Advanced Card Systems Ltd.
Card & Reader Technologies

ACR1283L Standalone Contactless Reader

Application Programming Interface V1.03

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

Table of Contents

List of Figures

List of Tables

1.0. Introduction

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

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

1.1. Symbols and Abbreviations

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

Table 1: Symbols and Abbreviations

2.0. Features

- Dual Operation Modes:
  - PC-linked
  - Standalone
- PC-linked Operation:
  - PC/SC and CCID Compliance
  - Supports CT-API (through wrapper on top of PC/SC)
- Standalone Operation:
  - Support for third-party application programming
  - Over 400 KB memory space for third-party application
  - Over 500 KB memory space for data storage
  - Supported development platform:
    - IAR Embedded Workbench version 5.50 or above
    - CoIDE (GCC) version 1.3.0 or above
- Smart Card Reader:
  - Contactless Interface Read/Write speed of up to 848 Kbps
  - Built-in antenna for contactless tag access, with reading distance of up to 50 mm (depending on tag type)
  - Support for ISO 14443 Part 4 Type A and B and MIFARE series
  - Built-in anti-collision feature (only one tag is accessed at any time)
  - Four ISO 7816-compliant SAM slots (MCU Card, T=0 and T=1)
- Built-in Peripherals:
  - Two-line graphic LCD
  - Four user-controllable LEDs
  - User-controllable buzzer
  - Twelve-key capacitive touch keypad
- USB 2.0 Full Speed Interface for Data Communication, Firmware Upgrade and Power
- In-device AES (128 and 256), DES and 3DES encryption
- Supports Android™ 3.1 and above
- Compliant with the following standards:
  - ISO 14443
  - ISO 7816 (for SAM slot)
  - CE / FCC
  - PC/SC
  - CCID
  - Microsoft® WHQL
  - RoHS 2

3.0. Hardware Design

3.1. USB

The ACR1283L is connected to a computer through a USB following the USB standard.

3.1.1. Communication Parameters

The ACR1283L is connected to a computer through USB as specified in the USB Specification 2.0. The ACR1283L is working in full speed mode, i.e. 12 Mbps.

Pin	Signal	Function
1	V_{BUS}	+5 V power supply for the reader.
2	D-	Differential signal transmits data between ACR1283L and PC.
3	D+	Differential signal transmits data between ACR1283L and PC.
4	GND	Reference voltage level for power supply.

Table 2: USB Interface Wiring

Note: *In order for ACR1283L to function properly through USB interface, the driver should be installed.*

3.1.2. Endpoints

The ACR1283L uses the following endpoints to communicate with the host computer:

Control Endpoint	For setup and control purpose.
Bulk OUT	For command to sent from host to ACR1283L (data packet size is 64 bytes).
Bulk IN	For response to sent from ACR1283L to host (data packet size is 64 bytes).
Interrupt IN	For card status message to sent from ACR1283L to host (data packet size is 8 bytes).

3.2. Contact Smart Card Interface

The interface between the ACR1283L and the inserted smart card follows the specifications of ISO 7816-3 with certain restrictions or enhancements to increase the practical functionality of the ACR1283L.

3.2.1. Smart Card Power Supply VCC (C1)

The current consumption of the inserted card must not be higher than 50 mA.

3.2.2. Card Type Selection

Before activating the inserted card, the controlling PC always needs to select the card type through the proper command sent to the ACR1283L. This includes both memory card and MCU-based cards.

For MCU-based cards the reader allows to select the preferred protocol, T=0 or T=1. However, this selection is only accepted and carried out by the reader through the PPS when the card inserted in the reader supports both protocol types. Whenever a MCU-based card supports only one protocol type, T=0 or T=1, the reader automatically uses that protocol type, regardless of the protocol type selected by the application.

3.2.3. Interface for Microcontroller-based Cards

For microcontroller-based smart cards only the contacts C1 (VCC), C2 (RST), C3 (CLK), C5 (GND) and C7 (I/O) are used. A frequency of 4.8 MHz is applied to the CLK signal (C3).

3.3. Contactless Smart Card Interface

The interface between the ACR1283L and the contactless interface follows the specifications of ISO 14443 with certain restrictions or enhancements to increase the practical functionality of the ACR1283L.

3.3.1. Carrier Frequency

The carrier frequency for ACR1283L is 13.56 MHz.

3.3.2. Card Polling

The ACR1283L automatically polls the contactless cards that are within the field. ISO 14443-4 Type A, ISO 14443-4 Type B and MIFARE cards are supported.

4.0. USB Communication Protocol

Communication data frame package follows CCID Rev. 1.1 standard. Please see the reference note and sample description below for more information.

Note: *Does not support extended APDU.*

4.1. Data Package Format

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

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

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

Notes:

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

4.2. Sample Code

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

Explanation:

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

Information Type 6Fh
Note: *For PC_to_RDR_XfrBlock, please refer to CCID Rev 1.1.*
Data Length 00000005 >> 5-byte user data
Card Slot 01h >> Data send to SAM1
Serial Number 11h
Special Usage 00 00 00h
User Data 80 84 00 00 08h

4.3. CCID Protocol

ACR1283L shall interface with the host with USB connection. A specification, namely CCID, has been released within the industry defining such a protocol for the USB chip-card interface devices. CCID covers all the protocols required for operating smart cards and PIN.

The configurations and usage of USB endpoints on ACR1283L shall follow CCID section 3. An overview is summarized below:

- *Control Commands* are sent on control pipe (default pipe). These include class-specific requests and USB standard requests. Commands that are sent on the default pipe report information back to the host on the default pipe.
- *CCID Events* are sent on the interrupt pipe.
- *CCID Commands* are sent on BULK-OUT endpoint. Each command sent to ACR1283L has an associated ending response. Some commands can also have intermediate responses.
- *CCID Responses* are sent on BULK-IN endpoint. All commands sent to ACR1283L have to be sent synchronously. (i.e. *bMaxCCIDBusySlots* is equal to 1 for ACR1283L).

The supported CCID features by ACR1283L are indicated in its class descriptor:

Offset	Field	Size	Value	Description
0	*bLength*	1	36h	Size of this descriptor, in bytes.
1	*bDescriptorType*	1	21h	CCID Functional Descriptor type.
2	*bcdCCID*	2	0110h	CCID Specification Release Number in Binary-Coded decimal.
4	*bMaxSlotIndex*	1	04h	Five slots are available on ACR1283L.
5	*bVoltageSupport*	1	07h	ACR1283L can supply 1.8 V, 3.0 V and 5.0 V to its slot.
6	*dwProtocols*	4	00000003h	ACR1283L supports T=0 and T=1 Protocol.
10	*dwDefaultClock*	4	00000FA0h	Default ICC clock frequency is 4 MHz.
14	*dwMaximumClock*	4	00000FA0h	Maximum supported ICC clock frequency is 4 MHz.
18	*bNumClockSupported*	1	00h	Does not support manual setting of clock frequency.
19	*dwDataRate*	4	00002A00h	Default ICC I/O data rate is 10752 bps.
23	*dwMaxDataRate*	4	0001F808h	Maximum supported ICC I/O data rate is 344100 bps.
27	*bNumDataRatesSupported*	1	00h	Does not support manual setting of data rates.
28	*dwMaxIFSD*	4	00000200h	Maximum IFSD supported by ACR1283L for protocol T=1 is 512.
32	*dwSynchProtocols*	4	00000000h	ACR1283L does not support synchronous card.

Offset	Field	Size	Value	Description
36	*dwMechanical*	4	00000000h	ACR1283L does not support special mechanical characteristics.
40	*dwFeatures*	4	00040040h	ACR1283L supports the following features: • Automatic parameters negotiation made by the CCID • Short and Extended APDU level exchange with CCID
44	*dwMaxCCIDMessageLength*	4	0000020Ah	Maximum message length accepted by ACR1283L is 522 bytes.
48	*bClassGetResponse*	1	00h	Indicates that the CCID echoes the class of the APDU.
49	*bClassEnvelope*	1	00h	Indicates that the CCID echoes the class of the APDU.
50	*wLCDLayout*	2	0000h	No LCD.
52	*bPINSupport*	1	00h	No PIN Verification.
53	*bMaxCCIDBusySlots*	1	01h	Only one slot can be simultaneously busy.

4.4. CCID Commands

4.4.1. CCID Command Pipe Bulk-OUT Messages

ACR1283L shall follow the CCID Bulk-OUT Messages as specified in CCID section 4. In addition, this specification defines some extended commands for operating additional features. This section lists the CCID Bulk-OUT Messages to be supported by ACR1283L.

4.4.1.1. PC_to_RDR_IccPowerOn

This command is used to activate the card slot, and then return ATR from the card.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	62h	
1	*dwLength*	4	00000000h	Size of extra bytes of this message.
2	*bSlot*	1		Identifies the slot number for this command.
5	*bSeq*	1		Sequence number for command.
6	*bPowerSelect*	1		Voltage that is applied to the ICC. 00h = Automatic Voltage Selection 01h = 5 V 02h = 3 V
7	*abRFU*	2		Reserved for future use.

The response to this message is the *RDR_to_PC_DataBlock* message and the data returned is the

Answer To Reset (ATR) data.

4.4.1.2. PC_to_RDR_IccPowerOff

This command is used to deactivate the card slot.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	63h	
1	*dwLength*	4	00000000h	Size of extra bytes of this message.
5	*bSlot*	1		Identifies the slot number for this command.
6	*bSeq*	1		Sequence number for command.
7	*abRFU*	3		Reserved for future use.

The response to this message is the *RDR_to_PC_SlotStatus* message.

4.4.1.3. PC_to_RDR_GetSlotStatus

This command is used to get the current status of the slot.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	65h	
1	*dwLength*	4	00000000h	Size of extra bytes of this message.
5	*bSlot*	1		Identifies the slot number for this command.
6	*bSeq*	1		Sequence number for command.
7	*abRFU*	3		Reserved for future use.

The response to this message is the *RDR_to_PC_SlotStatus* message.

4.4.1.4. PC_to_RDR_XfrBlock

This command is used to transfer data block to the ICC.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	6Fh	
1	*dwLength*	4		Size of *abData* field of this message
5	*bSlot*	1		Identifies the slot number for this command
6	*bSeq*	1		Sequence number for command
7	*bBWI*	1		Used to extend the CCIDs Block Waiting Timeout for this current transfer. The CCID will timeout the block after “this number multiplied by the Block Waiting Time” has expired.
8	*wLevelParameter*	2	0000h	RFU (TPDU exchange level)
10	*abData*	Byte array		Data block sent to the CCID. Data is sent “as is” to the ICC (TPDU exchange level)

The response to this message is the *RDR_to_PC_DataBlock* message.

4.4.1.5. PC_to_RDR_Escape

This command is used to access extended features.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	6Bh	
1	*dwLength*	4		Size of *abData* field of this message.
5	*bSlot*	1		Identifies the slot number for this command.
6	*bSeq*	1		Sequence number for command.
7	*abRFU*	3		Reserved for Future Use.
10	*abData*	Byte array		Data block sent to the CCID.

The response to this command message is the *RDR_to_PC_Escape* response message.

4.4.1.6. PC_to_RDR_GetParameters

This command is used to get slot parameters.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	6Ch	
1	*dwLength*	4	00000000h	Size of extra bytes of this message.
5	*bSlot*	1		Identifies the slot number for this command.
6	*bSeq*	1		Sequence number for command.
7	*abRFU*	3		Reserved for future use.

The response to this message is the *RDR_to_PC_Parameters* message.

4.4.1.7. PC_to_RDR_ResetParameters

This command is used to reset slot parameters to default value.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	6Dh	
1	*DwLength*	4	00000000h	Size of extra bytes of this message.
5	*BSlot*	1		Identifies the slot number for this command.
6	*BSeq*	1		Sequence number for command.
7	*AbRFU*	3		Reserved for future use.

The response to this message is the *RDR_to_PC_Parameters* message.

4.4.1.8. PC_to_RDR_SetParameters

This command is used to set the slot parameters.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	61h	
1	*dwLength*	4		Size of extra bytes of this message.

Offset	Field	Size	Value	Description
5	*bSlot*	1		Identifies the slot number for this command.
6	*bSeq*	1		Sequence number for command.
7	*bProtocolNum*	1		Specifies what protocol data structure follows. 00h = Structure for protocol T=0 01h = Structure for protocol T=1 The following values are reserved for future use: 80h = Structure for 2-wire protocol 81h = Structure for 3-wire protocol 82h = Structure for I2C protocol
8	*abRFU*	2		Reserved for future use.
10	*abProtocolDataStructure*	Byte array		Protocol Data Structure.

Protocol Data Structure for Protocol T=0 (*dwLength*=00000005h)

Offset	Field	Size	Value	Description
10	*bmFindexDindex*	1		B7-4 – FI – Index into the table 7 in ISO/IEC 7816-3:1997 selecting a clock rate conversion factor B3-0 – DI - Index into the table 8 in ISO/IEC 7816-3:1997 selecting a baud rate conversion factor
11	*bmTCCKST0*	1		B0 – 0b, B7-2 – 000000b B1 – Convention used (b1=0 for direct, b1=1 for inverse) ***Note:*** *The CCID ignores this bit.*
12	*bGuardTimeT0*	1		Extra Guardtime between two characters. Add 0 to 254 etu to the normal guardtime of 12etu. FFh is the same as 00h.
13	*bWaitingIntegerT0*	1		WI for T=0 used to define WWT
14	*bClockStop*	1		ICC Clock Stop Support. 00h = Stopping the Clock is not allowed 01h = Stop with Clock signal Low 02h = Stop with Clock signal High 03h = Stop with Clock either High or Low

Protocol Data Structure for Protocol T=1 (*dwLength*=00000007h)

Offset	Field	Size	Value	Description
10	*bmFindexDindex*	1		B7-4 – FI – Index into the table 7 in ISO/IEC 7816-3:1997 selecting a clock rate conversion factor B3-0 – DI - Index into the table 8 in ISO/IEC 7816-3:1997 selecting a baud rate

				conversion factor
11	*BmTCCKST1*	1		B7-2 – 000100b B0 – Checksum type (b0=0 for LRC, b0=1 for CRC B1 – Convention used (b1=0 for direct, b1=1 for inverse) ***Note:*** *The CCID ignores this bit.*
12	*BGuardTimeT1*	1		Extra Guardtime (0 to 254 etu between two characters). If value is FFh, then guardtime is reduced by 1 etu.
13	*BwaitingIntegerT1*	1		B7-4 = BWI values 0-9 valid B3-0 = CWI values 0-Fh valid
14	*bClockStop*	1		ICC Clock Stop Support. 00h = Stopping the Clock is not allowed 01h = Stop with Clock signal Low 02h = Stop with Clock signal High 03h = Stop with Clock either High or Low
15	*bIFSC*	1		Size of negotiated IFSC
16	*bNadValue*	1	00h	Only support NAD = 00h

The response to this message is the *RDR_to_PC_Parameters* message.

