Althico Lite-M API Programming Guide

V 0. 0. 3

HISTORY LIST

DATE	Version	Comments	Author
2023-03-19	V0.0. 1	Initial Version	Alex
2023-03-20	V0.0.2	Correct incorrect descriptions	Alex
2023-03-20	V0.0.3	1. Fixed print api description 2. Delete unwanted tables	Alex

directory

1 Basic system

1.1 Basic definition

System basic functions are functions used to control the main system of POS, such as clock, buzzer, product version management, and other general control functions.

1.1.1 Data definition

Macro definition:

```
#define    u8    unsigned char
#define    u16   unsigned short
#define    u32      unsigned int
```

1.2 LiteMapiInit

Prototype	**void LiteMapiInit(void);**	
Function	The Lite-M API library is initialized and called once before using the Lite-M API.	
Parameter	NULL	
return	NULL	
usage		

1.3 Beep

Prototype	**void Beep(void);**	
function	The buzzer immediately emits a "beep" for a duration of 100ms.	
parameter	NULL	
return	NULL	
usage	Hints for some terminal actions and event triggering.	

Example:

```
Beep();                     /* */
DelayMs(20);                /* A pause of 20ms */
Beep();                     /* */
```

1.4 BeepF

Prototype	**void BeepF(u8 mode, u16 DlyTime);**	
function	The buzzer sounds at the frequency and duration specified by the parameter. Abort Bee pF after pressing a key (no longer responds to keys when the key cache is full) and exit the function (except for the power on and off keys).	
parameter	mode [input].	Frequency setting, mode%7 can be a value of 0~6: 0 is the lowest frequency 6 is the highest frequency
	DlyTime [input]	Continuous vocalization time (unit: ms).
return	NULL	
usage	If the mode value is greater than 6, the function uses mode%7 as the sound frequency.	

1.5 SevenTime

Prototype	**u8 SetTime(u8 *time);**	
function	Set the date and time of the system, and the day of the week will be automatically calculated and set.	
parameter	time[input].	The pointer of the date time parameter, in the format YYMMDDhhmmss, the parameter is BCD code, a total of 6 bytes long. (Valid time range: year (19) 50 ~ (20) 49 ,

		month 1 ~ 12, day 1 ~ 31,.) Hours 0~24, minutes and seconds 0~59).
return	0 Non-0	Setup successful The datetime value is illegal
usage	When the return value is non-zero, the following value can be used to track the error: 1 The year in the parameter is illegal→ 2 Month in argument illegal→ 3 The date in the argument is illegal→ 4 Parameters in hours illegal→ 5 Minutes in parameter illegal→ 6 seconds in the argument is illegal→ 0xff Error reading and writing clock chip→ The error return value is prioritized as year, month, day, hour, minute, and second, that is, if the year is wrong, the month is also wrong, and only 1 is returned.	

1.6 GetTime

Prototype	**void GetTime(u8 *time);**	
function	Read the terminal date and time.	
parameter	Time[output]	A pointer to the datetime value, stored in the form of a BCD code: time[0] year→ time[1] month→ time[2] day→ time[3] hours→ time[4] minutes→ time[5] seconds→ time[6] week→
return	NULL	
usage	Definition of week value: 1 Monday, 2 Tuesday→→, ... 7 Sunday. →	

1.7 DelayMs

Prototype	**void DelayMs(u16 Ms);**	
function	Delay milliseconds.	
parameter	M s-milliseconds [input].	
return	NULL	

usage	The accuracy is 10ms class

1.8 TimerSet

Prototype	**void TimerSet(u8 TimerNo, u16 Cnts);**	
function	Starts a user-specified timer with a minimum timing unit of 100 milliseconds. Timer timing time = CNTS x 100 milliseconds.	
parameter	TimerNo [input]	User-specified timer number, valid value: 0~4
	CNTS [input]	The number of 100 milliseconds of the timing time, effective value: 1~65535
return	NULL	
usage		

CAUTION ⚠

For illegal TimerNo numbers, the system will call this function repeatedly with the TimerNo%5 set timer number, and the system will call this function with the last time takes precedence.

1.9 TimerCheck

Prototype	**u16 TimerCheck(u8 TimerNo);**	
function	Detects the current value of a specified timer (number of 100 milliseconds).	
parameter	TimerNo [input]	The timer number to be detected, effective value: 0~4
return	Time remaining on the timer	(Unit: 100ms)
usage	For illegal TimerNo numbers, the system will set the timer number with TimerNo%5. The remaining time is 0, indicating that the timer time has arrived. If the timer is NULL used, calling TimerCheck returns the default value of 0.	

Example:

```
TimerSet(0,100);

while(1)

{

if(kbhit()==0xff) break;                /* Exit    */ if no key is pressed
if(! TimerCheck(0)) break;            /* If there is a key press, the timer exits */
}
```

1.10 GetTimerCount

Prototype	**uintGetTimerCount(void);**
function	Gets the current timescale.
parameter	NULL
return	Returns the current timescale (unit: 1ms).
usage	Used to implement timing functions. For example, call this function to obtain the timescale T1, and then call this function to obtain the timescale T2, T2-T1 is the specific timing time.

1.11 ReadSN

Prototype	**voidReadSN(u8 *SerialNo);**	
function	Read the terminal serial number.	
parameter	SerialNo [output]	The buffer address used to hold the product serial number requires 32 bytes of space to be pre-allocated.
return	NULL	
usage	Returned serial number, string ending with '\0' (up to 32 bytes). If SerialNo[0]='\0', it is a serial number termination. Added support for 10-bit SN, all digital.	

NOTE

SN is the only distribution by the manufacturer when the product leaves the factory.

1.12 ReadVerInfo

Prototype	**u8ReadVerInfo(u8 *VerInfo);**	
function	Read the version information of the terminal.	
parameter	VerInfo [output]	Version information (8 bytes) VerInfo[0] BOOT major version number (incremented starting with 1).

		VerInfo[1]	BOOT minor version number (incremented by 1).
		VerInfo[2]	Monitor major version numbers (increment starting from 1)
		VerInfo[3]	Monitor minor version number 1 (0 starts incrementing).
		VerInfo[4]	Monitor minor version number 2 (incrementing from 0).
		VerInfo[5]	retain
return	Always 0	Other return values are retained	
usage	Please refer to the section on terminal build management. The version number is represented by a hexadecimal value.		

1.13 GetTermInfo

Prototype	**int GetTermInfo(u8 *out_info);**		
function	To read the terminal model and configuration information, the information buffer area should NULL be less than 30 bytes.		
parameter	out_info [output].	out_info[0]	Terminal model [0x00~0xFF]. 0xA5—A50
		out_info[1]	Printer type 0 - No printer 'S' - dot matrix printer 'T' - Thermal printer
		out_info[2]	Modem module configuration information 0 - NULL applicable to the Modem communication module Other-Modem module model code: 0x01-TDK 73K222 1200/1200 0x02-TDK 73K224 2400/2400 0x10—Silicon 2414 14.4k/2400 0x20—conexant 81802 33.6k/9600 0x40—conexant 93001-V92 56k/9600 0x41—conexant 93001-V32B 14.4K/9600 0x80-Zilog comes with a Modem 14.4K/2400
		out_info[3]	MODEM maximum synchronization rate information 1—1200 2—2400 3—9600 4—14400

		out_info[4]	MODEM maximum asynchronous rate information 1—1200 2—2400 3—4800 4—7200 5—9600 6—12000 7—14400 8—19200 9—24000 10—26400 11—28800 12—31200 13—33600 14—48000 15—56000
		out_info[5]	PCI configuration information 0 - No built-in PCI security module Other - PCI-compliant security module
		out_info[6]	USB host configuration information 0 - No USB host api Other-USB host api version code (if applicable USB1.1, USB2.0, USB-OTG, etc.)
		out_info[7]	USB device configuration information 0 - No from USB api Other - USB device api version code
		out_info[8]	LAN (TCP/IP) module configuration information 0 - No TCP/IP module Other - TCP/IP module version encoding
		out_info[9]	GPRS module configuration information 0 - No GPRS module Other - GPRS module version encoding
		out_info[10]	CDMA module configuration information 0 - No CDMA module Other - DMA module version code
		out_info[11]	WI-FI MODULE CONFIGURATION INFORMATION 0- NO WI-FI MODULE OTHER-WI-FI MODULE VERSION ENCODING
		out_info[12]	Contactless card reader module configuration information

			0 - No contactless card reader module Other - Contactless card reader module version coding.
		out_info[13]	Whether there is a Chinese font library 0 - Non-Chinese font 1-Chinese font library
		out_info[14]	Font version information 0 - No font file Other - Font version number
		out_info[15]	IC reader module configuration information 0x00 indicates that there is no IC card reader Non-0 indicates that there is an IC card reader
		out_info[16]	Magnetic card reader module 0x00 indicates that there is no magnetic card reader module Non-0 indicates that there is a magnetic card reader module
		out_info[17]	Whether or NULL there is a Tilt Sensor 0一无 Tilt Sensor 1　　一有 Tilt Sensor
		out_info[18]	WCDMA module configuration information 0 - No WCDMA module Other - WCDMA module version code
		out_info[19]	Bit0: Is there a touchscreen 0 - no touchscreen; 1- There is a touchscreen. Bit1: Whether there is a color screen 0 - no color screen; 1- There is a color screen. Bit2：reserve。 Bit3: Whether there is a Bluetooth module 0 - No Bluetooth module; 1- There is a Bluetooth module. Bit4: Whether the national secret algorithm is supported 0 - NULL supported; 1- Support.

		out_info[20]	4G module configuration information 0 - No 4G module Other - 4G module version coding
		out_info[21]	retain
return	Returns the effective byte length of terminal information.		
usage			

1.14 Reboot

Prototype	**int Reboot();**	
function	Reset the terminal.	
parameter	NULL	
return	NULL	
usage	For example, when the remote download is completed, the machine needs to be automatically restarted.	

1.15 GetTermInfoExt

Prototype	**int GetTermInfoExt (char *keyword,char *attr,u32 attrLen)**	
function	Get terminal extension information.	
parameter	aTTR [output].	attr: Output buffer
		attrLen: Output buffer length
	Keyword [type].	The name of the tag to be obtained
return	>=0 The actual length of the information in InfoOut -1 The output buffer (InfoOut) is NULL long enough -2 Other errors	
usage	A list of tag names #define GPRS_RSSI "GPRS_RSSI" #define GPRS_MCC "GPRS_MCC" #define GPRS_MNC "GPRS_MNC" #define GPRS_LAC "GPRS_LAC" #define GPRS_CI "GPRS_CI" #define GPRS_CID "GPRS_CID" #define GPRS_IMEI "GPRS_IMEI" #define GPRS_SN "GPRS_SN" #define GPRS_IMSI "GPRS_IMSI" #define USB_STA_NAME "USB_STA" #define LCD_W_NAME "LCD_W"	

	#define LCD_H_NAME	"LCD_H"
	#define LCD_RES_NAME	"LCD_RES"

1.16 PciGetRandom

Prototype	**void PciGetRandom(u8 *random);**	
function	The system generates a true random number of 8 bytes.	
parameter	random [output].	Buffer for output random numbers, 8 bytes
return	NULL	
usage		

1.17 SysSleep

Prototype	**int SysSleep(u8 *DownCtrl);**	
function	Put the terminal to sleep to reduce power consumption.	
parameter	DownCtrl [input]	DownCtrl - Input parameter pointing to the power-down control string (ending with '\0'). DownCtrl[0]—Sets whether the IC card powers down for the power-up state: '0' - no power-down, '1' - power-down. DownCtrl[1] - Sets whether the contactless card is powered down to the power-on state: '0' - no power-down, '1' - power-down. If the input parameter is NULL, the default is to power down, which is equivalent to input '11'.
return	1	Hard decode magnetic card swipe wake
	2	IC card plug-in card wake-up
	3	Press to wake
	4	RTC wake-up
	5	Bluetooth wake
	-1	Invalid parameter
	-2	The machine is locked (cannot hibernate).
	-3	Call too often (call the function again within two seconds of the key wake-up)
	-4	Invalid wakes (such as wakes caused by a key popping).
usage	In the sleep state, the CPU stops running, the LCD of the black and white screen does NULL turn off, the color screen defaults to the backlight completely turned off, (some models can be set by SysConfig() to NULL completely turn off the backlight, see SysConfig for specific usage.) ()) to keep the screen before calling the function; It can be woken up by pressing buttons, IC cards, etc., and after waking up, the	

	system continues to run from the stop breakpoint before sleeping, at the same time All application variables in memory remain unchanged before hibernation. Since the key takes a certain amount of time, this function deliberately adds two seconds of buffer time for the wake-up action. That is, if the function is called within two seconds after the key wakes up, it returns a failure (-2) instead of sleeping, lest the application fail to detect the key and fall back to sleep. A non-negative return code is used to indicate the reason for being woken up, and a negative return code indicates the cause of the error.

1.18 SysIdle

Prototype	**void SysIdle(void);**	
function	This function is used to reduce power consumption caused by the operation of the processor core.	
parameter	NULL	
return	NULL	
usage		

1.19 BatteryCheck

Prototype	**u8 BatteryCheck(void);**	
function	Read the system voltage level.	
parameter	NULL	
return	0	The small battery icon flashes to indicate that the battery voltage is low. It is recommended that this state do NULL trade, print, wireless communication, etc., should be charged in time, so as to avoid data loss due to low voltage shutdown.
	1	The small battery icon shows 1 tile
	2	The small battery icon shows 2 tiles
	3	The small battery icon shows 3 tiles
	4	The small battery icon shows 4 tiles
	5	Indicates that there is an external power supply and the battery is in a charging state.

		The battery icon is now cycling from space to full and the battery status indicator (if available) on the bottom of the machine is displayed in red.
	6	Indicates that there is an external power supply and the battery has been charged. The battery is now displayed as full and the battery status indicator (if available) on the bottom of the machine is green.
usage		

1.20 PowerOff

Prototype	**void PowerOff(void);**	
function	Power off the terminal.	
parameter	NULL	
return	NULL	
usage		

1.21 GetEnv

Prototype	**u8 GetEnv(char*filename, char *name,** **u8 *value);**	
function	Read an environment variable.	
parameter	Filename [input].	filename
	name	Parameter names, which can be up to 7bytes, names over 7bytes Only the first 7bytes are taken.
	value [output].	parameter value, space allocated by the application, can be up to 119 words A string of section lengths. End with 0x00.
return	0	succeed
	1	The parameter name was NULL found
usage		

1.22 PutEnv

Prototype	**u8 PutEnv(char*filename, char *name, u8 *value);**	
function	Write an environment variable.	
parameter	Filename [input].	filename
	name	Parameter names, which can be up to 7bytes, names over 7bytes Only the first 7bytes are taken.
	value [input].	Parameter values, which can be up to 119 bytes and more than 119bytes Only the first 119bytes are taken.
return	0	Indicates success
	1	Indicates that the parameter is illegal
	2	Indicates insufficient space
usage	Overwrite if the environment variable written already exists. The value of the parameter name is NULL allowed to be NULL or "". The value of Value is also NULL allowed to be NULL.	

