

Advanced Card Systems Ltd.



ACR30 Smart Card Reader/Writer



PC/SC MEMORY CARD ACCESS REFERENCE MANUAL

Version 1.1 10-2004

Advanced Card Systems Ltd.
Unit 2910-2913, 29/F, The Center,
99 Queen's Road Central, Hong Kong

Tel: +852 2796 7873
Website: www.acs.com.hk

Fax: +852 2796 1286
Email: info@acs.com.hk

Contents

1.	Introduction.....	3
2.	Supported Memory Cards and System Requirements	3
2.1	Supported Memory Cards.....	3
2.2	System Requirements	3
3.	Memory Card Access	4
3.1	Logical Flow of Memory Card Functions	4
3.2	PC/SC Functions for Memory Card Access	5
3.2.1	<i>SCardEstablishContext</i>	5
3.2.2	<i>SCardReleaseContext</i>	6
3.2.3	<i>SCardListReaders</i>	7
3.2.4	<i>SCardConnect</i>	8
3.2.5	<i>SCardDisconnect</i>	10
3.2.6	<i>SCardTransmit</i>	11
3.2.7	<i>SCardControl</i>	13
3.3	ACR30U Command Set for Memory Card Access.....	15
3.3.1	<i>SLE4406 / GPM103</i>	15
3.3.1.1	ACI_READ	15
3.3.1.2	ACI_WRITE.....	16
3.3.1.3	ACI_PCODE	17
3.3.2	<i>IIC</i>	18
3.3.2.1	ACI_READ	18
3.3.2.2	ACI_WRITE.....	19
3.3.3	<i>SLE4432 / SLE4442</i>	20
3.3.3.1	ACI_READ	20
3.3.3.2	ACI_WRITE.....	21
3.3.3.3	ACI_WRITEP	21
3.3.3.4	ACI_PCODE (only SLE4442)	22
3.3.3.5	ACI_CCODE (only SLE4442):.....	22
3.3.4	<i>SLE4418 / SLE4428</i>	23
3.3.4.1	ACI_READ	23
3.3.4.2	ACI_WRITE.....	24
3.3.4.3	ACI_WRITEP	24
3.3.4.4	ACI_PCODE (only SLE4428)	25

1. Introduction

PC/SC 1.0 is an architecture that defines an interface between smart card readers and a resource manager. The specification includes a description of the general-purpose interface as well as card authorization, PIN verification, file access and cryptographic services. However, PC/SC defines only the interface for access of MCU (asynchronous) smart cards. In order to use a memory (synchronized) smart card, vendor-specific implementation that makes use of the extension to the PC/SC specification is required.

There are two implementations of PC/SC namely, the WinSCard API provided by Microsoft under Windows and the "PC/SC Lite" provided by MUSCLE project under Linux. The support routines described in this reference manual is compatible with the Microsoft implementation only.

2. Supported Memory Cards and System Requirements

2.1 Supported Memory Cards

Access to memory cards requires the support from reader. Currently the following cards are supported by ACR30U.