4.4.2. CCID Bulk-IN Messages

The Bulk-IN messages are used in response to the Bulk-OUT messages. ACR1283L shall follow the CCID Bulk-IN Messages as specified in CCID section 4. This section lists the CCID Bulk-IN Messages to be supported by ACR1283L.

4.4.2.1. RDR_to_PC_DataBlock

This message is sent by ACR1283L in response to *PC_to_RDR_IccPowerOn*, *PC_to_RDR_XfrBlock* and *PC_to_RDR_Secure* messages.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	80h	Indicates that a data block is being sent from the CCID.
1	*dwLength*	4		Size of extra bytes of this message.
5	*bSlot*	1		Same value as in Bulk-OUT message.
6	*bSeq*	1		Same value as in Bulk-OUT message.
7	*bStatus*	1		Slot status register as defined in CCID section 4.2.1.
8	*bError*	1		Slot error register as defined in CCID section 4.2.1 and this specification section 5.2.8.
9	*bChainParameter*	1	00h	RFU (TPDU exchange level).
10	*abData*	Byte array		This field contains the data returned by the CCID.

4.4.2.2. RDR_to_PC_Escape

This message is sent by ACR1283L in response to *PC_to_RDR_Escape* messages.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	83h	
1	*dwLength*	4		Size of *abData* field of this message.
5	*bSlot*	1		Same value as in Bulk-OUT message.
6	*bSeq*	1		Same value as in Bulk-OUT message.
7	*bStatus*	1		Slot status register as defined in CCID section 4.2.1.
8	*bError*	1		Slot error register as defined in CCID section 4.2.1 and this specification section 5.2.8.
9	*bRFU*	1	00h	RFU.
10	*abData*	Byte array		This field contains the data returned by the CCID.

4.4.2.3. RDR_to_PC_SlotStatus

This message is sent by ACR1283L in response to *PC_to_RDR_IccPowerOff*, *PC_to_RDR_GetSlotStatus*, *PC_to_RDR_Abort* messages and class-specific ABORT request.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	81h	
1	*dwLength*	4	00000000h	Size of extra bytes of this message.
5	*bSlot*	1		Same value as in Bulk-OUT message.
6	*bSeq*	1		Same value as in Bulk-OUT message.
7	*bStatus*	1		Slot status register as defined in CCID section 4.2.1.
8	*bError*	1		Slot error register as defined in CCID section 4.2.1 and this specification section 5.2.8.
9	*bClockStatus*	1		Value: 00h = Clock running 01h = Clock stopped in state L 02h = Clock stopped in state H 03h = Clock stopped in an unknown state All other values are RFU.

4.4.2.4. RDR_to_PC_Parameters

This message is sent by ACR1283L in response to *PC_to_RDR_GetParameters*, *PC_to_RDR_ResetParameters* and *PC_to_RDR_SetParameters* messages.

Offset	Field	Size	Value	Description
0	*bMessageType*	1	82h	

Offset	Field	Size	Value	Description
1	*dwLength*	4		Size of extra bytes of this message.
5	*bSlot*	1		Same value as in Bulk-OUT message.
6	*bSeq*	1		Same value as in Bulk-OUT message.
7	*bStatus*	1		Slot status register as defined in CCID section 4.2.1.
8	*bError*	1		Slot error register as defined in CCID section 4.2.1 and this specification section 5.2.8.
9	*bProtocolNum*	1		Specifies what protocol data structure follows. 00h = Structure for protocol T=0 01h = Structure for protocol T=1 The following values are reserved for future use: 80h = Structure for 2-wire protocol 81h = Structure for 3-wire protocol 82h = Structure for I2C protocol
10	*abProtocolDataStructure*	Byte array		Protocol Data Structure as summarized in section 5.2.3.

5.0. Host Programming (PC-linked Mode) API

5.1. PCSC API

This section describes some of the PCSC API for application programming usage. For more details, please refer to the Microsoft MSDN Library or PCSC workgroup.

5.1.1. SCardEstablishContext

The *SCardEstablishContext* function establishes the resource manager context within which database operations are performed.

Refer: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379479%28v=vs.85%29.aspx

5.1.2. SCardListReaders

The *SCardListReaders* function provides the list of readers within a set of named reader groups, eliminating duplicates.

The caller supplies a list of reader groups, and receives the list of readers within the named groups. Unrecognized group names are ignored. This function only returns readers within the named groups that are currently attached to the system and available for use.

Refer to: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379793%28v=vs.85%29.aspx

5.1.3. SCardConnect

The *SCardConnect* function establishes a connection (using a specific resource manager context) between the calling application and a smart card contained by a specific reader. If no card exists in the specified reader, an error is returned.

Refer to: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379473%28v=vs.85%29.aspx

5.1.4. SCardControl

The *SCardControl* function gives you direct control of the reader. You can call it any time after a successful call to *SCardConnect* and before a successful call to *SCardDisconnect*. The effect on the state of the reader depends on the control code.

Refer to: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379474%28v=vs.85%29.aspx

Note: *Commands from* ***Section 5.4*** *– Peripherals Control are using this API for sending.*

5.1.5. ScardTransmit

The *SCardTransmit* function sends a service request to the smart card and expects to receive data back from the card.

Refer to: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379804%28v=vs.85%29.aspx

Note: *APDU Commands (i.e., commands sent to the connected card and* ***Section 5.3*** *– Pseudo APDUs for Contactless Interface") are using this API for sending.*

5.1.6. ScardDisconnect

The *SCardDisconnect* function terminates a connection previously opened between the calling application and a smart card in the target reader.

Refer to: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379475%28v=vs.85%29.aspx

5.1.7. APDU Flow

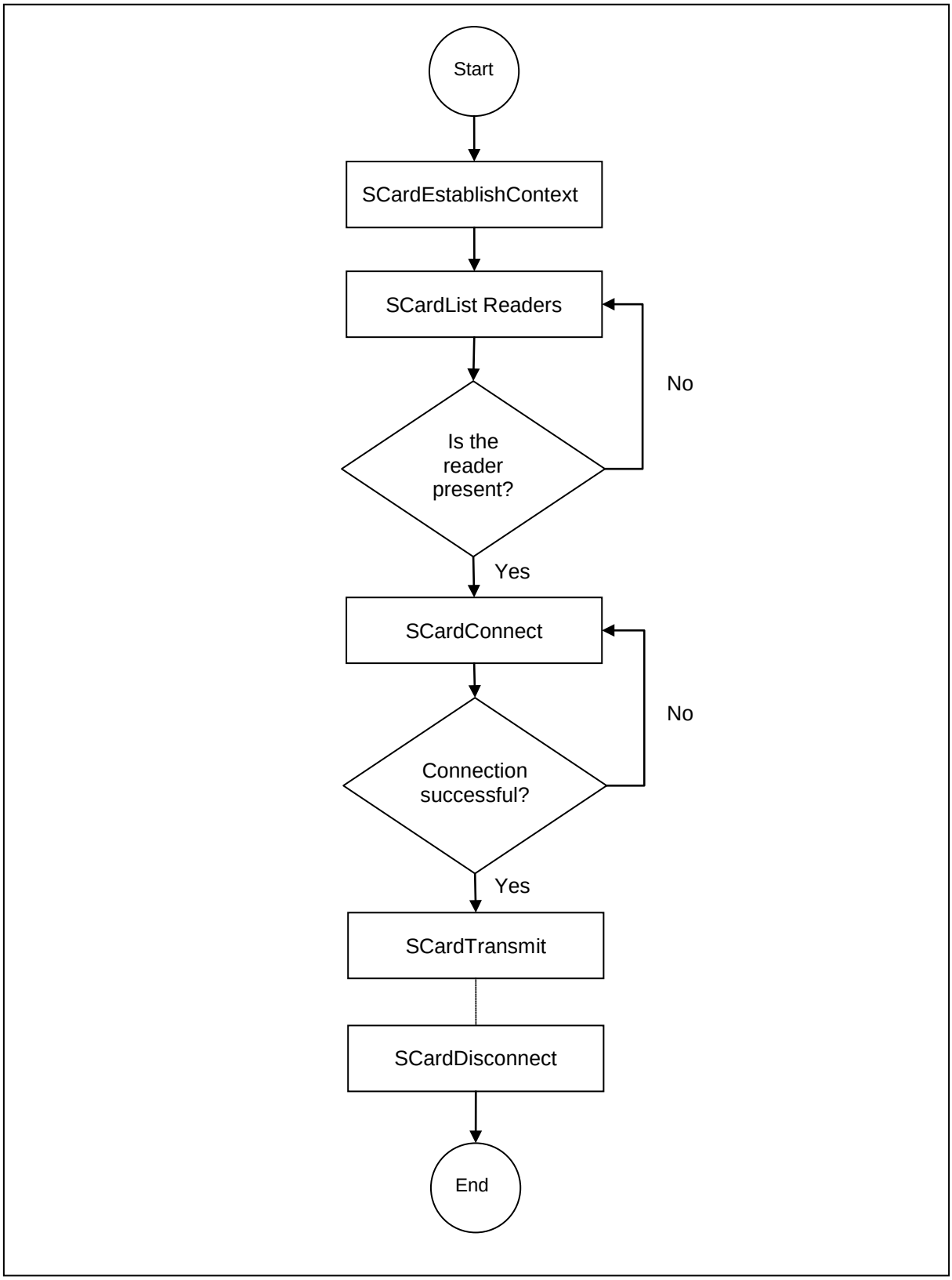


Figure 1: APDU Flow

5.1.8. Escape Command Flow

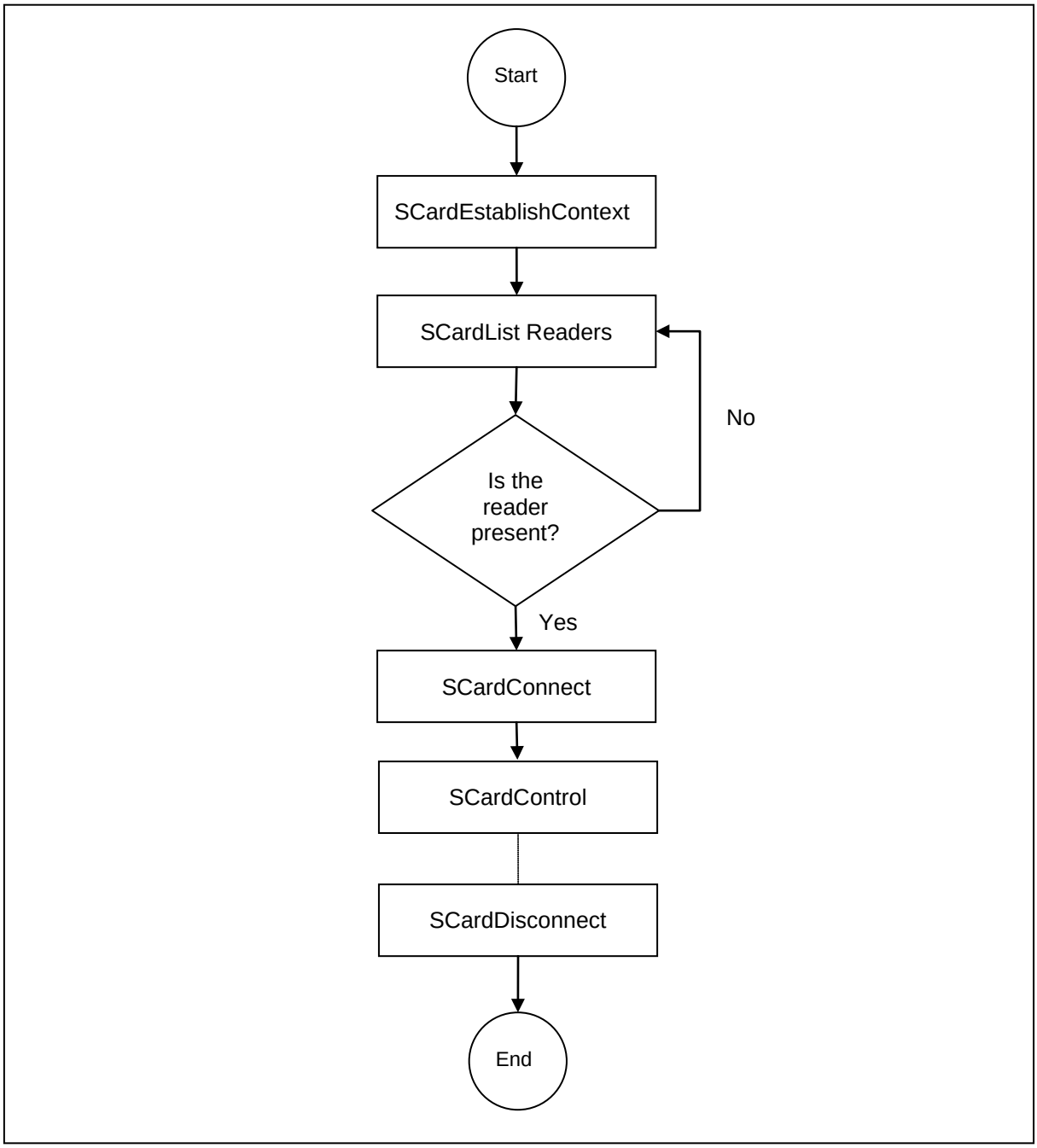


Figure 2: Escape Command Flow

5.2. Contactless Smart Card Protocol

5.2.1. ATR Generation

If the reader detects a PICC, an ATR is sent to the PCSC driver for identifying the PICC.