1.23 FileToApp

Prototype	**int FileToApp(u8 *FileName);**	
function	Update the app	
parameter	Filename [input].	The app file name to update
return	0	Indicates that the update was successful
	Other negative values	Indicates that the update failed
usage	Prompt that after the app update is complete, it will automatically shut down and restart	

1.24 SysReset

Prototype	**int SysRest(void);**	
function	Factory reset the system	
parameter		
return	0	Indicates that the recovery was successful
	Other negative values	Indicates failure

usage	This rebuilds the file system, rebuilds the key security module, and untriggers the TAMPER

2 Arithmetic

2.1 Basic definition

2.1.1 Returns a list of values

macro	numeric value	illustrate
RSA_KEY_RET_ERR_BIT	-1	The parameter iProtoKeyBit is wrong
RSA_KEY_RET_ERR_PUKE	-2	The parameter iPubEType is wrong
RSA_KEY_RET_ERR_DATA	-3	Failed to produce data
RSA_KEY_RET_ERR_RANDOM	-4	The data fails randomly
RSA_KEY_RET_ERR_ENCRY	-5	The encryption operation failed
RSA_KEY_RET_ERR_DECRY	-6	The decryption operation failed
RSA_KEY_RET_ERR_VERIF	-7	Encryption and decryption verification failed

2.1.2 SHA_TYPE table of values

macro	numeric value	illustrate
SHA_TYPE_1	0	SHA-1
SHA_TYPE_224	1	SHA-224
SHA_TYPE_256	2	SHA-256
SHA_TYPE_384	3	SHA-384
SHA_TYPE_512	4	SHA-512

2.1.3 Data definition

RSA Key Structure:

Public key structure

```
typedef struct {
        unsigned short int bits;                              /* length in bits of modulus */
        unsigned char modulus[MAX_RSA_MODULUS_LEN];          /* modulus */
        unsigned char exponent[MAX_RSA_MODULUS_LEN];        /* public exponent */
} R_RSA_PUBLIC_KEY;
```

Private key structure

```
typedef struct {
        unsigned short int bits;                              /* length in bits of modulus */
        unsigned char modulus[MAX_RSA_MODULUS_LEN];          /* modulus */
        unsigned char publicExponent[MAX_RSA_MODULUS_LEN];    /* public exponent */
        unsigned char exponent[MAX_RSA_MODULUS_LEN];              /* private exponent */
        unsigned char prime[2][MAX_RSA_PRIME_LEN];          /* prime factors */
        unsigned char primeExponent[2][MAX_RSA_PRIME_LEN];       /* exponents for CRT
*/
        unsigned char coefficient[MAX_RSA_PRIME_LEN];    /* CRT coefficient */
} R_RSA_PRIVATE_KEY;
```

2.2 Des

Prototype	**void des(u8 *input, u8 *output, u8 *deskey, int mode);**	
function	Perform DES encryption and decryption operations on 8 bytes of data.	
parameter	input [input].	8 bytes of input data
	output	8 bytes of output data
	deskey [input].	8-byte DES key
	mode [input].	0 - decryption; 1- Encryption.
return	NULL	

usage	This function performs encryption or decryption operations according to the mode selection.

2.3 Hash

Prototype	**void Hash(u8* DataIn,** **uint DataInLen,** **u8* DataOut);**	
function	For input data of any length, output a hash result of 20 bytes.	
parameter	DataIn [input]	Input data buffer pointer
	DataInLen [input]	Input data length in bytes
	DataOut [output]	Output data buffer pointer [output, 20 bytes in size].
return	NULL	
usage	The function uses the hash algorithm SHA-1 (according to ISO/IEC 10118-3 or FIPS 180-1).	

2.4 RSARecover

Prototype	**int RSARecover(u8 *pbyModule,** **uint dwModuleLen,** **u8 *pbyExp,** **uint dwExpLen,** **u8 *pbyDataIn,** **u8 *pbyDataOut);**	
function	Perform RSA encryption and decryption operations.	
parameter	pbyModule [input]	The modulo buffer pointer that holds the RSA operation (i.e. n=p*q). Store in the order of high first, low last.
	dwModuleLen [input]	Modulo byte length (value range 1~256).
	pbyExp [input]	Holds the exponential buffer pointer for RSA operations. That is, e. stored in the order of high position first and low bit last.
	dwExpLen [input]	Exponential byte length.
	pbyDataIn [input]	Input data buffer pointer, the same length as the modulo length.
	pbyDataOut [output]	Output data buffer pointer, the same length as the modulo length.
return	0	Indicates success.
	-1	Indicates that the input parameter is wrong, the pointer to the input parameter is empty, error.

	-2	Indicates that the length of the input module dwModuleLen or the exponential length dwExpLen is 0, error.
	-3	The length of the denote modulo dwModuleLen or the exponential length dwExpLen is too large, wrong.
	-4	Indicates that pbyDataIn is greater than or equal to pbyModule, error.
	-5	Indicates that dwExpLen is greater than dwModuleLen, error.
usage	1. This function performs RSA encryption and decryption operations on pbyDataIn, and outputs the operation result to pbyDataOut. 2. When (pbyModule, pby Exp) is the private key, if pbyDataIn is the cryptographic ciphertext corresponding to the public key, then pbyDataOut is the plaintext of pbyDataIn, otherwise pbyDataOut is the RSA ciphertext of pbyDataIn; 3. When (pbyModule, pby Exp) is the public key, if pbyDataIn is the cryptographic ciphertext corresponding to the private key, then pbyDataOut is the plaintext of pbyDataIn, otherwise pbyDataOut is the RSA ciphertext of pbyDataIn; 4. This function enables RSA operations up to 2048 bits in length.	

2.5 RSAKeyPairGen

Prototype	**int RSAKeyPairGen(R_RSA_PUBLIC_KEY *pPublicKeyOut,** **R_RSA_PRIVATE_KEY *pPrivateKeyOut,** **int iProtoKeyBit,** **int iPubEType);**		
function	Randomly generate 512, 1024, 2048-bit RSA public-private key pairs.		
parameter	pPublicKeyOut [output].	A pointer to the public key structure, which is referenced in this section Data definition	
	pPrivateKeyOut	A pointer to the private key structure, which is referenced in this section Data definition	
	iProtoKeyBit[input]	Modulo length of digits, supported: 512 1024 2048	
	iPubEType [input]	Index type: 0:0x00, 0x00, 0x00, 0x03 1：0x00,0x01,0x00,0x01	
return	#define RSA_KEY_RET_ERR_BIT	-1	The parameter iProtoKeyBit is wrong
	#define RSA_KEY_RET_ERR_PUKE	-2	The parameter iPubEType is wrong
	#define RSA_KEY_RET_ERR_DATA	-3	Failed to produce data
	#define RSA_KEY_RET_ERR_RANDOM	-4	The data fails randomly
	#define RSA_KEY_RET_ERR_PARA	-5	Parameter error with empty pointer

usage	Use this api to randomly generate a pair of RSA public-private keys with a specified exponential and modular length. The data in the key structure is a small tail. When using RSARecover, use data such as long copy mode and index. For example, copy the private key model data: memcpy(PriModulus, pvk.modulus + MAX_RSA_MODULUS_LEN -ModulLen, ModulLen);

2.6 RSAKeyPairVerify

Prototype	**intRSAKeyPairVerify(R_RSA_PUBLIC_KEY PublicKey, R_RSA_PRIVATE_KEY PrivateKey);**	
function	Verify 512, 1024, 2048-bit RSA public-private key pairs.	
parameter	PublicKey [input]	The structure of the public key needs to be verified, and the public key structure is described in this section Data definition
	PrivateKey [input]	The structure of the private key needs to be verified, and the private key structure is described in this section Data definition
return	#define RSA_KEY_RET_ERR_ENCRY-5	The encryption operation failed
	#define RSA_KEY_RET_ERR_DECRY-6	The decryption operation failed
	#define RSA_KEY_RET_ERR_VERIF-7	Encryption and decryption verification failed
usage	Put the generated key in, and the function randomly generates data for private key encryption and public key decryption operations to verify that the keys are a pair. 2048bit operations are generally slower.	

2.7 SHA

Prototype	**intSHA(int Mode, const unsigned char *DataIn, int DataInLen, unsigned char *ShaOut);**	
function	Calculate a safe hash value.	
parameter	Mode	SHA_TYPE_1SHA_TYPE_224SHA_TYPE_256SHA_TYPE_384SHA_TYPE_512
	DataIn【输入】	Enter the data
	DataInLen【输入】	Enter the data length
	ShaOut [Output].	The output value of SHA

return	0 -2	Indicates success Mode error
usage	Calculate the hash value of the SHA series.	

2.8 AES

Prototype	**int AES(const unsigned char *Input, unsigned char *Output, const unsigned char *AesKey, int KeyLen, int Mode);**	
function	Perform AES encryption and decryption operations.	
parameter	Input	16 bytes of input data
	Output [output].	16 bytes of output data
	AesKey [input].	AES key
	KeyLen	16, 24, or 32 (bytes).
	Mode	0 - decryption; 1- Encryption
return	0 -2 -3	Indicates success The key length is wrong Mode error
usage	This function supports 128-, 192-, or 256-bit AES encryption and decryption operations. When the parameter is invalid, no action will be taken.	

3 keyboard

3.1 Basic definition

3.2 kbhit

Prototype	**u8 kbhit(void);**	
function	Detects if there are pressed key values in the keyboard buffer that have NULL yet been taken.	
parameter	NULL	
return	0xFF 0x00	The buffer has no key value. The buffer has key values.
usage	The keyboard has a 32-byte key buffer. If the buffer has a key value, it can be read out by the getkey() function.	

3.3 kbflush

Prototype	**void kbflush(void);**	
function	Clears all unread keystrokes in the current keyboard buffer.	
parameter	NULL	
return	NULL	
usage	Use this function to empty the buffer, and then call kbhit to determine whether there is a key press event.	

3.4 getkey

Prototype	**u8 getkey(void);**		
function	Reads the first key value entered in the keyboard buffer. If the buffer is empty, wait for key input.		
parameter	NULL	The keyboard has a 32-byte key buffer, and when the buffer is full, the key is discarded.	
return	The key value read		
usage	The following is the definition of the key value:		
	‘1’:	KEY1	0x31
	‘2’:	KEY2	0x32
	‘3’:	KEY3	0x33
	‘4’:	KEY4	0x34
	‘5’:	KEY5	0x35
	‘6’:	KEY6	0x36
	‘7’:	KEY7	0x37
	‘8’:	KEY8	0x38
	‘9’:	KEY9	0x39
	‘0’:	KEY0	0x30
	‘▲’ :	KEYUP	0x05
	‘▼’ :	KEYDOWN	0x06
	'Cancel key':	KEYCANCEL	0x1B
	'Clear key':	KEYCLEAR	0x08
	'Confirm key':	KEYENTER	0x0D

3.5 kbmute

Prototype	**void kbmute(u8 flag);**	
function	Sets whether the keypad sounds when presses are pressed.	
parameter	flag [input]	0: No sound Non-0: vocalization
return	NULL	
usage	Set to 0, you can enter the mute state without generating a key sound.	

4Color screen

4.1 Basic definition

The api of the color screen module will adopt a dual-layer display scheme, with the background layer being the canvas and the foreground layer being transparent glass; The foreground layer is used to show the main text or shape (point, line, box), the background layer is used to display pictures or color blocks, and the canvas content of the background layer can be seen through the undrawn area of the foreground layer. Among them, CLcdSetBgColor, CLcdBgDrawBox, CLcdBgDrawImg, CLcdBgDrawGif, CLcdBgClrGif apis all act on the background layer of the screen, respectively, rendering the background layer of the whole layer of color, drawing a solid rectangle, drawing a background picture, and playing and clearing GIF pictures in the background layer; Other color screen apis are used for drawing and clearing foreground text, points, and rectangular boxes. CLcdPrint and CLcdTextOut, the API apis for drawing text, will output their strokes in the foreground layer of the screen according to the dot-to-pixel correspondence of the dot matrix of the character model, and the other spaces other than the strokes in the font mold are transparent.

4.1.1 Data definition

Structure purpose name:

Color screen information structure

```
typedef struct{
        unsigned int width;/* User usable area width (pixels) */
        unsigned int height;                    /* User Availability Area Height (pixels) */
        unsigned int ppl;                        /* Number of pixels occupied by a character line */
        unsigned int ppc;                        /* Number of pixels occupied by a character column
*/
        unsigned int fgColor;                    /* Foreground color */
        unsigned int bgColor;                   /* Screen background color */
        intreserved[8];/* reserved */
}ST_LCD_INFO;
```

NOTE

PPL represents the height of a character (pixels), and PPC represents the width of a character (pixels).

4.2 CLcdGetInfo

Prototype	**intCLcdGetInfo (ST_LCD_INFO *pstLcdInfo);**	
function	Get color screen information.	
parameter	pstLcdInfo [output].	Color screen information
return	0	succeed
	-1	The incoming pointer is empty
usage	Please refer to the definition of the color screen information structure and related descriptions.	

4.3 CLcdSetFgColor

Prototype	**intCLcdSetFgColor(uint color);**	
function	Sets the color of the text output from the foreground layer of the screen.	
parameter	color [input].	The color value of the text. The value consists of a 32-bit orthomorphic value in the format 0xFFRRGGBB where: BIT0 ~ BIT7 is the blue component; BIT8 ~ BIT15 is the green component; BIT16 ~ BIT23 is the red component; BIT24 ~ BIT31 is reserved bits, this value may be used for transparency in the future, so be sure to 0xFF.
return	The color value passed in the last time the app called the api.	
usage	1. The set text color is valid for text output in both color screen mode and black and white screen mode;	

	2. The text color you set is only valid for text that is output on subsequent screens, NULL for text that has been previously exported to the screen.	

4.4 CLcdSetBgColor

Prototype	**int CLcdSetBgColor (uint color);**	
function	Sets the background color of the screen.	
parameter	color [input].	The color value of the screen background color. The value consists of a 32-bit orthomorphic value in the format 0xFFRRGGBB where: BIT0 ~ BIT7 is the blue component; BIT8 ~ BIT15 is the green component; BIT16 ~ BIT23 is the red component; BIT24 ~ BIT31 is reserved bits, this value may be used for transparency in the future, so be sure to 0xFF.
return	The color value passed in the last time the app called the api	
usage	This function is used to change the color of the screen background immediately after the call, the function only changes the background color of the screen without changing the original foreground display content.	

4.5 CLcdBgDrawBox

Prototype	**int CLcdBgDrawBox (uint left,** **uint top,** **uint right,** **uint bottom,** **uint color);**	
function	Draws a solid rectangular patch of color in the background layer with the specified color.	
parameter	left [input].	The left boundary value
	top [input].	The upper boundary value
	right [input].	The right boundary value
	bottom [input].	Lower bounded value
	color [input].	Render color values. The value consists of 32-bit positive integer values, where: BIT0 ~ BIT7 is the blue component; BIT8 ~ BIT15 is the green component; BIT16 ~ BIT23 is the red component; BIT24 ~ BIT31 is reserved bits, this value may be used for transparency in the future, so be sure to 0xFF.

return	0	succeed
	-1	The specified region is NULL valid
usage	The display effect is a solid filled rectangular patch of color. If the coordinate value exceeds the screen range or the left coordinate is greater than the right coordinate or the top coordinate is greater than the bottom coordinate, it is an illegal parameter and no rectangle is drawn.	

4.6 CLcdDrawPixel

Prototype	**int CLcdDrawPixel (uint x,** **uint y,** **uint color);**	
function	Draw a dot at the specified location on the screen.	
parameter	x [input].	X coordinate value
	y [input].	Y coordinate value
	color [input].	The color value of the drawn point. The value consists of a 32-bit orthomorphic value in the format 0xFFRRGGBB where: BIT0 ~ BIT7 is the blue component; BIT8 ~ BIT15 is the green component; BIT16 ~ BIT23 is the red component; BIT24 ~ BIT31 is reserved bits, this value may be used for transparency in the future, so be sure to 0xFF.
return	0	Draw a point of success
	-1	Invalid parameter (invalid coordinates specified)
usage	This api operates on the foreground layer, but the color of the dots is reflected directly on the screen.	