- ✓ SLE4442
- ✓ SLE4432
- ✓ SLE4428
- ✓ SLE4418
- ✓ SLE4406
- ✓ GPM103
- ✓ I2C

2.2 System Requirements

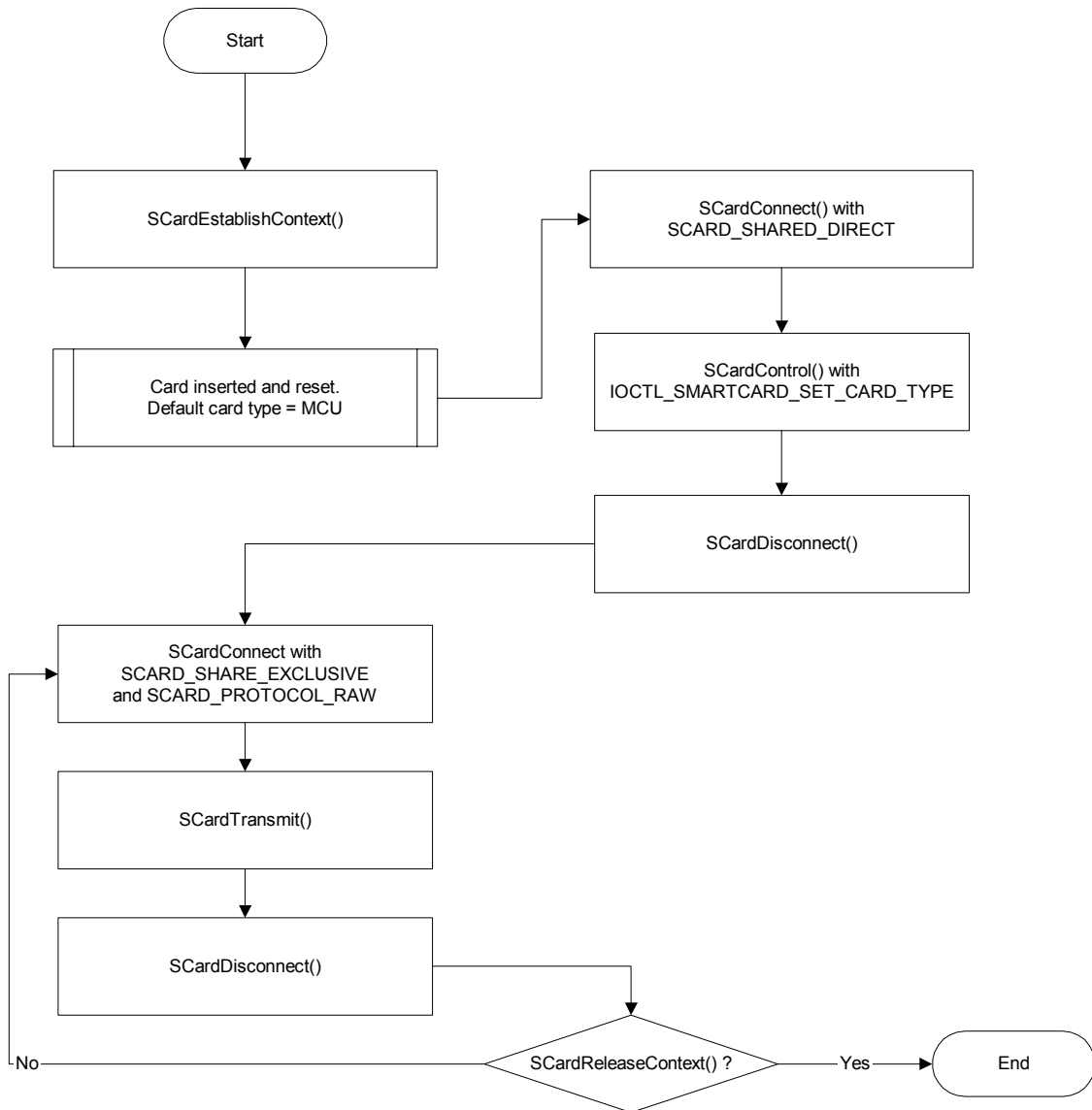
To use memory card functions, user requires ACR30U driver with version of 2.10.1+.

3. Memory Card Access

3.1 Logical Flow of Memory Card Functions

The goal of the memory card functions is to provide access to memory card using the standard PC/SC routines as much as possible. Before an application can make access to a memory card, it must notify the reader of the type of memory card that will be used. Then the application can perform data exchange with memory card.

The logical flow of the memory card functions are shown as follow:



3.2 PC/SC Functions for Memory Card Access

3.2.1 SCardEstablishContext

Descriptions:

This function establishes the communication context with the ACR30U reader. It must be the first function to call.

