Advanced Card Systems Limited

Card and Reader Technologies

Application Protocol Interface

ACR120U Contactless Reader/Writer

CONTENTS

SCOPES

The ACR120U USB High Level APIs are some standard functions for controlling the Reader and accessing the supported contactless-cards. By using the High Level APIs, the users can develop applications that involve the use of contactless-cards with minimum effort. For examples,

- Access control, Identification: Reading the serial numbers of all cards in the field.
- Data Storage: Performing encrypted read and write operations.
- Ticketing: Performing read, write, increment and decrement operations in an encrypted environment.
- Multi applications: Performing read, write, increment and decrement operations on various sectors of the card.

# The High Level APIs are available for Windows 98, ME, 2000 & XP Operating Systems.

USB INTERFACE

The ACR120U is connected to a computer through USB as specified in the USB Specification 1.1. The ACR120U is working in low speed mode, i.e. 1.5 Mbps.

USB Interface Wiring

Pin	Signal	Function
1	**V_{BUS}**	+5V power supply for the reader (~100mA)
2	**D-**	Differential signal transmits data between ACR120U and PC.
3	**D+**	Differential signal transmits data between ACR120U and PC.
4	**GND**	Reference voltage level for power supply

NOTE: In order for the ACR120U functioning properly through USB interface, ACS proprietary device drive has to be installed. Please refer to the *Device Driver Installation Guide* for more detail.

GROUP A. READER COMMANDS

1. ACR120_Open

High Level API:

DLLAPI INT16 AC_DECL ACR120_Open(INT16 ReaderPort);

Description	To open a port (connection) to Reader.	
Parameters	ReaderPort	The port number. Available choices are “ACR120_USB1” to “ACR120_USB8”.
Return Value	INT16	Handle for further operations. Error Code < 0

2. ACR120_Close

High Level API:

DLLAPI INT16 AC_DECL ACR120_Close(INT16 hReader);

Description	To close the port (connection) to Reader.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
Return Value	INT16	0 = success; Error Code < 0

3. ACR120_Reset

High Level API:

DLLAPI INT16 AC_DECL ACR120_Reset(INT16 hReader);

Description	To reset the Mifare Chip of the Reader, then restore the factory settings	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
Return Value	INT16	0 = success; Error Code < 0

Sample Code:

```
#include "acr120.h"

main()
{
        // Open a communication channel, the first USB Reader
        INT16 hReader=ACR120_Open(ACR120_USB1);

        // Reset the Reader to the initial state.
        if(hReader>0)
        {
                INT16 Status= ACR120_Reset(hReader);
        }
        else
        {
                // error happened
        }

        // some operations
        // Close the communication channel, the first USB Reader
        if(hReader>0)
        {
                Status= ACR120_Close(hReader);
                hReader = -1;
        }

}
```

GROUP A. READER COMMANDS

4. ACR120_Status

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Status(INT16             hReader,
              UINT8                 pFirmwareVersion[20],
              STRUCT_STATUS     pReaderStatus);
```

Description	Return the firmware version and the Reader status.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	pFirmwareVersion	The firmware version will be returned (20 bytes)
	pReaderStatus	The Reader status.
Return Value	INT16	0 = success; Error Code < 0

Sample Code:

```
#include “acr120.h”

// Obtain the Firmware version & Reader Status if the USB connection is already
// established
if(hReader>0)
{
        UINT8 FirmwareVersion[20];
        STRUCT_STATUS ReaderStatus;

        INT16 Status= ACR120_Status(hReader. FirmwareVersion, &ReaderStatus);

        If(Status== SUCCESS_READER_OP)
        {
                // do some operations if the operation is success
        }
        else
        {
                // error happened!!

        }

}
```

GROUP A. READER COMMANDS

(Cont.)

```
Struct STRUCT_STATUS
{
        // 0x01 = Type A; 0x02 = Type B; 0x03 = Type A + Type B
        UINT8           MifareInterfaceType;

        // Bit 0 = Mifare Light; Bit 1 = Mifare1K; Bit 2 = Mifare 4K; Bit 3 = Mifare DESFire
        // Bit 4 = Mifare UltraLight; Bit 5 = JCOP30; Bit 6 = Shanghai Transport
        // Bit 7 = MPCOS Combi; Bit 8 = ISO type B, Calypso
        // Bit 9 – Bit 31 = To be defined
        UINT32  CardsSupported;

        UINT8           CardOpMode; // To be defined

        UINT8           FWI;    // the current FWI value (time out value)

        UINT8           RFU; // To be defined

        UINT16  RFU2; // to be defined

} ReaderStatus;
```

GROUP A. READER COMMANDS

5. ACR120_ReadRC531Reg

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ReadRC531Reg(INT16                 hReader,
                    UINT8                 RegNo,
                    UINT8* pValue);
```

Description	To read the Mifare registers.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	RegNo	The register number.
	pValue	Mifare register's value.
Return Value	INT16	Result code. 0 means success.