4.7 CLcdDrawRect

Prototype	**int CLcdDrawRect (uint left,** **uint top,** **uint right,** **uint bottom,** **uint color);**	
function	Draw a rectangular box in the foreground of the screen.	
parameter	left [input].	The left boundary value
	Top [input]	The upper boundary value
	right [input].	The right boundary value

	bottom [input].	Lower bounded value
	color [input].	The color value of each pixel in the rectangle. The value consists of 32-bit positive integer values, where: BIT0 ~ BIT7 is the blue component; BIT8 ~ BIT15 is the green component; BIT16 ~ BIT23 is the red component; BIT24 ~ BIT31 is a reserved bit, this value may be used for transparency in the future, so be sure to 0xFF.
return	0	succeed
	-1	The parameter is invalid
usage	The display effect is a hollow unfilled rectangular border with a line width of 1 pixel; If you need to draw a rectangular border with a width greater than 1 pixel, you can set different coordinates to call this function multiple times to achieve it. If the coordinate value exceeds the screen range or the left coordinate is greater than the right coordinate or the top coordinate is greater than the bottom coordinate, it is an illegal parameter, and no rectangle drawing treatment will be made.	

4.8 CLcdClrRect

Prototype	**int CLcdClrRect (uint left, uint top, uint right, uint bottom, uint mode);**	
function	Clears the display information for the foreground of the screen in a rectangular area.	
parameter	left [input].	The left boundary value
	Top [input]	The upper boundary value
	right [input].	The right boundary value
	bottom [input].	Lower bounded value
	mode [input].	0: Clear the rectangle border and the contents of the box 1: Clear only the rectangular border
return	0	succeed
	-1	Invalid parameter (invalid coordinate value or mode).
usage	If the coordinate value exceeds the screen range or the left coordinate is greater than the right coordinate or the top coordinate is greater than the bottom coordinate, it is an illegal parameter and will NULL be cleared in any way.	

4.9 CLcdTextOut

Prototype	**int CLcdTextOut (uint x, orint and, char *fmt, …) ;**	
function	Formats the display string at the specified location in the foreground of the screen.	
parameter	x [input].	Starting row X-coordinate value (pixels)
	y [input].	Y coordinate value (pixels) in the starting row
	FMT [input].	Displays the string and format of the output
return	0	succeed
	-1	The input parameter is invalid
usage	1. Character output coordinates do NULL wrap when they exceed the right boundary of the screen. 2. Only the strokes of the text are output, and the other spaces of the font mold blocks other than the strokes are transparent and have an overlay effect with the background layer.	

4.10 CLcdDispStatus

Prototype	**Int CLcdDispStatus (int flag);**	
function	Displays the system status bar icon switch	
parameter	flag [input].	0 Off 1 On
return	0	Setup successful
	-1	Setup failed
usage	The status bar icon is displayed by default when the system boots	

4.11 CLcdDrawArea

Prototype	**int CLcdDrawArea(u32 left,u32 top, u32 width, u32 height, u8* data);**	
function	Load and start the drawing area.	
parameter	left [input].	The starting x-coordinate pixel value
	Top [input].	The starting Y-coordinate pixel value
	WiDTH [input].	X-axis direction length
	height [input].	Y-axis length

	data [input].	16BITS RGB (R5| G6| B5) bit stream
return	0	The data loads successfully and starts drawing.
	-1	The input parameter is invalid.
usage	1. data is 1 6-bit R GB (R 5| G6| B5) Bitstream data, with the upper eight bits first, the lower eight bits last.	

5 Magnetic card reader

5.1 MagOpen

Prototype	**void MagOpen(void);**	
function	Open the magnetic card reader.	
parameter	NULL	
return	NULL	
usage	Reading magnetic card data works in an interrupt mode, once the magnetic card reader is opened, even if the read function is NULL called, as long as the card is swiped, the magnetic head can read the magnetic card data, so when the magnetic card reader is NULL required, It is best to turn off the magnetic card reader.	

5.2 MagClose

Prototype	**void MagClose(void);**	
function	Turn off the magnetic card reader.	
parameter	NULL	

return	NULL
usage	When NULL using a magnetic card reader, turn off the function of reading the head.

5.3 MagReset

Prototype	**void MagReset(void);**	
function	Reset the head and clear the magnetic card buffer data.	
parameter	NULL	
return	NULL	
usage	In the case that the head is powered on, this function resets the head and clears the magnetic card buffer data; When the head is NULL powered on, only the magnetic card buffer data is cleared. To ensure that the data read in by the head is up-to-date, it is best to call this function once to clear the magnetic card buffered data before cyclically detecting whether the card is swiped.	

5.4 MagRead

Prototype	**u8 MagRead(u8 *Track1, u8 *Track2, u8 *Track3);**	
function	Read data for tracks 1, 2, and 3 of the magnetic card buffer.	
parameter	Track1 [Output]	A buffer pointer that holds 1 track data.
	Track2 [Output]	A buffer pointer that holds 2 track data.
	Track3 [Output]	A buffer pointer that holds 3-track data.
return	0x00	Reading error.

	other	bit0 = 1 reads 1 track data correctly
		bit1 = 1 reads out 2 track data correctly
		bit2 = 1 reads 3 track data correctly
		Bit4 = 1 1 Track data has a check error
		Bit5 = 1 2 Track data has a check error
		Bit6 = 1 3 Track data has a check error

usage	Works with the MagSwiped function. If you do NULL need a track data, you can set the pointer corresponding to the track to NULL, and the data for the track will NULL be output. Generally, when reading a magnetic card that complies with the ISO7812 standard: track1 is 79 bytes long track2 is 40 bytes long	

	track3 is 107 bytes long

6 Contact IC card reader

6.1 Basic definition

IC (Integrated Circuit Card) cards usually refer to contact same/asynchronous integrated circuit cards that comply with or are implemented based on the ISO7816 specification.

6.1.1 Data definition

IC card data structure:

Send data structure: APDU_SEND
typedef struct{ *unsigned char Command[4];   /*CLA, INS, P1, P2   */* *unsigned short Lc;   /*Data length sent to IC card*/* *unsigned char   DataIn[512];   /* Data sent to IC card*/* *unsigned short Le;   /* Length of expected returned data */* *}APDU_SEND;*
Receive data structure: APDU_RESP

```
typedef struct{
        unsigned short LenOut;                  /*Actual data length returned from IC card
                                                */
        unsigned char   DataOut[512];           /*Data pointer returned from IC card */
        unsigned char   SWA;                    /* IC card status word 1 */
        unsigned char   SWB;                    /* IC card status word 2 */
}APDU_RESP;
```

6.2 IccInit

Prototype	**u8 IccInit(u8 slot, u8 *ATR);**	
function	Resets the IC card and returns the reset answer content of the card.	
parameter	slot [input]	slot - The card channel number needs to be initialized bit[2:0]: The channel number of 0~7 number bit[4:3]: Indicates the operating condition • 00:5V • 01:1.8V • 10:3V • 11:5V bit[5]: Indicates support for the PPS protocol • 0: NULL supported • 1: Support bit[6]: Indicates the rate at which power-on reset is used • 0: Default (9600). • 1：38400 bit[7]: Indicates the supported specification type • 0：EMV • 1：ISO7816-3
	ATR [output]	Card reset answer. (Minimum 33bytes of space required) Its content is length (1 byte) + reset reply content [output]
return	0x00	Initialization succeeded
	0x02	Card pull-out (for SAM holders, 0 x33 error code is returned if no SAM card is present or NULL plugged in).
	0x04	The channel number is wrong
	0x06	Protocol error
	other	Communication failed

usage	● The length of the ATR information of the IC card varies depending on the card, so refer to the IC card used to define sufficient memory space (maximum length NULL exceeding 33). bytes). ● IC card channel number slot description: 0 is a semi-subducted large deck (user card). 1 is a large deck of total vandalism 2, 3, 4, 5 are small SAM card holders

6.3 IccClose

Prototype	**void IccClose(u8 slot);**	
function	Power down on cards in the specified deck.	
parameter	slot [input]	slot - The card channel number needs to be initialized bit[2:0]: The channel number of 0~7 number bit[4:3]: Indicates the operating condition ● 00:5V ● 01:1.8V ● 10:3V ● 11:5V bit[5]: Indicates support for the PPS protocol ● 0: NULL supported ● 1: Support bit[6]: Indicates the rate at which power-on reset is used ● 0: Default (9600). ● 1：38400 bit[7]: Indicates the supported specification type ● 0：EMV ● 1：ISO7816-3
return	NULL	
usage	Error: Illegal parameter no action.	

6.4 IccIsoCommand

Prototype	**u8 IccIsoCommand(u8 slot,** **APDU_SEND *ApduSend,** **APDU_RESP *ApduRecv);**

function	IC card operation function. This function supports the common api protocols (T=0 and T=1) of IC cards.	
parameter	slot [input]	slot - The card channel number needs to be initialized bit[2:0]: The channel number of 0~7 number bit[4:3]: Indicates the operating condition • 00:5V • 01:1.8V • 10:3V • 11:5V bit[5]: Indicates support for the PPS protocol • 0: NULL supported • 1: Support bit[6]: Indicates the rate at which power-on reset is used • 0: Default (9600). • 1：38400 bit[7]: Indicates the supported specification type • 0：EMV • 1：ISO7816-3
	ApduSend [output]	Command data structure sent to IC card
	ApduRecv [output]	The data structure returned from the IC card
return	0x00	Execution succeeded
	0x01	Communication timed out
	0x02	The card is dialed during the transaction
	0x03	Parity error
	0x04	Wrong channel selection
	0x05	Sending data too long (LC)
	0x06	Card protocol error (NULL T=0 or T=1)
	0x07	There is no reset card
	0xff	No communication or no power.

usage	The structure of the APDU_SEND is as follows:
	struct{
	unsigned charCommand[4];
	unsigned int Lc;
	unsigned char DataIn[512];
	unsigned int Le;
	};
	where Command[] = {CLA,INS,P1,P2}.
	Lc = length of DataIn.
	DataIn = Data pointer to be sent to the IC card.
	Le = the length of the expected returned data, the actual return data length card return is related to the specific command, here is just the expected length, the	

	actual return length can be obtained by the LenOut of the response structure parameter. **Box1:Lc=0; The=0** There is neither data sent nor returned **Box2:Lc=0; The>0** If no data is sent but the data is expected to return, if the number of data expected to be returned by the terminal in the actual application is unknown, Le=256 should be set; Otherwise, it is a determined numeric value. **Box 3: Lc>0; The=0** There is data sent but no data is expected to return **Case4: Lc>0; Le>0** If there is data sent and data is expected to return, if the number of data expected to be returned by the terminal in practical application is unknown, Le=256 should be set; Otherwise, it is a determined numeric value. **Notes**: Since Le**=0** indicates that data return is NULL required here, Le**=256** should be set when **LE="0"** in the command sent in practical applications. The structure of the APDU_RESP is as follows: *struct{* *unsigned intLenOut;* *unsigned char DataOut[512];* *unsigned char SWA;* *unsigned char SWB;* *};* Among them, LenOut = the length of data actually returned from the IC card. DataOut = Data pointer returned from the IC card. SWA = Status Byte 1. SWB = Status Byte 2.

6.5 IccDetect

Prototype	**u8 IccDetect(u8 slot);**	
function	Check if there is a card in the designated deck and use the presence detection for 0-1 decks. For the 2-5 deck, do power-on reset detection.	
parameter	slot [input]	slot - The card channel number needs to be initialized bit[2:0]: The channel number of 0~7 number bit[4:3]: Indicates the operating condition • 00:5V • 01:1.8V • 10:3V • 11:5V bit[5]: Indicates support for the PPS protocol • 0: NULL supported

		● 1: Support bit[6]: Indicates the rate at which power-on reset is used ● 0: Default (9600). ● 1：38400 bit[7]: Indicates the supported specification type ● 0：EMV ● 1：ISO7816-3
return	0x00	There is a card insertion.
	0xff	No card insertion.
usage	The function returns immediately, regardless of whether the deck has a card or NULL. Call this function to determine whether a card event has occurred.	

CAUTION

This api is only applicable to user card status detection.

7 RF card reader

7.1 Basic definition

Non-contact IC card, also known as radio frequency card, is composed of IC chip and induction antenna, packaged in a standard PVC card, and the chip and antenna do NULL have any exposed parts. It is a new technology developed in the world in recent years, which successfully combines radio frequency identification technology and IC card technology, ending the problem of passive (no power supply in the card) and contact-free, and is a major breakthrough in the field of electronic devices. The card is close to the surface of the reader within a certain distance range (usually 0-4cm), and the data reading and writing operation is completed through the transmission of radio waves.

PCD: Proximity Coupling Device

PICC: Proximity IC Card

7.1.1 Returns a list of values

numeric value	illustrate
0x00	The operation succeeded
0x01	Parameter error
0x02	The RF module is NULL turned on
0x03	Card NULL found (no card of the specified type in the sensing area)

0x04	Too many cards in the sensing area (communication conflicts)
0x06	Protocol errors (data that violates the protocol appears in the card answer)
0x13	The card is NULL activated
0x14	Doka conflict
0x15	Timeout is unresponsive
0x16	Protocol error
0x17	Communication transmission error
0x18	M1 card authentication failed
0x19	The sector is NULL certified
0x1A	The numeric block data is malformed
0x1B	The card is still in the sensing zone
0x1C	Card status error (e.g. A/B card calling M1 card api, or M1 card calling PiccIsoCommand api)
0xff	The api chip is NULL present or abnormal
other	Related to specific API functions

7.2 PiccOpen

Prototype	**u8 PiccOpen(void);**	
function	Power on and reset the contactless card module to check whether the initial state of the module is normal after reset.	
parameter	NULL	
return	RET_RF_OK	0x00 operation succeeded
	other	See List of return values
usage	● After the POS is powered on, the contactless card module is in the default power-down state. ● If this function is NULL called, calls to non-contact card classes other than PiccClose() will fail. ● When the contactless card module is NULL installed or fails, calling this function fails.	

7.3 PiccDetect

Prototype	**u8 PiccDetect(u8 Mode,** **u8 *CardType,** **u8 *SerialInfo,**

	u8 *CID, **u8 *Other);**		
function	Search for PICC cards by the specified pattern; Once you find a card, select it and activate it. Multi-card is NULL allowed in the sensing zone.		
parameter	Mode [input]	0x00	Search for Type A and Type B cards once, This mode is suitable for applications where enhanced multi-card detection is required. This mode is a card search mode that conforms to ISO14443 specifications;
		0x01	Search for Type A and Type B cards once; This mode is EMV card hunting mode, which is usually used;
		‘a’或 ‘A’	Search for Type A cards only once;
		‘b’或 ‘B’	Search for B cards only once;
		‘m’或 ‘M’	Search for M1 cards only once;
		Other values	retain
	CardType [output]	HOLDS THE BUFFER POINTER OF THE CARD TYPE, WHICH CAN BE NULL Both currently return one-byte type values	
		‘A’	Search for Type A cards
		‘B’	Found a Type B card
		‘M’	Search for M1 card
	SerialInfo [output]	The buffer pointer that holds the card serial number, which can be NULL.	
	CID [output]	The buffer pointer that stores the logical channel number of the card, which is internally allocated and specified by the driver, and the value range is 0~14, which can be NULL.	
	Other [output]	The buffer pointer that stores detailed error codes, card response information, and other contents can be NULL.	
return	RET_RF_OK	0x00 operation succeeded	
	other	See List of return values	
usage	● CardType: In the case of successful card finding, Mode=0 x00 or 0x01 may return 'A' or 'B'; Mode='a' or 'A' will only return 'A'; Mode='b' or 'B' will only return 'B'; Mode='m' or 'M' only returns 'M'. Note that here the 0 pattern or 1 pattern is a number, NULL the characters '0' or '1'. ● SerialInfo：L+V； L: The sequence number length of one byte, V: The sequence number information of length L. Typical serial numbers are 4 bytes, 7 bytes, or 10 bytes.		