Synopsis:

```
LONG SCardEstablishContext(  
    DWORD dwScope,  
    LPCVOID pvReserved1,  
    LPCVOID pvReserved2,  
    LPSCARDCONTEXT phContext  
);
```

Parameters:*dwScope*

Scope of communication context. Can be either SCARD_SCOPE_USER or SCARD_SCOPE_SYSTEM

pvReserved1

System reserved and must be NULL

pvReserved2

System reserved and must be NULL

phContext

Handle to the established communication context.

Example Code:

```
SCARDCONTEXT hContext;  
LONG rv;  
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);  
if (rv != SCARD_S_SUCCESS)  
{  
    printf("error establishing context!\n");  
    exit(1);  
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.2.2 SCardReleaseContext

Descriptions:

This function releases the communication context and any resource allocated previously with **SCardEstablishContext**.

Synopsis:

```
LONG SCardReleaseContext(  
    SCARDCONTEXT phContext  
);
```

Parameters:

hContext

Handle that identifies the communication context returned from a previous call to **SCardEstablishContext**.

Example Code:

```
SCARDCONTEXT hContext;  
LONG rv;  
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);  
rv = SCardReleaseContext(hContext);  
if (rv != SCARD_S_SUCCESS)  
{  
    printf("error releasing context!\n");  
    exit(1);  
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.2.3 SCardListReaders

Descriptions:

This function returns application with a list of readers known to the system. Application must parse the buffer storing names of readers to retrieve the name of individual reader.

Synopsis:

```
LONG SCardListReaders (
    SCARDCONTEXT hContext,
    LPCTSTR mszGroups,
    LPTSTR mszReaders,
    LPDWORD pcchReaders
);
```

Parameters:

hContext

Handle that identifies the communication context for the query (returned from a previous call to **SCardEstablishContext**).

mszGroups

Name of reader groups defined in the system. Use NULL here.

mszReaders

Multi-string buffer with list of readers. Reader names are separated by NULL characters.

pcchReaders

Length of the multi-string buffer in characters including NULL.

Example Code:

```
SCARDCONTEXT hContext;
LPSTR mszReaders;
DWORD pcchReaders;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &pcchReaders);
mszReaders = (LPSTR) malloc(sizeof(CHAR) * pcchReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &pcchReaders);
// Content of multi-string buffer, mszReaders, will look like:
// "ACS ACR30U Reader1\0ACS ACR38U Reader1\0\0"
if (rv != SCARD_S_SUCCESS)
{
    printf("error listing reader names!\n");
    // release context and quit
    ...
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.2.4 SCardConnect

Descriptions:

This function establishes a connection between the calling application and the smart card inserted in the specified reader.

Synopsis:

```
LONG SCardConnect (
    SCARDCONTEXT hContext,
    LPCTSTR szReader,
    DWORD dwShareMode,
    DWORD dwPreferredProtocol,
    LPSCARDHANDLE phCard,
    LPDWORD pdwActiveProtocol
);
```

Parameters:

hContext

Handle that identifies the communication context returned from a previous call to **SCardEstablishContext**.

szReader

Name of reader containing the target card.

dwShareMode

Set to SCARD_SHARE_EXCLUSIVE for memory card access.
Set to SCARD_SHARE_DIRECT for card type selection.

dwPreferredProtocol

Set to SCARD_PROTOCOL_RAW with SCARD_SHARE_EXCLUSIVE
Set to 0 with SCARD_SHARE_DIRECT

phCard

Handle that identifies the connection to the smart card in the specified reader.

pdwActiveProtocol

The smart card protocol to be used in the subsequent communication (the value will be SCARD_PROTOCOL_RAW).

Example Code:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
LPSTR mszReaders;
LPSTR szReader;
DWORD pcchReaders;
DWORD dwActiveProtocol;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &pcchReaders);
mszReaders = (LPSTR) malloc(sizeof(CHAR) * pcchReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &pcchReaders);
szReader = mszReaders; // Use the 1st reader in the list
rv = SCardConnect(
    hContext,
    szReader,
    SCARD_SHARE_EXCLUSIVE,
    SCARD_PROTOCOL_RAW,
    &hCard,
    &dwActiveProtocol);
```

```
if (rv != SCARD_S_SUCCESS)
{
    printf("error connecting to smart card!\n");
    // release context and quit
    ...
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.2.5 SCardDisconnect

Descriptions:

This function releases the connection established between the calling application and the smart card inserted in the specified reader in the previous call to **SCardConnect**.

Synopsis:

```
LONG SCardDisconnect(
    SCARDHANDLE hCard,
    DWORD dwDisposition
);
```

Parameters:

hCard

Handle to connection from **SCardConnect**, which is going to be released.

dwDisposition

Can be any one of SCARD_UNPOWER_CARD, SCARD_LEAVE_CARD, SCARD_RESET_CARD.

Example Code:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
LPSTR mszReaders;
LPSTR szReader;
DWORD pcchReaders;
DWORD dwActiveProtocol;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &pcchReaders);
mszReaders = (LPSTR) malloc(sizeof(CHAR)* pcchReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &pcchReaders);
szReader = mszReaders; // Use the 1st reader in the list
rv = SCardConnect(
    hContext,
    szReader,
    SCARD_SHARE_EXCLUSIVE,
    SCARD_PROTOCOL_RAW,
    &hCard,
    &dwActiveProtocol);

rv = SCardDisconnect(
    hCard,
    SCARD_UNPOWER_CARD
);

if (rv != SCARD_S_SUCCESS)
{
    printf("error connecting to smart card!\n");
    // release context and quit
    ...
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.2.6 SCardTransmit

Descriptions:

This function exchanges (sends and receives) a data unit (APDU) with the target smart card with which a connection is established using **SCardConnect**.

Synopsis:

```
LONG SCardTransmit(
    SCARDHANDLE hCard,
    LPCSCARD_IO_REQUEST pioSendPci,
    LPCBYTE pbSendBuffer,
    DWORD cbSendLength,
    LPSCARD_IO_REQUEST pioRecvPci,
    LPBYTE pbRecvBuffer,
    LPDWORD pcbRecvLength
);
```

Parameters:

hCard

Handle to the connection of target smart card returned from **SCardConnect**.

pioSendPci

Set to SCARD_PCI_RAW.

pbSendBuffer

Buffer containing data to be sent to the smart card.

cbSendLength

Length (in bytes) of *pbSendBuffer*.

pioRecvPci

Set to NULL.

pbRecvBuffer

Buffer containing data returned from smart card.

pcbRecvLength

Indicate the length (in bytes) of *pbRecvBuffer* and receives the actually number of bytes received.

Example Code:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
LPSTR mszReaders;
LPSTR szReader;
BYTE pbSendBuffer[512];
BYTE pbRecvBuffer[512];
DWORD cbSendLength=512;
DWORD cbRecvLength=512;
DWORD pcchReaders;
DWORD dwActiveProtocol;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &pcchReaders);
mszReaders = (LPSTR) malloc(sizeof(CHAR)* pcchReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &pcchReaders);
szReader = mszReaders; // Use the 1st reader in the list
rv = SCardConnect(
    hContext,
    szReader,
    SCARD_SHARE_EXCLUSIVE,
```

```
    SCARD_PROTOCOL_RAW,  
    &hCard,  
    &dwActiveProtocol);  
  
// Read 32 bytes from I2C card starting at address 00:10  
pbSendBuffer[0] = 0x00;    // CLA  
pbSendBuffer[1] = ACI_READ; // INS  
pbSendBuffer[2] = 0x00;    // P1  
pbSendBuffer[3] = 0x10;    // P2  
pbSendBuffer[4] = 0x20;    // Len  
cbSendLength = 5;  
rv = SCardTransmit(  
    hCard,  
    SCARD_PCI_RAW,  
    pbSendBuffer,  
    cbSendLength,  
    NULL,  
    pbRecvBuffer,  
    &cbRecvBuffer);  
  
if (rv != SCARD_S_SUCCESS)  
{  
    printf("error transmitting data!\n");  
    // release context and quit  
    ...  
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.2.7 SCardControl

Descriptions:

This function allows application to have direct control of the reader. Application can call it any time after a successful call to **SCardConnect** and before a successful call to **SCardDisconnect**.

Synopsis:

```
LONG SCardControl(
    SCARDHANDLE hCard,
    DWORD dwControlCode,
    LPCVOID lpInBuffer,
    DWORD nInBufferSize,
    LPCVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpBytesReturned
);
```

Parameters:

hCard

Handle to the connection of target smart card returned from **SCardConnect**.

dwControlCode

Set to IOCTL_SMARTCARD_SET_CARD_TYPE.

lpInBuffer

Buffer containing data to be sent to reader to perform the specific operation.

nInBufferSize

Length (in bytes) of *lpInBuffer*.

lpOutBuffer

Buffer containing the operation's output data.

nOutBufferSize

Length (in bytes) of *lpOutBuffer*.

lpBytesReturned

Receives the length (in bytes) of data stored into the buffer, *lpOutBuffer*.

Example Code:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
LPSTR mszReaders;
LPSTR szReader;
BYTE lpOutBuffer[10];
DWORD nOutBufferSize=10;
DWORD lpBytesReturned;
DWORD pcchReaders;
DWORD dwActiveProtocol;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &pcchReaders);
mszReaders = (LPSTR) malloc(sizeof(CHAR) * pcchReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &pcchReaders);
szReader = mszReaders; // Use the 1st reader in the list
rv = SCardConnect(
    hContext,
    szReader,
    SCARD_SHARE_DIRECT,
    0,
```

```
        &hCard,
        &dwActiveProtocol);

// Set the card type of memory card
rv = SCardControl(
    hCard,
    IOCTL_SMARTCARD_SET_CARD_TYPE,
    NULL,
    0,
    lpOutBuffer,
    nOutBufferSize,
    &lpBytesReturned);

if (rv != SCARD_S_SUCCESS)
{
    printf("error transmitting data!\n");
    // release context and quit
    ...
}
```

Return Values:

Success	SCARD_S_SUCCESS
Failure	An error code (see MSDN Smart Card Error Codes for detail)

3.3 ACR30U Command Set for Memory Card Access

The set of memory card commands is used to define the operations to be performed on the target memory card. The table below shows the full set of memory card commands and the corresponding type of memory cards that support these commands.

	SLE4442	SLE4432	SLE4428	SLE4418	SLE4406	GPM103	I2C
ACI_READ	X	X	X	X	X	X	X
ACI_WRITE	X	X	X	X	X	X	X
ACI_WRITEP	X	X	X	X			
ACI_PCODE	X		X		X	X	
ACI_CCODE	X						

3.3.1 SLE4406 / GPM103

`pbSendBuffer[]` is the buffer parameter of **SCardTransmit** that contains data to be sent to smart card. This is also the command buffer.

`pbRecvBuffer[]` is the buffer parameter of **SCardTransmit** that contains data returned from smart card. This is also the response buffer.

3.3.1.1 ACI_READ

Descriptions:

This command reads the specified number of bytes from the specified address of the inserted card. The bytes are read from the card with LSB first, i.e., the bit at card address 0 is regarded as the LSB of byte 0.

Command Buffer Format (`pbSendBuffer`):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_READ;
pbSendBuffer[2] = 0x00;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
```

Parameters:

P2 Starting address of the read operation

Len The number of bytes to read

Response Data Format (`pbRecvBuffer`):

BYTE 1	BYTE 2	BYTE 3	BYTE N

BYTE x Data bytes read from the card memory.

3.3.1.2 *ACI_WRITE*

Descriptions:

This command writes one byte to the specified address of the inserted card. The byte is written to the card with LSB first, i.e., the bit at card address 0 is regarded as the LSB of byte 0.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;  
pbSendBuffer[1] = ACI_WRITE;  
pbSendBuffer[2] = P1;  
pbSendBuffer[3] = P2;  
pbSendBuffer[4] = 1;  
pbSendBuffer[5] = Data;
```

Parameters:

P1

Mode of write operation.
Set to either ACM_WRITE or ACM_WRITEC.