6. ACR120_WriteRC531Reg

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_WriteRC531Reg(   INT16          hReader,
                        UINT8          RegNo,
                        UINT8          Value);
```

Description	To write the Mifare registers	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	RegNo	The register number.
	Value	Mifare register's value to write
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include "acr120.h"

// Read & Write the Reader Register if the USB connection is already established
if(hReader>0)
{
        UINT8 RegNo=0x05; // the register address
        UINT8 Value;            // the register value

        INT16 Status= ACR120_ReadRC531Reg(hReader, RegNo, &Value);

        If(Status== SUCCESS_READER_OP)
        {
                // Update the register value
                Value!=0x01;
                Status= ACR120_WriteRC531Reg(hReader, RegNo, Value);

        }

        if(Status!= SUCCESS_READER_OP)
        {
                // error happened!!

        }

}
```

#Users are not recommended to modify the internal register setting.

GROUP A. READER COMMANDS

7. ACR120_DirectSend

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_DirectSend(INT16       hReader,
                  UINT8       DataLength,
                  UINT8*      pData,
                  UINT8*      pResponseDataLength,
                  UINT8*      pResponseData,
                  UINT16      TimedOut);
```

Description	To send data to the Reader directly.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	DataLength (N)	The Data Length (maximum 66 bytes)
	Data	The Data to be sent
	pResponseDataLength (K)	The Response Data Length
	pResponseData	The Response Data
	TimedOut	The Time Out for waiting the response data in m-sec
Return Value	INT16	0 = success; Error Code < 0

8. ACR120_DirectReceive

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_DirectReceive(INT16       hReader,
                     UINT8       RespectedDataLength,
                     UINT8*      pReceivedDataLength,
                     UINT8*      pReceivedData,
                     UINT16      TimedOut);
```

Description	To receive data from the Reader directly.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	RespectedDataLength	The Respected Data Length to be received (maximum 64 bytes)
	pReceivedDataLength (K)	The Data Length of the received data
	pReceivedData	The Received Data
	TimedOut	The Time Out for waiting the received data in m-sec
Return Value	INT16	0 = success; Error Code < 0

# These two APIs are for special purposes.

GROUP A. READER COMMANDS

9. ACR120_RequestDLLVersion

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_RequestDLLVersion(UINT8*    pVersionInfoLength,
                         UINT8*        pVersionInfo);
```

Description	To get the reader's API DLL version information	
Parameters	pVersionInfoLength	
	pVersionInfo	It returns the DLL Version string.
Return Value	INT16	0 = success; Error Code < 0

Sample Code:

```
#include “acr120.h”

// Get the DLL Version

UINT8 Length;
UINT8 Version[40]; // the DLL Version string is less than 40 bytes long

INT16 Status=ACR120_RequestDLLVersion(&Length, Version);

if(Status== SUCCESS_READER_OP)
{
        // display the DLL version,

        Version[Length]=’\0’;      // add the terminator ‘\0’
        printf(“The DLL version is %s”, Version);
}
else
{
        // DLL Error !!

}
```

GROUP A. READER COMMANDS

10. ACR120_ReadEEPROM

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ReadEEPROM(INT16          hReader,
                  UINT8              RegNo,
                  UINT8*             pEEPROMData);
```

Description	Read the internal EEPROM.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	RegNo	The register number.
	pEEPROMData	Contain the EEPROM register’s value.
Return Value	INT16	Result code. 0 means success.

11. ACR120_WriteEEPROM

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_WriteEEPROM(INT16         hReader,
                   UINT8             RegNo,
                   UINT8             EEPROMData);
```

Description	Write the internal EEPROM.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	RegNo	The register number.
	EEPROMData	The EEPROM register’s value to write.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include "acr120.h"

// Read & Write the EEPROM if the USB connection is already established
if(hReader>0)
{
        UINT8 Address=0x04; // the address of the EEPROM to be accessed
        UINT8 Value;            // the value

        INT16 Status= ACR120_ReadEEPROM(hReader, Address, &Value);

        If(Status== SUCCESS_READER_OP)
        {
                // Update the register value
                Value &= 0x0F;
                Status= ACR120_WriteEEPROM(hReader, Address, Value);

        }

        if(Status!= SUCCESS_READER_OP)
        {
                // error happened!!
        }

}
```

GROUP A. READER COMMANDS

12. ACR120_ReadUserPort

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ReadUserPort(INT16          hReader,
                    UINT8*              pUserPortState);
```

Description	Read in the state of user port .	
Parameters	HReader	The handle to the Reader returned by ACR120_Open().
	pUserPortState	Contain the port state (only Bit 2 & Bit 6 are used).
Return Value	INT16	Result code. 0 means success.

13. ACR120_WriteUserPort

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_WriteUserPort(INT16         hReader,
                     UINT8              UserPortState);
```

Description	Update the state of user port.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	UserPortState	Contain the port state to write (only Bit 2 & Bit 6 are used).
Return Value	INT16	Result code. 0 means success.

