; the range is 0~63 for 1K capacity of M1 card.; stored in the block for twice ^ blk_no - Anti-code of the block number; stored in the block for twice

7) M1 Card Write Block

Write the specified contents to the specified block of M1 card (16 bytes).

Function	`int Lib_PiccM1WriteBlock(byte blkNo,` `byte[] blkValue);`
Parameter	blkNo (byte) [in] – The block No. (0~63 for 1K capacity of M1 card)
	blkValue (byte[]) [out] – The data from block. The size of the buffer should be 16 at least.
Return Value	0: Success Other: Failed (Refer to error code below)
Note	After authenticate successfully, then call this function to write wallet's initial value and other data in the specified block. When card personalization, we also use this function to update the control block;

8) Read T1T

Read the specified block of the T1T card.

Function	`int Lib_PiccReadT1T(byte blknum,`
	`byte[] outLen,`
	`byte[] dataOut);`

Parameter	blknum (byte) [in] – Block number
	outLen (byte[]) [out] – Block data length, 1 byte
	dataOut (byte[]) [out] – Block data
Return Value	0: Success Other: Failed (Refer to error code below)

9) Write T1T

Write to the specified block of the T1T card.

Function	`int Lib_PiccWriteT1T(byte blknum,`
	`byte inLen,`
	`byte[] dataIn,`
	`byte[] outLen,`
	`byte[] dataOut);`

Parameter	blknum (byte) [in] – Block number
	inLen (byte) [out] – Write block data length
	dataIn (byte[]) [out] – Block data
	outLen (byte[]) [out] – Data length return, 1 byte
	dataOut (byte[]) [out] – Data return from card
Return Value	0: Success Other: Failed (Refer to error code below)

10) Read 15693

Reads the specified block of the 15693 card.

Function	`int Lib_PiccRead15693(byte blknum,`
	`byte[] outLen,`
	`byte[] dataOut);`

Parameter	blknum (byte) [in] – Block number
	outLen (byte[]) [out] – Block data length, 1 byte
	dataOut (byte[]) [out] – Block data

Return Value	0: Success Other: Failed (Refer to error code below)

11) Write 15693

Write to the specified block of the 15693 card.

Function	`int Lib_PiccWrite15693(byte blknum,`
	`byte inLen,`
	`byte[] dataIn);`

Parameter	Blknum (byte) [in] – Block number
	inLen (byte) [out] – Write block data length
	dataIn (byte[]) [out] – Block data
Return Value	0: Success Other: Failed (Refer to error code below)

12) Read Felica

Read the Felica card.

Function	`int Lib_PiccReadFelica(byte[] outLen,` `byte[] dataOut);`
Parameter	outLen (byte[]) [out] – Block data length, 1 byte
	dataOut (byte[]) [out] – Block data
Return Value	0: Success Other: Failed (Refer to error code below)

13) Write Felica

Write to the Felica card.

Function	`int Lib_PiccWriteFelica(byte inLen,` `byte[] dataIn);`
Parameter	inLen (byte) [out] – Write block data length
	dataIn (byte[]) [out] – Block data
Return Value	0: Success Other: Failed (Refer to error code below)

14) Init Felica State

Initialize the Felica card status.

Function	`int Lib_PiccInitFelicaState();`
Parameter	Void
Return Value	0: Success Other: Failed (Refer to error code below)

15) Get Felica State

Get Felica card status.

Function	`int Lib_PiccGetFelicaState(byte[] state);`
Parameter	State (byte[]) [out] – Card state #define PHAL_TOP_STATE_NONE 0x00U /**<Default initial state. */ #define PHAL_TOP_STATE_INITIALIZED 0x01U /**< Initialized state. */ #define PHAL_TOP_STATE_READONLY 0x02U /**< Read Only state. */ #define PHAL_TOP_STATE_READWRITE 0x04U /**< Read/Write state. */
Return Value	0: Success Other: Failed (Refer to error code below)

16) Read T2T

Read the T2T card.