5.2.1.1. ATR Format for ISO 14443 Part 3 PICCs

Byte	Value (Hex)	Designation	Description
0	3B	Initial Header	
1	8N	T0	Higher nibble 8 means: no TA1, TB1, TC1 only TD1 is following. Lower nibble N is the number of historical bytes (HistByte 0 to HistByte N-1)
2	80	TD1	Higher nibble 8 means: no TA2, TB2, TC2 only TD2 is following. Lower nibble 0 means T = 0
3	01	TD2	Higher nibble 0 means no TA3, TB3, TC3, TD3 following. Lower nibble 1 means T = 1
4 to 3+N	80	T1	Category indicator byte, 80 means A status indicator may be present in an optional COMPACT-TLV data object.
	4F	Tk	Application identifier Presence Indicator.
	0C		Length.
	RID		Registered Application Provider Identifier (RID) # A0 00 00 03 06
	SS		Byte for standard.
	C0 .. C1		Bytes for card name.
	00 00 00 00	RFU	RFU # 00 00 00 00
4+N	UU	TCK	Exclusive-oring of all the bytes T0 to Tk

Example:

ATR for MIFARE 1K = {3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6Ah}

Length (YY) = 0Ch

RID = {A0 00 00 03 06h} (PC/SC Workgroup)

Standard (SSh) = 03h (ISO 14443A, Part 3)

Card Name (C0 .. C1h) = {00 01h} (Mifare 1K)

Card Name (C0 .. C1)

00 01: MIFARE 1K

00 02: MIFARE 4K

00 03: MIFARE Ultralight

FF 28: JCOP 30

FF [SAK]: Undefined tags

00 26: MIFARE Mini

5.2.1.2. ATR Format for ISO 14443 Part 4 PICCs

<table>
<tr><th>Byte</th><th>Value (Hex)</th><th>Designation</th><th>Description</th></tr>
<tr><td>0</td><td>3B</td><td>Initial Header</td><td></td></tr>
<tr><td>1</td><td>8N</td><td>T0</td><td>Higher nibble 8 means: no TA1, TB1, TC1 only TD1 is following.
Lower nibble N is the number of historical bytes (HistByte 0 to HistByte N-1)</td></tr>
<tr><td>2</td><td>80</td><td>TD1</td><td>Higher nibble 8 means: no TA2, TB2, TC2 only TD2 is following.
Lower nibble 0 means T = 0</td></tr>
<tr><td>3</td><td>01</td><td>TD2</td><td>Higher nibble 0 means no TA3, TB3, TC3, TD3 following.
Lower nibble 1 means T = 1</td></tr>
<tr><td rowspan="2">4 to 3 + N</td><td>XX</td><td>T1</td><td rowspan="2">Historical Bytes:

ISO 14443-A:
The historical bytes from ATS response. Refer to the ISO 14443-4 specification.

ISO 14443-B:
<table><tr><th>Byte1-4</th><th>Byte5-7</th><th>Byte8</th></tr><tr><td>Application Data from ATQB</td><td>Protocol Info Byte from ATQB</td><td>Higher nibble=MBLI from ATTRIB command
Lower nibble (RFU)=0</td></tr></table></td></tr>
<tr><td>XX
XX
XX</td><td>Tk</td></tr>
<tr><td>4+N</td><td>UU</td><td>TCK</td><td>Exclusive-oring of all the bytes T0 to Tk</td></tr>
</table>

Example 1:

ATR for MIFARE DESFire = { 3B 81 80 01 80 80h } // 6 bytes of ATR

Note: *Use the APDU "FF CA 01 00 00h" to distinguish the ISO 14443A-4 and ISO 14443B-4 PICCs, and retrieve the full ATS if available. ISO 14443A-3 or ISO 14443B-3/4 PICCs do have ATS returned.*

APDU Command = FF CA 01 00 00h

APDU Response = 06 75 77 81 02 80 90 00h

ATS = {06 75 77 81 02 80h}

Example 2:

ATR for EZ-Link = {3B 88 80 01 1C 2D 94 11 F7 71 85 00 BEh}

Application Data of ATQB = 1C 2D 94 11h

Protocol Information of ATQB = F7 71 85h

MBLI of ATTRIB = 00h

5.3. Pseudo APDUs for Contactless Interface

5.3.1. Get Data

This command returns the serial number or ATS of the “connected PICC.”

Get UID APDU Format (5 Bytes)

Command	Class	INS	P1	P2	Le
Get Data	FFh	CAh	00h 01h	00h	00h (Max. Length)

If P1 = 00h, Get UID Response Format (UID + 2 Bytes)

Response	Data Out					
Result	UID (LSB)	…	…	UID (MSB)	SW1	SW2

If P1 = 01h, Get ATS of a ISO 14443 A card (ATS + 2 Bytes)

Response	Data Out		
Result	ATS	SW1	SW2

Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Warning	62 82h	End of UID/ATS reached before Le bytes (Le is greater than UID Length).
Error	6C XXh	Wrong length (wrong number *Le*: ‘XXh’ encodes the exact number) if *Le* is less than the available UID length.
Error	63 00h	The operation is failed.
Error	6A 81h	Function not supported.

Examples:

// To get the serial number of the “connected PICC.”

UINT8 GET_UID[5]={FFh, CAh, 00h, 00h, 00h};

// To get the ATS of the “connected ISO 14443-A PICC.”

UINT8 GET_ATS[5]={FFh, CAh, 01h, 00h, 00h};

5.3.2. PICC Commands (T=CL Emulation) for MIFARE 1K/4K Memory Cards

5.3.2.1. Load Authentication Keys

This command loads the authentication keys into the reader. The authentication keys are used to authenticate the particular sector of the MIFARE 1K/4K Memory Card. Two kinds of authentication key locations are provided: volatile and non-volatile key locations.

Load Authentication Keys APDU Format (11 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Load Authentication Keys	FFh	82h	Key Structure	Key Number	06h	Key (6 bytes)

Where:

Key Structure (1 Byte) 00h = Key is loaded into the reader volatile memory.

20h = Key is loaded into the reader non-volatile memory.

Other = Reserved.

Key Number (1 Byte)

00h ~ 1Fh = Non-volatile memory for storing keys. The keys are permanently stored in the reader and will not disappear even the reader is disconnected from the PC. It can store up to 32 keys inside the reader non-volatile memory.

20h (Session Key) = Volatile memory for storing a temporary key. The key will disappear once the reader is disconnected from the PC. Only 1 volatile key is provided. The volatile key can be used as a session key for different sessions.

Note: *Default Value = {FF FF FF FF FF FFh}*

Key (6 Bytes) The key value loaded into the reader, e.g. {FF FF FF FF FF FFh}

Load Authentication Keys Response Format (2 Bytes)

Response	Data Out	
Result	SW1	SW2

Load Authentication Keys Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

Examples:

// Load a key {FF FF FF FF FF FFh} into the non-volatile memory location 05h.

APDU = {FF 82 20 05 06 FF FF FF FF FF FFh}

// Load a key {FF FF FF FF FF FFh} into the volatile memory location 20h.

APDU = {FF 82 00 20 06 FF FF FF FF FF FFh}

Notes:

1. *Basically, the application should know all the keys being used. It is recommended to store all the required keys to the non-volatile memory for security reasons. The contents of both volatile and non-volatile memories are not readable by the outside world.*
2. *The content of the volatile memory "Session Key 20h" remains valid until the reader is reset or power-off. The session key is useful for storing any key value that is changing from time to time. The session key is stored in the "Internal RAM", while the non-volatile keys are stored in "EEPROM" that is relatively slower than "Internal RAM".*
3. *It is not recommended to use the "non-volatile key locations 00h ~ 1Fh" to store any "temporary key value" that will be changed so often. The "non-volatile keys" are supposed to be used for storing any "key value" that will not change frequently. If the "key value" is supposed to be changed from time to time, please store the "key value" to the "volatile key location 020h."*

5.3.2.2. Authentication for MIFARE 1K/4K

This command uses the keys stored in the reader to authenticate the MIFARE 1K/4K card (PICC). Two types of authentication keys are used: TYPE_A and TYPE_B.

Load Authentication Keys APDU Format (6 Bytes) (Obsolete)

Command	Class	INS	P1	P2	P3	Data In
Authentication	FFh	88h	00h	Block Number	Key Type	Key Number

Load Authentication Keys APDU Format (10 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Authentication	FFh	86h	00h	00h	05h	Authenticate Data Bytes

Authenticate Data Bytes (5 Byte)

Byte1	Byte 2	Byte 3	Byte 4	Byte 5
Version 01h	00h	Block Number	Key Type	Key Number

Where:

Block Number (1 Byte) The memory block to be authenticated.

For MIFARE 1K card, it has totally 16 sectors and each sector consists of 4 consecutive blocks. For example, Sector 00h consists of Blocks {00h, 01h, 02h and 03h}; Sector 01h consists of Blocks {04h, 05h, 06h and 07h}; the last sector 0Fh consists of Blocks {3Ch, 3Dh, 3Eh and 3Fh}. Once the authentication is done successfully, there is no need to do the authentication again provided that the blocks to be accessed are belonging to the same sector. Please refer to the MIFARE 1K/4K specification for more details.

Note: *Once the block is authenticated successfully, all the blocks belonging to the same sector are accessible.*

Key Type (1 Byte) 60h = Key is used as a TYPE A key for authentication

61h = Key is used as a TYPE B key for authentication

Key Number (1 Byte)

00h ~ 1Fh = Non-volatile memory for storing keys. The keys are permanently stored in the reader and will not disappear even the reader is disconnected from the PC. It can store 32 keys into the reader non-volatile memory.

20h (Session Key) = Volatile memory for storing keys. The keys will disappear when the reader is disconnected from the PC. Only 1 volatile key is provided. The volatile key can be used as a session key for different sessions.

Load Authentication Keys Response Format (2 Bytes)

Response	Data Out	
Result	SW1	SW2

Load Authentication Keys Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

Sectors (Total 16 sectors. Each sector consists of 4 consecutive blocks)	Data Blocks (3 blocks, 16 bytes per block)	Trailer Block (1 block, 16 bytes)
Sector 0	00h ~ 02h	03h
Sector 1	04h ~ 06h	07h
..		
..		
Sector 14	38h ~ 0Ah	3Bh
Sector 15	3Ch ~ 3Eh	3Fh

1K Bytes

Table 3: MIFARE 1K Memory Map

Sectors (Total 32 sectors. Each sector consists of 4 consecutive blocks)	Data Blocks (3 blocks, 16 bytes per block)	Trailer Block (1 block, 16 bytes)
Sector 0	00h ~ 02h	03h
Sector 1	04h ~ 06h	07h
..		
..		
Sector 30	78h ~ 7Ah	7Bh
Sector 31	7Ch ~ 7Eh	7Fh
Sector 32	80h ~ 8Eh	8Fh
Sector 33	90h ~ 9Eh	9Fh
..		
..		
Sector 38	E0h ~ EEh	EFh
Sector 39	F0h ~ FEh	FF

4K Bytes

Table 4: MIFARE 4K Memory Map

Examples:

// To authenticate the Block 04h with a {TYPE A, key number 00h}.

// PC/SC V2.01, Obsolete

APDU = {FF 88 00 04 60 00h};

// To authenticate the Block 04h with a {TYPE A, key number 00h}.

// PC/SC V2.07

APDU = {FF 86 00 00 05 01 00 04 60 00h}

Note: *MIFARE Ultralight does not need to do any authentication. The memory is free to access.*

Byte Number	0	1	2	3	Page
Serial Number	SN0	SN1	SN2	BCC0	0
Serial Number	SN3	SN4	SN5	SN6	1
Internal/Lock	BCC1	Internal	Lock0	Lock1	2
OTP	OPT0	OPT1	OTP2	OTP3	3
Data read/write	Data0	Data1	Data2	Data3	4
Data read/write	Data4	Data5	Data6	Data7	5
Data read/write	Data8	Data9	Data10	Data11	6
Data read/write	Data12	Data13	Data14	Data15	7
Data read/write	Data16	Data17	Data18	Data19	8
Data read/write	Data20	Data21	Data22	Data23	9
Data read/write	Data24	Data25	Data26	Data27	10
Data read/write	Data28	Data29	Data30	Data31	11
Data read/write	Data32	Data33	Data34	Data35	12
Data read/write	Data36	Data37	Data38	Data39	13
Data read/write	Data40	Data41	Data42	Data43	14
Data read/write	Data44	Data45	Data46	Data47	15

512 bits
Or
64 bytes

Table 5: MIFARE Ultralight Memory Map

5.3.2.3. Read Binary Blocks

This command is used for retrieving a multiple of "data blocks" from the PICC. The data block/trailer block must be authenticated first before executing the *Read Binary Blocks* command.

Read Binary APDU Format (5 Bytes)

Command	Class	INS	P1	P2	Le
Read Binary Blocks	FFh	B0h	00h	Block Number	Number of Bytes to Read

Where:

Block Number (1 Byte) — The starting block.

Number of Bytes to Read (1 Byte) — Multiple of 16 bytes for MIFARE 1K/4K or multiple of 4 bytes for MIFARE Ultralight

- Maximum 16 bytes for MIFARE Ultralight.
- Maximum 48 bytes for MIFARE 1K (Multiple Blocks Mode; 3 consecutive blocks).
- Maximum 240 bytes for MIFARE 4K (Multiple Blocks Mode; 15 consecutive blocks).

Example 1: 10h (16 bytes). The starting block. (Single Block Mode)

Example 2: 40h (64 bytes). From the starting block to starting block +3. (Multiple Blocks Mode)

Note: *For safety reason, the Multiple Block Mode is used for accessing Data Blocks only. The Trailer Block is not supposed to be accessed in Multiple Blocks Mode. Please use Single Block Mode to access the Trailer Block.*

Read Binary Block Response Format (Multiple of 4/16 + 2 Bytes)

Response	Data Out		
Result	Data (Multiple of 4/16 Bytes)	SW1	SW2

Read Binary Block Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

Examples:

// Read 16 bytes from the binary block 04h (MIFARE 1K or 4K)

APDU = {FF B0 00 04 10h}

// Read 240 bytes starting from the binary block 80h (MIFARE 4K)

// Block 80h to Block 8Eh (15 blocks)

APDU = {FF B0 00 80 F0h}

5.3.2.4. Update Binary Blocks

This command is used for writing a multiple of "data blocks" into the PICC. The data block/trailer block must be authenticated first before executing the *Update Binary Blocks* command.

Update Binary APDU Format (Multiple of 16 + 5 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Update Binary Blocks	FFh	D6h	00h	Block Number	Number of Bytes to Update	Block Data (Multiple of 16 Bytes)

Where:

Block Number (1 Byte) The starting block to be updated.

Number of Bytes to Update (1 Byte)

- Multiple of 16 bytes for MIFARE 1K/4K or 4 bytes for MIFARE Ultralight
- Maximum 48 bytes for MIFARE 1K (Multiple Blocks Mode; 3 consecutive blocks)
- Maximum 240 bytes for MIFARE 4K (Multiple Blocks Mode; 15 consecutive blocks)

Example 1: 10h (16 bytes). The starting block only. (Single Block Mode)

Example 2: 30h (48 bytes). From the starting block to starting block +2. (Multiple Blocks Mode)

Note: *For safety reason, the Multiple Block Mode is used for accessing Data Blocks only. The Trailer Block is not supposed to be accessed in Multiple Blocks Mode. Please use Single Block Mode to access the Trailer Block.*

Block Data (Multiple of 16 + 2 Bytes, or 6 bytes) — The data to be written into the binary block/blocks.

