PMD841P-CA
04/02/2009

CSB6 FAMILY - PC/SC

Vendor specific attributes and commands

Headquarters, Europa

SpringCard
13 Voie la Cardon
Parc Gutenberg
91120 Palaiseau
FRANCE

Phone : +33 (0) 1 64 53 20 10
Fax : +33 (0) 1 64 53 20 18

Americas

SpringCard
694 Fifth Avenue
Suite 235
San Diego, CA 92101
USA

Phone : +1 (619) 544 1450
Fax : +1 (619) 573 6867

www.springcard.com

DOCUMENT INFORMATION

Category : Developer's manual
Group : CSB6
Reference : PMD841P
Version : AC
Status : draft

pmd841p-ac.doc
saved 05/09/08 - printed 04/02/09

Keywords :
CSB6, PC/SC, SCardControl, SCardGetAttrib, SCardSetAttrib

Abstract :
SpringCard has incorporated a few vendor attributes and some vendor commands in its PC/SC readers. This document lists both and explain how to use them.

REVISION HISTORY

Ver.	Date	Author	Valid. by Tech.	Valid. by Qual.	Approv. by	Remarks :
AA	21/04/08	JDA				Early draft
AB	30/05/08	JDA				Corrected to reflect some changes in the firmware itself
AC	05/09/08	JDA				New SpringCard template
BA	20/10/08	JDA				Written chapter 3, added paragraph 2.4 Corrected P1 possible values for load key command Add ASK CTSB, ST SR176 read compliance Add ISO/IEC 15693 and ICODE 1 compliance (For 15693 compliant reader) Available in firmware version 1.50 and up
CA	22/01/09	LTC				

TABLE OF CONTENT

1. INTRODUCTION

1.1. ABSTRACT

PC/SC is the de-facto standard to interface Personal Computers with Smart Cards (and smartcard readers of course). **SpringCard CSB6 Family** complies with this standard. This makes those products usable on most operating systems, using an high-level and standardized API.

Most of the time, developers will only use the SCardTransmit function to communicate with the card, seeing no difference whatever the card technology (ISO 7816-3 contact smartcard or ISO 14443-4 contactless smartcard), and whatever the reader (SpringCard CSB6, or other).

Anyway, in some specific situations, PC/SC standard functions are not enough to cover the whole functional field. This happens typically :

- When working with "false" smartcards, i.e. with memory cards or even micro-processor based cards not following the ISO 7816-4 standard (APDU formalism),
- When needing to perform actions onto the reader itself, and not onto the card (driving LEDs or buzzer, getting reader's serial number, and so on).

This document is the reference manual, for both usages.

1.2. SUPPORTED PRODUCTS

At the date of writing, this document refers to products in the CSB6 Family running a firmware version ≥ 1.47 :

- CSB6,
- Prox'N'Roll,
- CrazyWriter,
- EasyFinger.

Please review the datasheet of each product for accurate specification and a detailed list of features.

1.3. AUDIENCE

This manual is designed for use by application developers. He assumes that the reader has expert knowledge of computer development.

1.4. SUPPORT AND UPDATES

Interesting related materials (product datasheets, application notes, sample software, HOWTOs and FAQs…) are available at SpringCard's web site :

www.springcard.com

Updated versions of this document and others will be posted on this web site as soon as they are made available.

For technical support enquiries, please refer to SpringCard support page, on the web at address www.springcard.com/support .

1.5. USEFUL LINKS

- Microsoft's PC/SC reference documentation is included in most Visual Studio help system, and available online at http://msdn.microsoft.com . Enter "winscard" or "SCardTransmit" keywords in the search box.
- MUSCLE PCSC-Lite project : http://www.musclecard.com (direct link to PC/SC stack : http://pcsclite.alioth.debian.org)
- PC/SC workgroup : http://www.pcscworkgroup.com

