Advanced Card Systems Ltd.

ACR38 USB Smart Card Reader/Writer

APPLICATION PROGRAMMING INTERFACE

Version 2.2 11-2005

Unit 1008, 10th Floor, Hongkong International Trade and Exhibition Centre
1 Trademart Drive, Kowloon Bay, Hong Kong
Tel: +852 2796 7873 Fax: +852 2796 1286 Email: info@acs.com.hk Website: www.acs.com.hk

Contents

1. Introduction

This manual describes the use of ACR38 interface software to program the ACR38 smart card readers. It is a set of library functions implemented for the application programmers to operate the ACR38 smart card reader and the inserted smart cards. Currently, it is supplied in the form of 32-bit DLL (for Windows 95/98/NT). It can be programmed using the popular development tools like Visual C/C++, Borland C/C++, Visual Basic, Delphi, FoxPro, etc…

ACR38 series of smart card readers can be connected to the PC via the USB interface.

Even though the hardware communication interface can be different, application programs can still be using the same API (Application Programming Interface) for operating the smart card readers. Actually, the purpose of using the ACR38 library is to provide the programmer with a simple and consistent interface over all possible hardware. It is the responsibility of the ACR38 library to handle the communication details, parameter conversions and error handling. The architecture of the ACR38 library can be visualized as the following diagram:

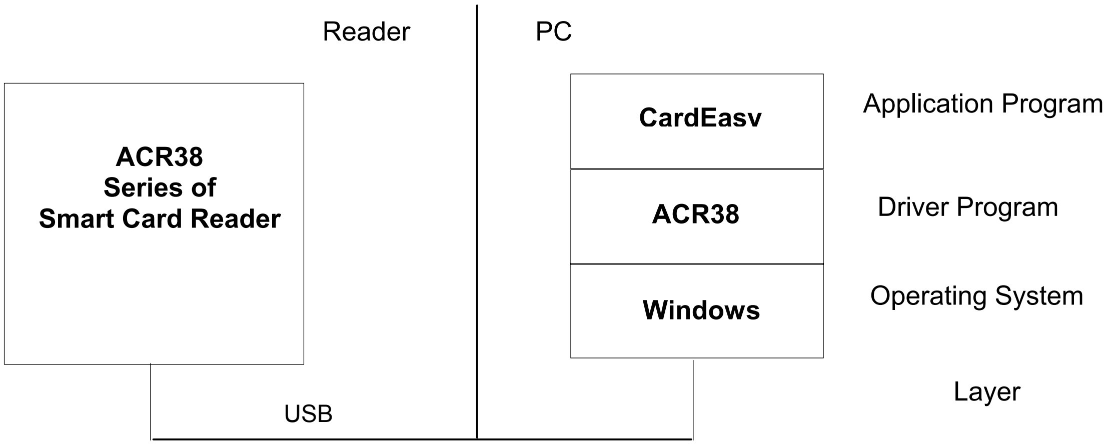

Figure 1.1

2. ACR38

2.1 Overview

ACR38 is a set of high-level functions provided for the application software to use. It provides a consistent application programming interface for the application to operate on the card reader and the corresponding inserted card. ACR38 communicates with the ACR38 reader via the communication port facilities provided by the operating system. ACR38 is supposed to be platform independent provided that there is a minor modification on the communication module of the ACR38 to adapt to different operating environment.

2.2 Communication Speed

The ACR38 library controls the communication speed between the reader and the PC. The speed is fixed at 1.5Mbps respectively.

3. ACR38 API

The ACR38 Application Programming Interface (API) defines a common way of accessing the ACR38 reader. Application programs invoke ACR38 through the interface functions and perform operations on the inserted card through the using of ACI commands. The header file ACR38.H is available for the program developer which contains all the function prototypes and macros described below.