Update Binary Block Response Codes (2 Bytes)

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

Examples:

// Update the binary block 04h of MIFARE 1K/4K with Data {00 01 .. 0Fh}

APDU = {FF D6 00 04 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0Fh}

// Update the binary block 04h of MIFARE Ultralight with Data {00 01 02 03h}

APDU = {FF D6 00 04 04 00 01 02 03h}

5.3.2.5. Value Block Operation (INC, DEC, STORE)

This command is used for manipulating value-based transactions (e.g., increment a value of the value block etc.).

Value Block Operation APDU Format (10 Bytes)

Command	Class	INS	P1	P2	Lc	Data In	
Value Block Operation	FFh	D7h	00h	Block Number	05h	VB_OP	VB_Value (4 Bytes) {MSB .. LSB}

Where:

Block Number (1 Byte) — The value block to be manipulated.

VB_OP (1 Byte) — 00h = Stores the *VB_Value* into the block. The block will then be converted to a value block.

01h = Increments the value of the value block by the *VB_Value*. This command is only valid for value block.

02h = Decrements the value of the value block by the *VB_Value*. This command is only valid for value block.

VB_Value (4 Bytes) — The value used for value manipulation. The value is a signed long integer (4 bytes).

Example 1: Decimal –4 = {FFh, FFh, FFh, FCh}

VB_Value			
MSB			LSB
FFh	FFh	FFh	FCh

Example 2: Decimal 1 = {00h, 00h, 00h, 01h}

VB_Value			
MSB			LSB
00h	00h	00h	01h

Value Block Operation Response Format (2 Bytes)

Response	Data Out	
Result	SW1	SW2

Value Block Operation Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.3.2.6. Read Value Block

This command is used for retrieving the value from the value block. It is only valid for value block.

Read Value Block APDU Format (5 Bytes)

Command	Class	INS	P1	P2	Le
Read Value Block	FFh	B1h	00h	Block Number	00h

Where:

Block Number (1 Byte) The value block to be accessed.

Read Value Block Response Format (4 + 2 Bytes)

Response	Data Out		
Result	Value {MSB .. LSB}	SW1	SW2

Where:

Value (4 Bytes) The value returned from the card. The value is a signed long integer (4 bytes).

Example 1: Decimal –4 = {FFh, FFh, FFh, FCh}

Value			
MSB			LSB
FFh	FFh	FFh	FCh

Example 2: Decimal 1 = {00h, 00h, 00h, 01h}

Value			
MSB			LSB
00h	00h	00h	01h

Read Value Block Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.3.2.7. Copy Value Block

This command is used to copy a value from a value block to another value block.

Copy Value Block APDU Format (7 Bytes)

Command	Class	INS	P1	P2	Lc	Data In	
Value Block Operation	FFh	D7h	00h	Source Block Number	02h	03h	Target Block Number

Where:

Source Block Number (1 Byte) The value of the source value block will be copied to the target value block.

Target Block Number (1 Byte) The value block to be restored. The source and target value blocks must be in the same sector.

Copy Value Block Response Format (2 Bytes)

Response	Data Out	
Result	SW1	SW2

Copy Value Block Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

Examples:

// Store a value "1" into block 05h

APDU = {FF D7 00 05 05 00 00 00 00 01h}

// Read the value block 05h

APDU = {FF B1 00 05 00h}

// Copy the value from value block 05h to value block 06h

APDU = {FF D7 00 05 02 03 06h}

// Increment the value block 05h by "5"

APDU = {FF D7 00 05 05 01 00 00 00 05h}

5.3.3. Accessing PCSC-compliant Tags (ISO 14443-4)

Basically, all ISO 14443-4 compliant cards (PICCs) understand the ISO 7816-4 APDUs. The ACR1283L reader has to communicate with the ISO 14443-4 compliant cards through exchanging ISO 7816-4 APDUs and responses. ACR1283L handles the ISO 14443 Parts 1-4 Protocols internally.

MIFARE 1K, 4K, Mini and Ultralight tags are supported through the T=CL emulation. Simply treat the MIFARE tags as standard ISO 14443-4 tags.

ISO 7816-4 APDU Format

Command	Class	INS	P1	P2	Lc	Data In	Le
ISO 7816 Part 4 Command					Length of the Data In		Expected length of the Response Data

ISO 7816-4 Response Format (Data + 2 Bytes)

Response	Data Out		
Result	Response Data	SW1	SW2

Common ISO 7816-4 Response Codes

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

Typical sequence may be:

1. Present the tag and connect the PICC Interface.
2. Read/Update the memory of the tag.

Step 1: Connect the tag.

The ATR of the tag is 3B 88 80 01 00 00 00 00 33 81 81 00 3Ah

In which,

The Application Data of ATQB = 00 00 00 00h, protocol information of ATQB = 33 81 81h. It is an ISO 14443-4 Type B tag.

Step 2: Send an APDU, Get Challenge.

<< 00 84 00 00 08h

>> 1A F7 F3 1B CD 2B A9 58h [90 00h]

Note: *For ISO 14443-4 Type A tags, the ATS can be obtained by using the APDU "FF CA 01 00 00h."*

Example:

// To read 8 bytes from an ISO 14443-4 Type B PICC (ST19XR08E)

APDU ={80 B2 80 00 08h}

Class =	80h
INS =	B2h
P1 =	80h
P2 =	00h
Lc =	None
Data In =	None
Le =	08h

Answer: 00 01 02 03 04 05 06 07h [$9000h]

5.4. Peripherals Control

The reader's peripherals control commands are implemented by using *PC_to_RDR_Escape*. The vendor IOCTL for the escape commands is 3500.

5.4.1. Get Firmware Version

This command is used to get the reader's firmware version.

Get Firmware Version Format

Command	Class	INS	P1	P2	Lc
Get Firmware Version	E0h	00h	00h	18h	00h

Get Firmware Version Response Format

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	Number of bytes to be received	Firmware Version

5.4.2. Set Default LED and Buzzer Behaviors

This command is used to set the default behaviors for LEDs and Buzzer.

Set Default LED and Buzzer Behaviors Format (6 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Set Default LED and Buzzer Behaviors	E0h	00h	00h	21h	01h	Default Behaviors

Where:

Default Behaviors 1 Byte.

Default Behaviors	Mode	Description
Bit 0	RFU	RFU
Bit 1	PICC Polling Status LED	To show the PICC Polling Status. 1 = Enable; 0 =Disable
Bit 2	PICC Activation Status LED	To show the activation status of the PICC interface. 1 = Enable; 0 =Disable
Bit 4	Card Insertion and Removal Events Buzzer	To make a beep whenever a card insertion or removal event is detected. 1 = Enable; 0 =Disabled
Bit 3, 5 - 6	RFU	RFU
Bit 7	Card Operation Blinking LED	To make the LED blink whenever the card is being accessed.

Note: *Default value of Default Behaviors = FFh*

Set Default LED and Buzzer Behaviors Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Default Behaviors

5.4.3. Read Default LED and Buzzer Behaviors

This command is used to read the current default behaviors for LEDs and Buzzer.

Read Default LED and Buzzer Behaviors Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Read Default LED and Buzzer Behaviors	E0h	00h	00h	21h	00h

Read Default LED and Buzzer Behaviors Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Default Behaviors

Where:

Default Behaviors 1 Byte.

Default Behaviors	Mode	Description
Bit 0	RFU	RFU
Bit 1	PICC Polling Status LED	To show the PICC Polling Status. 1 = Enable; 0 =Disable
Bit 2	PICC Activation Status LED	To show the activation status of the PICC interface. 1 = Enable; 0 =Disable
Bit 4	Card Insertion and Removal Events Buzzer	To make a beep whenever a card insertion or removal event is detected. 1 = Enable; 0 =Disabled
Bit 3, 5 - 6	RFU	RFU
Bit 7	Card Operation Blinking LED	To make the LED blink whenever the card is being accessed.

Note: *Default value of Default Behaviors = FFh*

5.4.4. Set Automatic PICC Polling

This command is used to set the reader's polling mode.

Whenever the reader is connected to the PC, the PICC polling function will start the PICC scanning to determine if a PICC is placed on/removed from the built-in antenna.

We can send a command to disable the PICC polling function. The command is sent through the PC/SC Escape Command interface. To meet the energy saving requirement, special modes are provided for turning off the antenna field whenever the PICC is inactive, or no PICC is found. The reader will consume less current in power saving mode.

Set Automatic PICC Polling Format

Command	Class	INS	P1	P2	Lc	Data In
Set Automatic PICC Polling	E0h	00h	00h	23h	01h	Polling Setting

Set Automatic PICC Polling Response Format

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Polling Setting

Where:

Polling Setting Default value = 8Fh (1 Byte)

Polling Setting	Description	Description
Bit 0	Auto PICC Polling	1 = Enable 0 = Disable
Bit 1	Turn off Antenna Field if no PICC found	1 = Enable 0 = Disable
Bit 2	Turn off Antenna Field if the PICC is inactive	1 = Enable 0 = Disable
Bit 3	RFU	RFU (Preferably set to 0).
Bit 5 – 4	PICC Polling Interval for PICC	Bit 5 – Bit 4: 0 – 0 = 250 ms 0 – 1 = 500 ms 1 – 0 = 1000 ms 1 – 1 = 2500 ms
Bit 6	RFU	RFU
Bit 7	Enforce ISO 14443A Part 4	1 = Enable 0 = Disable

Notes:

1. *It is recommended to enable the option "Turn off Antenna Field if the PICC is inactive," so that the "Inactive PICC" will not be exposed to the field all the time so as to prevent the PICC from "warming up."*
2. *The longer the PICC Poll Interval, the more efficient of energy saving. However, the response time of PICC Polling will become longer. The Idle Current Consumption in Power Saving Mode is about 60 mA, while the Idle Current Consumption in Non-Power Saving mode is*

about 130 mA. Idle Current Consumption = PICC is not activated.

3. *The reader will activate the ISO 14443A-4 mode of the "ISO 14443A-4 compliant PICC" automatically. Type B PICC will not be affected by this option.*
4. *The JCOP30 card comes with two modes: ISO 14443A-3 (MIFARE 1K) and ISO 14443A-4 modes. The application has to decide which mode should be selected once the PICC is activated.*

5.4.5. Read Automatic PICC Polling

This command is used to read the current automatic PICC polling setting.

Read Automatic PICC Polling Format

Command	Class	INS	P1	P2	Lc
Read Automatic PICC Polling	E0h	00h	00h	23h	00h

Read Automatic PICC Polling Response Format

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Polling Setting

Where:

Polling Setting Default value = 8Fh (1 Byte)

Polling Setting	Description	Description
Bit 0	Auto PICC Polling	1 = Enable 0 = Disable
Bit 1	Turn off Antenna Field if no PICC found	1 = Enable 0 = Disable
Bit 2	Turn off Antenna Field if the PICC is inactive	1 = Enable 0 = Disable
Bit 3	RFU	RFU (Preferably set to 0).
Bit 5 - 4	PICC Polling Interval for PICC	Bit 5 – Bit 4: 0 – 0 = 250 ms 0 – 1 = 500 ms 1 – 0 = 1000 ms 1 – 1 = 2500 ms
Bit 6	RFU	RFU
Bit 7	Enforce ISO 14443A Part 4	1 = Enable 0 = Disable

5.4.6. Set the PICC Operating Parameter

This command is used to set the PICC operating parameter.

Set the PICC Operating Parameter Format (6 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Set the PICC Operating Parameter	E0h	00h	00h	20h	01h	Operation Parameter

Set the PICC Operating Parameter Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Operation Parameter

Where:

Operating Parameter 1 Byte.

Operating Parameter	Parameter	Description	Option
Bit 0	ISO 14443 Type A	The Tag Types to be detected during PICC Polling.	1 = Detect 0 = Skip
Bit 1	ISO 14443 Type B		1 = Detect 0 = Skip
Bit 2 - 7	RFU	RFU	RFU

Note: *Default value of Operation Parameter = 03h*

5.4.7. Read the PICC Operating Parameter

This command is used to read the current PICC operating parameter.

Read the PICC Operating Parameter Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Read the PICC Operating Parameter	E0h	00h	00h	20h	00h

Read the PICC Operating Parameter Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Operation Parameter

Where:

Operating Parameter 1 Byte.

Operating Parameter	Parameter	Description	Option
Bit 0	ISO 14443 Type A	The Tag Types to be detected during PICC Polling.	1 = Detect 0 = Skip
Bit 1	ISO 14443 Type B		1 = Detect 0 = Skip
Bit 2 - 7	RFU	RFU	RFU

5.4.8. Set Auto PPS

This command is used to set the reader's PPS setting.

Whenever a PICC is recognized, the reader tries to change the communication speed between the PCD and PICC defined by the maximum connection speed. If the card does not support the proposed connection speed, the reader tries to connect the card with a slower speed setting.

Set Auto PPS Format (7 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Set Auto PPS	E0h	00h	00h	24h	01h	Max Speed

Set Auto PPS Response Format (9 Bytes)

Response	Class	INS	P1	P2	Le	Data Out	
Result	E1h	00h	00h	00h	02h	Max Speed	Current Speed

Where:

Max Speed 1 Byte. Maximum Speed.

Current Speed 1 Byte. Current Speed.

Value can be: 106 Kbps = 00h (equal to No Auto PPS)
212 Kbps = 01h
424 Kbps = 02h (default setting)
848 Kbps = 03h

Notes:

1. *Normally, the application should know the maximum connection speed of the PICCs being used. The environment also affects the maximum achievable speed. The reader just uses the proposed communication speed to talk with the PICC. The PICC will become inaccessible if the PICC or environment does not meet the requirement of the proposed communication speed.*
2. *The reader supports different speed between sending and receiving.*

5.4.9. Read Auto PPS

This command is used to read the current auto PPS setting.

Read Auto PPS Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Read Auto PPS	E0h	00h	00h	24h	00h

Set Auto PPS Response Format (9 Bytes)

Response	Class	INS	P1	P2	Le	Data Out	
Result	E1h	00h	00h	00h	02h	Max Speed	Current Speed

Where:

Max Speed 1 Byte. Maximum Speed.

Current Speed 1 Byte. Current Speed.