2. EMBEDDED APDU INTERPRETER

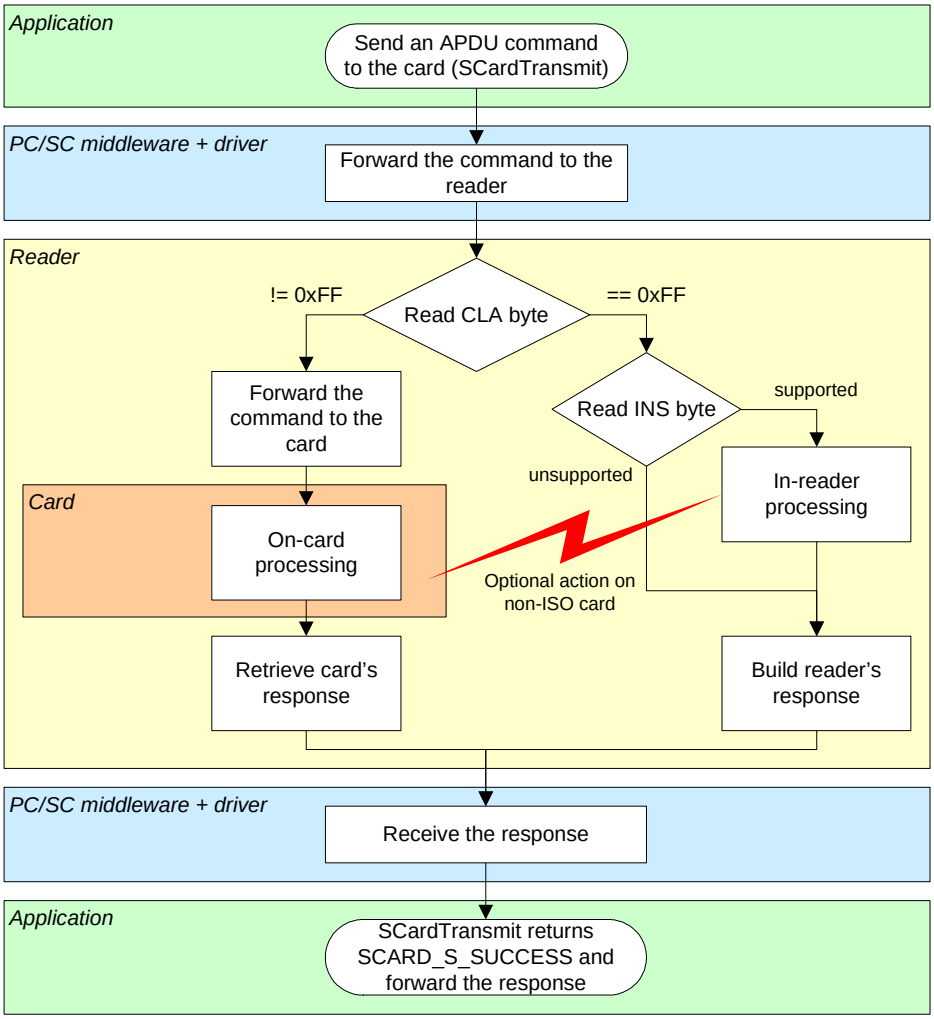
2.1. BASIS

In PC/SC architecture, the **SCardTransmit** function implements the dialog between an application and a card, through a "passive" reader. The reader only transmit frames in both directions, without any specific processing.

This simple scheme is not suitable for all kind of cards :

- SCardTransmit requires that the frames follow the ISO 7816-4 APDU rules (CLA, INS, P1, P2, and so on), and not every smartcard use this formalism
- SCardTransmit is designed to exchange commands with a smartcard, but commonly used cards (especially in the contactless world) are memory cards, and not smartcards. This means that specific card functions (read, write, …) must be implemented by the reader itself.

For both reasons, the CSB6 family features an embedded **APDU interpreter**, which overcomes some limitations of the PC/SC architecture when working with non-standard smartcards or with memory cards.



2.1.1. CLA byte of the embedded APDU interpreter

Default class is 0xFF. This means that every APDU starting with CLA=0xFF will be interpreted by the reader, and not forwarded by the card.

a. Changing the CLA byte of the embedded APDU interpreter

The CLA byte of the embedded APDU interpreter is stored at ident=0xB2 in reader's non volatile memory ("FEED").

Use the escape sequence depicted in 3.3.4.b to write the new value, and restart your CSB6.