3.1 Interface Data Structure

The ACR38 API makes use of several data structures to pass parameters between application programs and the library driver. These data structures are defined in the header file ACR38.H and they are discussed below:

3.1.1 AC_APDU

```
typedef     struct {
      BYTE        CLA;
      BYTE        INS;
      BYTE        P1;
      BYTE        P2;
      INT16       Lc;
      INT16       Le;
      BYTE        DataIn[256];
      BYTE        DataOut[256];
      WORD16      Status;
} AC_APDU;
```

The AC_APDU data structure is used in the AC_ExchangeAPDU function for the passing of commands and data information into the smart card. For memory card operation, please refer to section **3.3** for the definition of fields' value. For MCU card (T=0,T=1) operation, these value are specific to the smart card operating system. You must have the card reference manual before you can perform any valid operations on the card. Please notice that Lc representing the data length going into the card and Le representing the data length expecting from the card.

Name	Input/Output	Description
CLA	I	Instruction Class
INS	I	Instruction Code
P1	I	Parameter 1
P2	I	Parameter 2
Lc	I	Length of command data (DataIn)
Le	I/O	Length of response data (DataOut)
DataIn	I	Command data buffer
DataOut	O	Response data buffer
Status	O	Execution status of the command

3.1.2 AC_SESSION

```
typedef     struct {
      BYTE  CardType;    // Card type selected
       BYTE SCModule;    // Selected security module.
                         //Use only when card type = AC_SCModule
      BYTE  ATRLen;      // Length of the ATR
      BYTE  ATR[128];    // ATR string
      BYTE  HistLen;     // Length of the Historical data
      BYTE  HistOffset;  // Offset of the Historical data
                         // from the beginning of ATR
      INT16 APDULenMax;  // Max. APDU supported
} AC_SESSION;
```

The AC_SESSION data structure is used in the AC_StartSession function call for the retrieval of ATR information from the smart card. Before calling AC_StartSession, the program needs to specify the value of CardType. After calling the function, the ATR string can be found in ATR field and the length is stored in ATRLen.

Name	Input/Output	Description
CardType	I	The card type selected for operation.
SCModule	I	The security module selected for operation.
ATRLen	O	Length of the ATR string
ATR	O	Attention to reset (ATR) string
HistLen	O	Obsolete field – not used anymore
HistOffset	O	Obsolete field – not used anymore
APDULenMax	O	Obsolete field - not used anymore

3.1.3 AC_INFO

```
typedef     struct {
      INT16 nMaxC;       // Maximum number of command data bytes
      INT16 nMaxR;       // Maximum number of data bytes that
                         // can be requested in a response
      INT16 CType;       // The card types supported by the reader
      BYTE  CStat;       // The status of the card reader
      BYTE  CSel;        // The current selection of card type
      BYTE  szRev[10];   // The 10 bytes firmware type and
                         // revision code
      INT16  nLibVer;    // Library version
      Long   lBaudRate;  // Current Running Baud Rate
} AC_INFO;
```

The AC_INFO data structure is used in the AC_GetInfo function call for the retrieval of reader related information. Their meaning are described as follow:

Name	Input/Output	Description
nMaxC	O	The maximum number of command data byte (DataIn) that can be accepted in the ExchangeAPDU command
nMaxR	O	The maximum number of response data byte (DataOut) that will be appeared in the ExchangeAPDU command
CType	O	The card types supported by the reader (For details, please look at the ACR20 reference manual)

Cstat	O	The status of the card reader
		Bit0 = card present (1) or absent (0)
		Bit1 = card powered up (1) or powered down (0)

szRev[10]	O	The firmware revision code
nLibVer	O	Library version (e.g. 310 is equal to version 3.10)

3.2 Interface Function Prototypes

Generally, a program is required to call AC_Open first to obtain a handle. The handle is required for subsequent calls to AC_StartSession, AC_ExchangeAPDU, AC_EndSession and AC_Close. The inserted card can be powered up by using the AC_StartSession function and card commands can be exchanged with the inserted card using the AC_ExchangeAPDU function. Moreover, AC_SetOptions and AC_GetInfo are two commands that can be used to set and read the various information of the reader.

3.2.1 AC_Open

This function opens a port and returns a valid reader handle for the application program.