P2

Starting address of memory location to write.

Data

Byte value to be written to the memory location.

Remarks:

Two different WRITE modes are available for this card type, which are distinguished by a flag in the command data field:

a) Write (ACM_WRITE)

The byte value specified in the command is written to the specified address. This command can be used for writing personalization data and counter values to the card.

b) Write with carry (ACM_WRITEC)

The byte value specified in the command is written to the specified address and the command is sent to the card to erase the next lower counter stage. This write mode can therefore only be used for updating the counter value in the card.

With either write mode, the byte at the specified card address is not erased prior to the write operation and, hence, memory bits can only be programmed from '1' to '0'.

Response Data Format (pbRecvBuffer):

No Response Data

3.3.1.3 *ACI_PCODE*

Descriptions:

This command submits the transport code to the card in order to enable the card personalization mode. The following actions are executed by the ACR30:

- Search a '1' bit in the presentation counter and write the bit to '0'.
- Present the specified code to the card.

The ACR30 does not try to erase the presentation counter after the code submission! This must be done by the application software through a separate 'Write with carry' command.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_PCODE;
pbSendBuffer[2] = 0x00;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
pbSendBuffer[5] = Data00;
pbSendBuffer[6] = Data01;
      :
      :
      :
```

Parameters:

P2
The byte address of the presentation counter in the card.

Len
Byte value to be written to the memory location.

Data x
The transport code.

Response Data Format (pbRecvBuffer):

No Response Data

3.3.2 IIC

`pbSendBuffer[]` is the buffer parameter of **SCardTransmit** that contains data to be sent to smart card. This is also the command buffer.

`pbRecvBuffer[]` is the buffer parameter of **SCardTransmit** that contains data returned from smart card. This is also the response buffer.

3.3.2.1 ACI_READ

This command reads the specified number of bytes from the specified address of the inserted card.

Command Buffer Format (`pbSendBuffer`):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_READ;
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
```

Parameters:

P1

The byte address of the first byte to be read from card (high order byte).

P2

The byte address of the first byte to be read from card (low order byte).

Len

The number of bytes to be read from the card ($0 < N \leq \text{MAX_R}$).

Response Data Format (`pbRecvBuffer`):

BYTE 1	BYTE 2	BYTE 3	BYTE N

BYTE x Data bytes read from the card memory.

3.3.2.2 *ACI_WRITE*

This command writes the specified data bytes to the specified address of the inserted card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;  
pbSendBuffer[1] = ACI_WRITE  
pbSendBuffer[2] = P1;  
pbSendBuffer[3] = P2;  
pbSendBuffer[4] = Len;  
pbSendBuffer[5] = Data00;  
pbSendBuffer[6] = Data01;  
:  
:  
:  
:
```

Parameters:

P1

The byte address of the first byte to be written to card (high order byte).

P2

The byte address of the first byte to be written to card (low order byte).

Len

The number of bytes to be written to the card.

Data x

The byte value to be written to card at starting address ADDR (P1:P2). BYTE 1 is written to address ADDR; BYTE N is written to address ADDR+Len-1.

Response Data Format (pbRecvBuffer):

No Response Data

3.3.3 SLE4432 / SLE4442

pbSendBuffer[] is the buffer parameter of **SCardTransmit** that contains data to be sent to smart card. This is also the command buffer.

pbRecvBuffer[] is the buffer parameter of **SCardTransmit** that contains data returned from smart card. This is also the response buffer.

3.3.3.1 ACI_READ

This command reads the specified number of bytes from the specified address of the inserted card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_READ
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
```

Parameters:

- P1 The byte address of the first byte to be written to card (high order byte).
- P2 The byte address of the first byte to be written to card (low order byte).
- Len The number of bytes to be read from the card (0<N≤MAX_R).