Note : in the following paragraphs, documentation of the APDUs is written with CLA=0xFF. Change this to match your own CLA if necessary.

b. Disabling the embedded APDU interpreter

Define CLA byte = 0x00 (value 0x00 for ident 0xB2 in reader's FEED) to disable the embedded APDU interpreter.

2.1.2. Status words returned by the embedded APDU interpreter

SW1	SW2	Meaning
0x90	0x00	Success
0x67	0x00	Wrong length (Lc incoherent with Data In)
0x68	0x00	CLA byte is not correct
0x6A	0x81	Function not supported (INS byte is not correct), or not available for the selected card
0x6B	0x00	Wrong parameter P1-P2

Some functions provided by the embedded APDU interpreter may return specific status words. This behaviour is documented within the paragraph dedicated to each function.

2.1.3. Summary of embedded APDU interpreter command list

Command	INS	Contactless	Contact
LOAD KEY	0x82	✓	
GENERAL AUTHENTICATE	0x86	✓	
READ BINARY	0xB0	✓	
GET DATA	0xCA	✓	✓
UPDATE BINARY	0xD6	✓	
READER CONTROL	0xF0	✓	✓
RC CONTROL	0xF1	✓	
GEMCORE CONTROL	0xF1		✓
MIFARE CLASSIC READ	0xF3	✓	
MIFARE CLASSIC WRITE	0xF4	✓	
SET DATA	0xFA	✓	
RFU (CALYPSO)	0xFC		
TEST	0xFD	✓	✓
ENCAPSULATE	0xFE	✓	✓

2.2. COMMANDS AVAILABLE ON ALL SLOTS

Those commands allow to interact with the reader within a card connection (using SCardTransmit). This is of course only possible when a card has been activated (SCardConnect).

If you need to interact with the reader even when there's no card inserted, please refer to chapter 3.

2.2.1. READER CONTROL instruction

The **READER CONTROL** instruction allows to drive the global behavior of the CSB6 reader (LEDs, buzzer, etc depending on product physical characteristics).

For advanced operation, use SCardControl instead (see chapter 3).

If you CSB6 reader is a multi-slot device (contactless, contact, SAM…), the READER CONTROL instruction is send to one slot (a logical reader) but may have a global impact to the whole physical reader.

In other words, sending a READER CONTROL instruction in one card connection may have an impact on another card.

It is highly recommended to use a synchronisation object (mutex, critical section, …) to prevent any concurrent access to the same physical reader while a READER CONTROL instruction is pending.

READER CONTROL command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xF0	0x00	0x00	See below	See below	See below

a. Driving reader's LEDs

For a reader with only red and green LEDs, send the APDU :

```
FF F0 00 00 03 1E <red> <green>
```

For a reader with red, green and yellow / blue LEDs, send the APDU :

```
FF F0 00 00 04 1E <red> <green> <yellow/blue>
```

Choose values for red, green and yellow/blue in this table :

0x00	LED is switched OFF
0x01	LED is switched ON
0x02	LED blinks slowly
0x03	LED is driven automatically by reader's firmware *(default behaviour)*
0x04	LED blinks quickly
0x05	LED performs the "heart-beat" sequence

b. Driving reader's buzzer

Some hardware feature a single tone beeper. To start the buzzer, send the APDU :

```
FF F0 00 00 03 1C <duration MSB> <duration LSB>
```

Where duration specifies the length of the tone, in milliseconds (max is 60000ms).

Set duration to 0 if you need to stop the buzzer before the duration started in a previous call.

c. Others

The data block in the READER CONTROL command is forwarded "as is" to the reader control interpreter, as documented in chapter 3.

Therefore, every command documented in 3.3 and starting with code 0x58 may be transmitted in the SCardTransmit link using the READER CONTROL command, exactly as if it were transmitted in a SCardControl link.

Once again, do not use this feature unless you know exactly what you are doing.

2.2.2. ENCAPSULATE instruction

The **ENCAPSULATE** instruction has been designed to help the application working with cards that don't comply with ISO 7816-4[1].

ENCAPSULATE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xFE	See below	See below	XX	XX … XX	XX

Data In is the frame to be sent to the card.