Value can be: 106 Kbps = 00h (equal to No Auto PPS)
212 Kbps = 01h
424 Kbps = 02h (default setting)
848 Kbps = 03h

5.4.10. Set Antenna Field

This command is used for turning on/off the antenna field.

Antenna Field Control Format (6 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Antenna Field Control	E0h	00h	00h	25h	01h	Status

Antenna Field Control Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Status

Where:

Status 1 Byte.

01h = Enable Antenna Field

00h = Disable Antenna Field

Note: *Make sure the Auto PICC Polling is disabled before turning off the antenna field.*

5.4.11. Read Antenna Field Status

This command is used to read the current status of the antenna field.

Read Antenna Field Status Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Read Antenna Field Status	E0h	00h	00h	25h	00h

Read Antenna Field Status Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	Status

Where:

Status 1 Byte.

01h = Enable Antenna Field

00h = Disable Antenna Field

5.4.12. Two LEDs Control

This command is used to control the LEDs output.

LED Control Format (6 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
LED Control	E0h	00h	00h	29h	01h	LED Status

LED Control Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	LED Status

Where:

LED Status 1 Byte.

LED Status	Description	Description
Bit 0	Blue LED	1 = ON; 0 = OFF
Bit 1	Orange LED	1 = ON; 0 = OFF
Bit 2 - 7	RFU	RFU

5.4.13. LED Status

This command is used to check the status of the existing LEDs.

LED Status Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
LED Status	E0h	00h	00h	29h	00h

LED Status Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	LED Status

Where:

LED Status 1 Byte.

LED Status	Description	Description
Bit 0	Blue LED	1 = ON; 0 = OFF
Bit 1	Orange LED	1 = ON; 0 = OFF
Bit 2 - 7	RFU	RFU

5.4.14. Four LEDs Control

This command is used to control the four LEDs.

LEDs Control Command Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
LEDs Control	FFh	00h	44h	bLEDsState	00h

Where:

P2 *bLEDsState*

LED_0, LED_1, LED_2 and LED_3 Control Format (1 Byte)

CMD	Item	Description
Bit 0	LED_0 State	1 = On; 0 = Off
Bit 1	LED_1 State	1 = On; 0 = Off
Bit 2	LED_2 State	1 = On; 0 = Off
Bit 3	LED_3 State	1 = On; 0 = Off
Bits 4 – 7	Reserved	-

Where:

Data Out SW1 SW2

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.15. Buzzer Control

This command is used to control the buzzer output.

Buzzer Control Format (6 Bytes)

Command	Class	INS	P1	P2	Lc	Data In
Buzzer Control	E0h	00h	00h	28h	01h	Buzzer On Duration

Where:

Buzzer On Duration 1 Byte.
00h = Turn OFF
01h - FEh = Duration (unit: 10 ms)
FFh = Turn ON

Buzzer Control Response Format (6 Bytes)

Response	Class	INS	P1	P2	Le	Data Out
Result	E1h	00h	00h	00h	01h	00h

Where:

Buzzer Remain Duration 1 Byte. The remained *turn on* duration (unit: 10 ms)

5.4.16. LCD Control Commands

5.4.16.1. Clear LCD

This command is used to clear all contents shown in the LCD.

Clear LCD Command Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Clear LCD	FFh	00h	60h	00h	00h

Where:

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.2. LCD Display (ASCII Mode)

This command is used to display LCD message in ASCII mode.

LCD Display Command Format (5 Bytes + LCD Message Length)

Command	Class	INS	P1	P2	Lc	Data In (Max. 16 Bytes)
LCD Display	FFh	Option Byte	68h	LCD XY Position	LCD Message Length	LCD Message

Where:

INS Option Byte (1 Byte).

CMD	Item	Description
Bit 0	Character Bold Font	1 = Bold; 0 = Normal
Bit 1 - 3	Reserved	-
Bit 4 - 5	Table Index	00 = Fonts Set A 01 = Fonts Set B 10 = Fonts Set C
Bits 6 – 7	Reserved	-

P2 LCD XY Position. The character to be displayed on the LCD position specified by DDRAM Address

Please follow the DDRAM table below for the LCD character position's representation.

For Fonts Set 1 and 2:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DISPLAY POSITION
1st LINE	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	LCD XY POSITION
2nd LINE	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	

For Fonts Set 3:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DISPLAY POSITION
1st LINE	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	LCD XY POSITION
2nd LINE	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	
3rd LINE	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	
4th LINE	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	

Where:

Lc The length of the LCD message (max. 10h). If the message length is longer than the number of character that the LCD screen's can be shown, then the redundant character will not be shown on the LCD.

Data In LCD Message. The data to be sent to LCD (maximum character is 16 for each line)

Please follow the fonts tables (selected by INS Bit 4 - 5) below for the LCD character index.

Note: *Size of the characters in Fonts Set A and Fonts Set B is 8 x 16, but size of the characters in Fonts Set C is 8 x 8.*

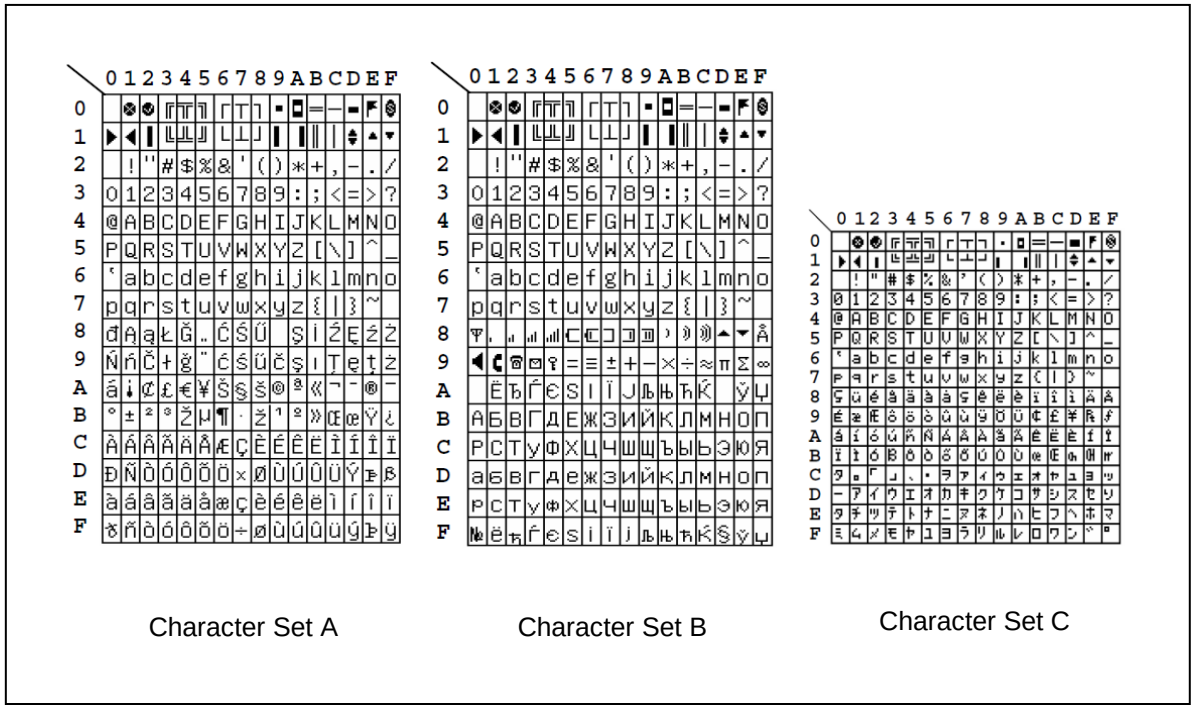


Figure 3: LCD Display Font Table

Where:

Data Out SW1 SW2

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.3. LCD Display (GB Mode)

This command is used to display LCD message in GB Mode.

LCD Display Command Format (5 Bytes + LCD Message Length)

Command	Class	INS	P1	P2	Lc	Data In (Max. 16 Bytes)
LCD Display	FFh	Option Byte	69h	LCD XY Position	LCD Message Length	LCD Message

Where:

INS Option Byte (1 Byte).

CMD	Item	Description
Bit 0	Character Bold Font	1 = Bold; 0 = Normal
Bit 1 - 7	Reserved	-

P2 LCD XY Position. The character to be displayed on the LCD position specified by DDRAM Address.

Please follow the DDRAM table below for the LCD character position's representation.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DISPLAY POSITION
FIRST LINE	00		01		02		03		04		05		06		07		LCD XY POSITION
SECOND LINE	40		41		42		43		44		45		46		47		

Where:

Lc The length of the LCD message (max. 10h). If the message length is longer than the number of character that the LCD screen's can be shown, then the redundant character will not be shown on the LCD.

The length of the LCD message should be multiplied by 2 because each Chinese character (GB code) should contain two bytes.

Data In LCD Message. The data to be sent to LCD, maximum 8 (2 x 8 bit each character) character for each line. Please follow the fonts table of GB Coding.

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.4. LCD Display (Graphic Mode)

This command is used to display LCD message in graphic mode.

LCD Display Command Format (5 Bytes + LCD Message Length)

Command	Class	INS	P1	P2	Lc	Data In (Max. 128 Bytes)
LCD Display	FFh	00h	6Ah	Line Index	Pixel Data Length	Pixel Data

Where:

P2 Line Index. This is used to set which line in the LCD display should start to update.

Please refer to below LCD display position.

Lc Pixel Data Length. The length of the pixel data (max. 80h)

Data In Pixel Data. The pixel data to be sent to LCD for display.

LCD Display Position (Total LCD Size: 128 x 32)

	Byte 00h (X = 00h)								Byte 01h (X = 01h)								…	Byte 0Fh (X = 0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
00h																									
01h																									
02h																									
03h																									
04h																									
05h																									
06h																									
07h																									
08h																									
09h																									
…	…																								
1Fh																									

Where:

Data Out SW1 SW2

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.5. Scroll Current LCD Display

This command is used to set the scrolling feature of the current LCD display.

Scrolling LCD Command Format (5 Bytes + LCD Message Length)

Command	Class	INS	P1	P2	Lc	Data In (6 Bytes)
Scrolling LCD	FFh	00h	6Dh	00h	06h	Scroll Ctrl

Where:

Scroll Ctrl 6 Bytes. Scrolling control format.

Scrolling Control Format (6 Bytes)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
X Position	Y Position	Scrolling Range (Horizontal)	Scrolling Range (Vertical)	Refresh Speed Ctrl	Scrolling Direction

Where:

X Position Horizontal start up position. Please refer to the LCD display position below.

Y Position Vertical start up position. Please refer to the LCD display position below.

LCD Display Position (Total LCD Size: 128 x 32)

	Byte 00h (X = 00h)								Byte 01h (X = 01h)								…	Byte 0Fh (X = 0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
00h																									
01h																									
02h																									
03h																									
04h																									
05h																									
06h																									
07h																									
08h																									
09h																									
…	…																								
1Fh																									

Where:

Scrolling Range (Horizontal) Number of pixels (in multiple of 8) in horizontal after X position is scrolled.

Scrolling Range (Vertical) Number of pixels in Vertical after Y position is scrolled.

Refresh Speed Ctrl Bit 0~Bit 3 – Number of pixel moves pre-scrolling.

Bit 4~Bit 7 – Scrolling period.

Bit 7	Bit 6	Bit 5	Bit 4	Scrolling Period
0	0	0	0	1 Unit
0	0	0	1	3 Units
0	0	1	0	5 Units
0	0	1	1	7 Units
0	1	0	0	17 Units
0	1	0	1	19 Units
0	1	1	0	21 Units

Bit 7	Bit 6	Bit 5	Bit 4	Scrolling Period
0	1	1	1	23 Units
1	0	0	0	129 Units
1	0	0	1	131 Units
1	0	1	0	133 Units
1	0	1	1	135 Units
1	1	0	0	145 Units
1	1	0	1	147 Units
1	1	1	0	149 Units
1	1	1	1	151 Units

Table 6: Scrolling Period

Bit 1	Bit 0	Scrolling Direction
0	0	From Left to Right
0	1	From Right to Left
1	0	From Top to Bottom
1	1	From Bottom to Top

Table 7: Scrolling Direction

Where:

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.6. Pause LCD Scrolling

This command is used to pause the LCD scrolling set. To resume the scrolling, send again the scrolling LCD command to perform.

Pause Scrolling Command Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Pause LCD Scrolling	FFh	00h	6Eh	00h	00h

Where:

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.7. Stop LCD Scrolling

This command is used to stop the LCD scrolling set before the LCD display goes back to normal display position.

Stop Scrolling LCD Command Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
Stop Scrolling LCD	FFh	00h	6Fh	00h	00h

Where:

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.8. LCD Contrast Control

This command is used to control the LCD contrast.

LCD Contrast Control Command Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
LCD Contrast Control	FFh	00h	6Ch	Contrast Control	00h

Where:

P2 Contrast Control. The value range is between 00h to 0Fh. It is as large as brightened on contrast. Otherwise the contrast is darkened.

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

5.4.16.9. LCD Backlight Control

This command is used to control the LCD backlight.

LCD Backlight Control Command Format (5 Bytes)

Command	Class	INS	P1	P2	Lc
LCD Backlight Control	FFh	00h	64h	Backlight Control	00h

Where:

P2 Backlight Control.

Backlight Control Format (1 Byte)

CMD	Description
00h	LCD Backlight Off
FFh	LCD Backlight On

Where:

Data Out SW1 SW2.

Status Code

Results	SW1 SW2	Meaning
Success	90 00h	The operation is completed successfully.
Error	63 00h	The operation is failed.