	● CID: Currently, only one card is allowed in the sensing area, so CID[0] always returns 0x00. ● Other: len (1 byte) + errcode (2 bytes) + card-related information 1. LEN represents the total length of the returned information, excluding 1 byte of the len itself 2. 对 A 型卡，Other 表 示：len[1]+errorcode[2]+ ATQA[2] + SAK1 + [SAK2] + [SAK3]+ ATS SAK1/SAK2/SAK3 are both 1 byte, [SAK2] and [SAK3] are optional, only SAK1 when SerialInfo is 4 bytes, SAK1 and SAK2 when SerialInfo is 7 bytes, and SAK1, SAK2 and SAK3 when SerialInfo is 10 bytes. ATQA is 2 bytes, where the length of ATS is len minus the length of known information 3. For M-type cards, Other means: len[1] + errcode[2] + ATQA[2] + SAK1 4. For B-type cards, Other means: len[1] + errcode[2] + ATQB[12] + ATTRIB The length of ATQB is 12 bytes, and the length of ATTRIB is len minus the length of known information For more information about ATS, ATQB, and ATQA, please refer to the relevant sections of ISO14443-3 and ISO14443-4. ● If there is no card of the specified type in the sensing area, the function will exit with an error after searching once, and will NULL search continuously in a loop; The continuous loop of hunting is done and controlled by the application itself.

NOTE

Call this function to wake up and activate the standby card of the specified type in the sensing zone, after which the card enters the activated state. For cards in the activated state, calling the function again returns a failure and the card goes into a down state.

7.4 PiccIsoCommand

Prototype	**u8 PiccIsoCommand(u8 cid, APDU_SEND *ApduSend, APDU_RESP *ApduRecv);**	
function	On the specified channel, data in APDU format is sent to the card and a response is received.	
parameter	CID [input]	Used to specify the card logical channel number; The channel number is determined by The CID parameter output of PiccDetect( ) has a value range of 0~14, the current values are 0.
	ApduSend [input]	Send the PICC card command data structure
	ApduRecv [output]	Data structures returned from the PICC card
	RET_RF_OK	0x00 operation succeeded

return	other	Failed, see List of return values
usage	APDU: Application Protocol Data Unit, which refers to the card data format that complies with the ISO-14443-4 protocol. For APDU_SEND and APDU_RESP data structure definition, see the structure definition section of the IccIsoCommand() function in the contact IC card reader.	

NOTE

1. The function can only be called after the PiccDetect( ) call is successful, otherwise it cannot succeed.
2. ApduSend–>LC should be no greater than 255, otherwise a parameter error will be returned.

7.5 PiccRemove

Prototype	**u8 PiccRemove(u8 mode, u8 cid);**	
function	According to the specified mode, send a shutdown command to the card; or send a deactivation command; Or reset the carrier and determine if the card has moved away from the sensing area.	
parameter	mode [input]	'h'或'H' It means HALT, which only sends a stop command to the card and then exits; Damn it The program does NULL perform card removal detection
		'r'或'R' REMOVE， Send a stop command to the card and perform card removal detection;
		'e'或' IS Shifter mode in accordance with EMV non-connector specifications Reset the carrier and perform a card move detection
	CID [input]	The PiccDetect( ) call returns the logical channel number of the card, currently All are 0.
	RET_RF_OK	0x00 operation succeeded

return	other	0x01 parameter is incorrect 0x02 module is NULL turned on 0x03 card does NULL wake up 0x04 transmission error 0x05 protocol error 0x06 card is still in the sensing zone and only returns in R mode or E mode
usage		

7.6 PiccClose

Prototype	**void PiccClose(void);**	
function	Shut down the PICC module so that it is powered down.	
parameter	NULL	
return	NULL	
usage	After calling this function, the contactless card module becomes turned off and the module no longer radiates RF carriers to the outside. After that, all calls except PiccOpen( ) fail.	

NOTE

Recommendation: After the transaction is completed, call this function to close the RF card module, and then call the PiccOpen() function to restart the RF card module before the next transaction starts.

7.7 PiccLight

Prototype	**void PiccLight(u8 ucLedIndex,** **u8 ucOnOff);**	
function	Controls the on-off and off-state of the RF module's 4 LEDs.	
parameter	ucLedIndex [input]	Lamp index, each representing a lamp of one color
	ucOnOff [input]	Light or turn off the sign 0 Off Non-0 lit
return	NULL	
usage	a) ucLedIndex, its values are as follows: BIT0：red BIT1：green BIT2：yellow	

	BIT3：blue BIT4~BIT7: Retained #define PICC_LED_RED 0x01 #define PICC_LED_GREEN 0x02 #define PICC_LED_YELLOW 0x04 #define PICC_LED_BLUE 0x08 b) Machine PICC LEDs are virtualized by an LCD screen. After calling this function, the screen displays a virtual LED, and after calling PiccClose, the screen turns off the virtual LED light.

7.8 PiccCmdExchange

Prototype	**u8 PiccCmdExchange(uint uiSendLen,** **u8* paucInData,** **uint* puiRecLen,** **u8* paucOutData);**	
function	The APDU data is exchanged with the card, and the terminal sends the data from the paucInData directly to the card and receives the response data of the card.	
parameter	uiSendLen [input].	The length of the command data to be sent
	pauclnData [input].	Command data to be sent
	puiRecLen [output].	The length of the data received for the card
	paucOutData [Output].	Card data received
return	0x00	succeed
	0x01	Parameter error
	0x02	The module is NULL turned on
	0x03	The card is NULL activated
	0x04	Transmission error
	0x05	Protocol error
	other	Error, refer to the list of returned values

usage	If PCD sends a select file command to PICC, the usage of calling the PiccIsoCommand() api and calling the PiccCmdExchange() api is as follows:
	1. Using the PiccIsoCommand() api:
	APDU_SEND ApduSend;
	APDU_RESP ApduRecv;
	memcpy(ApduSend.Command,"\x00\xA4\x04\x00",4);
	ApduSend.Lc = 0x0E;
	memset(ApduSend.DataIn,0,sizeof(ApduSend.DataIn));
	strcpy(ApduSend.DataIn,"1PAY.SYS. DDF01");
	ApduSend.Le = 256;
	PiccIsoCommand(0, &ApduSend, &ApduRecv);	

	2. Using the PiccCmdExchange() api:
	u8 aucDataIn[256], aucDataOut[256];
	uint uiLenLen=0, respLength=0;
	memset(aucDataIn, 0, sizeof(aucDataIn));
	memset(aucDataOut, 0, sizeof(aucDataOut));
	memcpy(aucDataIn, "\x00\xA4\x04\x00", 4);
	uiLenIn += 4;
	aucDataIn[uiLenIn++] = 0x0E;
	memcpy(aucDataIn+uiLenIn, "2PAY.SYS. DDF01", 0x0E);
	uiLenIn += 0x0E;
	aucDataIn[uiLenIn++] = 0;
	PiccCmdExchange(uiLenIn, aucDataIn, &respLength, aucDataOut);

8 printer

8.1 Basic definition

After calling the print initialization function, make the necessary print control settings, use the print string function to arrange the print content, and finally start the print function to print and return to the printing status after printing. You can also query by querying the print status function.

NOTE

The following functions are NULL available for models without printer modules, i.e. calling the following functions does nothing.

8.1.1 Returns a list of values

numeric value	illustrate
0x00	Print successfully
0x01	The printer is busy
0x02	The printer is out of paper
0x03	The print packet is malformed
0x04	Printer failure

0x08	The printer is overheating
0x09	The printer voltage is too low
0xf0	Printing is NULL complete
0xfc	The printer does NULL have a font library
0xfe	The packet is too long
Other values	Other errors

8.2 PrnInit

Prototype	**u8 PrnInit(void);**	
function	Print initialization, called once before each print.	
parameter	NULL	
return	0x00	The command was executed successfully
	Other values	mistake
usage	Restore the printer's default settings after initialization; The data in the print buffer is emptied after initialization.	

8.3 PrnStart

Prototype	**u8 PrnStart(u8* DotBitmap, u32 lines);**	
function	Start the printer and print the data in the DotBitmap buffer.	
parameter	DotBitmap [input].	Printer point bitmap data, 3 84 dots per line, i.e. 3 84 BITS, 48 bytes
	lines [input].	The number of point rows in the cache.
return	See List of return values	
usage	1. After calling this function, the printer prints and does NULL return until the printing is complete; 2. After printing, the function returns the print status, so you don't have to query the print status anymore; 3. If the printer calls the function again after successfully completing the print task, the document is retyped.	

8.4 PrnStatus

Prototype	**u8 PrnStatus(void);**
function	Query the current print status.

parameter	NULL	
return	Refer to the return value of PrnStart	
usage	Use this function to check whether there is a print library, whether it is loaded with paper, and whether the print buffer is full.	

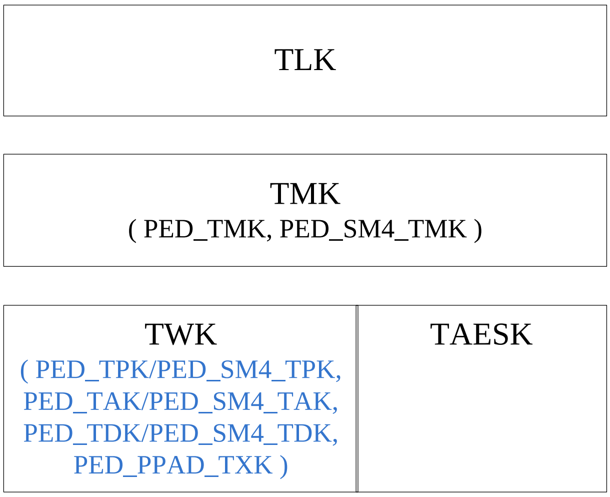
8.5 PrnSetGray

Prototype	**void PrnSetGray(int Level);**	
function	Sets the print blackness level.	
parameter	Level [input]	Level=0 reserved
		Level=1 The default blackness, which is the normal print sheet
		Level=2 retained
		Level=3 Double-layer thermal printing
		Level=4 Double-layer thermal printing, higher blackness than 3
		Level=[50~500] The blackness is set according to the default blackness percentage, such as 50 is to set the blackness to 50% of the default value, and 500 is to set the blackness to 500%.
		Other values are retained and the changes are invalid.
return	NULL	
usage	When level=3, the blackness will be 2.5~3 times deeper than level=1; When level=4, the blackness will be 3.5~4 times deeper than level=1; When Level=50~500, the blackness is the percentage corresponding to the default blackness, such as Level=50, the blackness is half of the default, and Level=100 is set as the default A Level setting of 300 is equivalent to Level=3, i.e. both increase blackness by 3 times.	

9 PED

9.1 Basic definition

PED adopts a three-layer key system, from top to bottom, in order:



TLK

TMK
(PED_TMK, PED_SM4_TMK)

TWK
(PED_TPK/PED_SM4_TPK,
PED_TAK/PED_SM4_TAK,
PED_TDK/PED_SM4_TDK,
PED_PPAD_TXK)

TAESK

- *TLK —Terminal Key Loading Key*

  The private key of the acquirer or POS operator, written directly by the acquirer or POS operator in a secure environment.

  The key has only one per PED terminal, with index numbers from 1 to 1.

- *TMK —Terminal Master Key =Acquirer Master Key*

  Terminal master key, or acquirer master key.

There can be 40 such keys, with index numbers from 1 to 40.

TMK can be written directly in a secure environment, directly written to TMK, and TWK can be decentralized through TMK.

- *TWK —Transaction working key = Transaction Pin Key + Transaction MAC Key +Terminal DES Key*

The terminal working key, the key for PIN ciphertext, MAC and other operations.

TPK (PED_TPK): Used to calculate the PIN Block after the application enters the PIN.

TAK (PED_TAK): Used to calculate MAC in application message communication.

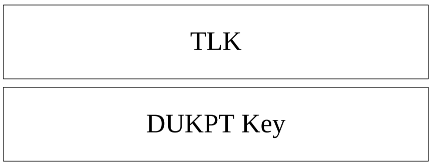
TDK(PED_TDK): Used to encrypt the transmission or storage of sensitive data in applications.

TWK can be written directly in a secure environment, and direct writing to TWK is consistent with the Fixed Key key system.

Each key has its index number, length, purpose, and label. The tag of the key is set through the API before writing the key to authorize the use of the key and ensure that the key will NULL be abused.

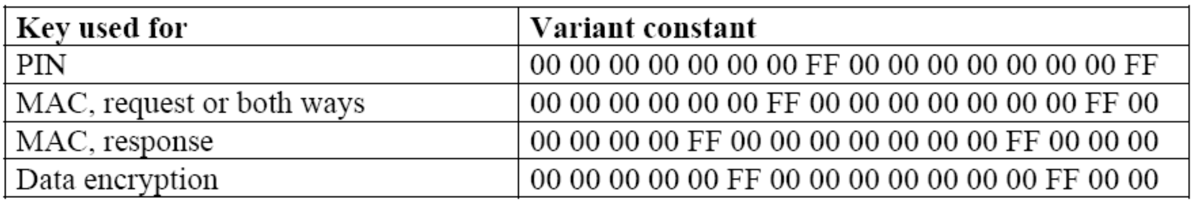
These seven types share index numbers from 1 to 40 and key storage for 40 groups.

- *TAESK —Terminal AES Key*
- AES algorithm key, AES algorithm encryption transmission or storage of sensitive data in the application.
- DUKPT key mechanism:



DUKPT [Derived Unique Key Per Transaction] key system is a key system of one key at a time, and the working key [PIN, MAC] of each transaction is different. It introduces the concept of KSN [Key Serial Number], which is the key factor to achieve one secret at a time.

The key corresponding to each KSN is generated according to the key purpose and the following components.



Key used for	Variant constant
PIN	00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 FF
MAC, request or both ways	00 00 00 00 00 00 FF 00 00 00 00 00 00 00 FF 00
MAC, response	00 00 00 00 FF 00 00 00 00 00 00 00 FF 00 00 00
Data encryption	00 00 00 00 00 FF 00 00 00 00 00 00 00 FF 00 00

There can be 40 sets of keys. When writing a TIK, you need to select the index number of the group and select the corresponding group index when using the DUKPT key.