ENCAPSULATE command parameter P1 for the contactless slot

P1	
0x00	Send the frame in the T=CL stream, using the ISO 14443-4 protocol. Data In shall not include PCB nor CRC fields
0x01	Send the frame "as is" using the ISO 14443-3 A protocol. Data In shall not include the CRC field (CRC added by the reader)
0x02	Send the frame "as is" using the ISO 14443-3 B protocol. Data In shall not include the CRC field (CRC added by the reader)
0x09	Send the frame "as is" using the ISO 14443-3 A protocol. Data In shall include the CRC field (CRC computed by the application)
0x0A	Send the frame "as is" using the ISO 14443-3 B protocol. Data In shall include the CRC field (CRC computed by the application)

ENCAPSULATE command parameter P1 for the contact slots

P2	
0x00	Send the frame in the T=0 or T=1 stream Other values are RFU

[1] ISO 7816-4 –and PC/SC as an extension- assumes that every command sent to the card use the APDU format, and therefore the PC/SC layer will prevent the application from sending a proprietary frame that doesn't follow this rule.

ENCAPSULATE command parameter P2 for the contactless slot

P2 encodes the frame timeout.

P2	
0x-0	If P1 = 0x00, use default T=CL timeout defined by the card If P1 ≠ 0x00, this value shall not be used
0x-1	Timeout = 106 ETU ≈ 1ms
0x-2	Timeout = 212 ETU ≈ 2ms
0x-3	Timeout = 424 ETU ≈ 4ms
0x-4	Timeout = 848 ETU ≈ 8ms
0x-5	Timeout = 1696 ETU ≈ 16ms
0x-6	Timeout = 3392 ETU ≈ 32ms
0x-7	Timeout = 6784 ETU ≈ 65ms
0x-8	Timeout = 13568 ETU ≈ 0,125s
0x-9	Timeout = 27136 ETU ≈ 0,250s
0x-A	Timeout = 54272 ETU ≈ 0,500s
0x-B	Timeout = 108544 ETU ≈ 1s
0x-C	Timeout = 217088 ETU ≈ 2s
0x-D	Timeout = 434176 ETU ≈ 4s
0x0-	Set status word = 0x6F XX , XX being the contactless specific error
0x8-	Set status word = 0x63 00 on any contactless specific error

ENCAPSULATE command parameter P2 for the contact slot

P2	
0x00	Other values are RFU

ENCAPSULATE response

Data Out	SW1	SW2
XX … XX	See below	

Data Out is the frame returned by the card.

- If Data In did include the CRC field (as indicated by P1), then Data Out also includes the CRC field (and CRC is not verified by the reader).
- If Data In did not include the CRC field, then CRC is verified by the reader and not provided in Data Out.

ENCAPSULATE status word

SW1	SW2	Meaning
0x90	0x00	Success
0x6F	XX	Error reported by the contactless interface (only allowed if high-order bit of P2 is 0). See chapter 6 for the list of possible values and their meaning.
0x63	0x00	Error reported by the contactless interface (when high-order bit of P2 is 1).
0x62	0x82	Le is greater than actual response from card
0x6C	XX	Le is shorter than actual response from card

2.2.3. TEST instruction

The **TEST** instruction has been designed to test the driver and/or the applications, with arbitrary length of data (in and out).

TEST command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xFD	See below	See below	XX	XX … XX	XX

TEST command parameters

Parameter P1 specifies the length of Data Out the application wants to receive from the reader :

- 0x00 : empty Data Out, only SW returned
- 0xFF : 255 bytes of data + SW
- All values between 0x00 and 0xFF are allowed

6 low-order bits of P2 specify the delay between command and response.

- 0x00 : no delay, response comes immediately
- 0x37 : 63 seconds between command and response
- All values between 0 and 63 are allowed

2 high-order bits of P2 are RFU and must be set to 0.

TEST response

Data Out	SW1	SW2
XX … XX	See below	

Content of Data Out is not specified, and may contain either "random" or fixed data, depending on the reader implementation and current status.

TEST status word

When 2 high-order bits of P2 are 0, the embedded APDU interpreter analyzes the format of the APDU, and return appropriate status word. On the other hand, if at least one of those bits is 1, status word is fixed whatever the APDU format.