Response Data Format (pbRecvBuffer):

BYTE 1	BYTE 2	BYTE 3	...	BYTE N	PROT 1	...	PROT L

BYTE x Data bytes read from card memory
 PROT y Bytes containing the protection bits of the data bytes read (0 ... 4 bytes)

The protection bits are only returned in the response data if the start address ADDR specified in the command is < 20H, i.e., it is lying within the first 32 bytes of card memory which can be write protected.

Accordingly, the number of PROT bytes returned depends on how many of the data bytes read lie within the protectable area. If all data bytes read are outside the protectable area, only the data bytes read from the card are returned in the response, no PROT bytes are returned.

The arrangement of the protection bits in the PROT bytes is as follows:

PROT 1								PROT 2															
P8	P7	P6	P5	P4	P3	P2	P1	P16	P15	P14	P13	P12	P11	P10	P9	P1	P17
									5	4	3	2											8	

Px is the protection bit of BYTE x in the response data
 '0' : byte is write protected
 '1' : byte can be written

Example:
 XX

3.3.3.2 *ACI_WRITE*

This command writes the specified data bytes to the specified address of the inserted card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_WRITE;
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
pbSendBuffer[5] = Data00;
pbSendBuffer[6] = Data01;
.
.
.
```

Parameters:

P1

The byte address of the first byte to be written to card (high order byte).

P2

The byte address of the first byte to be written to card (low order byte).

Len

The number of bytes to be written to the card.

Data x

The byte value to be written to card at starting address ADDR (P1:P2). BYTE 1 is written to address ADDR; BYTE N is written to address ADDR+Len-1.

Response Data Format (pbRecvBuffer):

No response data

Example:

XX

3.3.3.3 *ACI_WRITEP*

This command writes the protection bits for the specified addresses in the card.

Each of the bytes specified in the command is internally in the card compared with the byte stored at the specified address and if the data match, the corresponding protection bit is irreversibly programmed to '0'.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_WRITEP;
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
pbSendBuffer[5] = Data00;
pbSendBuffer[6] = Data01;
.
.
.
```

Parameters:

P1

The byte address of the first byte to be written to card (high order byte).

P2

The byte address of the first byte to be written to card (low order byte).

Len

The number of bytes to be written to the card.

Data x

The byte value to be written to card at starting address ADDR (P1:P2). BYTE 1 is written to address ADDR; BYTE N is written to address ADDR+Len-1.

Response Data Format (pbRecvBuffer):

No response data

Example:

XX

3.3.3.4 ACI_PCODE (only SLE4442)

This command submits secret code to card to enable the write operation with the SLE4442 card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_PCODE;
pbSendBuffer[2] = 0x00;
pbSendBuffer[3] = 0x00;
pbSendBuffer[4] = 0x03;
pbSendBuffer[5] = Data00;
pbSendBuffer[6] = Data01;
pbSendBuffer[6] = Data02;
```

Parameters:

Data x

The secret code.

Response Data Format (pbRecvBuffer):

ERRCNT	CODE		

ERRCNT

The value of presentation error counter after the code presentation.

CODE

The three bytes secret code read from card. If the correct code has been presented to the card, the value of ERRCNT is 07H and the value of CODE is identical to the code data specified in the command.

Example:

XX

3.3.3.5 ACI_CCODE (only SLE4442):

This command writes the specified data as new secret code in the card.

The current secret code must have been presented to the card with the *PRESENT_CODE* command prior to the execution of this command!

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_CCODE;
pbSendBuffer[2] = 0x00;
pbSendBuffer[3] = 0x00;
pbSendBuffer[4] = 0x03;
pbSendBuffer[5] = Data00;
pbSendBuffer[6] = Data01;
```

```
pbSendBuffer[6] = Data02;
```

Parameters:

Data x
The secret code.