UserPortState:
Bit 0: Not Used
Bit 1: Not Used
Bit 2: Buzzer (0 = OFF; 1 = ON)
Bit 3: Not Used
Bit 4: Not Used
Bit 5: Not Used
Bit 6: LED (0 = OFF; 1 = ON)
Bit 7: Not Used

Sample Code:

```
#include “acr120.h”

// Turn on the LED if the USB connection is already established
if(hReader>0)
{
        UINT8 PortValue;                // the value of the user port

        INT16 Status= ACR120_ReadUserPort(hReader, &PortValue);

        If(Status== SUCCESS_READER_OP)
        {
                // Turn on the LED only
                PortValue |= 0x40;
                Status= ACR120_WriteUserPort(hReader, PortValue);

        }

}
```

GROUP A. READER COMMANDS

14. ACR120_Power

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Power(INT16          hReader,
             INT8           State);
```

Description	Turn on or off the antenna power.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	State	Turn OFF (0) or ON (1).
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include "acr120.h"

// Turn off the Antenna Power for power saving
if(hReader>0)
{
        INT16 Status= ACR120_Power(hReader, 0x00);
}

// The Antenna Power will be turned on automatically if any Card Operations is started.
// E.g. ACR120_Select(). Don't need to turn on the Antenna Power manually.
// However, the Antenna Power cannot be turned off while any Card Operations is running.
```

GROUP B. GENERAL CARD COMMANDS

NOTE: All Card API's involving SECTOR and BLOCK parameters please refer to APPENDIX 5 for further explanation

1. ACR120_Select

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Select(INT16          hReader,
              UINT8*                pResultTagType,
              UINT8*                pResultTagLength,
              UINT8                 pResultSN[10]);
```

Description	Select a single card and return the card ID (Serial Number)	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	pResultTagType	Contain the selected Tag Type
	pResultTagLength	Contain the Length of the selected TAG.
	pResultSN	If the pResultTagLength = 4 or 7 or 10, the pSN contains the selected card ID (Serial Number). The ID may be 4 or 7 or 10 Bytes long.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include "acr120.h"

// Select a TAG on the reader

if(hReader>0)
{
        UINT8 TagType;                  // the Tag Type
        UINT8 TagLength;                // the length of the Tag SN
        UINT8 TagSN[10];                // The SN of the Tag

        // This API is useful for selecting a TAG in which the SN is not known in advance.
        INT16 Status= ACR120_Select(hReader, &TagType, &TagLength, TagSN);

        If(Status== SUCCESS_READER_OP)
        {
                // Now the TagSN[10] contains the SN of the Tag
                // Please check the TagLength to determine the actual length of the SN
                // e.g for Mifare 1K card, the TagLength will be equal to 0x04.
                // the TagType will be equal to 0x02;

        }
        else
        {
                // No TAG is found!!

        }
}
```

GROUP B. GENERAL CARD COMMANDS

2. ACR120_ListTags

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ListTags(INT16              hReader,
                UINT8*             pNumTagFound,
                UINT8              pTagType[4],
                UINT8              pTagLength[4],
                UINT8              pSN[4][10]);
```

Description	List out the serial numbers of all tags, which are in readable antenna range.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	pNumTagFound	Contains of number of TAG listed.
	pTagType[4]	Contains the TAG Type
	pTagLength[4]	Contains the length of the serial number.
	pSN[4][10]	The flat array of serial numbers. All serial numbers are concatenated with fixed length – 10 bytes.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include "acr120.h"

UINT8 TagFound;                 // number of TAG found
UINT8 TagType[4];               // the Tag Type
UINT8 TagLength[4];             // the length of the Tag SN
UINT8 TagSN[4][10];             // The SN of the Tag

// Find all the TAGs placed on the reader antenna. Maximum 4 TAGs can be recognized
// by the reader at the same time.
INT16 Status= ACR120_ListTags(hReader,
                 &TagFound, TagType, TagLength, TagSN);

If(Status== SUCCESS_READER_OP)
{
        // Now the TagFound contains the number of TAG recognized by the reader

        // Assume the TagFound is equal to two, Two TAGs are found
        // the TagSN[0][10] contains the SN of the first Tag
        // the TagLength[0] contains the actual length of the SN of the first TAG
        // the TagType[0] contains the TAG Type of the first TAG

        // the TagSN[1][10] contains the SN of the second Tag
        // the TagLength[1] contains the actual length of the SN of the second TAG
        // the TagType[1] contains the TAG Type of the second TAG

        // the content of TagSN[2][10], TagLength[2], TagType[2] have no meaning
        // Similarly, the content of TagSN[3][10], TagLength[3], TagType[3] have no
        // meaning
}
else { // No TAG is found!! }
```