SW1	SW2	Meaning
0x90	0x00	Success, APDU correctly formatted
0x67	0x00	APDU is badly formatted (total length incoherent with Lc value)
0x6A	0x82	Le is greater than data length specified in P1
0x6C	P1	Le is shorter than data length specified in P1

2.3. COMMANDS AVAILABLE ONLY ON CONTACTLESS SLOT

2.3.1. GET DATA instruction

The **GET DATA** instruction retrieves information regarding the inserted card. It can be used with any kind of contactless cards, but the returned content will vary with the type of card actually in the slot.

GET DATA command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xCA	See below	0x00	–	–	0x00

GET DATA command parameters

P1	P2	Will return in Data Out
0x00	0x00	Card's serial number - ISO 14443-A : UID (4, 7 or 11 bytes) - ISO 14443-B : PUPI (4 bytes) - ISO 15693 : UID (8 bytes) - others : see chapter 5 for details
0x01	0x00	- ISO 14443-A : historical bytes from the ATS - ISO 14443-B : INF field in ATTRIB response - others : see chapter 5 for details
0xF0	0x00	Card's complete identifier : - ISO 14443-A : ATQ (2 bytes) + SAK (1 byte) + UID - ISO 14443-B : full REQB response (11 bytes) - others : see chapter 5 for details
0xF1	0x00	Card's type, according to PC/SC part 3 supplemental document : PIX.SS (standard, 1 byte) + PIX.NN (card name, 2 bytes) See chapter 5.1 for details
0xF2	0x00	Card's short serial number - ISO 14443-A : UID truncated to 4 bytes, in "classical" order - others : same as P1,P2=0x00,0x00
0xFF	0x00	Reader's serial number (UID of the NXP RC chipset)

GET DATA response

Data Out	SW1	SW2
XX … XX	See below	

GET DATA status word

SW1	SW2	Meaning
0x90	0x00	Success
0x62	0x82	End of data reached before Le bytes (Le is greater than data length)
0x6C	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

2.3.2. LOAD KEY instruction

The **LOAD KEY** instruction loads a Mifare access key in reader's memory.

LOAD KEY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0x82	Key location	Key index	0x06	Key value (6 bytes)	-

LOAD KEY command parameter P1 (key location)

P1	
0x00	The key is to be loaded in reader's volatile memory
0x20	The key is to be loaded in reader's non-volatile memory (secured E2PROM of RC chipset)

LOAD KEY command parameter P2 (key index)

When P1 = 0x00, P2 is the identifier of the key into reader's volatile memory. This memory can store 4 "A" keys, and 4 "B" keys.

P2	Type of key	Index of key
0x00	A	0
…	*(…)*	*(…)*
0x03	A	3
0x10	B	0
…	*(…)*	*(…)*
0x13	B	3

When P1 = 0x20, P2 is the identifier of the key into RC chipset secured E2PROM. This memory can store 16 "A" keys, and 16 "B" keys.

P2	Type of key	Index of key
0x00	A	0
…	*(…)*	*(…)*
0x0F	A	15
0x10	B	0
…	*(…)*	*(…)*
0x1F	B	15

LOAD KEY response

SW1	SW2
See below	

LOAD KEY status word

SW1	SW2	Will return in Data Out
0x90	0x00	Success
0x69	0x86	Volatile memory is not available
0x69	0x87	Non-volatile memory is not available
0x69	0x88	Key number is not valid
0x69	0x89	Key length is not valid

2.3.3. GENERAL AUTHENTICATE instruction

The **GENERAL AUTHENTICATE** instruction performs Mifare authentication explicitly. Application provides the number of the key used for the Mifare authentication. The specific key must be already in the reader.

> When working with a Mifare classic card, the application must be authenticated on a sector before being able to read or write its content.
>
> One must always call GENERAL AUTHENTICATE instruction (with the right sector's key) before calling READ BINARY or UPDATE BINARY instructions (on the same sector).