6.0. Embedded Programming (Standalone Mode) Design

6.1. Operation Mode Selection

The start-up routine of the ACR1283L calls the function API_SysInit() as shown below:

```
int main(void)
{
  if(API_SysInit())
  {
    GoToMain();
  }
  else
  {
    standalone_mode();
  }

  return 0;
}
```

If the API, SysInit(), returns a value not equal to zero, it means that the USB is connected to PC, then user can use the API GoToMain() make ACR1283L run in PC-Linked mode. Otherwise, it will call the API standalone_mode() which will make ACR1283L run in Standalone Mode. This is also shown in the figure below:

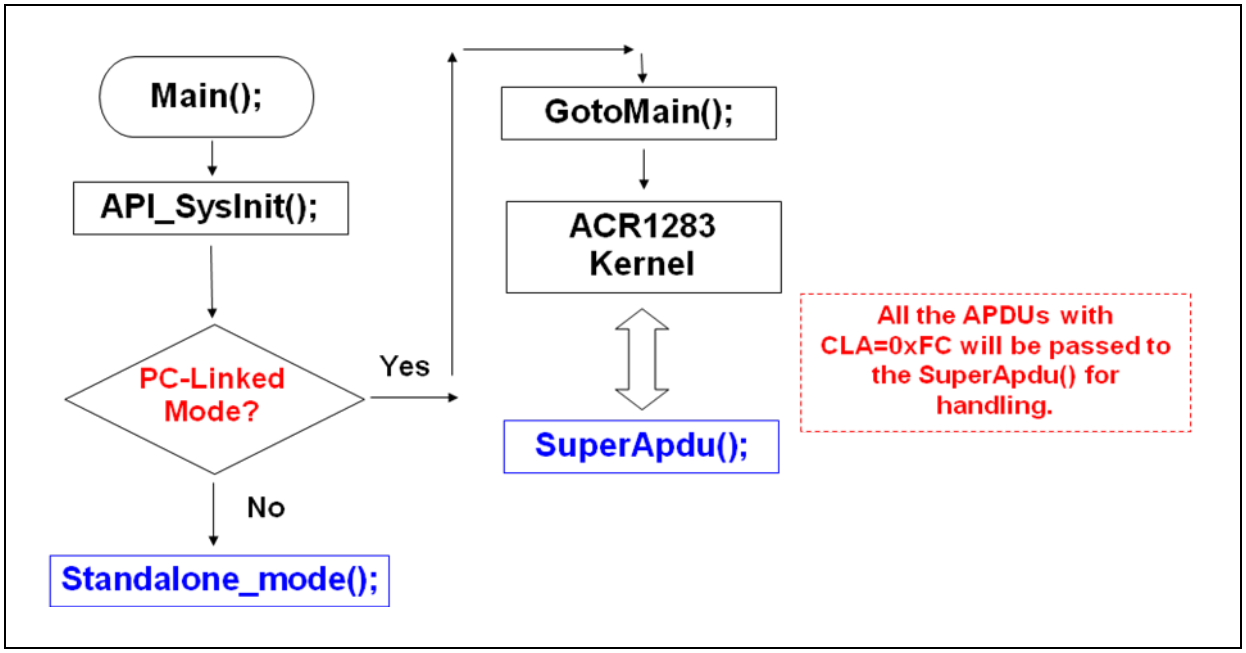


Figure 4: Entry Point Flowchart

6.1.1. Accessing Standalone Commands while running in PC-Link Mode

The SuperApdu() API is a callback function in ACR1283L. This is automatically called and performed by the ACR1283L when the PC-Linked application sends an APDU command with a starting byte of FCh.

Users can define the tasks to be performed by the ACR1283L by creating specific escape commands within the SuperApdu() API. Any APDU with a starting byte of FCh is considered an escape command by ACR1283L.

See below for an example:

```
UINT8 SuperApdu(UINT8 *Cmdbuffer,UINT8 *Resbuffer,UINT16 *pReslen)
{
   if((Cmdbuffer[0] ==0xFC) &&(Cmdbuffer[1]==0xCD))
   {
      standalone_mode();
   }
    else if((Cmdbuffer[0] ==0xFC) &&(Cmdbuffer[1]==0xFF))
   {
     API_LcdCLear();
     API_LcdDisplay(0x01,"API Testing...",17,2,0);
     *pReslen = APITest(&Cmdbuffer[2],Resbuffer);
     API_LcdCLear();
     API_LcdDisplay(0x01,"API Test Done",13,2,0);
     return API_OK;
   }
}
```

The example above means that if the APDU command from the PC-Linked application is FC CDh, the ACR1283L will perform the standalone_mode() function and if the command is FC FFh it will perform the commands specified in the routine. (See **Section 7.9.5** for more information)

6.1.2. Standalone Mode Programming

In standalone mode, users have program their routine within the standalone_mode() API. Please be reminded that, in the standalone_mode() API, you need to have a loop in order to make it work continuously. See below for an example:

```
void standalone_mode()
{
  UINT8 status;
  while(1) // infinite Loop
  {
    status = API_PollPICC(0,5,4,UIDBuf,&UIDLen);
    if (status == API_OK)
    {
       if (Cnt ==1)
       {
         Cnt = 0;
         API_LedCtrl(0,1);
         API_LedCtrl(1,2);
       }
       else
       {
         Cnt = 1;
         API_LedCtrl(1,1);
         API_LedCtrl(0,2);
      }
    }
  }
}
```

6.2. Architecture

The figure below shows the Third Party Programming (Embedded Programming) flow for the ACR1283L.

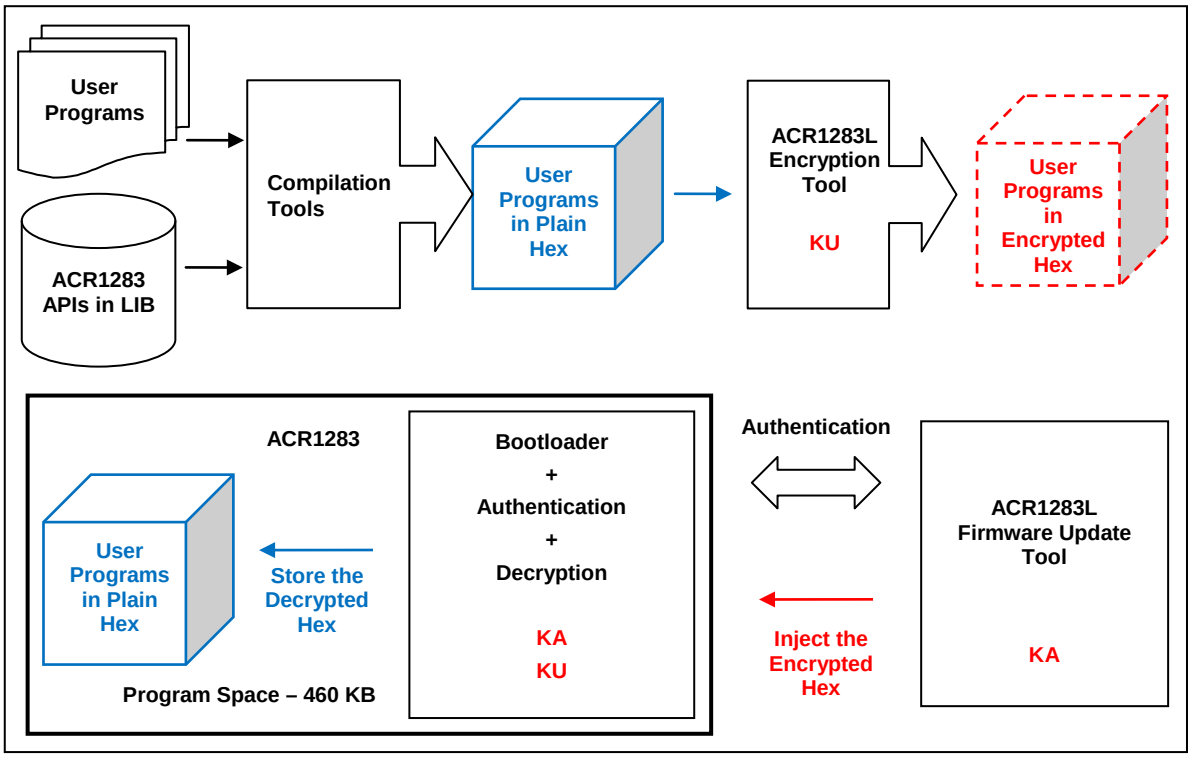


Figure 5: Third-party Programming Architecture

User Programs	The program that is written by third party, which performs the application that the user defined.
ACR1283 APIs in LIB	The internal program for controlling all the peripherals (which is not an open source), and provides the API for User Program Use.
Compilation Tools	Used to compile the standalone applications. IAR and GCC compilation tools are supported.
ACR1283L Encryption Tool	Encrypts the compiled hex file using AES algorithm to protect the IP rights of developers.
ACR1283L Firmware Upgrade Tool	Uploads the encrypted standalone file (which is also a hex file or a firmware file) to the ACR1283L.

7.0. Embedded Programming (Standalone Mode) API

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

7.1. Data Type

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

7.2. Card Operation API Functions

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

ucSlot	Card Slot
00h	PICC
01h	SAM1
02h	SAM2
03h	SAM3
04h	SAM4

Table 8: Card Interface Parameters

7.2.1. Data Exchange

This function is used to access APDU and pseudo APDU, and then receives a return message.

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

Parameters

ucSlot Card slot index.

pucApdu Address of the buffer where the data is stored; to send.

uiApduLen Length of the data (16 bits).

pucRsp Address of the buffer for storing the return message.

puiRspLen Address of the buffer for storing the length of the return message (16 bits).

pucSw Address of the buffer for storing the return status (SW0 SW1).

Note: *Return status length is 2 bytes.*

Return Parameters

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

description of the status codes.

Sample Code

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

slot = 0x01;
senddatalen = 0x05;

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

}
```

Get Challenge command is sent to SAM1 in order to get an 8-byte random number.

APDU = “80 84 00 00 08h”

Response = 8 bytes random number

SW1 SW2 = “90 00h”

7.2.2. Activating a card

This function activates the card and receives ATR.

Note: *The PICC slot cannot use this API function to power up during Standalone Mode.*

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

Parameters

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

Return Parameters

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

Sample Code

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

/* status = 00, mean success, if failed, please refer to appendix 1 for
```

```
the status code contend of atr_array, for example:0x3b 0x8f 0x80 0x01
0x80 0x4f 0x0c 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00
0x00 0x6a
artlen = 0x14 */
……
 }
```

Example of ATR: "3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6Ah"

7.2.3. Deactivating a card

This function deactivates the card.

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

Parameter

ucSlot Card slot index.

Return Parameters

Successful API_OK

Failed API_ERR_INDEX/API_ERR_CMDFAIL

7.2.4. Reactivating a card

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

Note: *This function does not return ATR.*

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

Parameters

ucSlot Card slot index.

pucStatus Card slot's current status.

pucStatus Bit definition:

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

Note: *All 0 means no card or failed to activate the slot.*

Return Parameters

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

7.2.5. Setting the PICC communication speed

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

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

Parameters

ucPar Communication speed set by user

- 00h = 106 Kbps
- 01h = 212 Kbps
- 02h = 424 Kbps (default setting)
- 03h = 848 Kbps

Return Parameter

None.

7.2.6. Setting the SAM1~SAM4 communication speed

This function is used to set the ICC communication speed.

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

Fi and *f* (max.)

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

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

Di

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

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

Figure 6: ICC Communication Speed

Parameters

ucSlot Card slot index.

pucFiDi Communication speed between card and reader.

- Bit 7 to Bit 4 for *Fi*
- Bit 3 to Bit 0 for *Di*

ucPar Speed type.

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

Setting the ATR should be done before activation by the following:

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

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

Return Parameters

Successful API_OK

Failed API_ERR_INDEX/API_ERR_CMDFAIL

7.2.7. Setting the Auto PPS

This function enables/disables the Auto PPS:

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

Parameters

ucSlot Card slot index.

ucPar Enables/disables the parameter.

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

Return Value

Successful 00h

Failed 01h

7.2.8. Getting the ATR

This function returns ATR but does not activate the card slot. This also activates the SAM2.

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

Parameters

ucSlot Card slot index.

pucATR Address of the buffer for storing current card ATR.

pucATRLen Address of the buffer for storing ATR length.

Note: *Returns 00h if there is no card or other error.*

Return Value

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

Sample Code

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

Example of ATR: "3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6Ah"

7.2.9. Getting the card slot status

This function retrieves the status of the card slot.

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

Parameters

ucSlot Card slot index.

pucStatus Card slot status.

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

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

Return Value

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

7.2.10. Getting the card UID

This function gets the card IUD.

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

Parameters

pucUIDBuf Buffer for storing card UID.

pucUIDLength Card UID length.

Return Value

None.

7.2.11. Getting Volatile Key

This function gets the volatile key of the card.

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

Parameters

pucKeyBuf Buffer for storing 6-byte volatile key.

Return Value

None.

7.2.12. Saving Volatile Key

This function saves the card's volatile key.

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

Parameters

pucKeyBuf Buffer for storing 6-byte volatile key.

Return Value

None.

7.2.13. Polling a card

This function is used to poll a card.

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

Parameters

ucTagType Card type to be polled.

- 0 = Type A
- 1 = Type B

ucPollRetry Number of times of polling a card.

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

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

pucUIDBuf Buffer for storing card UID.

pucUIDLength Card UID length.

Return Value

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

7.2.14. Setting MIFARE authentication

This function sets authentication for a MIFARE card.

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

Parameters

ucCmd 0 = Does not require MIFARE card authentication.

1 = Requires MIFARE card authentication.

Return Value

None.

7.2.15. Setting the parameter for polling a card

This function sets the parameter for polling a card.

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

Parameters

ucAutoPolling 0 = Turn off auto polling card.

1 = Turn on auto polling card.

ucCardTypes Card type to be polled

- 01h = Type A
- 02h = Type B
- 03h = Type A + Type B

ucPart4Enter Enter OSP/IEC 14443 part 4,

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

ucPollingInterval Interval for polling card (unit is 125 ms)

- 0 = For polling card continuously.

Return Value

None.

7.2.16. Halting a MIFARE card

This function sets the selected MIFARE card to halt mode.

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

Parameters

No parameters.

Return Value

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

7.2.17. Waking up a MIFARE card

This function is used to wake up a halted MIFARE card.

```
UINT8 API_WupMF(void)
```

Parameters

No parameters.

Return Value

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

7.2.18. Checking the SAM file status

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

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

Parameters

ucSlot Card slot index.

Return Value

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

7.2.19. Setting the SAM file status

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

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

Parameters

ucSlot Card slot index.

ucCmd 0 = Key file is closed.
1 = Key file is open.

Return Value

Successful API_OK

Failed API_ERR_INDEX

7.3. Reader API Functions

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

7.3.1. Checking the firmware version

This function is used to read the firmware version.

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

Parameters

pucBuffer	Buffer for store firmware version. 17 bytes of firmware version length. The format is as follows: ACR1283U Vx.x.x.x; where x represents version number. ***Note:*** *Each character of the bytes is using ASCII (e.g. Y = 59h, S = 53h, T = 54h)*
pucLen	Firmware version length.

Return Value

None.

7.3.2. Resetting the reader

This function resets the reader.

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

Parameters

No parameters.

Return Value

None.

7.3.3. Entering firmware upgrade mode

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

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

Parameters

No parameters.

Return Value

Successful	No return.
Failed	API_ERR_CMDFAIL

7.3.4. Controlling the buzzer

This function sets the buzzer operation mode.