All keys share index numbers from 1 to 4 0 and key storage space for groups 40

9.1.1 Returns a list of values

macro	numeric value	illustrate
PED_RET_ERR_START	-300	The starting value of the PEDAPI error code
PED_RET_ERR_END	-500	The end of the PEDAPI error code
PED_RET_OK	0	The PED function returns correctly
PED_RET_ERR_NO_KEY	PED_RET_ERR_START-1	The key does NULL exist
PED_RET_ERR_KEYIDX_ERR	PED_RET_ERR_START-2	The key index is wrong, and the parameter index is NULL in scope
PED_RET_ERR_DERIVE_ERR	PED_RET_ERR_START-3	When the key is written, the source key is less hierarchical than the destination key
PED_RET_ERR_CHECK_KEY_FAIL	PED_RET_ERR_START-4	Key validation failed
PED_RET_ERR_NO_PIN_INPUT	PED_RET_ERR_START-5	No PIN entered
PED_RET_ERR_INPUT_CANCEL	PED_RET_ERR_START-6	Cancel entering the PIN
PED_RET_ERR_WAIT_INTERVAL	PED_RET_ERR_START-7	Function calls are less than the minimum interval
PED_RET_ERR_CHECK_MODE_ERR	PED_RET_ERR_START-8	KCV mode is wrong and NULL supported
PED_RET_ERR_NO_RIGHT_USE	PED_RET_ERR_START-9	The key is NULL authorized to be used, and the key is returned when the key label is incorrect or when the value of the source key type is greater than the destination key type
PED_RET_ERR_KEY_TYPE_ERR	PED_RET_ERR_START-10	The key type is wrong
PED_RET_ERR_EXPLEN_ERR	PED_RET_ERR_START-11	The length string of the expected PIN is wrong
PED_RET_ERR_DSTKEY_IDX_ERR	PED_RET_ERR_START-12	The destination key is indexed incorrectly and is NULL in scope

PED_RET_ERR_SRCKEY_IDX_ERR	PED_RET_ERR_START-13	The source key is indexed incorrectly and is NULL in scope
PED_RET_ERR_KEY_LEN_ERR	PED_RET_ERR_START-14	The key length is wrong
PED_RET_ERR_INPUT_TIMEOUT	PED_RET_ERR_START-15	Enter the PIN timeout
PED_RET_ERR_NO_ICC	PED_RET_ERR_START-16	The IC card is NULL present
PED_RET_ERR_ICC_NO_INIT	PED_RET_ERR_START-17	The IC card is NULL initialized
PED_RET_ERR_GROUP_IDX_ERR	PED_RET_ERR_START-18	The DUKPT group index number is wrong
PED_RET_ERR_PARAM_PTR_NULL	PED_RET_ERR_START-19	The pointer parameter is illegally empty
PED_RET_ERR_LOCKED	PED_RET_ERR_START-20	PED is locked
PED_RET_ERROR	PED_RET_ERR_START-21	PED generic error
PED_RET_ERR_NOMORE_BUF	PED_RET_ERR_START-22	There is no free buffer
PED_RET_ERR_NEED_ADMIN	PED_RET_ERR_START-23	Advanced permissions are required
PED_RET_ERR_DUKPT_OVERFLOW	PED_RET_ERR_START-24	DUKPT has overflowed
PED_RET_ERR_KCV_CHECK_FAIL	PED_RET_ERR_START-25	KCV validation failed
PED_RET_ERR_SRCKEY_TYPE_ERR	PED_RET_ERR_START-26	The source key type is wrong
PED_RET_ERR_UNSPT_CMD	PED_RET_ERR_START-27	Command NULL supported
PED_RET_ERR_COMM_ERR	PED_RET_ERR_START-28	Communication error
PED_RET_ERR_NO_UAPUK	PED_RET_ERR_START-29	There is no user authentication public key
PED_RET_ERR_ADMIN_ERR	PED_RET_ERR_START-30	Failed to fetch system-sensitive service
PED_RET_ERR_DOWNLOAD_INACTIVE	PED_RET_ERR_START-31	The PED is in the download inactive state
PED_RET_ERR_KCV_ODD_CHECK_FAIL	PED_RET_ERR_START-32	KCV parity failed
PED_RET_ERR_PED_DATA_RW_FAIL	PED_RET_ERR_START-33	Failed to read PED data
PED_RET_ERR_ICC_CMD_ERR	PED_RET_ERR_START-34	Card operation errors (offline plaintext, ciphertext password verification)
PED_RET_ERR_INPUT_CLEAR	PED_RET_ERR_START-39	The user presses the CLEAR key to exit entering the PIN

PED_RET_ERR_NO_FREE_FLASH	PED_RET_ERR_START-43	There is NULL enough storage space for the PED
PED_RET_ERR_DUKPT_NEED_INC_KSN	PED_RET_ERR_START-44	DUKPT KSN needs to add 1 first
PED_RET_ERR_KCV_MODE_ERR	PED_RET_ERR_START-45	KCV MODE error
PED_RET_ERR_DUKPT_NO_KCV	PED_RET_ERR_START-46	NO KCV
PED_RET_ERR_PIN_BYPASS_BYFUNKEY	PED_RET_ERR_START-47	Press the FN/ATM4 key to cancel the PIN entry
PED_RET_ERR_MAC_ERR	PED_RET_ERR_START-48	Data MAC check error
PED_RET_ERR_CRC_ERR	PED_RET_ERR_START-49	Data CRC checks errors
PED_RET_ERR_PARAM_INVALID	PED_RET_ERR_START-50	The parameter is incorrect or invalid

9.1.2 Data definition

Key Type:

macro	numeric value	illustrate
#define PED_TLK	0x01	TSP
#define PED_TMK	0x02	Master Key
#define PED_TPK	0x03	PIN Key
#define PED_TAK	0x04	MAC Key
#define PED_TDK	0x05	DES Key
#define PED_TIK	0x07	DUKPT Key

Structure purpose name:

Write the key structure	
typedef struct{	
u8 ucSrcKeyType;	*/* The key type of the source key that diverges* the key, *PED_TLK, PED_TMK, PED_TPK, PED_TAK, PED_TDK, must NULL be lower than* the *key level */*, where *ucDstKeyType* is located
u8ucSrcKeyIdx;	*/* The index of the source key of the key is divergent,* the *index generally starts from 1, if the variable is 0, it means that the key is written in clear text */*
u8 ucDstKeyType;	*/* Key type of the purpose key, PED_TLK, PED_TMK, PED_TPK, PED_TAK, PED_TDK */*

u8 ucDstKeyIdx;	*/* Purpose Key Index */*
int iDstKeyLen;	*/* Destination key length, 8,16,24*/*
u8 aucDstKeyValue[24];	*/* Write the contents of the key */*
}ST_KEY_INFO;	
KCV structure	
typedef struct{	
int iCheckMode;	*/*Check Mode*/*
u8 szCheckBuf[128];	*/*Check data buffer*/*
}ST_KCV_INFO;	

9.2 PedWriteKey

Prototype	**int PedWriteKey(ST_KEY_INFO * KeyInfoIn,** **ST_KCV_INFO * KcvInfoIn);**	
function	Write a key, including the writing and divergence of TLK, TMK, and TWK, and optionally use KCV to verify the correctness of the key.	
parameter	KeyInfoIn [input]	**ucSrcKeyType:** PED_TLK PED_TMK PED_TPK PED_TAK PED_TDK

		ucSrcKeyIdx:
		When ucSrcKeyType = PED_TLK
		UcSrcKeyIdx = 0;
		When ucSrcKeyType = PED_TMK,
		ucSrcKeyIdx = 0;
		When ucSrcKeyType = PED_TPK or PED_TAK or PED_TDK, ucSrcKeyIdx = [1~40];

		ucDstKeyType: PED_TLK PED_TMK/PED_SM4_TMK PED_TPK/PED_SM4_TPK PED_TAK/PED_SM4_TAK PED_TDK/PED_SM4_TDK
		ucDstKeyIdx:

		When ucDtKeyType = PED_TLK
		ucDstKeyIdx = 1;
		When ucDstKeyType = PED_TMK,
		ucDstKeyIdx = [1~40];
		When ucDstKeyType = PED_TPK or PED_TAK or PED_TDK, ucDstKeyIdx = [1~40];

		iDstKeyLen :8/16/24
		aucDstKeyValue: Plaintext or ciphertext of the destination key; If the destination key is a DES key, the destination key is a ciphertext encrypted by the source key in 3DES ECB mode.
	KcvInfoIn [input]	**iCheckMode:** Authentication mode
		0x00 no verification 0x01 calculate DES/TDES encryption for an 8-byte 0x00, obtained The first 4 bytes of ciphertext are KCV 0x02 first perform odd checks on the plaintext of the key, and then check "\x12\x34." x56\x78\x90\x12\x34\x56" for DES/TDES Encryption operation, the first 4 bytes of the ciphertext is KCV 0x03 pass in a string of data KcvData, using the source key pair [aucDstKeyValue (ciphertext) + KcvData]. Perform the MAC operation of the specified mode, and the 8-byte MAC is KCV
		aucCheckBuf: When iCheckMode=0x00 The value of aucCheckBuf is invalid, and the system considers KCV NULL validated AucCheckBuf can be invalid data
		When iCheckMode=0x01 aucCheckBuf[0] = length of KCV (4). aucCheckBuf + 1 points to the value of KCV
		When iCheckMode=0x02 aucCheckBuf[0] = length of KCV (4). aucCheckBuf + 1 points to the value of KCV
		When iCheckMode=0x03 aucCheckBuf[0] = KcvData length (KcvDataLen). aucCheckBuf+1: KcvData aucCheckBuf[1+KcvDataLen]=MAC operation mode value [its value refers to PEDGetMac]. aucCheckBuf[2+KcvDataLen]=KCV length

		aucCheckBuf + 3 + KcvDataLen points to the value of KCV
		When iCheckMode=0x04 aucCheckBuf[0] = length of KCV (4). aucCheckBuf + 1 points to the value of KCV
return	See List of return values	
usage	Writes the ciphertext or plaintext of a key to the specified index in the specified key type area. The usage of this function has the following points: 1. PED_TLK length can only be 16 or 24 bytes. Allow any key to be written or downloaded in clear text when PED_TLK does NULL exist; When PED_TLK exists, only ciphertext writing keys is allowed and keys up to 8 bytes in length are NULL allowed. When writing a secret in ciphertext, we recommend that the length of the source key be greater than or equal to the length of the destination key for security reasons. 2. When writing to PED_TLK, the PED first formates, erases all downloaded keys, and then writes to the PED_TLK. 3. When ucSrcKeyIdx is a legal value in the parameter description, the system believes that the aucDstKeyValue in KeyInfoIn is the ciphertext of the source key written by ucSrcKeyIdx and ucSrcKeyType in 3DES ECB mode The aucDstKeyValue is decrypted by the ucSrcKeyIdx key in the ucSrcKeyType key area, and written to the ucDstKeyIdx location in the ucDstKeyType area. where **ucDstKeyType >= ucSrcKeyType**. 4. ucDstKeyLen can only be 8 or 16,24, this key can only be used for DES calculations when ucDstKeyLen is 8, and 16 or 24 when ucDstKeyLen is 16 or 24, can be used for TDES calculations 。 The SM4 key length is fixed 16 bytes and is used for SM4 calculations. 5. ucDstKeyType specifies the key type, and when ucDstKeyType=PED_TPK/PED_SM4_TPK, this key can only be used to calculate PIN Block. When ucDstKeyType = PED_TAK/PED_SM4_TAK, this key can only be used to compute MAC. When ucDstKeyType=PED_TDK/PED_SM4_TDK, this key can only be used for encryption and decryption operations. This limits the use of the working key and guarantees the uniqueness of the working key function. 6. Note that if the master key is written at an index number (for example, index 1) in the master key area, then writing any type of master key (for example, PED_TMK) at the same index number will overwrite the previous key; If the working key is written at an index number in the working key area (for example, index 1), then writing either type of work key (for example, PED_TPKK, PED_TAK, PED_TDK) at the same index number overwrites the previous key.	

NOTE

KCV is to verify the key plaintext or ciphertext, if the iCheckMode in the structure in KcvInfoIn is NULL 0, the system follows the iCheckMode The specified mode performs KCV verification of the key plaintext or ciphertext.

9.3 PedWriteTIK

Prototype	**int PedWriteTIK(u8 GroupIdx, u8 SrcKeyIdx, u8 KeyLen, u8 * KeyValueIn, u8 * KsnIn, ST_KCV_INFO * KcvInfoIn);**	
function	Write a TIK and optionally use KCV to verify key correctness.	
parameter	GroupIdx [input]	[1~40] DUKPT key group index number
	SrcKeyIdx [input].	[0~1] Key index for decentralized keys
	KeyLen [input].	16 TIK length, now DUKPT algorithm supports 8/16 bytes long The key of the degree.
	KeyValueIn [input].	Ciphertext pointing to TIK
	KsnIn [input].	Point to Initialize KSN
	KcvInfoIn [input]	For details, see PedWriteKey
return	See List of return values	
usage	1. When SrcKeyIdx is 0, it means that KeyValueIn is a TIK in plaintext, and the system directly writes KeyDataIn as the plaintext of TIK, and can write it in plaintext PED_TLK does NULL exist 。 2. When SrcKeyIdx is 1, it means that the ciphertext of the TIK is decrypted and converted into a TIK using TLK. 3. The value of GroupIdx corresponds to the parameter GroupIdx in PedGetPinDukpt, PedGetMacDukpt.	

9.4 PedGetPinBlock

Prototype	**int PedGetPinBlock(u8 KeyIdx, u8 *ExpPinLenIn, u8 * DataIn, u8 *PinBlockOut, u8 Mode, ulong TimeOutMs);**	
function	Within the specified time limit, scan the PIN entered on the keypad and output the PIN BLOCK encrypted data block. Enter a PIN of the length specified by ExpPinLenIn and output a PIN BLOCK encrypted by the Mode specified algorithm.	
parameter	KeyIdx [input].	[1~40] Index of TPK
	Mode [input]	Select the format of the PIN BLOCK

		0x00 0x00 ISO9564 format 0 0x01 0x01 ISO9564 format 1 0x02 0x02 ISO9564 format 3 0x03 0x03 HK EPS Specific Format [see Appendix for details].
	ExpPinLenIn	The legal password length string that can be entered, the application enumerates all the allowed password lengths, and separates each length with a ", " sign, and the valid value of the password length is: 0, 4~12。 If 4 or 6 digit passwords are allowed and no password is allowed to press confirm directly, the string should be set to "0,4,6". The length of the enumeration 0 indicates that you can press the confirm key directly to return without entering any number. If there is an invalid value of length in the enumeration string, such as "2,6,7,10", the invalid value is NULL valid will be ignored.
	DataIn [input]	When Mode=0x00, DataIn points to the 16-digit primary account number generated after the card number is shifted. Does NULL contain a check digit. When Mode=0x01, DataIn is NULL. When Mode=0x02, DataIn points to the 16-digit primary account number generated after the card number is shifted, without the check digit. When Mode=0x03, the transaction pipeline number ISN [6 Bytes, ASCII code] is the transaction number
	PinBlockOut	8bytes points to the generated PINBlock
	TimeOutMs [input]	Enter the timeout period for the PIN in milliseconds The maximum value is 300000ms 0: Indicates that there is no timeout period, and PED does NULL do timeout control.
return	See List of return values	
usage	1. Password length limit: Specify the password length by enumeration to reduce the chance of customer mistyping, such as ExpPinLenIn is "0,4, 6", the cardholder can only enter the 4-digit or 6-digit password or press the "Confirm" key directly to complete the PIN entry If you enter a 5-digit or other digit passcode, you will NULL be able to end the input normally. 2. Cancel Input: You can press the "Cancel" key to abort the operation during the input process.	