GROUP B. GENERAL CARD COMMANDS

3. ACR120_MultiTagSelect

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_MultiTagSelect(INT16          hReader,
                      UINT8                 TagLength,
                      UINT8                 SN[10],
                      UINT8*                pResultTagType,
                      UINT8*                pResultTagLength,
                      UINT8*                pResultSN);
```

Description	To select a TAG with specific serial number.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	TagLength (N)	Contains the length of the serial number of the TAG to be selected. The TagLength may be 4, 7 or 10 bytes long.
	SN	Contain the serial number of the TAG to be selected.
	pResultTagType	Contain the selected Tag Type
	pResultTagLength (K)	Contain the length of the serial number of the selected TAG. The pResultTagLength may be 4, 7 or 10 bytes long.
	pResultSN	The serial number of the selected TAG.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include "acr120.h"

UINT8 ResultTagType;            // the Tag Type detected by the reader
UINT8 ResultTagLength;                  // the Tag length detected by the reader
UINT8 ResultTagSN[10];                  // the Tag SN detected by the reader

// The SN of the Tag is "A6 2D EA 92", the length is 4 bytes
// Fill the rest of the array with zeros
UINT8 TagSN[10]={       0xA6, 0x2D, 0xEA, 0x92, 0x00,
                        0x00, 0x00, 0x00, 0x00, 0x00};

// Select an arbitrary TAG if the SN of the TAG is known already. E.g. By using
// ACR120_ListTags()
// This API is useful for selecting an arbitrary TAG among all the TAGs.

INT16 Status= ACR120_MultiTagSelect(hReader,
                        0x04, TagSN,
                        ResultTagType, ResultTagLength, ResultTagSN);

If(Status== SUCCESS_READER_OP)
{
        // the ResultTagSN[10] contains the SN of the Tag detected by the reader
        // it must be the same as the TagSN[10]

        // the ResultTagLength contains the actual length of the SN of the TAG detected by
        //the reader. it must be the same as the TagLength

        // the ResultTagType contains the TAG Type of the TAG detected by the reader


}
else
{
        // No TAG is selected!!
}
```

GROUP B. GENERAL CARD COMMANDS

4. ACR120_TxDataTelegram

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_TxDataTelegram(INT16              hReader,
                      UINT8              SendDataLength,
                      UINT8* pSendData
                      UINT8* pReceivedDataLength,
                      UINT8* pReceivedData);
```

Description	Send data to the Selected Card.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	SendDataLength (N)	The length of the data to be sent
	pSendData	The data to be sent
	pReceivedDataLength (K)	The length of the received data
	pReceivedData	The received data
Return Value	INT16	Result code. 0 means success.

Sample Code: None (please refer to the related document for more detailed information)

The Parameter “SendData” has the following format:

Telegram Length (1 Byte)	Option Byte (1 Byte)	Data (K Bytes)
K	#	Telegram Data

Telegram Length (K): This byte is transferred too for compatibility reasons even though it could be calculated with the SendDataLength**. SendDataLength (N) = Telegram Length (K) + 2**

Option byte:
This bytes holds transfer options.

Bit 0:	if set Parity generation is enabled
Bit 1:	if set Parity is odd, otherwise Parity bit is even
Bit 2:	if set CRC generation for transmission is enabled
Bit 3:	if set CRC checking for receiving is enabled
Bit 4:	if set Crypto unit is deactivated before transmission start Activation of the Crypto unit is only possible by using the login instruction
Bit 5,6,7:	Bit Framing (Number of Bits from last Byte transmitted)

Data: The telegram data to be sent

Sample Code:

E.g. To send "RATS". {0x02, 0x0F, 0xE0, 0x50}

In which,
0x02: The DataTelegram Length
0x0F: The DataTelegram Option. Pls refer to the API Document for more detailed info.
{0xE0, 0x50}: RATS Command <DataTelegram to be sent>

```
// Sample Code for sending "RATS" to DESFire Card

        UINT8 GetRATS[]={0x02,0x0F,0xE0,0x50};
        UINT8 BlockData[64], BlockDataLength;

        CMDStatus=ACR120_TxDataTelegram(ReaderHandle, 0x04, GetRATS, &BlockDataLength,
                                                    BlockData);

        // If the command is successfully executed,
        // the BlockDataLength will be equal to 0x06
        // And the Block Data will have the data {0x06, 075, 0x77, 0x81, 0x02, 0x80}
```

#Common TeleDatagram Option Bytes Setting

- MIFare 1K/4K: 0xF3
- DESFire: 0x0F
- ISO Type B: 0x0C

GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

1. ACR120_Login

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Login(INT16          hReader,
             UINT8          Sector,
             UINT8          KeyType,
             INT8           StoredNo,
             UINT8          pKey[6]);
```

Description	Perform an authentication to access one sector of the card. Only one sector can be accessed at a time.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Sector	The sector no. to login.
	KeyType	The type of key. It can be AC_MIFARE_LOGIN_KEYTYPE_A, AC_MIFARE_LOGIN_KEYTYPE_B, AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_A, AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_B, AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_F, AC_MIFARE_LOGIN_KEYTYPE_STORED_A and AC_MIFARE_LOGIN_KEYTYPE_STORED_B
	StoredNo	The stored no of key if keyType = AC_MIFARE_LOGIN_KEYTYPE_STORED_A or AC_MIFARE_LOGIN_KEYTYPE_STORED_B.
	pKey	The login key if keyType = AC_MIFARE_LOGIN_KEYTYPE_A or AC_MIFARE_LOGIN_KEYTYPE_B. It's AC_MIFARE_KEY_LEN(6) bytes long.
Return Value	INT16	Result code. 0 means success.