GENERAL AUTHENTICATE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0x86	0x00	0x00	0x05	See below	-

GENERAL AUTHENTICATE Data In bytes

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
0x01	0x00	Block number	Key location	Key index

- Block number (byte 2) is the address on the card, where we try to be authenticated *(note : this is not the sector's number, but the block's)*.
- Key location (byte 3) is the same as P1 parameter used in the LOAD KEY command.
- Key index (byte 4) is the same as P2 parameter used in the LOAD KEY command.

GENERAL AUTHENTICATE response

SW1	SW2
See below	

GENERAL AUTHENTICATE status word

SW1	SW2	Meaning
0x90	0x00	Success
0x69	0x82	Authentication failed
0x69	0x86	Key type is not valid
0x69	0x88	Key number is not valid

2.3.4. READ BINARY instruction

The **READ BINARY** instruction retrieves data from a (supported) contactless memory card.

> When working with a Mifare classic card, the application must be authenticated on a sector before being able to read or write its content.
>
> One must always call GENERAL AUTHENTICATE instruction (with the right sector's key) before calling READ BINARY or UPDATE BINARY instructions (on the same sector).

READ BINARY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xB0	Address MSB	Address LSB	-	-	XX

READ BINARY response

Data Out	SW1	SW2
XX ... XX	See below	

READ BINARY status word

SW1	SW2	Will return in Data Out
0x90	0x00	Success
0x62	0x82	End of data reached before Le bytes (Le is greater than data length)
0x69	0x81	Command incompatible
0x69	0x82	Security status not satisfied
0x6A	0x82	Wrong address (no such block or no such offset in the card)
0x6C	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

a. Using READ BINARY with a Mifare Classic card

For Mifare Classic cards,

- P1-P2 is the address of the first block to be read (0 to 63 for a Mifare 1k, 0 to 255 for a Mifare 4k),
- Since the size of each block is 16, Le must be a multiple of 16 (48 bytes for standard sectors, 240 bytes for the largest sectors in Mifare 4k).

Note than when a sector trailer (security block) is read, the keys are always masked by the card.

The READ BINARY instruction can't cross sector boundaries ; the GENERAL AUTHENTICATE instruction must be called for each sector immediately before READ BINARY.

Tip : using the READ MIFARE CLASSIC instruction instead is easier and may shorten the transaction time.

b. Using READ BINARY with a Mifare UltraLight card

For Mifare UltraLight cards,

- P1-P2 is the address of the first page to be read (0 to 15),
- Since the size of each page is 4, Le must be multiple of 4 (64 bytes for the full card).

c. Using READ BINARY with ASK CTSB, ST SR176 cards

For ASK CTSB, ST SR176 cards,

- P1-P2 is the address of the first block to be read, please read documentation of your card to know the number of blocks,
- Since the size of each block is 2, Le must be 2, exactly.

d. Using READ BINARY with a 15693 card

You can read 15693 card only if your reader is 15693 compliant.

For ICODE SLI and TAG-IT cards,

- P1-P2 is the address of the first block to be read, please read documentation of your 15693 card to know the number of blocks,

Since the size of each block is 4, Le must be multiple of 4.

For ST LRI64 cards,

- P1-P2 is the address of the first block to be read (0 to 14),

Since the size of each block is 1, Le must be 14 bytes for the full card.

e. Using READ BINARY with a ICODE 1 card

You can read ICODE 1 card only if your reader is 15693 compliant.

For ICODE 1 cards,

- P1-P2 is the address of the first block to be read (0 to 15),

Since the size of each block is 4, Le must be multiple of 4 (64 bytes for the full card).

2.3.5. MIFARE CLASSIC READ instruction

The **MIFARE CLASSIC READ** instruction retrieves data from a Mifare Classic (e.g. standard 1k or 4k) contactless card.

The difference with READ BINARY lies in the authentication scheme :

- With the READ BINARY instruction, authentication must be performed before, using the GENERAL AUTHENTICATE instruction,
- With the MIFARE CLASSIC READ instruction, the authentication is performed automatically by the reader, trying every keys one after the other, until one succeed.

This "automatic" authentication makes MIFARE CLASSIC READ instruction an interesting helper to read Mifare data easily.