Format:

INT16 AC_DECL AC_Open (INT16 ReaderType, INT16 ReaderPort);

Input Parameters:

The table below lists the parameters for this function (you can refer to ACSR20.H for the corresponding value):

Parameters	Definition / Values	
ReaderType	The target reader type:	
	Value	Meaning
	ACR30	Target reader is ACR30
	ACR38	Target reader is ACR38
	ACR_AUTODETECT	Auto detect the target reader
ReaderPort	The port connected with the reader:	
	Value	Meaning
	AC_COM1	Standard communication port 1
	AC_COM2	Standard communication port 2
	AC_COM3	Standard communication port 3
	AC_COM4	Standard communication port 4
	AC_USB	USB communication port

Returns:

The return value is negative and contains the error code when the function encounters an error during operation. Otherwise, it returns a valid reader handle. Please refer to appendix A for the detailed description and meaning of the error codes.

Examples:

```
// open a port to a ACR30 reader connected to COM1
INT16   hReader;

hReader = AC_Open(ACR30,AC_COM1);
```

3.2.2 AC_Close

This function closes a previously opened reader port.

Format:

```
INT16 AC_DECL AC_Close (INT16 hReader);
```

Input Parameters:

The table below lists the parameters for this function

Parameter	Definition / Values
hReader	A valid reader handle previously opened by AC_Open

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.

Examples :

```
// Close a previously opened port
INT16 RtnCode;
RtnCode = AC_Close(hReader);
```

3.2.3 AC_StartSession

This function starts a session with a selected card type and updates the session structure with the values returned by the card Answer-To-Reset (ATR). A session is started by a card reset and it is ended by either another card reset, a power down of the card or the removal of a card from the reader. Note that this function will power up the card and perform a card reset.

Format:

```
INT16 AC_DECL AC_StartSession (INT16 hReader, AC_SESSION FAR *Session);
```

Input Parameters:

The table below listed the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle previously opened by AC_Open

<table>
<tr><th>Parameters</th><th>Definition / Values</th></tr>
<tr><td>Session.CardType</td><td>Card type for the session: (X – Support)
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>AC_AUTO</td><td>Auto-select T=0 or T=1 communication protocol</td></tr>
<tr><td>AC_I2C_1K_16K</td><td>I2C memory card (1k, 2k, 4k, 8k and 16k bits)</td></tr>
<tr><td>AC_I2C_32K_1024K</td><td>I2C memory card (32k, 64k, 128k, 256k, 512k and 1024k bits)</td></tr>
<tr><td>AC_AT88SC153</td><td>Atmel AT88SC153 secure memory card</td></tr>
<tr><td>AC_AT88SC1608</td><td>Atmel AT88SC1608 secure memory card</td></tr>
<tr><td>AC_SLE4418</td><td>Infineon SLE4418</td></tr>
<tr><td>AC_SLE4428</td><td>Infineon SLE4428</td></tr>
<tr><td>AC_SLE4432</td><td>Infineon SLE4432</td></tr>
<tr><td>AC_SLE4442</td><td>Infineon SLE4442</td></tr>
<tr><td>AC_MCU_T0</td><td>MCU-based cards with T=0 communication protocol</td></tr>
<tr><td>AC_MCU_T1</td><td>MCU-based cards with T=1 communication protocol</td></tr>
<tr><td>AC_SAM_T0</td><td>SAM Slot MCU-based cards with T=0 communication protocol</td></tr>
<tr><td>AC_SAM_T1</td><td>SAM Slot MCU-based cards with T=1 communication protocol</td></tr>
</table>
</td></tr>
</table>

Output Parameters:

The table below listed the parameters returned by this function

Parameters	Definition / Values
Session.ATR	Answer to Reset (ATR) returned by the card
Session.ATRLen	Length of the ATR

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.

Examples:

```
// Prepare Session structure for SLE 4442 memory card
INT16   RtnCode,i;
AC_SESSION Session;

Session.CardType = AC_SLE4442;  // Card type = SLE4442

//Start a session on previously opened port
RtnCode = AC_StartSession(hReader, &Session);

// Print the card ATR
```

```
printf("Card Answer to Reset : ");
for (i = 0; i < (INT16) Session.ATRLen; i++)
       printf(" %02X",Session.ATR[i]);
```

Remarks:

1)
When AC_AUTO is selected, the reader will try to detect the inserted card type automatically (in main slot). However, while the reader can distinguish the T=0 and T=1 card. It cannot distinguish different types of memory card.