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

Parameters

ucOnTime Buzzer on time.

ucOnTime Setting range is 01h to FFh (unit is 10 ms).

(e.g., if *ucOnTime* equal to 64h, buzzer on time is 100 x 10 ms = 1000 ms)

ucOffTime Buzzer off time.

- ucOffTime setting range is 01h to FFh (unit is 10 ms)

ucCnt Repeat time.

- 00h = Buzzer off.
- 01h~FEh = The number of repeat tuning buzzer on.
- FFh = Buzzer on continuously.

ucMode Mode setting.

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

Return Value

None.

Sample Code

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

7.3.5. Setting the LED status

This function sets the status of LEDs 1 to 4:

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

Parameters

ucStatus Turn on/off LED 1 to 4

ucNbr 0 = All D1 to D4

1 = D1 3 = D3

2 = D2 4 = D4

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

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

Return Value

None.

Sample Code

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

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

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

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

7.3.6. Reading LED's current status

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

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

Parameters

No parameters.

Return Value

Return the current LED status.

- Bit 0 <-------> LED1 (PCB print : D1)
- Bit 1 <-------> LED2 (PCB print : D2)
- Bit 2 <-------> LED3 (PCB print : D3)
- Bit 3 <-------> LED4 (PCB print : D4)
- "1" = LED is turned on
- "0" = LED is turned off

7.4. LCD API Functions

This section introduces the LCD-related API functions.

7.4.1. Setting the LCD backlight

This function sets the LCD's backlight status:

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

Parameters

ucCtrlCode 0 = turns backlight off.

1 = turns backlight on.

Return Value

None.

7.4.2. Displaying characters in LCD

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

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

Parameters

Table Table = 0Xh for ASCII Table 1 (Refer to Figure 7)

Table = 1Xh for ACSII Table 2 (Refer to Figure 8)

Table = 2Xh for ACSII Table 3 (Refer to Figure 9)

Table = 4Xh for GB

Table = X1h bold character

Table = X0h normal character

pLCDBuf Show the address of the message

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

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

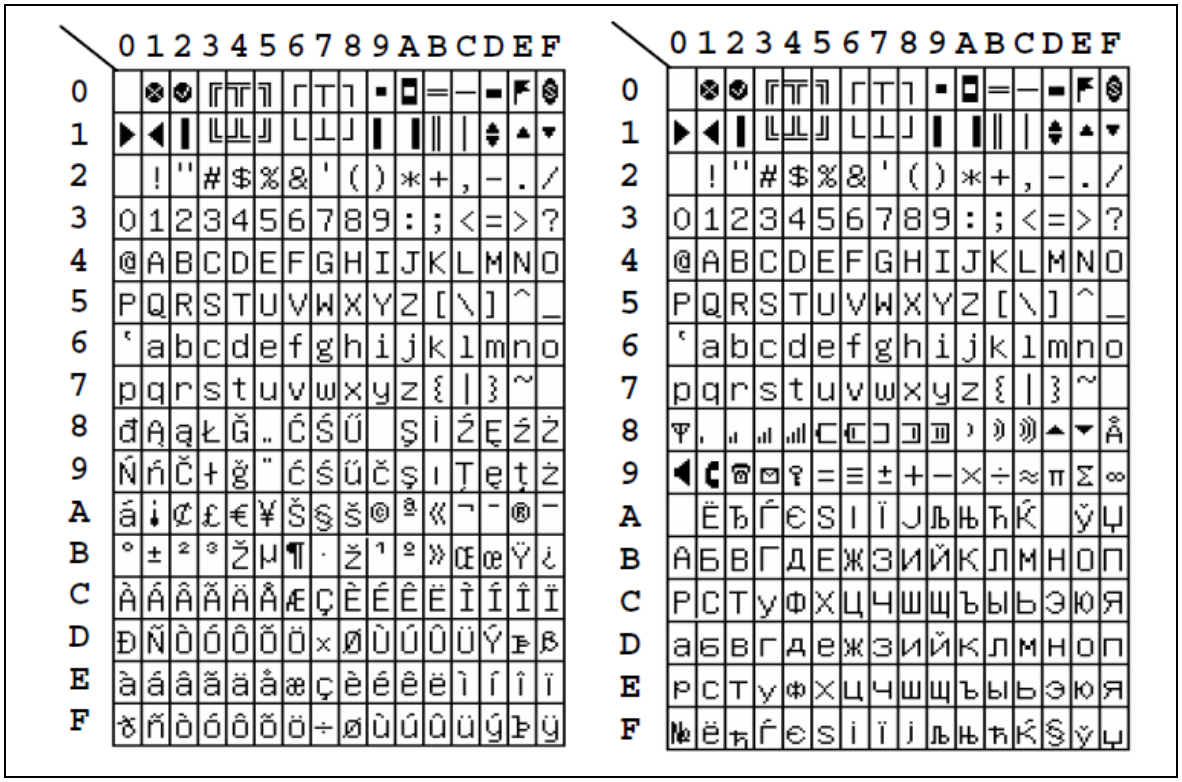


Figure 7: Character Display Table (Table 1)

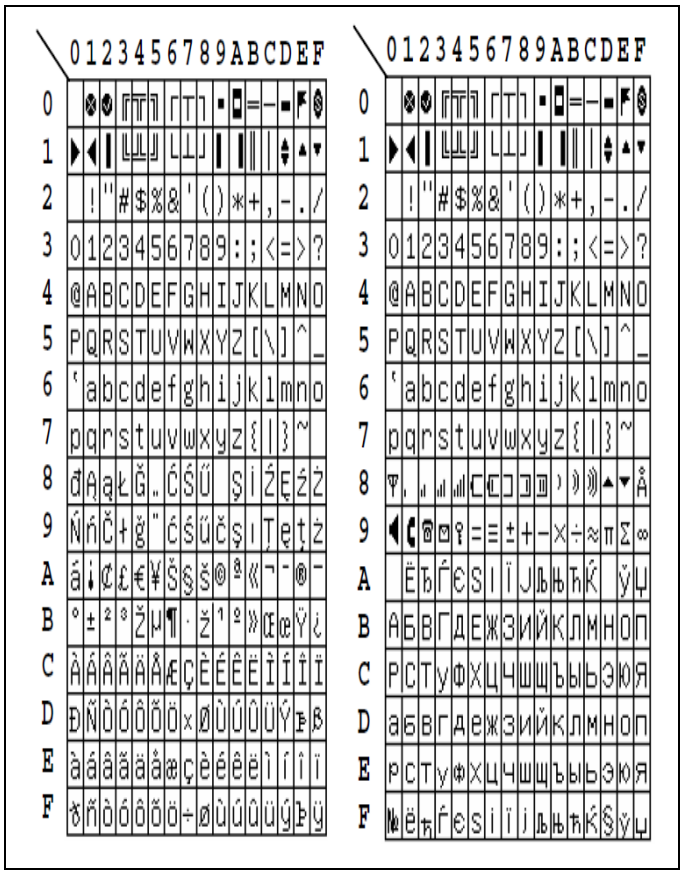


Figure 8: Character Display Table (Table 2)

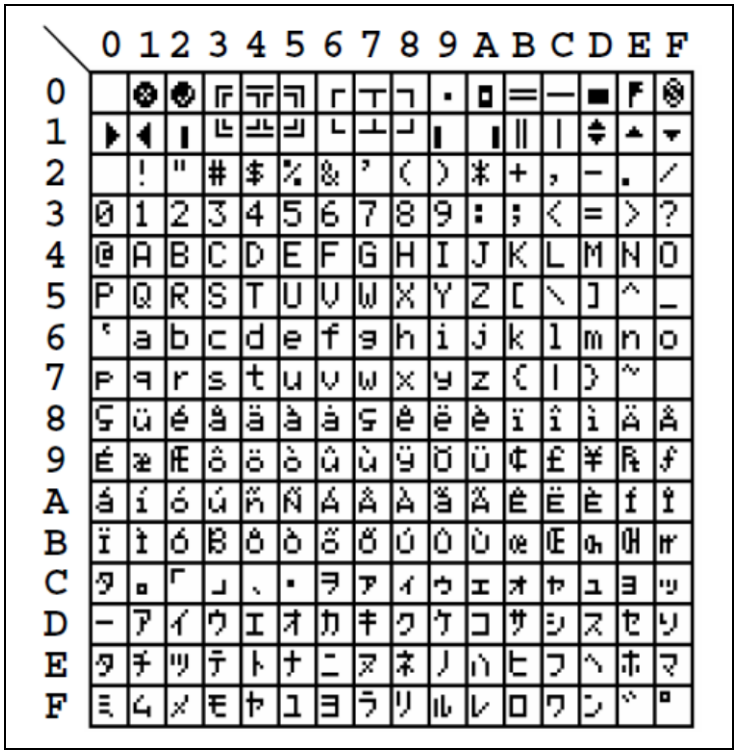


Figure 9: Character Display Table (Table 3)

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

Sample Code

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

7.4.3. Displaying graphs in LCD

This function displays graph in the LCD.

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

Parameters

Line	Line index (Refer to **Table 9**).
Len	Length of pixel data.
pMessage	Buffer for storing the message.

	Byte 00h (X = 00h)								Byte 01h (X = 01h)								…	Byte 0Fh (X = 0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
00h																									
01h																									
02h																									
03h																									
04h																									
05h																									
06h																									
07h																									
08h																									
09h																									
…	…																								
1Fh																									

Table 9: LCD Line Index

7.4.4. Controlling the LCD scroll

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

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

Parameters

ScrollDirect Direction of scroll.

Bit 1	Bit 0	Scrolling Direction
0	0	From Left to Right
0	1	From Right to Left
1	0	From Top to Bottom
1	1	From Bottom to Top

ScrollSpeed Speed of scroll.

Bit 0~Bit 3 – number of pixels pre-scrolling

Bit 7	Bit 6	Bit 5	Bit 4	Scrolling period
0	0	0	0	1 Unit
0	0	0	1	3 Units
0	0	1	0	5 Units
0	0	1	1	7 Units
0	1	0	0	17 Units

Bit 7	Bit 6	Bit 5	Bit 4	Scrolling period
0	1	0	1	19 Units
0	1	1	0	21 Units
0	1	1	1	23 Units
1	0	0	0	129 Units
1	0	0	1	131 Units
1	0	1	0	133 Units
1	0	1	1	135 Units
1	1	0	0	145 Units
1	1	0	1	147 Units
1	1	1	0	149 Units
1	1	1	1	151 Units

X_Start start at X coordination.
Y_Start start at Y coordination.
X_end end of X coordination.
Y_end end of Y coordination.

LCD Display Position Total LCD Size: 128 x 32.

	Byte 00h (X = 00h)								Byte 01h (X = 01h)								…	Byte 0Fh (X = 0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
00h																									
01h																									
02h																									
03h																									
04h																									
05h																									
06h																									
07h																									
08h																									
09h																									
…	…																								
1Fh																									

Table 10 : LCD Display Position

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.4.5. Clearing the LCD

This function is used to clear the LCD display.

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

Parameters

No parameters.

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.4.6. Pausing the LCD scroll

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

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

Parameters

No parameters.

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.4.7. Stopping the LCD scroll

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

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

Parameters

No parameters.

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.5. Keypad API Functions

This section introduces the keypad-related API functions.

7.5.1. Resetting keypad flash

This function is used to reset keypad parameters in flash (i.e., Set the keypad parameters to the last setting via `API_ConfigKeypadFlashParam`.

```
UINT8 API_ResetKeypadFlash(void)
```

Parameters

No parameters.

Return Parameters

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.5.2. Configuring keypad flash parameters

This function is used to configure keypad parameters in flash.

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

Parameters

pfbuffer Keypad parameters. Please refer to **Table 11** for the details of each parameter.

Note: *Keypad parameters include:*

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

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

Return Parameters

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.5.3. Configuring keypad parameters

This function is used to configure keypad parameters:

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

Parameters

pebuffer Keypad parameters. Please refer to **Table 11** for the details of each parameter.

Note: *Keypad parameters include:*

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

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

Return Parameters

Successful API_OK

Failed API_ERR_CMDFAIL

7.5.4. Resetting the keypad

This function resets the keypad.

```
UINT8 API_ResetKeypad(void)
```

Parameters

No parameters.

Return Parameters

Successful API_OK

Failed API_ERR_CMDFAIL

7.5.5. Configuring baseline control parameters

This function is used to configure baseline control parameters.

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

Parameters

pbbuffer Baseline control parameters. Please refer to **Table 11** for the details of each parameter.

Note: *Baseline control parameters include:*

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

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

Return Parameters

Successful API_OK

Failed API_ERR_CMDFAIL

7.5.6. Configuring threshold parameters

This function is used to configure threshold parameters.

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

Parameters

psbuffer Threshold parameters. Please refer to **Table 11** for the details of each parameter.

Note: *Threshold parameters include:*

- *E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH, E10TTH, E10RTH, E11TTH, E11RTH*

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

Return Parameters

Successful API_OK

Failed API_ERR_CMDFAIL

7.5.7. Configuring CDC parameters

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

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

Parameters

pcbuffer Charge/discharge current parameters. Please refer to **Table 11** for the details of each parameter.

Note: *Charge/discharge current parameters include:*

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

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

Return Parameters

Successful API_OK

Failed API_ERR_CMDFAIL

7.5.8. Configuring CDT parameters

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

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

Parameters

ptbuffer Charge/discharge time parameters. Please refer to **Table 11** for the details of each parameter.

Note: *Charge/discharge time parameters include:*

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

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

Return Parameters

Successful	API_OK
Failed	API_ERR_CMDFAIL

7.5.9. Reading key status

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

```
void API_ReadKeyStatus(void)
```

Parameters

No parameters.

Return Parameters

Status of the keys.

7.5.10. Reading keys

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

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

Parameters

Mode	0Xh: NORMAL, save the keys touched only. 1Xh: DISPLAY, display and save the keys touched. 2Xh: PASSWORD, display '*' and save the keys touched. 4Xh: Beep after Key Press. X0h: Keys 1, 2 …9, *, 0, # X1h: Keys A, B…I, *, _, # X, Y: Refer to section **7.4.2**.
ptkey	Point of the keys touched.
keylength	Length of keys.

Leavecondition Complete sign.

- 01h: Key-10
- 02h: Key-11
- 04h: Key-12

waittime Time for user to touch keys. The range is 0~254s and FFh is infinite.

Return Parameters

API_OK

7.6. Flash Memory API Functions

This section introduces the flash memory access API functions.