Constant Definition:

AC_MIFARE_LOGIN_KEYTYPE_A	0xAA
AC_MIFARE_LOGIN_KEYTYPE_B	0xBB
AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_A	0xAD
AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_B	0xBD
AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_F	0xFD
AC_MIFARE_LOGIN_KEYTYPE_STORED_A	0xAF
AC_MIFARE_LOGIN_KEYTYPE_STORED_B	0xBF

Sample Code:

```
#include "acr120.h"

// Login the selected TAG on the reader
if(hReader>0)
{
        UINT8 TagType;                  // the Tag Type
        UINT8 TagLength;                // the length of the Tag SN
        UINT8 TagSN[10];                // The SN of the Tag

        // Select a Tag
        INT16 Status= ACR120_Select(hReader, &TagType, &TagLength, TagSN);

        // Assume a Tag is successfully selected
        // Login the Sector 0x02 with a given key (Key A Login)

        UINT8 Key[6]={ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06}; // the key used for login

        Status= ACR120_Login(hReader, 0x02,
                        AC_MIFARE_LOGIN_KEYTYPE_A, 0, Key);

        If(Status== SUCCESS_READER_OP)
        {
                // Now the Sector 0x02 is successfully authenticated (login success)
        }
        else
        {
                // The Sector 0x02 is not authenticated (login fail)!!
        }

        // some operations
        //
        //

        // Assume the Tag is still selected
        // Login the Sector 0x08 with a MasterKey 0x01 stored in Reader (Key B Login)

        Status= ACR120_Login(hReader, 0x08, AC_MIFARE_LOGIN_KEYTYPE_STORED_B,
                0x01, NULL);

        If(Status== SUCCESS_READER_OP)
        {
                // Now the Sector 0x08 is successfully authenticated (login success)
        }
        else
        {
                // The Sector 0x08 is not authenticated (login fail)!!
        }

}
```

GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

2. ACR120_Read

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Read(INT16          hReader,
            UINT8               Block,
            UINT8               pBlockData[16]);
```

Description	Read a block.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Block	The block number.
	pblockData	Contain the data read. It's AC_MIFARE_DATA_LEN(16) bytes long.
Return Value	INT16	Result code. 0 means success.

3. ACR120_ReadValue

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ReadValue(INT16          hReader,
                 UINT8          Block,
                 INT32*                 pValueData);
```

Description	Read a value.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Block	The block number.
	pValueData	Contains the value read. It's 32 bit signed integer.
Return Value	INT16	Result code. 0 means success.

GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

4. ACR120_Write

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Write(INT16            hReader,
             UINT8                   Block,
             UINT8                   pBlockData[16]);
```

Description	Write a block.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Block	The block number.
	pBlockData	Contain the data to write. It's AC_MIFARE_DATA_LEN(16) bytes long.
Return Value	INT16	Result code. 0 means success.

5. ACR120_WriteValue

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_WriteValue(INT16            hReader,
                  UINT8            Block,
                  INT32            ValueData);
```

Description	Write a value.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Block	The block number.
	ValueData	Contain the value to write. It's 32 bit signed integer.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include “acr120.h”

// Read & Write the Block if the USB connection is already established
if(hReader>0)
{
        UINT8 BlockData[16] // the data stored in the “Data Block”
        UINT8 BlockValue;       // the value stored in the “Value Block”

        // Assume the sector 0x02 is authenticated already
        // Read the block 0x08 of sector 0x02, each sector contains 4 blocks
        // Sector 0x02 consists of Blocks 0x08, 0x09, 0x0A & 0x0B

        // Assume the Block 0x08 is a “Data Block”, read the content
        INT16 Status= ACR120_Read(hReader, 0x08, BlockData);

        // update the block with a new content
        UINT8 NewBlockData[16];
        memset(NewBlockData, 0x00, 16);
        Status= ACR120_Write(hReader, 0x08, NewBlockData);

        //

        // Assume the Block 0x09 is a “Value Block”, read the value first
        Status= ACR120_ReadValue(hReader, 0x09, &BlockValue);

        // update the block with a new value. Decrease the value by 50
        Status= ACR120_WriteValue(hReader, 0x09, BlockValue-50);


}
```

GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

6. ACR120_WriteMasterKey

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_WriteMasterKey(      INT16           hReader,
                            UINT8           KeyNo,
                            UINT8           pKey[6]);
```

Description	Write master keys.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	KeyNo	The master key number.
	pKey	The key to write. It’s AC_MIFARE_KEY_LEN(6) bytes long.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
        #include “acr120.h”

        // Store a master key into the reader
        // There are totally 32 Masterkey storage space in the reader. From location 0x00 to 0x1F

        if(hReader>0)
        {

                UINT8 MasterKey[6]={0x00, 0x01, 0x02, 0x03, 0x04, 0x05};
                KeyStored=0x01;                 // The MasterKey location in the reader.

                Status= ACR120_WriteMasterKey(hReader, KeyStored,  MasterKey);

                If(Status== SUCCESS_READER_OP)
                {
                        // Now the Masterkey is successfully stored at location 0x01
                }
                else
                {
                        // The Masterkey is not stored!!
                }

        }
```

GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

7. ACR120_Inc

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Inc     (INT16  hReader,
                 UINT8  Block,
                INT32          Value,
                INT32*  pNewValue);
```

Description	Increment a value block by adding a value.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Block	The block number.
	Value	The value added to the block value.
	pNewValue	The updated value after increment.
Return Value	INT16	Result code. 0 means success.