2)
For accessing the MCU card in SAM slot, besides opening a port, you can need select the AC_SAM_T0 (for T=0 card) and AC_SAM_T1 (for T=1 card) in calling AC_StartSession.

3.2.4 AC_EndSession

This function ends a previously started session and powers off the card.

Format:

```
INT16 AC_DECL AC_EndSession (INT16 hReader);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()

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.

Examples:

```
//End session on a previously started session
RtnCode = AC_EndSession(hReader);
```

3.2.5 AC_ExchangeAPDU

This function sends an APDU command to a card via the opened port and returns the card's response. Please refer Section 2.3.3 ACI Commands for detail description on how to fill in the parameters.

Format:

```
INT16 AC_DECL AC_ExchangeAPDU (INT16 hReader, AC_APDU FAR *Apdu);
```

Input Parameters:

The table below listed the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()
Apdu.CLA	Instruction Class (Please refer Section 2.3.3 ACI Commands for detail description)
Apdu.INS	Instruction Code (Please refer Section 2.3.3 ACI Commands for detail description)
Apdu.P1	Parameter 1 (Please refer Section 2.3.3 ACI Commands for detail description)
Apdu.P2	Parameter 2 (Please refer Section 2.3.3 ACI Commands for detail description)
Apdu.DataIn	Data buffer to send
Apdu.Lc	Number of bytes in Apdu.DataIn to be sent
Apdu.Le	Number of bytes expected to receive

Output Parameters:

The table below listed the parameters returned by this function

Parameters	Definition / Values
Apdu.DataOut	Data buffer containing the card response
Apdu.Le	Number of bytes received in Apdu.DataOut
Apdu.Status	Status bytes SW1, SW2 returned by the card

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.

Examples:

```
// Read 8 bytes from SLE4442 from address 0
INT16 RtnCode, i;
APDU  apdu;

apdu.CLA     = 0x00;
apdu.INS     = ACI_Read;
apdu.P1      = 0;
apdu.P2      = 0;
apdu.Lc      = 0;
apdu.Le      = 8;
  RTNCODE    = AC_EXCHANGEAPDU(HREADER, &APDU);
If (RtnCode == 0)
{
      // print the data
      printf("Data :");
      for (i = 0; i < apdu.Le; i++)
      {
            printf(" %02X", apdu.DataOut[i]);
      }
      printf("Card Status (SW1 SW2) = %04X", apdu.Status);
}
```

3.2.6 AC_GetInfo

This function retrieve information related to the currently selected reader.

Format :

```
INT16 AC_DECL AC_GetInfo (INT16 hReader, AC_INFO FAR *Info);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()
Info	Pointer to the AC_INFO structure

Output Parameters:

The table below lists the parameters returned by this function

<table>
<tr><th>Parameters</th><th>Definition / Values</th></tr>
<tr><td>Info.szRev</td><td>Revision code for the selected reader.</td></tr>
<tr><td>Info.nMaxC</td><td>The maximum number of command data bytes.</td></tr>
<tr><td>Info.nMaxR</td><td>The maximum number of data bytes that can be requested to be transmitted in a response</td></tr>
<tr><td>Info.Ctype</td><td>The card types supported by this reader</td></tr>
<tr><td>Info.Cstat</td><td>The current status of the reader:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>No card Inserted</td></tr>
<tr><td>01</td><td>Card Inserted but Not Power Up</td></tr>
<tr><td>03</td><td>Card Inserted and Powered Up</td></tr>
</table></td></tr>
<tr><td>Info.CSel</td><td>The currently selected card type</td></tr>
<tr><td>Info.nLibVer</td><td>Current library version. E.g. 310 means version 3.10</td></tr>
<tr><td>Info.lBaudRate</td><td>The current running baud rate</td></tr>
</table>

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.

Examples:

```
// Get the revision code of the currently selected reader
INT16 RtnCode;
AC_INFO Info;

RtnCode = AC_GetInfo(hReader, &Info);
printf("Reader Operating System ID : %s",Info.szRev);
```

3.2.7 AC_SetOptions

This function set various options for the reader.