As a consequence, it may be slower than an explicit GENERAL AUTHENTICATE instruction followed by a READ BINARY instruction.

a. MIFARE CLASSIC READ using reader's keys

In this mode, the application does not specify anything. The reader tries every key he knows (both permanent keys in E2PROM and temporary keys previously loaded in volatile memory) until one succeed.

As the reader has to try all the keys, the ordering of the keys in reader's memory is also very important to reduce this side-effect (the upper the correct key is in the list, the faster the transaction is done).

The reader tries all "A" keys at first, and only after, it tries all the "B" keys.

This behaviour has been chosen because in 95% of Mifare application, the "A" key is the preferred key for reading (where the "B" key is used for writing).

To get optimal performance, it is recommended to put the correct "A" key in reader's memory, even if reading is also possible with one of the "B" keys.

MIFARE CLASSIC READ command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xF3	0x00	Block Number	-	-	XX

Refer to the READ BINARY command (§ 2.3.4) for response and status words.

b. MIFARE CLASSIC READ with specified key

In this mode, the application provides the key to the reader.

The reader tries the supplied key first with an "A" authentication, and only after, it tries is with a "B" authentication.

Optimal performance is reached when the provided key is actually the "A" key of the sector.

MIFARE CLASSIC READ command APDU, with specified key

CLA	INS	P1	P2	Lc	Data In	Le

0xFF	0xF3	0x00	Block Number	0x06	Key value (6 bytes)	XX

Refer to the READ BINARY command (§ 2.3.4) for response and status words.

2.3.6. UPDATE BINARY instruction

The **UPDATE BINARY** instruction writes data into a (supported) contactless card.

When working with a Mifare classic card, the application must be authenticated on a sector before being able to read or write its content.

One must always call GENERAL AUTHENTICATE instruction (with the right sector's key) before calling READ BINARY or UPDATE BINARY instructions (on the same sector).

UPDATE BINARY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xD6	Address MSB	Address LSB	XX	Data	-

UPDATE BINARY response

SW1	SW2
See below	

UPDATE BINARY status word

SW1	SW2	Will return in Data Out
0x90	0x00	Success
0x69	0x82	Security status not satisfied
0x6A	0x82	Wrong address (no such block or no such offset in the card)
0x6A	0x84	Wrong length (trying to write too much data at once)

a. Using UPDATE BINARY with a Mifare Classic card

For Mifare Classic cards,

- P1-P2 is the address of the first block to be written (1 to 63 for a Mifare 1k, 1 to 255 for a Mifare 4k),
- Since the size of each block is 16, Lc must be a multiple of 16 (48 bytes for standard sectors, 240 bytes for the largest sectors in Mifare 4k).

Writing sector trailers (security blocks) is possible as long as the sector's current access conditions allow it. BUT Mifare sector trailers must follow a specific formatting rule ("mash-up" of the access conditions bits) to be valid.

Writing a badly formatted security block means making the sector permanently unusable (and writing a key you don't remember later has the same impact).

Please read Mifare 1k or 4k documentation carefully, to understand how the data must be arranged in the security blocks.

The UPDATE BINARY instruction can't cross sector boundaries ; the GENERAL AUTHENTICATE instruction must be called for each sector immediately before UPDATE BINARY.

Tip : using the WRITE MIFARE CLASSIC instruction instead is easier and may shorten the transaction time.

b. Using UPDATE BINARY with a Mifare UltraLight card

For Mifare UltraLight cards,

- P1-P2 is the address of the first page to be written (2 to 15),
- Since the size of each page is 4, Lc must be 4, exactly.

Pages 2 and 3 holds lock and OTP (one-time-programming) bits.

Please read Mifare UltraLight documentation carefully, to understand the impact of writing those pages.

c. Using UPDATE BINARY with a 15693 card

You can write 15693 card only if your reader is 15693 compliant.

For ICODE SLI and TAG-IT cards,

- P1-P2 is the address of the first block to be written, please read documentation of your 15693 card to know the number of blocks,
- Since the size of each block is 4, Lc must be 4, exactly.

For ST LRI64 cards,

- P1-P2 is the address of the first block to be written (8 to 14),
- Since the size of each block is 1, Lc must be 1, exactly.

Blocks 8 to 14 are Write-Once Read-Many (WORM) memory.
Please read ST LRI64 documentation carefully, to understand the impact of writing those blocks.