Example:

```
unsigned char ucRet,PinBlk[9];
                memset(PinBlk,0);
inRet = PedGetPinBlock(1,"4,6","4000682502342834",0,PinBlk,12000);
/* Enter the PIN, the length can be 4 or 6, and the index of the TPK is 1 */
if(ucRet)
{
        /* Error *    / is displayed
        ...
        Return 2;
}
...
```

9.5 PedGetMac

Prototype	**int PedGetMac(u8 KeyIdx, u8 *DataIn, u16 DataInLen, u8 *MacOut, u8 Mode);**	
function	Use the MAC key specified by KeyIdx to perform MAC operations on the DataIn Mode-specified algorithm, and output the 8-byte MAC result to MacOut.	
parameter	KeyIdx [input]	Index of 1~40 TAK
	Mode [input]	0x00/0x01/0x02 0x00: ANSI X9.9 specification, BLOCK1 is encrypted with MAC key for DES/TDES, and the encryption result is bit-by-bit OR with BLOCK2 and then DES/with TAK TDES encryption, sequentially to obtain 8-byte encryption results. 0x01: Hypercom Fast Mode, BLOCK1 and BLOCK2 are bitwise XOR, XOR results are bitwise XOR with BLOCK3, and finally 8 bytes of XOR are obtained. The result is encrypted with DES/TDES using TAK. 0x02: ANSIX9.19 specification, BLOCK1 is encrypted with TAK for DES (only the first 8 bytes of the key), and the encryption result is bit-by-bit or BLOCK2 and then used TAK for DES encryption. The result of 8-byte encryption is obtained sequentially until the last DES/TDES encryption is used.
	DataInLen [input]	<=2048bytesMAC operation packet length [input], length If it is NULL divisible by 8 bytes, "\x00" is automatically filled.
	DataIn [input]	Packets that require MAC operations
	MacOut	MAC output
return	See List of return values	
usage		

Example:

```
unsigned char MacOut[9];

memset(MacOut,0, sizeof(MacOut));
inRet = PedGetMac(1,"1234567890123456",16,MacOut,0);
/* Using the TAK index of 1, algorithm 1, pair the data of length 16 "1234567890123456"
Calculate MAC */
if(iRet)
{
/* Error *    / is displayed
…
return 1;
}
…
```

9.6 PedVerifyOfflinePin

Prototype	**int PedVerifyOfflinePin (u32 offline_type, ST_SCPINKEY *RsaKey, u8 *status_word);**	
function	Implement plaintext/ciphertext offline PIN verification function. GET THE PIN, AND THEN SEND THE PLAIN/CIPHERTEXT PIN BLOCK DIRECTLY TO THE CARD BY FOLLOWING THE CARD COMMAND AND CARD CHANNEL NUMBER PROVIDED BY THE APP	
parameter	offline_type [input].	0: plaintext 1: Ciphertext
	RsaKey [input].	The data structures required for encryption, as defined in Usage, are described in Usage
	status_word [Output].	The status code of the card response (2 bytes: SW1+SW2).
return	See List of return values	
usage	typedef struct{ u32 modlen; /* Cryptographic public key modulus length */ u8 mod[256]; /* Cryptographic public key modulus, high byte first, low byte last, no Supplement 0 */ u8 exponent[4]; /* Cryptographic public key exponent, high byte first, low byte last, no Supplement 0 */ u8 randlen; /* Random number obtained from the card is long */ u8 random[8]; /* Random number taken from the card */ }ST_SCPINKEY;	

9.7 PedCalcDES

Prototype	**int PedCalcDES(u8 KeyIdx,** **u8 * DataIn,** **u16 DataInLen,** **u8 * DataOut,** **u8 Mode);**	
function	Use TDK to perform DES/TDES operations on DataInLen length data, using DES or TDES depending on the length of the key.	
parameter	KeyIdx [input]	Index of TDK or TXK
	DataInLen [input]	Data length <=2048,8 divisible
	Mode [input]	Encryption: 0x01 Decryption: 0x00
	DataIn [input]	Points to the data that needs to be calculated
	DataOut [output]	Points to the data that has been calculated
return	See List of return values	
usage		

9.8 PedGetPinDukpt

Prototype	**int PedGetPinDukpt(u8 GroupIdx,** **u8 *ExpPinLenIn,** **u8 * DataIn,** **u8* KsnOut,** **u8 * PinBlockOut,** **u8 Mode,** **ulong TimeoutMs);**	
function	Enter the PIN on the PED and make the PINBlock for the PIN key of the DUKPT.	
parameter	GroupIdx [input]	1~40 DUKPT key group index number
	ExpPinLenIn [input]	The legal password length string that can be entered, the application enumerates all the allowed password lengths, and separates each length with ", " sign, and the valid value of password length is: 0, 4~12. If 4 or 6 digit passwords are allowed and no password is allowed to press confirm directly, the string should be set to "0,4,6". If the length of 0 is enumerated, it indicates that you can press the confirm key directly without entering any number. If there is an invalid value of length in the enumeration string, such as "2,6,7,10", the invalid value will be ignored.

	DataIn [input]	When Mode=0x00, DataIn points to the 16-digit primary account number generated after the card number is shifted, without the check digit. When Mode=0x01, DataIn is NULL. When Mode=0x02, DataIn points to the 16-digit primary account number generated after the card number is shifted, without the check digit. When Mode=0x03, the transaction serial number ISN [6 Bytes, ASCII code] is the transaction number
	KsnOut [output]	Point to the current KSN
	PinBlockOut	Point to the generated PIN Block 8bytes
	Mode [input]	Select the format of PIN BLOCK, 00 ISO9564 format 0 KSN automatically add 1 01 ISO9564 format 1 KSN automatically adds 1 02 ISO9564 format 2 KSN automatically adds 1 03 HK EPS format KSN automatically adds 1
		Select the format of PIN BLOCK, 20 ISO9564 format 0 KSN does NULL automatically add 1 21 ISO9564 format 1 KSN does NULL automatically add 1 22 ISO9564 format 2 KSN does NULL automatically add 1 23 HK EPS format KSN does NULL automatically add 1
	TimeoutMs [input]	Enter the timeout period for the PIN in milliseconds The maximum value is 300000ms 0: Indicates that there is no timeout period, and PED does NULL do timeout control.
return	See List of return values	
usage		

9.9 PedGetMacDukpt

Prototype	**int PedGetMacDukpt (u8 GroupIdx,** **u8 *DataIn,** **u16 DataInLen,** **u8 *MacOut,** **u8 * KsnOut,**

	u8 Mode);		
function	MAC is calculated using DUKPT's MAC key.		
parameter	GroupIdx [input]	1~40 DUKPT key group index number	
	DataIn [input]	Point to the data content that needs to compute the MAC	
	DataInLen [input]	The length of the data [input] <=2048, and the length is NULL divisible by 8 bytes Autofill "\x00"	
	MacOut	Point to the resulting MAC	
	KsnOut [output]	Point to the current KSN	
	Mode [input]	00 01 02	Request and reply MAC key 00: ANSI X9.9 specification, BLOCK1 is encrypted with MAC key for TDES, and the encryption result is bit-by-bit or bit-by-bit with BLOCK2, and then TDES encryption is done with MAC key Proceed sequentially to obtain an 8-byte encryption result. KSN automatically adds 1.
			01: Hypercom Fast Mode, BLOCK1 and BLOCK2 are bitwise XOR, XOR results and BLOCK3 are bitwise XOR, sequentially, and finally obtain 8 bytes of XOR result, The result is performed with the MAC key for TDES encryption. KSN automatically adds 1.
			02: ANSIX9.19 specification, BLOCK1 with MAC key for DES encryption (only take the first 8 bytes of key), encryption results and BLOCK2 bit-by-bit oror before using MAC The key is DES encrypted, and 8 bytes of encryption result is obtained sequentially until the last TDES encryption is used. KSN automatically adds 1.
		20 21 22	Request and reply MAC key 20: ANSI X9.9 specification, BLOCK1 is encrypted with MAC key for TDES, and the encryption result is bit-by-bit or BLOCK2 and then TDES encryption with MAC key, and 8-byte encryption is obtained in turn Fruit. KSN does NULL automatically add 1.
			21: Hypercom Fast Mode, BLOCK1 and BLOCK2 are bitwise XOR, XOR results and BLOCK3 are bitwise XOR, sequentially, and finally obtain 8 bytes of XOR result, The result is performed with the MAC key for TDES encryption. KSN does NULL automatically add 1.
			22: ANSIX9.19 specification, BLOCK1 with MAC key for DES encryption (only take the first 8 bytes of key), encryption results and BLOCK2 bit-by-bit or, and then use MAC key Do DES encryption, and obtain 8-byte encryption results in turn until the last TDES encryption is used. KSN does NULL automatically add 1.

			Other values are reserved for the extended MAC algorithm.
		40 41 42	Answer the MAC key
			40: ANSI X9.9 specification, BLOCK1 is encrypted with MAC key for TDES, and the encryption result is bit-by-bit OR with BLOCK2 and then TDES encryption with MAC key Proceed sequentially to obtain an 8-byte encryption result. KSN does NULL automatically add 1.
			41: Hypercom Fast Mode, BLOCK1 and BLOCK2 are bitwise XOR, the result is bitwise XOR with BLOCK3, in turn, and finally obtains the 8-byte XOR result, The result is performed with the MAC key for TDES encryption. KSN does NULL automatically add 1.
			42: ANSIX9.19 specification, BLOCK1 is encrypted with MAC key as DES (only the first 8 bytes of the key), and the encryption result is bit-by-bit or BLOCK2 and then DES encryption is done with TAK. Successively obtain 8-byte encryption results until the last TDES encryption is used. KSN does NULL automatically add 1.
			Other values are reserved for the extended MAC algorithm.
return	See List of return values		
usage			

9.10 PedGetKcv

Prototype	**int PedGetKcv(u8 KeyType, u8 KeyIdx, ST_KCV_INFO *KcvInfoOut);**	
function	Obtain the KCV value of the key for both parties to the conversation to verify the key, encrypt a piece of data with the specified key and algorithm, and return part of the data ciphertext.	
parameter	KeyType [input]	PED_TLK PED_TMK PED_TAK PED_TPK PED_TDK PED_TIK
	KeyIdx [input]	The index number of the key. The value can be 1~40.

	KcvInfoOut	[input]	iCheckMode = 0x00: Use this key to perform DES/TDES encryption operations on a piece of data, and the first 4 bytes of the generated ciphertext are KCV.
		[output]	aucCheckBuf When iCheckMode = 0, aucCheckBuf[0] is the length of the data to be operated. aucCheckBuf+1 points to the data to be operated [input] When the function returns correctly, aucCheckBuf points to a 4-byte length KCV. [Output]. The data to be performed must be a multiple of 8. When KeyType is PED_TIK, the returned KCV value is the KCV value when the PedWriteTIK api is written. If PedWriteTIK injects the key without a KCV checksum, the KCV checksum cannot be returned.
return	See List of return values		
usage			

9.11 PedGetVer

Prototype	**int PedGetVer(u8 * VerInfoOut);**	
function	Returns the version of the PED.	
parameter	VerInfoOut [output].	Pointer to the version information of the current PED, up to 16 bytes.
return	See List of return values	
usage		

9.12 PedErase

Prototype	**int PedErase();**	
function	Clear all key information from the PED.	
parameter	NULL	
return	See List of return values	
usage		

9.13 PedDukptDes

Prototype	**int PedDukptDes(u8 GroupIdx, u8 KeyVarType, u8 * pucIV, u16 DataInLen, u8 *DataIn, u8 * DataOut, u8*KsnOut , u8 Mode);**	
function	Using DUKPT's MAC key or DES key, the data in the input cache is encrypted or decrypted.	
parameter	GroupIdx [input]	1~40 DUKPT key group index number
	KeyVarType	0x00, with the request and reply MAC key 0x01, operate with the DUKPT DES key 0x02, use the DUKPT PIN key for ECB encryption (that is, when taking the key component type, Mode can only take the value 0x01: ECB encryption. ）
	pucIV [input]	8-byte initial vector, required for CBC encryption and decryption, if NULL is passed, "\x00\x00\x00\x00\x00\x00\x00\x00\x00" will be used as the initial vector by default.
	DataInLen [input]	Data length <=8192,8 divisible
	DataIn [input]	Points to the data that needs to be calculated
	DataOut [output]	Points to the data that has been calculated
	KsnOut [output]	10 bytes Current KSN
	Mode [input]	0x00: ECB decryption 0x01: ECB encryption 0x02: CBC decryption 0x03: CBC encryption
return	See List of return values	
usage		

9.14 PedGetDukptKSN

Prototype	**int PedGetDukptKSN(u8 GroupIdx, u8 * KsnOut);**
function	Read the KSN for the next calculation.

parameter	GroupIdx [input]	1~40 DUKPT key group index number
	KsnOut [output]	10 bytes Current KSN
return	See List of return values	
usage		

9.15 PedDukptIncreaseKsn

Prototype	**int PedDukptIncreaseKsn(u8 GroupIdx);**	
function	KSN plus 1.	
parameter	GroupIdx [input].	1~40 DUKPT key group index number
return	See List of return values	
usage	Each KSN corresponds to a DUKPT key that can only be used up to 256 times. When a single key is used 256 times, a PED_RET_ERR_DUKPT_NEED_INC_KSN error is returned, and you need to call this api to add 1 to KSN.	

10 File operations

10.1 Basic definition

When a file operation fails (returns -1), the specific reason for the failure can be read from the global variable errno, using GetLastError(). The value of errno, which is encoded as follows:

	macro	numeric value
#define	FILE_EXIST	1
#define	FILE_NOEXIST	2
#define	MEM_OVERFLOW	3
#define	TOO_MANY_FILES	4
#define	INVALID_HANDLE	5
#define	INVALID_MODE	6
#define	NO_FILESYS	7
#define	FILE_NULL_OPENED	8
#define	FILE_OPENED	9
#define	END_OVERFLOW	10
#define	TOP_OVERFLOW	11
#define	NO_PERMISSION	12
#define	FS_CORRUPT	13

10.1.1 Returns a list of values

numeric value	illustrate
>=0	Success, return the handle number (0~255).
-1	Failed, error code placed in errno
INVALID_MODE	Mode is NULL O_RDWR/O_CREATE
INVALID_FILEID	Invalid file handle
FILE_NOEXIST	The file does NULL exist
FILE_EXIST	Opens in a O_CREATE manner when the file exists
MEM_OVERFLOW	Lack of space
END_OVERFLOW	An offset that exceeds the length of the file when moving back
TOP_OVERFLOW	An error occurred while moving forward
TOO_MANY_FILES	There are too many file handles, more than 255, to create a new file
FILE_OPENED	The file is open
FILE_NULL_OPENED	The file is NULL open
NO_FILESYS	The file system is NULL established

10.1.2 Main parameters

The parameter name	illustrate
Filename	The file name, which can be up to 16 characters, ends with '\x00', and only the first 16 characters are taken for more than 16 characters.
Mode	NULL yet used.
Attr	NULL yet used.
In	File handle number, up to 64 files, so FID can take the value 0-63.
That	Data buffers, where space is allocated by the application.
Only	The length (in bytes) of the data to read/write
Offset	The number of bytes (signed value) from the position specified from where to move from. offset>0 moves forward and backward from fromwhere, and if the parameter is set SEEK_END and offset is greater than 0, the function returns failure. Offset<0 moves in the opposite direction, moving forward from fromwhere, and if the parameter is set SEEK_SET and the offset is less than 0, the function returns failure.
Fromwhere	You can take the values SEEK_CUR, SEEK_SET, and SEEK_END. SEEK_CUR means starting from the current file pointer; SEEK_SET means starting from the file header; SEEK_END means starting at the end of the file.