Response Data Format (pbRecvBuffer):

No response data

3.3.4 SLE4418 / SLE4428

pbSendBuffer[] is the buffer parameter of **SCardTransmit** that contains data to be sent to smart card. This is also the command buffer.

pbRecvBuffer[] is the buffer parameter of **SCardTransmit** that contains data returned from smart card. This is also the response buffer.

3.3.4.1 ACI_READ

This command reads the specified number of bytes from the specified address of the inserted card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_READ;
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
```

Parameters:

P1
The byte address of the first byte to be read from card (high order byte).

P2
The byte address of the first byte to be read from card (low order byte).

Len
The number of bytes to be read from the card (Len < 224 bytes)

Response Data Format (pbRecvBuffer):

BYTE 1	BYTE 2	BYTE 3	...	BYTE N	PROT 1	...	PROT L

BYTE x
Data bytes read from the card memory

PROT y
Bytes containing the protection bits of the data bytes read (1...4 bytes)

The number L of protection bytes returned in the response is determined by the number N of data bytes read from the card as follows:

$$L = 1 + \text{INT}(N/8) \quad \text{if } N \text{ is not multiplies of } 8$$

$$L = \text{INT}(N/8) \quad \text{if } N \text{ is multiplies of } 8$$

The arrangement of the protection bits in the PROT bytes is as follows:

PROT 1								PROT 2																
P8	P7	P6	P5	P4	P3	P2	P1	P1	P1	P1	P1	P1	P1	P1	P1	P9	P1	P1
								6	5	4	3	2	1	0									8	7	

Px

The protection bit of BYTE x in the response data

'0' : byte is write protected

'1' : byte can be written

Example:

XX

3.3.4.2 ACI_WRITE

This command writes the specified data bytes to the specified address of the inserted card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_WRITE;
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
pbSendBuffer[4] = Len;
pbSendBuffer[6] = Data00;
pbSendBuffer[7] = Data01;
.
.
.
```

Parameters:

P1

The byte address of the first byte to be written to card (high order byte).

P2

The byte address of the first byte to be written to card (low order byte).

Len

The number of bytes to be written to the card

Data x

The byte value to be written to card at starting address P1P2.

Response Data Format (pbRecvBuffer):

No response data

Example:

XX

3.3.4.3 ACI_WRITEP

This command writes the protection bits for the specified addresses in the card

Each of the bytes specified in the command is internally in the card compared with the byte stored at the specified address and if the data match, the corresponding protection bit is irreversibly programmed to '0'.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_WRITEP;
pbSendBuffer[2] = P1;
pbSendBuffer[3] = P2;
```

```
pbSendBuffer[4] = Len;
pbSendBuffer[6] = Data00;
pbSendBuffer[7] = Data01;
.
.
.
```

- P1 The byte address of the first byte to be written to card (high order byte).
- P2 The byte address of the first byte to be written to card (low order byte).
- Len The number of bytes to be written to the card
- Data x The byte value to be written to card at starting address P1P2.

Response Data Format (pbRecvBuffer):
 No response data

Example:
 XX

3.3.4.4 ACI_PCODE (only SLE4428)

This command submits the secret code to the card to enable the write operation with the SLE4428 card.

Command Buffer Format (pbSendBuffer):

```
pbSendBuffer[0] = 0x00;
pbSendBuffer[1] = ACI_PCODE;
pbSendBuffer[2] = 0x00;
pbSendBuffer[3] = 0x00;
pbSendBuffer[4] = 0x02;
pbSendBuffer[6] = Data00;
pbSendBuffer[7] = Data01;
```

Parameters:

- Data x The secret code.

Response Data Format (pbRecvBuffer):

ERRCNT	CODE

ERRCNT

The value of the presentation error counter after the code presentation.

CODE

The two bytes secret code read from the card. If the correct code has been presented to the card, the value of ERRCNT is FF_H and the value of CODE is identical to the code data specified in the command.

Example:
 XX