Function	`int Lib_PiccReadT2T(byte[] outLen,` `byte[] dataOut);`
Parameter	outLen (byte[]) [out] – Block data length, 1 byte
	dataOut (byte[]) [out] – Block data
Return Value	0: Success Other: Failed (Refer to error code below)

17) Write T2T

Write to the T2T card.

Function	`int Lib_PiccWriteT2T(byte inLen, byte[] dataIn);`
Parameter	inLen (byte) [in] – Write block data length
	dataIn (byte[]) [in] – Block data
Return Value	0: Success Other: Failed (Refer to error code below)

18) Init T2T State

Initialize the T2T card status.

Function	`int Lib_PiccInitT2TState();`
Parameter	void
Return Value	0: Success Other: Failed (Refer to error code below)

19) Get T2T State

Get T2T card status.

Function	`int Lib_PiccGetT2TState(byte[] state);`
Parameter	State (byte[]) [out] #define PHAL_TOP_STATE_NONE 0x00U /**< Default initial state. */ #define PHAL_TOP_STATE_INITIALIZED 0x01U /**< Initialized state. */ #define PHAL_TOP_STATE_READONLY 0x02U /**< Read Only state. */ #define PHAL_TOP_STATE_READWRITE 0x04U /**< Read/Write state. */
Return Value	0: Success Other: Failed (Refer to error code below)

20) Check Emv

Search the EMV mode card according to the specified mode, select and active the card when searched.

Function	`int Lib_PiccCheckEmv(byte mode,`
	`byte[] cardType,`
	`byte[] serialNo,`
	`byte[] ats);`

Parameter	mode (byte) [in] – 0
	cardType (byte[]) [out] – card type byte buffer (2 bytes), cardType[0~1] "AC": A-type card searched "BC": B-type card searched
	serialNo (byte[]) [out] – card serial number information. This information includes the length of the serial number and the content of serial number; serialNo[0]: Length of the serial number serialNo[1~n]: To save the serial number (left justified)
	ats (byte[]) [out] ats[0]: Length of the ats ats[1]: To save the ats (left justified)
Return Value	0: Success Other: Failed (Refer to error code below)

- **ERROR_CODE**

PICC_OK	(0)
PICC_ChipIDErr	(-3500)
PICC_OpenErr	(-3501)
PICC_NotOpen	(-3502)
PICC_ParameterErr	(-3503)
PICC_TxTimerOut	(-3504)
PICC_RxTimerOut	(-3505)
PICC_RxDataOver	(-3506)
PICC_TypeAColl	(-3507)
PICC_FifoOver	(-3508)
PICC_CRCErr	(-3509)
PICC_SOFErr	(-3510)
PICC_ParityErr	(-3511)
PICC_KeyFormatErr	(-3512)
PICC_RequestErr	(-3513)
PICC_AntiCollErr	(-3514)
PICC_UidCRCErr	(-3515)
PICC_SelectErr	(-3516)
PICC_RatsErr	(-3517)
PICC_AttribErr	(-3518)
PICC_HaltErr	(-3519)
PICC_OperateErr	(-3520)
PICC_WriteBlockErr	(-3521)
PICC_ReadBlockErr	(-3522)
PICC_AuthErr	(-3523)
PICC_ApduErr	(-3524)
PICC_HaveCard	(-3525)

PICC_Collision	(-3526)
PICC_CardTypeErr	(-3527)
PICC_CardStateErr	(-3528)
PICC_RxTimerOut2	(-3529)
PICC_RxErr	(-3530)
PICC_RxOverFlow	(-3531)
PICC_ProtocolErr	(-3532)
PICC_FastOut	(-3533)
PICC_Fsderror	(-3533)
PICC_CRCErr2	(-3534)
PICC_Continue	(-3535)
PICC_RxBlockErr	(-3536)
PICC_ApduErr1	(-3540)
PICC_ApduErr2	(-3541)
PICC_ApduErr3	(-3542)
PICC_ApduErr4	(-3543)
PICC_ApduErr5	(-3544)
PICC_ApduErr6	(-3545)
PICC_ApduErr7	(-3546)
PICC_ApduErr8	(-3547)
PICC_ApduErr9	(-3548)
PICC_ApduErr10	(-3549)
PICC_ApduErr11	(-3550)
PICC_ApduErr12	(-3551)
PICC_ApduErr13	(-3552)
PICC_ApduErr14	(-3553)
PICC_ApduErr15	(-3554)
PICC_ApduErr16	(-3555)