10.1.3 Data definition

Structure purpose name:

File information structure

```
typedef struct{
      unsigned char    fid;             /* File identification number */
      unsigned char    attr;            /* The owner of the file */
      unsigned char    type;            /* File Type      */
      char             name[17];        /* filename        */
      unsigned long    length;          /* File length     */
} FILE_INFO;
```

10.2 open

Prototype	**int open(char *filename,** **u8 mode);**	
function	Only files that are currently applied can be opened.	
parameter	filename [input]	The file name, which can be up to 16 characters and ends with '\x00' If it exceeds 16 characters, only the first 16 characters will be taken.
	mode [input]	BIT0 O_RDWR means to open the file for read/write BIT1 O_CREATE means creating a new file Other bits: reserved.
return	>=0	Success, return the handle number (0~63).
	-1	Failed, error code placed in errno
usage	If you start with O_CREATE | Open the file O_RDWR way, or O_RDWR if the file already exists, otherwise create the file. Open files can be opened repeatedly, with the file pointer moving to the beginning of the file each time you open it. The returned handle is incremented from 0, in fact, since monitor itself has system files, the file handles created by the user layer are NULL 0-based.	

10.3 ex_open

Prototype	**int ex_open(char *filename,** **u8 mode,** **u8* attr);**	
function	Access files from other apps.	
parameter	filename [input]	The file name, which can be up to 16 characters, ends with '\x00', and only the first 16 characters are taken for more than 16 characters.
	mode [input]	Retain.
	ATTR [input]	Retain.
return	Same as the return value of the open function	
usage		

10.4 read

Prototype	**int read(int fid,** **u8 *dat,** **int len);**
function	Reads data of the specified length in the file.

parameter	fid [input]	File handle number, can have up to 256 files, so FID can The value is 0~63.
	dat [output]	Data buffers, where space is allocated by the application.
	len [input]	Length of data to read (bytes)
return	>=0	The read succeeds, returning the actual number of bytes read.
	-1	Failed, error code placed in errorno
	FILE_NULL_OPENED	The file is NULL open
	INVALID_FILEID	Invalid file handle
usage	You must first get the file handle, that is, open a file. Compare the return value with len, if NULL equal, the read data may be wrong.	

10.5 write

Prototype	**int write(int fid,** **u8 *dat,** **int len);**	
function	Writes data of the specified length to the file.	
parameter	fid [input]	File handle number, can have up to 256 files, so FID can The value is 0~63.
	dat [input]	The data buffer to write.
	len [input]	The number of bytes of data to write.
return	>=0	The write succeeds, returning the actual number of bytes written.
	-1	Failed, error code placed in errno
	INVALID_FILEID	Invalid file handle
	FILE_NULL_OPENED	The file is NULL open
	MEM_OVERFLOW	Insufficient space or too many file handles
usage	You must first get the file handle, that is, open a file Compare the return value with len, if NULL equal, the data written may be wrong.	

10.6 close

Prototype	**int close(int fid);**	
function	Close the file handle.	
parameter	fid [input]	The file handle number can be 0~255.
return	0	Close the file successfully
	-1	Failed, error code placed in errno.

	INVALID_FILEID	Invalid file handle
	FILE_NULL_OPENED	The file is NULL open
usage	You must first get the file handle, that is, open a file. After you close a file handle, you can only reopen the file to operate on it. Even if an error is returned, there is no impact on the file system.	

10.7 seek

Prototype	**int seek(int fid, long offset, u8 fromwhere);**	
function	Moves the file pointer to the specified location.	
parameter	fid [input]	The file handle number can be 0~63.
	offset [input]	The number of bytes (signed value) from the position specified from where to move from. offset>0 moves forward and backward from fromwhere, and if the parameter is set SEEK_END and offset is greater than 0, the function returns failure. Offset<0 moves in the opposite direction, moving forward from fromwhere, and if the parameter is set SEEK_SET and the offset is less than 0, the function returns failure.
	fromwhere [input]	You can take the values SEEK_CUR, SEEK_SET, and SEEK_END. SEEK_CUR means starting from the current file pointer; SEEK_SET means starting from the file header; SEEK_END means starting at the end of the file.
return	0	succeed
	-1	Failed, error code placed in errno.
	INVALID_FILEID	Invalid file handle
	FILE_NULL_OPENED	The file is NULL open
	END_OVERFLOW	An offset that exceeds the length of the file when moving back
	TOP_OVERFLOW	An error occurred while moving forward
usage	You must first get the file handle, that is, open a file.	

10.8 filesize

Prototype	**long filesize(char *filename);**
function	Gets the number of bytes for a file.

parameter	filename [input]	The file name, which can be up to 16 characters and ends with '\x00' If it exceeds 16 characters, only the first 16 characters will be taken.
return	>=0	File size (bytes)
	-1	Failed, error code placed in errno.
	FILE_NOEXIST	The file does NULL exist
usage		

10.9 truncate

Prototype	**int truncate(int fid, long len);**	
function	Truncates the file to the specified length.	
parameter	fid [input]	The file handle number can be 0~63.
	len [input]	The truncated file length, in bytes.
return	0	Truncation succeeded.
	-1	Failed, error code placed in errno.
	NO_FILESYS	The file system is NULL established
	INVALID_FILEID	Invalid file handle
	FILE_NULL_OPENED	The file is NULL open
	TOP_OVERFLOW	The length is less than 0
	END_OVERFLOW	The length exceeds the file length
usage	You must first get the file handle, that is, open a file. This function truncates the file to len length, and all content in the original file from len to the end is truncated. The file pointer moves to the end of the truncated file.	

10.10 remove

Prototype	**int remove(const char *filename);**	
function	Delete a file.	
parameter	filename [input]	File name, maximum length 16 bytes, greater than 16 bytes only taken first 16 bytes.
return	0	succeed
	-1	Failed, error code placed in errno.
	FILE_OPENED	The file is open

	FILE_NOEXIST	The file does NULL exist
usage		

10.11 freesize

Prototype	**long freesize(void);**	
function	Gets the amount of storage space remaining in the file system.	
parameter	NULL	
return	Returns the total amount of space remaining in the file system (bytes).	
usage		

10.12 fexist

Prototype	**int fexist(char *filename);**	
function	Determine whether a file exists in the current application.	
parameter	filename [input]	File name, maximum length 16 bytes, greater than 16 bytes only taken first 16 bytes.
return	-1	There is no file specified in the current app
	Other(0-255)	The file serial number.
usage		

10.13 GetFileInfo

Prototype	**int GetFileInfo(FILE_INFO* finfo);**	
function	Get the file information that the current POS has.	
parameter	finfo [output]	A pointer to the buffer address that holds the returned file information, which must be called by the function A buffer large enough to accommodate the maximum number of files possible is required. (sizeof(FILE_INFO)×63＝1920 bytes) 。
return	>0	The number of files returned
	-1	Failed, error code placed in errno.
	NO_FILESYS	The file system is NULL established
usage	Returns information for all files.	

10.14 GetLastError

Prototype	**int GetLastError(void);**
function	To read the value of the error code, the error variable cannot be accessed directly in the application, but the errno must be accessed by calling the function.
parameter	NULL
return	Error code errno for the last function execution.
usage	

NOTE

1. Limited to file system operations.
2. The value of the error is NULL taken after using GetLastError, and the value of the error persists until the POS is restarted.

11 Communication API

11.1 Basic definition

Call this type of function to control the opening, closing, reading and writing operations of RS232 communication serial port and USB communication port.

NOTE

The correct way to remove USB devices from Windows PCs is to safely remove USB devices by clicking on the green icon at the bottom right corner of your computer screen. Abrupt unplugging of the device is an abnormal operation.

11.1.1 Returns a list of values

numeric value	illustrate
0x00	The operation succeeded
0x01	The send buffer is NULL empty (remaining data to be sent)
0x02	Illegal channel number
0x03	The channel is NULL open and is NULL connected to any physical port
0x04	Send buffer error (500ms full state)
0x05	No physical ports available
0xff	Data receive timeout
0xf0	The channel is being occupied by the system
0xf2	The system does NULL support it
0xfe	Invalid communication parameters, communication parameters do NULL conform to string rules or data is outside the normal range.
0xef	USB to serial device mounted unsuccessful (return value used only by FTDI USB serial port)
0xee	reset USB to serial device error (return value used only by FTDI USB adapter serial port).
0xec	FTDI USB adapter serial port chip communication blocking (only the return value used by FTDI USB adapter serial port)
0xed	Error setting the baud rate, check digit, and stop bit of the USB to serial device (only the return value used by FTDI USB serial port)
11	Device did NULL complete enumeration and configuration process (USB DEV only)
12	Device powered off and lost connection to host (USB DEV only)
13	Unplug the device from the console and plug it back in (USB DEV only)
14	Device turned off (USBDEV only)

11.1.2 A list of communication ports

Port number	port	use	Applicable models
11	USBDEV	USB device mode port	A50

11.2 PortOpen

Prototype	**u8 PortOpen(u8 channel,** **u8 *Attr);**
function	Open the specified communication port and set the communication parameters.

parameter	Channel [input]	Channel number: The logical number of the communication port
		11: USBDEV (USB Device Mode Port)
	Attr [input]	obligate
return	See List of return values	
usage	See the list of communication ports	

11.3 PortClose

Prototype	**u8 PortClose(u8 channel);**	
function	Close the specified communication port.	
parameter	Channel [input]	Channel number: The logical number of the communication port
		11: USBDEV (USB Device Mode Port)
		See the list of communication ports
return	See List of return values	
usage	● If there is still data in the sending queue that has NULL been sent, the operation will wait for all data to be sent before exiting. ● The failure of the communication port closure is meaningless, only has an indicative effect, and the underlying logical closure of the channel will always be successful. For example, if the send buffer data times out abnormally, the channel will still be closed.	

11.4 PortSend

Prototype	**u8 PortSend(u8 channel,** **u8 ch);**	
function	One byte of data is sent using the specified communication port.	
parameter	channel	Channel number: The logical number of the communication port
		11: USBDEV (USB Device Mode Port)
		See the list of communication ports
	ch [input]	Data to be sent (one byte)
return	See List of return values	
usage	Communication sending buffers are all 1k bytes;	

11.5 PortRecv

Prototype	**u8 PortRecv(u8 channel, u8 *ch, uint ms);**	
function	Within a set period of time, one byte of data is received from the specified communication port.	
parameter	channel	Channel number: The logical number of the communication port
		11: USBDEV (USB Device Mode Port)
		See the list of communication ports
	MS [input]	Receive timeout (in milliseconds)
	ch [output]	Holds the pointer to the received character.
return	See List of return values	
usage	1. If the receive timeout is set to 0, the receive failure (0xff) is returned immediately when the receive buffer is empty. 2. The communication receive buffers are all 1K bytes;	

11.6 PortReset

Prototype	**u8 PortReset(u8 channel);**	
function	Reset the communication port, which will clear all data in the communication port receive buffer.	
parameter	Channel [input]	Channel number: The logical number of the communication port.
		11: USBDEV (USB Device Mode Port)
		See the list of communication ports
return	See List of return values	
usage	1. The function does NULL clear the data in the send buffer;	

11.7 PortSends

Prototype	**u8 PortSends(u8 channel, U8 *str, u16 str_len);**	
function	Sends bytes of data using the specified communication port.	
parameter	Channel [input]	Channel number: The logical number of the communication port.

		11: USBDEV (USB Device Mode Port)
		See the list of communication ports
	str [input]	The pointer to the data string to be sent
	str_len [input]	The number of bytes of the string of data to be sent
return	See List of return values	
usage		

11.8 PortTxPoolCheck

Prototype	**u8 PortTxPoolCheck(u8 channel);**	
function	Check whether the send buffer of the specified port has no data to send.	
parameter	channel	Channel number: The logical number of the communication port
		11: USBDEV (USB Device Mode Port)
		See the list of communication ports
return	0x00	The send buffer is empty
	other	See List of return values
usage		

11.9 PortRecvs

Prototype	**int PortRecvs(u8 channel,** **u8 * pszBuf,** **u16 usBufLen,** **u16 usTimeoutMs);**	
function	Receive up to the desired length of data within a given time frame.	
parameter	channel	Channel number: The logical number of the communication port
		11:USB_DEV (USB device mode port)
		See the list of communication ports
	pszBuf [output]	Receive buffer first address
	usBufLen [input]	The number of data bytes expected to be received
	usTimeoutMs [input]	Receive timeout (unit: milliseconds); If it is 0, it will exit immediately after receiving data, and exit immediately if there is no data, both returning the number of bytes received
return	>=0	The actual number of bytes received (no larger than the expected length)

	<0	The error code is the result of the negative error code of the PortRecv( ) function
usage	1. The communication receive buffers are all 1K bytes;	

12 Network protocol stack

12.1 Basic definition

12.1.1 Returns a list of values

macro	numeric value	illustrate
NET_OK	0	No errors, normal
NET_ERR_MEM	-1	NULL enough memory
NET_ERR_BUF	-2	Buffer error
NET_ERR_ABRT	-3	An attempt to establish a connection failed
NET_ERR_RST	-4	The connection is reset by the other party (receiving the other party's Reset).
NET_ERR_CLSD	-5	The connection was closed

NET_ERR_CONN	-6	The connection was NULL established successfully
NET_ERR_VAL	-7	Error variables
NET_ERR_ARG	-8	Error parameter
NET_ERR_RTE	-9	Wrong route
NET_ERR_USE	-10	Address and port in use
NET_ERR_IF	-11	The underlying hardware is wrong
NET_ERR_ISCONN	-12	The connection is established
NET_ERR_TIMEOUT	-13	Timeout
NET_ERR_AGAIN	-14	The requested resource does NULL exist, please try again
NET_ERR_EXIST	-15	Already exists
NET_ERR_SYS	-16	The system does NULL support it
NET_ERR_PASSWD	-17	Bad password
NET_ERR_MODEM	-18	Dialing failed
NET_ERR_LINKDOWN	-19	The data link is down, please dial it again
NET_ERR_LOGOUT	-20	Logout
NET_ERR_PPP	-21	PPP disconnects
NET_ERR_STR	-22	The string is too long
NET_ERR_DNS	-23	Domain name resolution error
NET_ERR_INIT	-24	The corresponding functional system is NULL initialized
NET_ERR_NEED_DHCP	-25	DHCP is NULL turned on
NET_ERR_SERV	-30	PPPoE server NULL found
NET_ERR_IRDA_COMM	-54	Communication between landline and mobile phone failed.

12.1.2 Data definition

Structure purpose name:

IP module Socket address structure

```
struct sockaddr_in {

char sin_len;                        /* Length */

char sin_family;                     /* Address cluster */

short sin_port;                      /* Port number */

struct in_addr sin_addr;             /* Network address */
char sin_zero[8];                          /* Alternate domain */
};
struct sockaddr{
char sa_len;                               /*length*/
char sa_family;                            /* Address cluster */
    char sa_data[14];                      /* Protocol address */
};
```

12.2 NetSocket

Prototype	**int NetSocket(int domain, int type, int protocol);**	
function	Creating a network socket is equivalent to creating a connection handle.	
parameter	domain [input]	Fixed value: NET_ AF_INET

	type [input]	Fixed value: NET_SOCK_STREAM;
		NET_ SOCK_STREAM, use TCP;

	protocol [input]	is a fixed value: 0
return	>=0	Success, the value is the created socket
	<0	Failed, the value is the error code
usage	A maximum of 4 sockets can be established on a link established by the module.	