Format:

INT16 AC_DECL AC_SetOptions (INT16 hReader, WORD16 Type, WORD16 Value);

Input Parameters:

The table below lists the parameters for this function

Parameter	Definition / Values
hReader	A valid reader handle returned by AC_Open()
Type	Type of options that is going to set
Value	Value parameter for the selected option type

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value meaning that the option setting is not available.

Options :

Type	Option	Value
ACO_SET_BAUD_RATE	Set the communication baud rate between the reader and the host	ACO_B9600 ACO_B14400 ACO_B19200 ACO_B28800 ACO_B38400 ACO_B57600 ACO_B115200
ACO_SET_BAUD_HIGHEST	Set the communication to highest baud rate.	0
ACO_SET_CHAR_DELAY	Set the communication inter character delay between the reader and the host	0 – 255
ACO_ENABLE_GET_RESPONSE	Enable the reader to issue the GET_RESPONSE command automatically (only valid for the MCU card)	SW1 + “00” (GET_RESPONSE will be issued automatically when this SW1 is returned from the card)
ACO_DISABLE_GET_RESPONSE	Disable the automatic issue of the GET_RESPONSE command (this is the default option of the reader).	0
ACO_EJECT_CARD	Eject the card	0
ACO_ENABLE_INIT_DO_PPS	Enable the reader to do PPS negotiation with the card in AC_StartSession.	0
ACO_DISABLE_INIT_DO_PPS	Disable the reader to do PPS negotiation with the card in AC_StartSession.	0

* Function return 0 when that option is supported, otherwise it is not supported

Examples:

```
// Set the communication baud rate to the highest possible setting
INT16 RtnCode;

RtnCode = AC_SetOption(hReader, ACO_SET_BAUD_HIGHEST, 0);
if (RtnCode < 0)
      printf("Set option failed\n");
```

3.3 ACI Commands

ACI commands are provided to support the standard operations of a wide range of memory cards. Because of the different nature of different memory cards and their capabilities, not all commands a available to different type of cards. The table below list out the supported commands for different type of cards.

3.3.1 AC_I2C_1K_16K / AC_I2C_32K_1024K

3.3.1.1 ACI_Read

It is used to read data from certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Read	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	0	No input data is required
DataIn	-	Don't Care
Le	Variable	Number of bytes to be read

The data read will be stored in DataOut field.

3.3.1.2 ACI_Write

It is used to write data to certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Write	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	Variable	Number of byte to be written
DataIn	Data	Data to be written
Le	0	No response data expected.

3.3.1.3 ACI_CardOptions

It is used to change the page size.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_CardOptions</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>Page Size:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>AC_APDU_CARDOPTION_PAGESIZE_8</td><td>8 Bytes</td></tr>
<tr><td>AC_APDU_CARDOPTION_PAGESIZE_16</td><td>16 Bytes</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td></td><td></td><td><table>
<tr><td>AC_APDU_CARDOPTION_PAGESIZE_32</td><td>32 Bytes</td></tr>
<tr><td>AC_APDU_CARDOPTION_PAGESIZE_64</td><td>64 Bytes</td></tr>
<tr><td>AC_APDU_CARDOPTION_PAGESIZE_128</td><td>128 Bytes</td></tr>
</table></td></tr>
<tr><td>P2</td><td>-</td><td>Don’t care</td></tr>
<tr><td>Lc</td><td>-</td><td>No input data is required.</td></tr>
<tr><td>DataIn</td><td>-</td><td>Don’t care.</td></tr>
<tr><td>Le</td><td>-</td><td>No response data expected.</td></tr>
</table>

3.3.2 AT88SC153

3.3.2.1 ACI_Read

It is used to read data from certain address.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Read</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>Zone Address:<table>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_00</td><td>Zone 0</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_01</td><td>Zone 1</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_10</td><td>Zone 2</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_11</td><td>Zone 3</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_FUSE</td><td>Zone Fuse</td></tr>
</table></td></tr>
<tr><td>P2</td><td>Variable</td><td>Starting Address</td></tr>
<tr><td>Lc</td><td>0</td><td>No input data is required</td></tr>
<tr><td>DataIn</td><td>-</td><td>Don’t Care</td></tr>
<tr><td>Le</td><td>Variable</td><td>Number of bytes to be read</td></tr>
</table>

The data read will be stored in DataOut field.