PICC_ApduErr17	(-3556)
PICC_ApduErr18	(-3557)
PICC_ApduErr19	(-3558)
PICC_ApduErr20	(-3559)
PICC_ApduErr21	(-3560)
PICC_ApduErr22	(-3561)
PICC_ApduErr23	(-3562)
PICC_ApduErr24	(-3563)

Class Printer

- **Constructor**: Print();
- **Methods**
  - Initialization
  - Set Space
  - Set Font
  - Get Font
  - Offset in Buffer
  - Send String Data to Print Buffer
  - Send Logo Dot Data to Print Buffer
  - Start Printing
  - Set the Left Margin
  - Set Gray
  - Check Print Status
  - Print Barcode
  - Print Bitmap

1) Initialization

Restore printer's default settings and clear the contents of the print buffer.

Function	`int Lib_PrnInit();`
Parameter	Void
Return Value	0: Success (-4007): No font library

2) Set Space

Set the line spacing and column spacing.

Function	`int Lib_PrnSetSpace(byte x,` `byte y);`
Parameter	x (byte) [in] – Column spacing [dots]. The default spacing is 0, maximum is 255
	y (byte) [in] – Line spacing [dots]. The default spacing is 0, maximum is 255.

Return Value	0: Success Other: failed
Note	The setting will be effective until the next call Lib_PrnInit();

3) Set Font

Set the print font and zoom parameters.

Function	`int Lib_PrnSetFont(AssetManager ass,`
	`String filename,`
	`byte asciiFontHeight,`
	`byte extendFontHeight,`
	`byte zoom);`

Parameter	ass (AssetManager) [in] – getAssets()
	filename (String) [in] – Font file name under assets. “BBFontUnicode.bin”
	asciiFontHeight (byte) [in] – ASCII character height: PRN_FONT_SMALL (8X16) PRN_FONT_BIG (12X24) - Default (Others are illegal values)
	extendFontHeight (byte) [in] – Extended character height PRN_FONT_SMALL (16X16) PRN_FONT_BIG (24X24) – Default (Others are illegal values)
	zoom (byte) [in] – Font zoom parameters. Default value is 0, which means no zoom in or zoom out. PRN_ASCII_X_ENLARGE: ASCII characters enlarge doubled in the X direction PRN_ASCII_Y_ENLARGE: ASCII characters enlarge doubled in the Y direction PRN_EXT_X_ENLARGE: Extended characters enlarge doubled in the X direction PRN_EXT_Y_ENLARGE: Extended characters enlarge doubled in the Y direction
Return Value	0: Success (-4009): Parameters error
Note	• The setting will be effective until the next call Prn_Init(); • the following macro defined in header file "Paymentapi.h": # define PRN_FONT_SMALL 16 # define PRN_FONT_BIG 24 # define PRN_ASCII_X_ENLARGE 0x01 # define PRN_ASCII_Y_ENLARGE 0x02 # define PRN_EXT_X_ENLARGE 0x10 # define PRN_EXT_Y_ENLARGE 0x20

4) Get Font

Get the current font and zoom parameters.

Function	`int Lib_PrnGetFont(byte asciiFontHeight[],`
	`byte extendFontHeight[],`
	`byte[] zoom);`

Parameter	asciiFontHeight (byte) [out] – ASCII character height: PRN_FONT_SMALL (8X16) PRN_FONT_BIG (12X24)
	extendFontHeight (byte) [out] – Extended character height PRN_FONT_SMALL (16X16) PRN_FONT_BIG (24X24)
	zoom (byte[]) [out] – Font zoom parameters. Default value is 0, which means no zoom in or zoom out. PRN_ASCII_X_ENLARGE: ASCII characters enlarge doubled in the X direction PRN_ASCII_Y_ENLARGE: ASCII characters enlarge doubled in the Y direction PRN_EXT_X_ENLARGE: Extended characters enlarge doubled in the X direction PRN_EXT_Y_ENLARGE: Extended characters enlarge doubled in the Y direction
Return Value	0: Success Other: Failed

5) Offset in Buffer

Move to the specified pixel in buffer.