12.3 NetConnect

Prototype	**int NetConnect(int socket, struct net_sockaddr *addr, socklen_t addrlen);**	
function	As a client, start connecting to the server.	
parameter	socket [input]	socket
	addr [input]	Server address information, including IP address and Port, is recommended to initialize using the system-provided api SockAddrSet.
	addrlen [input]	The value is fixed to sizeof(struct net_sockaddr).
	0	succeed

return	<0	Failed, the value is the error code
usage		

12.4 NetSend

Prototype	**int NetSend(int socket,** **void *buf,** **int size,** **int flags);**	
function	Send data to the connecting party.	
parameter	socket [input]	Socket handle
	buf [input]	Holds buffers for sending data
	size [input]	Data length, which must be <=2048(2K).
	flags [input]	The current value is 0
return	>=0	Successfully returns the length of bytes sent
	<0	fail
usage		

12.5 NetRecv

Prototype	**int NetRecv(int socket,** **void *buf,** **int size,** **int flags);**	
function	Receive data.	
parameter	socket [input].	socket
	buf [output].	buffers that receive data; NULL is NULL allowed for this domain.
	size [input].	The length of the buffer
	flags [input].	The current value is 0
return	>=0	Successfully returns the received byte length
	<0	Failed, the value is the error code
usage		

12.6 NetCloseSocket

Prototype	**int NetCloseSocket (int socket);**	
function	Close the socket.	
parameter	socket [input]	Socket handle.
return	0	succeed
	<0	fail
usage		

CAUTION

Handles created through NetScoket must be closed via NetCloseSocket, otherwise the socket creation will fail.

12.7 Netioctl

Prototype	**int Netioctl(int socket,** **int cmd,** **int arg);**	
function	Set up and get information about the socket.	
parameter	socket [input].	socket
	cmd [input].	Operation commands, currently supported: 1. CMD_IO_SET: Set the I/O mode (blocking mode and asynchronous mode); 2. CMD_IO_GET: Get I/O mode; 3. CMD_TO_SET: Set the timeout period, which is effective when the I/O mode is blocked, and the default timeout time time of the system is 20 seconds; 4. CMD_TO_GET: Get the timeout period; 5. CMD_IF_SET: socket binding network api, when the socket is used as a server, the command is invalid; 6. CMD_IF_GET: Get the network api of the socket binding, the command does NULL work when the socket is used as a server; 7. CMD_EVENT_GET: Get socket events, the command is only valid for NET_SOCK_STREAM, possible events are: • SOCK_EVENT_READ: There is data to read; • SOCK_EVENT_WRITE: Data can be sent; • SOCK_EVENT_CONN: Connection succeeded;

		- SOCK_EVENT_NETACCEPT: There are new clients connecting to come; - SOCK_EVENT_ERROR: An error occurred and the connection was disconnected.
	arg [input].	This value has different meanings for different commands; 1. cmd=CMD_IO_SET, arg=1 indicates non-blocking (asynchronous) mode, arg=0 indicates blocking mode; 2. cmd = CMD_IO_GET, arg is meaningless; 3. cmd=CMD_TO_SET, arg>0 means waiting time in milliseconds, arg<=0 means waiting indefinitely; 4. cmd = CMD_TO_GET, arg is meaningless; 5. cmd=CMD_IF_SET, arg represents the network device api index number, arg=0 indicates Ethernet network card, arg=10 indicates PPP link; 6. cmd=CMD_IF_GET, arg is meaningless; 7. cmd=CMD_EVENT_GET, arg=0 means to get events, arg=SOCK_EVENT_READ means to get the current readable data length, arg=SOCK_EVENT_WRITE It means to get the current data length that can be sent, arg=SOCK_EVENT_ERR means to get the error code, arg=SOCK_EVENT_ACCEPT means to get the number of currently waiting for connections;
return	>=0	Succeed. When returned successfully, the return value has different meanings for different commands: 1. cmd=CMD_IO_SET, success returns 0; 2. cmd=CMD_IO_GET, return 1 for non-blocking (asynchronous) mode, return 0 for blocking mode; 3. cmd=CMD_TO_SET, success returns 0; 4. cmd=CMD_TO_GET, return 0 means indefinite wait, return > 0 means waiting time (in milliseconds); 5. cmd=CMD_IF_SET, successfully returns 0; 6. cmd=CMD_IF_GET, which returns the network device api index; 7. cmd=CMD_EVENT_GET, depending on the value of arg, the return value is different: arg=0 returns the NetSocket event;
	<0	Failed with an error code

usage	If you need to keep the connection active, use the following:
	err = Netioctl(s, CMD_KEEPALIVE_SET, ms);//ms is the time in milliseconds
	To view the connection status:
	event = Netioctl(s, CMD_EVENT_GET, 0);
	if(event<0)
	{
	// Code Error
	}else if(event&SOCK_EVENT_ERROR)
	{
	The connection has been disconnected	

	}else if(event&SOCK_EVENT_READ)
	{
	There is data to read
	}

12.8 SockAddrSet

Prototype	**int SockAddrSet(struct net_sockaddr *addr,** **char *ip_str,** **unsignedshort port);**	
function	An api that encapsulates setting socket address information.	
parameter	addr [output]	Used to store socket address information, as an output parameter; NULL is NULL allowed for this domain.
	ip_str [input]	The IP address string, such as "192.168.0.5", is the input parameter; If NULL, the address is 0; The address string cannot be "255.255.255.255".
	port[input]	Port number, such as FTP port number 21.
return	0	succeed
	<0	fail
usage		

12.9 SockAddrGet

Prototype	**int SockAddrGet(struct sockaddr *addr,** **char *ip_str,** **unsignedshort *port);**	
function	Obtain the IP address information and Port information contained in the socket address information.	
parameter	addr [input]	Stores socket address information as an input parameter. NULL is NULL allowed for this domain.
	ip_str [output]	Store the IP address string, such as 192.168.1.5, as the output parameter. The domain is NULL allowed to be NULL; The maximum length of the string is 15 characters.
	port [output]	Store the port number, such as FTP port number 21, as the output parameter. NULL is NULL allowed for this domain.
	0	succeed

return	<0	fail
usage		

12.10 RouteSetDefault

Prototype	**int RouteSetDefault(intifIndex);**	
function	Configure the default route used by the system.	
parameter	ifIndex [input]	Network device index number 0: Indicates an Ethernet network card 1: Represents PPPoE 10: Indicates the modem PPP link 11: Indicates wireless (GPRS/CDMA) PPP link 12: Indicates WIFI link
return	0	succeed
	<0	fail
usage		

WARNING

1. If you want to use the protocol stack api under a network channel, you must call the api RouteSetDefault to set the default network channel, otherwise the protocol stack api pointing error will occur, resulting in communication exceptions.
2. If you want to establish sockets under two different network channels at the same time for communication, you need to call RouteSetDefault at any time to switch the network channel before calling the protocol stack api.

12.11 RouteGetDefault

Prototype	**int RouteGetDefault(void);**	
function	Gets the default route used by the system.	
parameter	NULL	
return	>=0	Network device index number 0: Indicates an Ethernet device 1: Represents PPPoE 10: Indicates the modem PPP link

		11: Indicates wireless (GPRS/CDMA) PPP link 12: Indicates WIFI link
	<0	Error code
usage	If NULL specified by the user, the system uses the default route as follows Ethernet devices are preferred, but if PPPoE is enabled, they are preferred; If the Ethernet device is NULL available, the modem PPP link is used.	

13 Wireless module

13.1 Basic definition

13.1.1 Data definition

Structure purpose name:

```
typedef struct
{
        in32    SimCardSel;
        char SimPin[16];
        char AccessPointName[64];
        char UserName[16];
        char UserKey[16];
        char DialNumber[16];
        char ServerDomain[64];
}ST_WLAN_CFG;
```

13.2 WlInit

Prototype	**int WlInit(ST_WLAN_CFG cfg);**	
function	Initialize the wireless module global variables.	
parameter	cfg [input].	Communication Rotation Arrangement
return	0	Initialization succeeded
	<0	Initialization failed, see Return Value List
usage	1. Call this function once before using the wireless communication module.	

13.3 WlGetSignal

Prototype	**Int WlGetSignal(u8 * SignalLevelOut);**	
function	Query the network signal strength, return the RET_OK function, and then obtain the signal strength value through the parameter SignalLevelOut.	
parameter	SignalLevelOut	A pointer that receives the signal strength Returns a list of values and the corresponding module values and meanings 0x05 (99 or 0): No signal from the network 0x04(1~7): The signal is weak 0x03(8~13): The signal is weak 0x02(14~19): General signal 0x01(20~25): Strong signal 0x00(26~31): The signal is very strong
return	0	Get the signal successfully
	<0	Gets a list of signal failures and network stack module return values
usage	1. The signal icon on the screen is obtained by the application according to the call of the function;	

14 WIFI

14.1 Basic definition

It includes apis for WIFI module scanning, connection, status detection and control.

14.1.1 Returns a list of values

macro	numeric value	illustrate
WIFI_RET_OK	0	No errors, normal
WIFI_RET_ERROR_NODEV	-1	Device error
WIFI_RET_ERROR_NORESPONSE	-2	The WIFI module is unresponsive
WIFI_RET_ERROR_NOTOPEN	-3	The WIFI module is NULL turned on
WIFI_RET_ERROR_NOTCONN	-4	The WIFI module is NULL connected to the AP
WIFI_RET_ERROR_NULL	-5	The parameter is empty
WIFI_RET_ERROR_PARAMERROR	-6	Parameter error
WIFI_RET_ERROR_STATUSERROR	-7	The current state of the WIFI module does NULL support the operation of the api
WIFI_RET_ERROR_AUTHERROR	-9	Password authentication error
WIFI_RET_ERROR_NOAP	-10	Scan AP for abnormalities
WIFI_RET_ERROR_IPCONF	-11	Failed to set or obtain IP address
WIFI_RET_ERROR_NOTSUPPORT	-13	The WIFI module does NULL support this feature
IPLEN	4	IP length
KEY_WEP_LEN_MAX	16	Maximum WEP key length
KEY_WEP_LEN_64	5	
KEY_WEP_LEN_128	13	

KEY_WEP_LEN_152	16	
KEY_WPA_MAXLEN	63	WPA key maximum length
SSID_MAXLEN	32	SSID maximum length
SCAN_AP_MAX	15	The maximum number of APs to be scanned
WLAN_SEC_UNSEC	0	No encryption
WLAN_SEC_WEP	1	WEP encryption
WLAN_SEC_WPA_WPA2	2	WPA/WPA2 encryption
WLAN_SEC_WPAPSK_WPA2PSK	3	WPA-PSK/WPA2-PSK encryption
WLAN_SEC_WPAPSK	4	WPA-PSK encryption
WLAN_SEC_WPA2PSK	5	WPA2-PSK encryption
WIFI_ROUTE_NUM	12	WIFI routing number

14.1.2 Data definition

Structure purpose name:

WIFI module connection parameters and hotspot structure

```
typedef struct {

        u8 Key[4][KEY_WEP_LEN_MAX];         /* WEP password data */

        int KeyLen;                          /* WEP password length 5 or 13 or
                                             16*/

        int Idx;                             /* WEP Key Index [0..3] */

} ST_WEP_KEY;


typedef struct {

        int DhcpEnable;                      /* DHCP enabled, 0 - off, 1 - on */

        u8 Ip[IPLEN];                        /* Static IP*/

        u8 Mask[IPLEN];                      /* mask */

        u8 Gate[IPLEN];                      /* gateway */

        u8 Dns[IPLEN];                       /* DNS */

        U8 psw;                              /* wep password */
        u8 Wpa[KEY_WPA_MAXLEN];                    /* WPA connection password */
}ST_WIFI_PARAM;


typedef struct {
        u8 apName[50+1];                           WIFI hotspot name
        u8 apSignal[5+1];                          WIFI signal strength
        u8 apMode[32+1];                           WIFI hotspot key mode
}ST_WIFIAP;
```

14.2 WifiOpen

Prototype	**int WifiOpen(void);**	
function	Turn on the WIFI module.	
parameter	NULL	
return	0	succeed
	<0	See List of return values
usage	The power-on module includes power-on and initialization	

NOTE

After the wifi module is powered on, it needs to wait 3-5 seconds to work properly. It is recommended to open the wifi module in advance when in use.

14.3 WifiClose

Prototype	**int WifiClose(void);**	
function	Turn off the WIFI module.	
parameter	NULL	
return	0	succeed

usage	Close the module

14.4 WifiScan

Prototype	**int WifiScan (ST_WIFI_AP *outAps, unsigned int ApCount);**	
function	Scan for surrounding hotspots.	
parameter	outAps [output].	List of hotspots to scan
	ApCount [input].	The number of hotspots expected to scan (ApCount > 0).
return	>=0	The number of hotspots actually scanned
	<0	See List of return values
usage	1. Scan hot spots, the array of scanned hotspot structures can be directly used as parameters for connection; If the number of hotspots scanned is greater than ApCount, ApCount is returned. Up to 40 hotspots can be scanned. 。 2. The value of ApCount cannot be greater than the number of user-defined ST_WIFI_AP_EX structs. Failure to do so will cause a memory overflow.	

NOTE

This api can only be called when the WIFI module is turned on and no hotspot is connected.

14.5 WifiConnect

Prototype	**int WifiConnect(ST_WIFI_AP *Ap, ST_WIFI_PARAM *WifiParam);**	
function	Connect the scanned hotspot with certain connection parameters.	
parameter	AP [input].	Scanned hotspots
	WifiParam [input].	Parameters for connecting hotspots
return	0	succeed
	<0	See List of return values
usage	1. When the connection parameter DHCP is enabled, the static IP does NULL need to be set; 2. The WEP password and WPA password settings need to be set which one needs to refer to the encryption method of the Ap	

	3. When the WEP password is set for WEP mode, the password is NULL set for non-encryption methods, and WPA password is set for other encryption methods; 4. The api is non-blocking, and it returns immediately after calling the connection, and then calls WifiCheck to check the connection status.

NOTE

This api can only be called when the WIFI module is open and no hotspot is connected, if the hotspot is connected, you need to call WifiDisconnect before calling this api.

14.6 WifiDisconnect

Prototype	**int WifiDisconnect(void);**	
function	Break the hotspot link.	
parameter	NULL	
return	0	succeed
	<0	See List of return values
usage	Disconnect	

14.7 WifiCheck

Prototype	**int WifiCheck(ST_WIFI_AP *Ap);**	
function	Check the current status of the WIFI module.	
parameter	AP Output	If the pointer is NULL null, the connected hotspot information is returned, see ST_WIFI_AP structure
return	=0	Indicates that you are connecting and need to continue waiting.
	>0	The signal strength level of a successful connection = 1 :< -75dB = 2 ： >= -75dB &&< -65dB = 3 ： >= -65dB
	<0	See List of return values
usage	1. When Ap=NULL only returns the connection status, otherwise the hotspot information is populated; 2. The api returns immediately.	

15 Voice function module

15.1 SoundPlay

Prototype	**int SoundPlay(char *param, char type);**	
function	Audio playback.	
parameter	param [input].	A list of file names or frequencies
	type	1 -- Wave format 2 -- Play audio by frequency list
return	0	succeed
	-1	The input file name is empty, the audio file does NULL exist, or the frequency list parameter is invalid
	-2	Audio formats are NULL supported, including file format errors, frequency NULL supported, interpretation bits are NULL supported, etc
	-3	The system volume is 0
	-4	The device is busy
usage	When Type=1: 1) param is an audio file name, which can be up to 16 characters, the file name length includes the suffix, ending with '\x00', and only the first 16 characters are taken for more than 16 characters. Type=2 时： 1) Param is a list of audio frequencies in the format "Frequency: duration; Frequency: Duration.... ”。 The audio frequency unit is hertz, the effective range is 50~3000, and the duration unit is milliseconds. Supports up to 100 group frequencies. The frequency list parameter is "750:100; 1500:200; ", which means playing 100 milliseconds at 750 Hz, and then 200 ms at 1500 Hz. 2) This function is non-blocking playback.	