8. ACR120_Dec

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Dec    (INT16  hReader,
                     UINT8          Block,
                     INT32          Value,
                     INT32*  pNewValue);
```

Description	Decrement a value block by subtracting a value.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Block	The block number.
	Value	The value subtracts.
	pNewValue	The updated value after decrement.
Return Value	INT16	Result code. 0 means success.

GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

9. ACR120_Copy

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Copy(INT16          hReader,
            UINT8                srcBlock,
            UINT8                desBlock,
            INT32*               pNewValue);
```

Description	Copy a value block to another value block of the same sector.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	srcBlock	The source block number.
	tgtBlock	The target block number.
	pNewValue	The updated value of the desBlock after copy.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
#include “acr120.h”

// Read & Write the Value Blocks if the USB connection is already established
if(hReader>0)
{
        UINT8 Block;     // the block number within the sector
        UINT8 BlockValue;        // the value stored in the “Value Block”

        // Assume the sector 0x02 is authenticated already
        // each sector contains 4 blocks
        // Sector 0x02 consists of Blocks 0x08, 0x09, 0x0A & 0x0B

        // Assume the Blocks 0x08 and 0x0A are “Value Block”, copy the value from block 0x08 to
        block 0x0A first.
        INT16 Status= ACR120_Copy(hReader, 0x08, 0x09, &BlockValue);
        // now the BlockValue contains the updated value of Block 0x0A

        // update the block 0x0A with a new value. Decrease the value by 100 (decimal)
        Status= ACR120_Dec(hReader, 0x0A, 100,  &BlockValue);
        // now the BlockValue contains the updated value of Block 0x09

        // update the block 0x08 with a new value. Increase the value by 56 (decimal)
        Status= ACR120_Inc(hReader, 0x08, 56, &BlockValue);
        // now the BlockValue contains the updated value of Block 0x08
}
```

GROUP D. CARD COMMANDS FOR ASK CTS256B/512B CARDS (ONLY FOR SOME SPECIAL VERSIONS)

1. ACR120_ASKSectorWrite

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ASKSectorWrite(     INT16        hReader,
                           UINT8        Sector,
                           UINT8        pBlockData[2]
                           UINT8        UpdateMode);
```

Description	Write a block.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Sector	The Sector number. For CTS256B, 0 <= Sector <= 15 For CTS512B, 0 <= Sector <= 31
	pBlockData	Contain the data to write. It's 2 bytes long.
	UpdateMode	'0' = Write. It will write '1's at the specified memory location, but not '0'. '1' = Update/Erase. It will update the specified location with the data provided. It is the only way to write '0's to the specified memory location
Return Value	INT16	Result code. 0 means success.

2. ACR120_ASKSectorRead

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ASKSectorRead(      INT16        hReader,
                           UINT8        Sector,
                           UINT8        pBlockData[2]);
```

Description	Read a block.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Sector	The Sector number. For CTS256B, 0 <= Sector <= 15 For CTS512B, 0 <= Sector <= 31
	pBlockData	Contain the data received. It's 2 bytes long.
Return Value	INT16	Result code. 0 means success.

GROUP D. CARD COMMANDS FOR ASK CTS256B/512B CARDS (ONLY FOR SOME SPECIAL VERSIONS)

3. ACR120_ASKSectorMultiRead (for CTS512B only)

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ASKSectorRead(      INT16          hReader,
                           UINT8          Sector,
                           UINT8          pBlockData[8]);
```

Description	Read 4 consecutive sector blocks.	
Parameters	hReader	The handle to the Reader returned by ACR120_Open().
	Sector	The Sector number. For CTS512B only, 0 <= Sector <= 27
	pBlockData	Contain the data received. It's 8 bytes long.
Return Value	INT16	Result code. 0 means success.

Sample Code:

```
        #include "acr120.h"

        // Read & Write the Block if the USB connection is already established
        if(hReader>0)
        {
                UINT8 BlockData[2] ={0x12, 0x34}; // the data stored in the "Data Block"
                UINT8 MultiBlockData[8];

                // Assume a Tag is selected
                // Read the 4 consecutive sector blocks starting from Sector 0x00
                INT16 Status= ACR120_ASKSectorMultiRead(hReader,
                                                0x00, MultiBlockData);

                // Read the content of Sector 0x05
                Status= ACR120_ASKSectorRead(hReader, 0x05, BlockData);

                BlockData[0] |= 0xAA;
                BlockData[1] |= 0x55;

                // Write the new BlockData to Sector 0x05, Write Mode
                Status= ACR120_ASKSectorWrite(hReader, 0x05, BlockData, 0);

                BlockData[0]=0x00; BlockData[1]=0x00;

                // Erase the content of Sector 0x05, Update Mode
                Status= ACR120_ASKSectorWrite(hReader, 0x05, BlockData, 1);


        }
```

GROUP E. CARD COMMANDS FOR ISO 14443-4 INTERFACE

1. PICC_InitBlockNumber

Format:

DLLAPI INT16 AC_DECL PICC_InitBlockNumber (INT16 FrameSizeIndex);

Function Description:

This function resets the block number to be used during the ISO14443 part 4 (T=CL) communication. This function also sets the frame length of the Card (PICC). By default the frame length is 16 bytes. The frame length of the card is reported by the ATS in type A and the ATQB in type B cards.

Parameters	Description	
Frame Size Index	An index to a maximum frame size which the card can accept	
Return Value	INT16	The actual frame length selected.