3.3.2.2 ACI_Write

It is used to write data to certain address.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Write</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>Zone Address:<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_00</td><td>Zone 0</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_01</td><td>Zone 1</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_10</td><td>Zone 2</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_11</td><td>Zone 3</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td></td><td></td><td><table><tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_11</td><td>Zone 3</td></tr><tr><td>AC_APDU_ACI_RW_AT88SC153_ZONE_FUSE</td><td>Zone Fuse</td></tr></table></td></tr>
<tr><td>P2</td><td>Variable</td><td>Starting Address</td></tr>
<tr><td>Lc</td><td>Variable</td><td>Number of bytes to be written</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Data to be written</td></tr>
<tr><td>Le</td><td>0</td><td>No response data expected.</td></tr>
</table>

3.3.2.3 ACI_Verify

It is used to submit transport code to the card in order to enable the card personalization mode.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Verify</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>Password set number</td></tr>
<tr><td>P2</td><td>Variable</td><td>Verify Mode:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>AC_APDU_ACI_VERIFY_AT88SC153_WRITE</td><td>Write</td></tr><tr><td>AC_APDU_ACI_VERIFY_AT88SC153_READ</td><td>Read</td></tr></table></td></tr>
<tr><td>Lc</td><td>3</td><td>Transport code length (3 bytes)</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Transport code (3 bytes)</td></tr>
<tr><td>Le</td><td>0</td><td>No response data expected.</td></tr>
</table>

3.3.2.4 ACI_Authenticate

It is used to generate a card authentication certificate.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Authenticate</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>Authenticate Mode:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>AC_APDU_ACI_AUTH_AT88SC153_INIT</td><td>Initialize</td></tr><tr><td>AC_APDU_ACI_AUTH_AT88SC153_VERIFY</td><td>Verify</td></tr></table></td></tr>
<tr><td>P2</td><td>-</td><td>Don’t Care</td></tr>
<tr><td>Lc</td><td>8</td><td>Host random number/challenge length (8 bytes)</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Host random number/challenge length (8 bytes)</td></tr>
<tr><td>Le</td><td>0</td><td>No response data expected.</td></tr>
</table>

3.3.3 AT88SC1608

3.3.3.1 ACI_Read

It is used to read data from certain address.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Read</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>
<table>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Value</td><td>-</td><td>-</td><td colspan="2">ZZ</td><td>-</td><td colspan="3">AAA</td></tr>
</table>
ZZ -- Zone Address (2 bits):
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC1608_ZONE_USER (0x10)</td><td>User</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC1608_ZONE_CONFIG (0x20)</td><td>Config</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC1608_ZONE_FUSE (0x30)</td><td>Fuse</td></tr>
</table>
AAA – 3 bit address (MSB)</td></tr>
<tr><td>P2</td><td>Variable</td><td>Starting Address (LSB)</td></tr>
<tr><td>Lc</td><td>0</td><td>No input data is required</td></tr>
<tr><td>DataIn</td><td>-</td><td>Don’t Care</td></tr>
<tr><td>Le</td><td>Variable</td><td>Number of bytes to be read</td></tr>
</table>

The data read will be stored in DataOut field.

Examples:

```
// Reading 10 bytes from 0x31234 on User Zone
AC_APDU      Apdu;
INT16        Result;

Apdu.CLA     = 0x00;
Apdu.INS     = ACI_Read;
Apdu.P1      = AC_APDU_ACI_RW_AT88SC1608_ZONE_USER | 0x03;
Apdu.P2      = 0x1234;
Apdu.Lc      = 0;
Apdu.Le      = 10;
Result = AC_ExchangeAPDU(hReader, &Apdu);
```