Function	`int Lib_PrnStep(int pixel);`
Parameter	pixel (int) [in] – The number of pixels to be offset, which could be positive or negative.
Return Value	0: Success Other: failed
Note	When users print to note the following points: 1, the pixels to be offset can be positive or negative. Positive to move forward; negative to move backward. 2, assuming that there are already 100 pixels in buffer, then the legitimate range for “pixel” is [-100,4900], out of the range will no action;

6) Send String Data to Print Buffer

Send the data to be printed to print buffer.
(The system will automatically convert the string to dot matrix data and stored within the print buffer.)

Function	`int Lib_PrnStr(String strInUTF8);`
Parameter	strInUTF8 – The data to be printed
Return Value	0: Success (-4008): Buffer overflow
Note	1) Support variable parameters, please refer to printf () function of standard C; 2) Support '\ n' [line], '\ f' [feed] control characters in the buffer; 3) Wrap automatically; 4) Maximum 2047 bytes for once. 5) Not support the paramenters with "% f" format, otherwise it will cause system crashes, which is due to arm-elf-gcc compiler.

7) Send Logo Dot Data to Print Buffer

Send the logo dot data to the print buffer.

Function	`int Lib_PrnLogo(byte logo[]);`
Parameter	logo (byte) [in] – The logo data to be printed
Return Value	0: Success (-4003): Logo size error (-4004): Decompression error (-4008): Buffer overflow (-4012): Printer temperature is too low
Note	The logo to be printed must be in BMP format, the maximum size is 384*400.

8) Start Printing

Printer starting to print.

Function	`int Lib_PrnStart();`
Parameter	void
Return Value	0: Success (-4001): Printer is busy (-4002): Printer is lack of paper (-4003): Printer data format error (-4004): Printer is broken (-4005): Printer is too hot (-4006): Print is unfinished (-4007): No print font library (-4008): Print buffer overflow (-4011): Low battery (-4012): Printer temperature is too low Other: Other errors
Note	• After calling this function, it will return until completed • After finished, this function will return the state of printer • After calling this function, the print buffer will be cleared

9) Set the Left Margin

Set the left margin printing characters.

Function	`int Lib_PrnSetLeftIndent(int x);`
Parameter	x (int) [in] – Point blank left margin (0~336)
Return Value	0: Success Other: Failed
Note	The default boundary is 0

10) Set Gray

Set gray.

Function	`int Lib_PrnSetGray(byte nLevel);`
Parameter	nLevel (byte) [in] – 1~6. >= 6: nLevel = 6 Default: 3
Return Value	0: Success Other: Failed

11) Check Print Status

Check the current status of printer.

Function	`int Lib_PrnCheckStatus();`
Parameter	void
Return Value	Refer to Lib_PrnStart's return value

12) Print Barcode

Function	`int Lib_PrnBarcode(String contents,`
	`int desiredWidth,`
	`int desiredHeight,`
	`BarcodeFormat barcodeFormat);`

Parameter	contents (string) [in] – The contents to be printed
	desiredWidth (int) [in] – Barcode width
	desiredHeight (int) [in] – Barcode height
	barcodeFormat (BarcodeFormat) [in] – Support format include CODE_128 CODE_39 EAN_8 EAN_13 CODABAR UPC_A ITF QR_CODE PDF_417 AZTEC
Return Value	0: Success (-4001): Printer is busy (-4002): Printer is lack of paper (-4003): Printer data format error (-4004): Printer is broken (-4005): Printer is too hot (-4006): Print is unfinished (-4007): No print font library (-4008): Print buffer overflow (-4012): Printer temperature is too low Other: Other errors
Note	For different format, the contents to be printed should follow its rules.

13) Print Bitmap

Print bitmap.

Function	`int Lib_PrnBmp(Bitmap bitmap);`
Parameter	bitmap (Bitmap) [in] – BMP format image file
Return Value	0: success Others: failed
Note	BMP picture size 384*500, must be a unit color map