The argument only accepts the followings:

Frame Size Index	Frame Length (in bytes)
0	16
1	24
2	32
3	40
4	48
5	64
6	96
7	128
8	256
otherwise	16

Returns:
The actual frame length selected will be returned as a confirmation. e.g. if 4 is used as calling parameter, the value 48 is returned.

Notes:
This function should be called after each time with the ACR120_Select() or ACR120_MultiTagSelect() function.

Example:

```
        ACR120_Select();
        PICC_InitBlockNumber(3);
/* Reset block number and set card max frame size to 48 bytes */
```

GROUP E. CARD COMMANDS FOR ISO 14443-4 INTERFACE

2. PICC_Xch_APDU

Format:

```
DLLAPI INT16 AC_DECL PICC_Xch_APDU (
                INT16 rHandle,
                BOOL typeA,
                INT16 *pTransmitLength,
                UINT8 *pxData,
                INT16 *pReceiveLength,
                UINT8 *prData);
```

Function Description:

This function handles the APDU exchange in T=CL protocol. This routine will handle the Frame Waiting Time Extension (WTX) and chaining for long messages.

Parameters	Description	
rHandle	The handle to our reader returned by ACR120_Open	
typeA	A Boolean value indicates the card type, TRUE for type A cards, FALSE for type B cards	
pTransmitLength	A pointer to the location storing the length of the data to transmit, in bytes	
pxData	A pointer to the transmit data storage	
pReceiveLength	A pointer to the location storing the length of the data received, in bytes	
prData	A pointer to the receive data storage	
Return Value	INT16	Result code. 0 means success.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Notes:

1) The function PICC_InitBlockNumber( ) should be called each time between the ACR120_Select() or ACR120_MultiTagSelect() function and this function.
2) In many cases, the status code SW1 and SW2 are the last 2 bytes of the received data.

Example:

```
        INT16   rHandle;
        UINT8   SID;
        BOOT    typeA;
        INT16   xLen, rLen;
        UINT    rData[100];
        UINT8   Cmd[5]={0x94, 0xb2, 0x01, 0x3c, 0x1D};
        INT16   RetCode;

        xLen=5;
        SID=1;

        typeA = FALSE;          // Type B card

//Selects a single card and returns the card ID (Serial Number)
        retcode = ACR120_Select(rHandle, SID, &HaveTag, &tmpbyte, tmpArray);

if (retcode == 0)
{
        // If a card is selected, proceed to issue an APDU of 94B2013C1D
        PICC_InitBlockNumber(0);

        retcode = PICC_Xch_APDU(rHandle, SID, typeA, &xLen, Cmd, &rLen, rData);
        //check if retcode is error

        if(retcode < 0){
                // Exchange APDU failed
        } else{
                // Exchange APDU successful
        }
}
```

3. PICC_RATS

Format:

```
DLLAPI INT16 AC_DECL PICC_RATS (
        INT16 rHandle,
        UINT8   FSDI,
        UINT8 *pATSlen,
        UINT8 *pATS);
```

Function Description:

This function is only valid for ISO14443 type A cards. It requests an Answer-to-Select (ATS) message from the card after doing the ACR120_Select( ) operation. It tells the card how many bytes the reader can handle in a frame and also gets the operation parameters of the card when communicating in ISO14443 mode.

Parameters	Description	
rHandle	The handle to our reader returned by ACR120_Open	
FSDI	An index to a maximum frame size which the reader can accept. The value should not exceed 4, i.e. 48 bytes.	
pATSlen	A pointer to the location storing the length of the ATS received	
pATS	A pointer to the ATS received	
Return Value	INT16	Result code. 0 means success.

The FSDI to (Frame Size for proximity coupling Device) FSD conversion:

FSDI	FSD (in bytes)
0	16
1	24
2	32
3	40
4	48
5	64
6	96
7	128
8	256
otherwise	RFU

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A. For detailed meaning of the ATS, please refer to corresponding documents.

Note:

There is no need for calling this function in type B cards.

4. PICC_Deselect

Format:

```
DLLAPI INT16 AC_DECL PICC_Deselect(
        INT16 rHandle,
        BOOL typeA);
```

Function Description:

This function sends DESELECT (card close) signal to the cards running ISO14443 part 4 (T=CL) protocol.

Parameters	Description	
rHandle	The handle to our reader returned by ACR120_Open	
typeA	A Boolean value indicates the card type, TRUE for type A cards, FALSE for type B cards	
Return Value	INT16	Result code. 0 means success.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

APPENDIX:

1. Error Codes returned by High Level APIs

SUCCESS_READER_OP(0)
Successful operation. No Error Found.
#Handled by the DLL. The DLL has to do the consistent checking even a "Success Response Status" is returned by the device.
#Corresponding to the << Response Status 'L', 'P', 'A' & 'G' >>.

ERR_INTERNAL_UNEXPECTED(-1000)
Library internal unexpected error.
#Handled by the DLL

ERR_PORT_INVALID(-2000)
The port is invalid.
#Handled by the DLL

ERR_PORT_OCCUPIED(-2010)
The port is occupied by another application.
#Handled by the DLL

ERR_HANDLE_INVALID(-2020)
The handle is invalid.
#Handled by the DLL

ERR_INCORRECT_PARAM(-2030)
Incorrect Parameter.
#Handled by the DLL.