3.3.3.2 ACI_Write

It is used to write data to certain address.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Write</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>
<table>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Value</td><td>-</td><td>-</td><td colspan="2">ZZ</td><td>-</td><td colspan="3">AAA</td></tr>
</table>
ZZ -- Zone Address (2 bits):
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC1608_ZONE_USER (0x10)</td><td>User</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC1608_ZONE_CONFIG (0x20)</td><td>Config</td></tr>
<tr><td>AC_APDU_ACI_RW_AT88SC1608_ZONE_FUSE (0x30)</td><td>Fuse</td></tr>
</table>
AAA – 3 bit address (MSB)</td></tr>
<tr><td>P2</td><td>Variable</td><td>Starting Address (LSB)</td></tr>
<tr><td>Lc</td><td>Variable</td><td>Number of bytes to be written</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Data to be written</td></tr>
<tr><td>Le</td><td>0</td><td>No response data expected.</td></tr>
</table>

Examples:

```
// Writing 10 bytes from 0x31234 on User Zone
AC_APDU        Apdu;
INT16          Result;

Apdu.CLA       = 0x00;
Apdu.INS       = ACI_Write;
Apdu.P1        = AC_APDU_ACI_RW_AT88SC1608_ZONE_USER | 0x03;
Apdu.P2        = 0x1234;
Apdu.Lc        = 10;
memcpy(Apdu.DataIn, data, 10);
Apdu.Le        = 0;
Result = AC_ExchangeAPDU(hReader, &Apdu);
```

3.3.3.3 ACI_Verify

It is used to submit transport code to the card in order to enable the card personalization mode.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Verify	Instruction Code

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>P1</td><td>Variable</td><td>Password set number</td></tr>
<tr><td>P2</td><td>Variable</td><td>Verify Mode:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>AC_APDU_ACI_VERIFY_AT88SC1608_WRITE</td><td>Write</td></tr>
<tr><td>AC_APDU_ACI_VERIFY_AT88SC1608_READ</td><td>Read</td></tr>
</table></td></tr>
<tr><td>Lc</td><td>3</td><td>Transport code length (3 bytes)</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Transport code (3 bytes)</td></tr>
<tr><td>Le</td><td>0</td><td>No response data expected.</td></tr>
</table>

3.3.3.4 ACI_Authenticate

It is used to generate a card authentication certificate.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Authenticate</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>Variable</td><td>Authenticate Mode:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>AC_APDU_ACI_AUTH_AT88SC1608_INIT</td><td>Initialize</td></tr>
<tr><td>AC_APDU_ACI_AUTH_AT88SC1608_VERIFY</td><td>Verify</td></tr>
</table></td></tr>
<tr><td>P2</td><td>-</td><td>Don’t Care</td></tr>
<tr><td>Lc</td><td>8</td><td>Host random number/challenge length (8 bytes)</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Host random number/challenge length (8 bytes)</td></tr>
<tr><td>Le</td><td>0</td><td>No response data expected.</td></tr>
</table>

3.3.4 SLE4418 / SLE4428

3.3.4.1 ACI_Read

It is used to read data from certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Read	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	0	No input data is required
DataIn	-	Don’t Care
Le	Variable	Number of bytes to be read

The data read will be stored in DataOut field.

3.3.4.2 ACI_Write

It is used to write data to certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Write	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	Variable	Number of bytes to be written
DataIn	Data	Data to be written
Le	0	No response data expected.

3.3.4.3 ACI_WritePr

It is used to write protected data to certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_WritePr	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	Variable	Number of bytes to be written
DataIn	Data	Data to be written
Le	0	No response data expected.

3.3.4.4 ACI_Verify [SLE4428 Only]

It is used to submit transport code to the card in order to enable the card personalization mode.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Verify	Instruction Code
P1	-	Don’t care
P2	-	Don’t care
Lc	2	Transport code length (2 bytes)
DataIn	Data	Transport code
Le	3	Error Count (1 bytes) + Transport code read from the card (2 bytes)