7.6.1. Writing to flash memory

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

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

Parameters

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

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

Sample Code

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

7.6.2. Reading flash memory

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

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

Parameters

pBuffer	Save the data buffer address.
ReadAddr	Read the data address (Please refer to M25P40 datasheet).
NumByteToRead	Read the data length.

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

Sample Code

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

7.6.3. Erasing flash memory

This function erases the flash memory.

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

Parameters

sector sector = 0 Erase the whole flash memory

sector = 1 Erase the 1st sector

sector = 2 Erase the 2nd sector

sector = 3 Erase the 3rd sector

sector = 4 Erase the 4th sector

sector = 5 Erase the 5th sector

sector = 6 Erase the 6th sector

sector = 7 Erase the 7th sector

sector = 8 Erase the 8th sector

sector > 8 neglect

Note: *Please refer to M25P40 datasheet.*

Return Value

Successful API_OK

Failed API_ERR_CMDFAIL

Sample Code

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

7.7. Backup Registry API Functions

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

7.7.1. Writing data to backup registry

This function is used to store data to backup registry.

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

Parameters

Addr	Address for the data (0-83).
pData	Pointer address of the data to be written.
Len	Data length.

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

Sample Code

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

7.7.2. Reading data from the backup registry

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

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

Parameters

Addr	Address of the data (0-83).
pData	Pointer address of the data to be read.
Len	Data length.

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

Sample Code

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

7.8. Encryption API Functions

This section introduces DEC, 3DEC and AES functions.

7.8.1. Encrypting DES

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

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

Parameters

pucData	Save address of encrypted/decrypted data or operation result
ulDataLen	Data length of the encrypted/decrypted data
pucKey	Save address of the encryption key (64 bits)
pucInitialVector	Save address of initial vector value
ucMode	Mode of operation: • mode = 0Eh (encryption) • mode = 0Dh (decryption) • mode = E0h (EBC) • mode = D0h (CBC)

Return Value

None.

7.8.2. Encrypting 3DES

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

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

Parameters

pucData	Save address of encrypted/decrypted data or operation result.
ulDataLen	Data length of the encrypted/decrypted data.
pucKey	Save address of the encryption key (192 bits).
pucInitialVector	Save address of initial vector value.
ucMode	Mode of operation: • mode = 0Eh (encryption) • mode = 0Dh (decryption) • mode = E0h (EBC) • mode = D0h (CBC)

Return Value

None.

7.8.3. Encrypting AES

This function is used for AES encryption and decryption process.

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

Parameters

pucData	Save address of encrypted/decrypted data or operation result
ulDataLen	Data length of the encrypted/decrypted data
pucKey	Save address of the encryption key
uiKeylen	Length of encryption key (128 bits or 256 bits)
pucInitialVector	Save address of initial vector value
ucMode	Mode of operation: • mode = 0Eh (encryption) • mode = 0Dh (decryption) • mode = E0h (EBC) • mode = D0h (CBC)

Return Value

None.

7.9. Other API Functions

7.9.1. Delaying ms

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

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

Parameters

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

ucMode Delay mode:

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

Return Value

None.

7.9.2. Delay us

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

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

Parameters

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

ucMode Delay mode:

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

Return Value

None.

7.9.3. Interfacing with a PC-linked application

The SuperApdu() API allows the ACR1283L to receive commands from a PC-Linked application and perform various tasks as defined inside the function.

```
UINT8 SuperApdu(UINT8 *Cmdbuffer,UINT8 *Resbuffer,UINT16 *pReslen)
```

Parameters

Cmdbuffer Variable that contains the APDU command from the PC application. The first byte of the command should start with FCh.

Resbuffer Variable that contains the APDU response to the command from the PC application

pReslen Length of the APDU repsonse

Return Value

Successful	API_OK
Failed	API_ERR_CMDFAIL

Sample Code

```
UINT8 SuperApdu(UINT8 *Cmdbuffer,UINT8 *Resbuffer,UINT16 *pReslen)
{
   if((Cmdbuffer[0] ==0xFC) &&(Cmdbuffer[1]==0xCD))
   {
      standalone_mode();
   }
    else if((Cmdbuffer[0] ==0xFC) &&(Cmdbuffer[1]==0xFF))
   {
     API_LcdCLear();
     API_LcdDisplay(0x01,"API Testing...",17,2,0);
     *pReslen = APITest(&Cmdbuffer[2],Resbuffer);
     API_LcdCLear();
     API_LcdDisplay(0x01,"API Test Done",13,2,0);
     return API_OK;
   }
}
```

Note: *You can define the functions the ACR1283L will perform inside the SuperApdu() API as long as the `Cmdbuffer[0]` is FCh.*

Appendix A. Status Codes

Status Code (Hex)	Description		Notes
FFh	API_ERR_CMDABORT	Command aborted.	The current command is stopped by reader.
FEh	API_ERR_MUTE	No return.	CCID timeout.
FDh	API_ERR_PARITE	Parity error.	Parity error during communication with the card.
FCh	API_ERR_LENGTH	Length error.	Data length error message.
FBh	API_ERR_HARDWARE	Hardware error.	
F8h	API_ERR_TS	TS error.	
F7h	API_ERR_TCK	TCK error.	
F6h	API_ERR_PROTOCOL	Protocol not supported.	
F5h	API_ERR_CLASS	Class not supported.	
E1h~F4h	RFU	Undefined.	Future use.
E0h	API_ERR_SLOT_BUSY	Card slot busy.	Previous command is running and new command was received.
DFh~87h	RFU	Undefined.	For future use.
86h	API_ERR_TIMEOUT	Timeout.	Execute command timeout.
85h	API_ERR_CMDFAIL	Command failed.	
84h	API_ERR_FRAM_LEN	Read/write FRAM length error.	Start address plus length is greater than 1FFFh.
83h	API_ERR_FRAM_ADDR	Read/write FRAM address error.	Start address is greater than 1FFFh.
82h	API_ERR_CMDNOTSUPPORT	Command not supported.	
81h	API_OK	Successful.	Command executed successful.
80h	API_ERR_INDEX	Card slot index error.	Slot_ Number > 05h except 08h
7Fh~00h	RFU	Undefined.	For future use.

Table 11: Status Codes

Appendix B. Keypad Configure Parameter

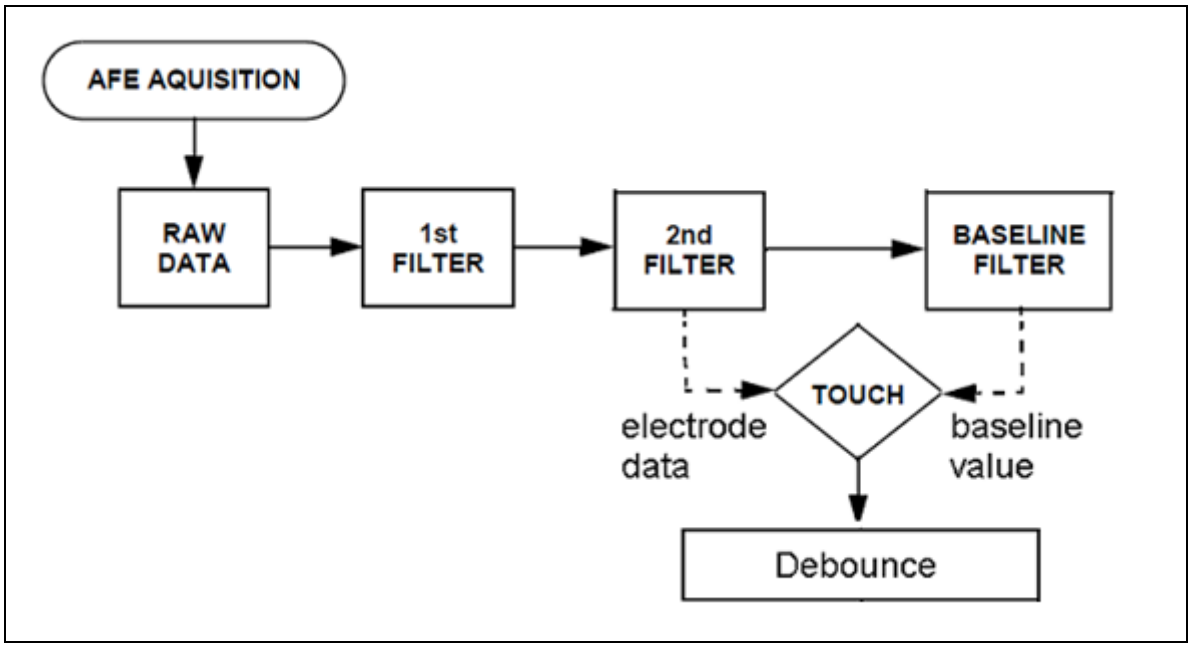


Figure 10: Capacitance Measurement Flow

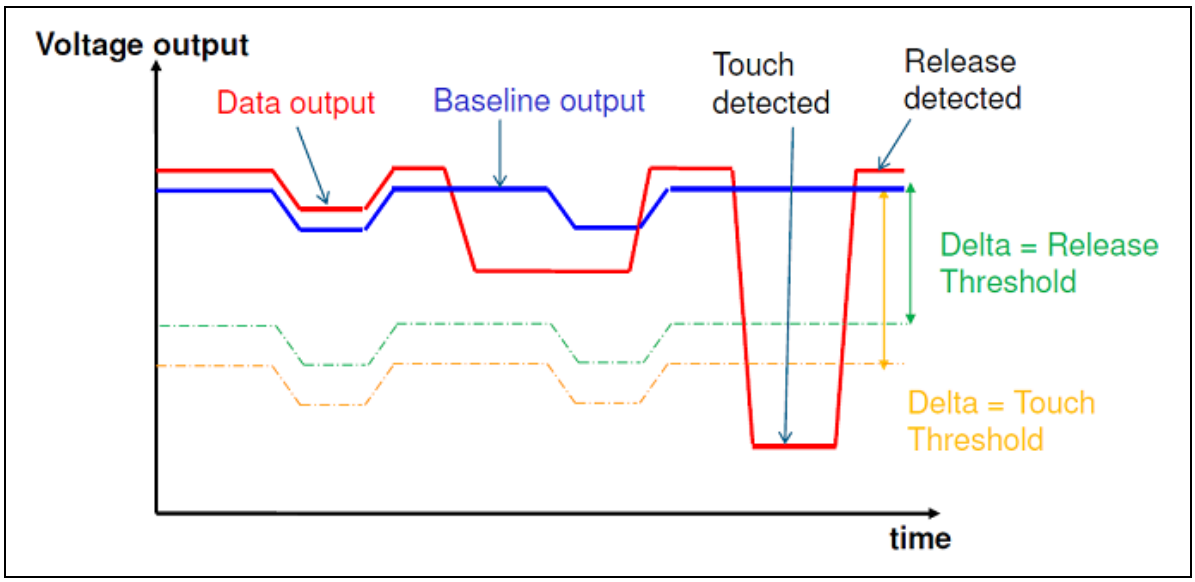


Figure 11: Touch and Release Detection Diagram

- **Baseline Control Parameters**

  Background baseline value tracking is operated by the internal third level filter.

  - **Maximum Half Delta (MHD):** Determines the largest magnitude of variation to pass through the baseline filter. The range of the effective value is 1~63.

    (MHDR = Max Half Delta for Rising, MHDF = Max Half Delta for Falling, MHDT = Max Half Delta for Touched)

  - **Noise Half Delta (NHD):** Determines the incremental change when non-noise drift is detected. The range of the effective value is 1~63.

(NHDR = Noise Half Delta for Rising, NHDF = Noise Half Delta for Falling, NHDT = Noise Half Delta for Touched)

- **Noise Count Limit (NCL):** Determines the number of samples consecutively greater than the Max Half Delta necessary before it can be determined that it is non-noise. The range of the effective value is 0~255.

  (NCLR = Noise Count Limit for Rising, NCLF = Noise Count Limit for Falling, NCLT = Noise Count Limit for Touched)

- **Filter Delay Count Limit (FDL):** Determines the rate of operation of the filter. The range of the effective value is 0~255.

  (FDLR = Filter Delay Count Limit for Rising, FDLF = Filter Delay Count Limit for Falling, FDLT = Filter Delay Count Limit for Touched)

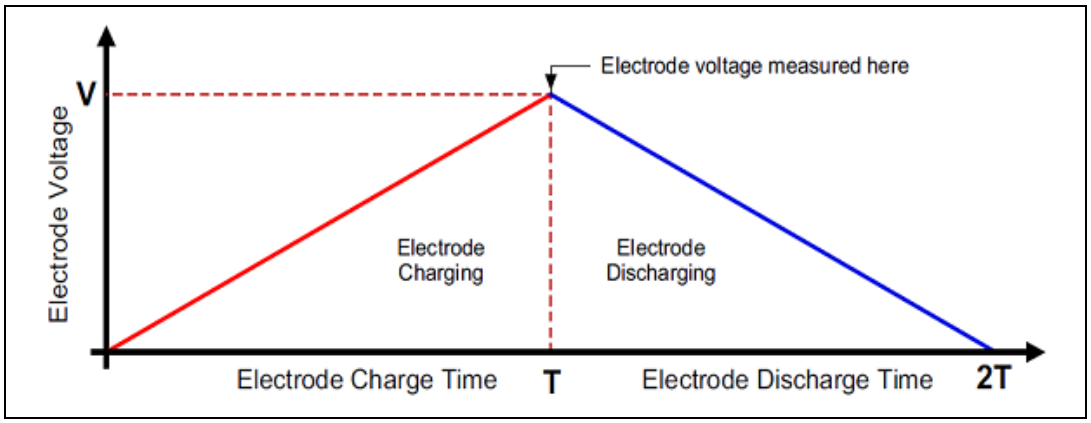
- **Threshold Parameters**

  Each of the twelve keys has its own set of touch threshold and release threshold registers. The threshold is defined as a deviation value from the baseline value, so it remains constant even the baseline value changes. Typically, the touch threshold is a little bigger than the release threshold to touch debounce and hysteresis.

  - **Touch Threshold (ExTTH)** - The range of the value is 0~255.
  - **Release Threshold (ExRTH)** - The range of the value is 0~255.

- **Charge/Discharge Current Parameters**

  Individual charge/discharge current value for each channel is the amount of current used in charging the electrode.



  - **Individual Charge Discharge Current (CDCx)** – selects the supply current to be used when charging and discharging a specific channel. Programmable in 1 µA step. The range of the effective value is 0~63.

- **Charge/Discharge Time Parameters**

  - **Individual Charge Discharge Time (CDTx)** – selects the amount of charge time for individual channels. Programmable to 0.5 (2^{n-1}) µs. The range of the effective value is 0~7.