ERR_READER_NO_TAG(-3000, or 0xF448)
No TAG in reachable range / selected.
#Corresponding to the << Response Status 'N' >>.

ERR_READER_OP_FAILURE(-3030, or 0xF42A)
Operation failed.
#Corresponding to the << Response Status 'F' >>.

ERR_READER_UNKNOWN(-3040, or 0xF420)
Reader unknown error.
#Corresponding to the << Response Status 'C', 'O', 'X' & '?' >>.

APPENDIX:

1. ***Error Codes returned by High Level APIs (Cont.)***

ERR_READER_LOGIN_INVALID_STORED_KEY_FORMAT(-4010, or 0xF056)
Invalid stored key format in login process.
#Handled by the DLL.

ERR_READER_LOGIN_FAIL(-4011, or 0xF055)
Login failed.
#Corresponding to the << Response Status 'I' >>.

ERR_READER_OP_AUTH_FAIL(-4012, or 0xF054)
The operation or access is not authorized.
#Corresponding to the << Response Status 'I' >>.

ERR_READER_VALUE_DEC_EMPTY(-4030, or 0xF042)
Decrement failure (empty).
#Corresponding to the << Response Status 'E' >>.

ERR_READER_VALUE_INC_OVERFLOW(-4031, or 0xF041)
Increment Overflow.
#Corresponding to the << Response Status 'E' >>.

ERR_READER_VALUE_OP_FAILURE (-4032, 0xF040)
Value Operations failure. E.g. Value Increment
#Corresponding to the << Response Status 'I' >>.

ERR_READER_VALUE_INVALID_BLOCK(-4033, 0xF03F)
Block doesn't contain value.
#Corresponding to the << Response Status 'F' >>.

ERR_READER_VALUE_ACCESS_FAILURE (-4034, 0xF03E)
Value Access failure.
#Corresponding to the << Response Status 'U' >>.

APPENDIX:

2. Possible TAG Types

TAG Type Value	TAG Type Description	TAG SN Length
0x01	Mifare Light	4
0x02	Mifare 1K	4
0x03	Mifare 4K	4
0x04	Mifare DESFire	7
0x05	Mifare Ultralight	7
0x06	JCOP30	4
0x07	Shanghai Transport	4
0x08	MPCOS Combi	4
0x80	ISO Type B, Calypso	4
0x81	ASK CTS256B, Type B	8
0x82	ASK CTS512B, Type B	8

#The TAG SN Format of ASK CTS256B and CTS512B Cards

1st Byte	2nd Byte	3rd Byte	4th Byte	5th to 8th Bytes			
Manufacturing Code	Product Code	Embedded Code	Application Code	MSB (H)	MSB (L)	LSB (H)	LSB (L)
XX	0x50 (CTS256B) or 0x60 (CTS512B)	XX	XX	XX	XX	XX	XX

3. USB ID and Drivers for ACR120U

- VID_0x072F & PID_0x8003 as the USB ID of ACR120U
- ACR120.SYS will be used as the driver name for ACR120U based on ST7263
- ACR120U.DLL will be used as the DLL name for ACR120U based on ST7263.

APPENDIX:

4. Standard Program Flow

1) Before executing any Card Commands, get the Reader Handle first.
2) Select a TAG
3) Login the TAG
4) Access the TAG
5) Close the Reader Handle

// ACR120_Sample.c; a very simple program for accessing Philips MIFare 1K Tags

```
#include “acr120.h”

void main(void)
{
        INT16 hReader = -1;
        UINT8 Length, SN[10], Data[16], Type;

        // Get the Reader Handle first. Open a communication channel (USB Interface)
        hReader=ACR120_Open(ACR120_USB1);

        if(hReader<0){ // error happened!!! };

        // Assume the Reader Handle is ready, then “Select a TAG”
        ACR120_Select(hReader, &Type, &Length, SN);

        // Assume a TAG is selected, then “Login Sector 0x02” using “Default Key F”
        ACR120_Login(hReader, 0x02, AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_F, 0, NULL);

        // Assume the Sector is authorized, then “Read data from Block 0x08 of Sector 0x02”
        ACR120_Read(hReader, 0x08, Data);

        /*
        Some operations.
        */

        ACR120_Close(hReader); // Close the port and quit the program

        return;
}
```

APPENDIX:

5. Physical and Logical Block/Sector Calculation

1. Mifare 1K

- Logical Sector is equal to Physical sector, which are 0 to 15.
- Logical block of each sector is from 0 to 3.
- Physical blocks = ((Sector * 4) + Logical block)

2. Mifare 4K

- **Case 1: If {0 <= Logical Sector <= 31}**
  - Physical sector is equal to Logical.
  - Logical block of each sector is from 0 to 3.
  - Physical blocks = ((Sector * 4) + Logical block)

- **Case 2: If {32 <= Logical Sector <= 39}**
  - Physical Sector = Logical Sector + ((Logical Sector - 32) * 3)
  - Logical block of each sector is from 0 to 15.
  - Physical blocks = ((Logical Sector - 32) * 16) + 128 + Logical block