3.3.4.5 *ACI_ReadProtect [SLE4428 Only]*

It is used to read in the error count and transport code from the card.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_ReadProtect</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>-</td><td>Don’t care</td></tr>
<tr><td>P2</td><td>-</td><td>Don’t care</td></tr>
<tr><td>Lc</td><td>0</td><td>No input data is required</td></tr>
<tr><td>DataIn</td><td>-</td><td>Don’t care</td></tr>
<tr><td>Le</td><td>3</td><td>Error Count (1 bytes) + Transport code read from the card (2 bytes)</td></tr>
<tr><td>DataOut</td><td>Data</td><td><table><tr><td>Byte</td><td>1</td><td>2</td><td>3</td></tr><tr><td>Data</td><td>ErrCnt</td><td colspan="2">Transport Code</td></tr></table></td></tr>
</table>

3.3.5 SLE4432 / SLE4442

3.3.5.1 *ACI_Read*

It is used to read data from certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Read	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	0	No input data is required
DataIn	-	Don’t Care
Le	Variable	Number of bytes to be read

The data read will be stored in DataOut field.

3.3.5.2 *ACI_Write*

It is used to write data to certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_Write	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	Variable	Number of bytes to be written
DataIn	Data	Data to be written
Le	0	No response data expected.

3.3.5.3 ACI_WritePr

It is used to write protected data to certain address.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_WritePr	Instruction Code
P1	Variable	Starting Address (MSB)
P2	Variable	Starting Address (LSB)
Lc	Variable	Number of bytes to be written
DataIn	Data	Data to be written
Le	0	No response data expected.

3.3.5.4 ACI_Verify [SLE4442 Only]

It is used to submit transport code to the card in order to enable the card personalization mode.

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td>CLA</td><td>0x00</td><td>Instruction Class</td></tr>
<tr><td>INS</td><td>ACI_Verify</td><td>Instruction Code</td></tr>
<tr><td>P1</td><td>-</td><td>Don’t care</td></tr>
<tr><td>P2</td><td>-</td><td>Don’t care</td></tr>
<tr><td>Lc</td><td>3</td><td>Transport code length (3 bytes)</td></tr>
<tr><td>DataIn</td><td>Data</td><td>Transport code</td></tr>
<tr><td>Le</td><td>4</td><td>Error Count (1 bytes) + Transport code read from the card (3 bytes)</td></tr>
<tr><td>DataOut</td><td>4</td><td><table><tr><td>Byte</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>Data</td><td>ErrCnt</td><td colspan="3">Transport Code</td></tr></table></td></tr>
</table>

3.3.5.5 ACI_ReadProtect [SLE4442 Only]

It is used to read in the error count and transport code from the card.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_ReadProtect	Instruction Code
P1	-	Don’t care
P2	-	Don’t care
Lc	0	No input data is required
DataIn	-	Don’t care
Le	4	Error Count (1 bytes) + Transport code read from the card (4 bytes)

3.3.5.6 ACI_ChangePIN [SLE4442 Only]

It is used to change the PIN code stored in the card.

Field	Value	Description
CLA	0x00	Instruction Class
INS	ACI_ChangePIN	Instruction Code
P1	-	Don’t care
P2	-	Don’t care
Lc	3	New PIN code length (3 bytes)
DataIn	Data	New PIN code (3 bytes)
Le	0	No response data expected.

Appendix A: Table of error codes

Code	Meaning
-603	Error in the reader handle
-600	Session parameter is null
-108	No free handle left for allocation
-100	Selected port is invalid
-101	Selected reader is invalid
-102	Selected port is occupied
-1001	No card type selected
-1002	No card is inserted
-1003	Wrong card type
-1004	Card not powered up
-1005	INS is invalid
-1006	Card failure
-1007	Protocol error
-1008	Card type not supported
-1009	Incompatible command
-1010	Error in address
-1011	Data length error
-1012	Error in response length
-1013	Secret code locked
-1014	Invalid SC module number
-1015	Incorrect password
-1050	Error in CLA
-1051	Error in APDU parameters
-1052	Communication buffer is full
-1053	Address not align with word boundary
-1080	Protocol frame error
-1081	No response from reader
-1082	Error found in the calling function’s parameters
-1083	Specified function not supported
-1084	Connector short circuit
-1085	Unexpected internal error
-1086	A required DLL file is missing
-1099	Unknown response
-2000	USB internal error
-2001	Error in memory allocation
-2002	Error in linking USB library
-2003	Error in locating window system directory
-3000	Error found in PCSC smart card manager