2.3.7. MIFARE CLASSIC WRITE instruction

The **MIFARE CLASSIC WRITE** command writes data into a Mifare Classic (e.g. standard 1k or 4k) contactless card.

The difference with UPDATE BINARY lies in the authentication scheme :

- With the UPDATE BINARY instruction, authentication must be performed before, using the GENERAL AUTHENTICATE instruction,
- With the MIFARE CLASSIC WRITE instruction, the authentication is performed automatically by the reader, trying every keys one after the other, until one succeed.

This "automatic" authentication makes MIFARE CLASSIC WRITE instruction an interesting helper to write Mifare data easily.

As a consequence, it may be slower than an explicit GENERAL AUTHENTICATE instruction followed by a WRITE BINARY instruction.

a. MIFARE CLASSIC WRITE using reader's keys

In this mode, the application does not specify anything. The reader tries every key he knows (both permanent keys in E2PROM and temporary keys previously loaded in volatile memory) until one succeed.

As the reader has to try all the keys, the ordering of the keys in reader's memory is also very important to reduce this side-effect (the upper the correct key is in the list, the faster the transaction is done).

The reader tries all "B" keys at first, and only after, it tries all the "A" keys.

This behaviour has been chosen because in 95% of Mifare application, the "B" key is the preferred key for writing (where the "A" key is used for reading).

To get optimal performance, it is recommended to put the correct "B" key in reader's memory, even if writing is also possible with one of the "A" keys[2].

MIFARE CLASSIC WRITE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xF4	0x00	Block Number	XX	XX … XX	-

Lc must be a multiple of 16.

Refer to the WRITE BINARY command (§ 2.3.6) for response and status words.

b. MIFARE CLASSIC WRITE with specified key

In this mode, the application provides the key to the reader.

[2] Mifare Classic cards issued by NXP are delivered in "transport configuration", with no "B" key and an "A" key allowed for both reading and writing. This "transport configuration" gives poorest writing performance ; card issuer must start the card personalisation process by enabling a "B" key for writing.

The reader tries the supplied key first with a "B" authentication, and only after, it tries is with an "A" authentication.

Optimal performance is reached when the provided key is actually the "B" key of the sector.

MIFARE CLASSIC WRITE command APDU, with specified key

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xF4	0x00	Block Number	XX	See below	-

MIFARE CLASSIC WRITE command APDU Data In bytes when key is specified

Bytes 0 to Lc-7	Bytes Lc-6 to Lc-1
Data to be written (multiple of 16 bytes)	Key value (6 bytes)

Lc = 6 + 16 x (number of blocks to be written).

Refer to the WRITE BINARY command (§ 2.3.6) for response and status words.

2.4. COMMANDS AVAILABLE ONLY ON CONTACT SLOTS

2.4.1. CONFIGURE CALYPSO SAM specific instruction

This command is only available on devices having the Calypso option enabled.

The **CONFIGURE CALYPSO SAM** instruction activates internal shortcuts to speed-up Calypso transactions.

CONFIGURE CALYPSO SAM command APDU

CLA	INS	P1	P2	Lc	Data In	Le
0xFF	0xFC	See below	See below	0x00	-	-

CONFIGURE CALYPSO SAM command parameters

P1	P2	Will return in Data Out
0x04	0x00	Configure Calypso SAM for 9600 bps communication
0x04	0x01	Configure Calypso SAM for 115200 bps communication
0x08	0x00	Disable Calypso internal DigestUpdate mode
0x08	0x01	Enable Calypso internal DigestUpdate mode When this mode is enabled, every APDU exchanged on the other slots is forwarded to the SAM within 2 Calypso DigestUpdate commands.

CONFIGURE CALYPSO SAM response

SW1	SW2
See below	

CONFIGURE CALYPSO SAM status word

SW1	SW2	Meaning
0x90	0x00	Success
0x6B	0x00	Wrong value for P1
0x6F	0xE7	SAM didn't answer with 9000 (maybe this is not a Calypso SAM !)
0x6F	XX	Error code returned by the Gemcore

3. DIRECT CONTROL OF THE READER

3.1. BASIS

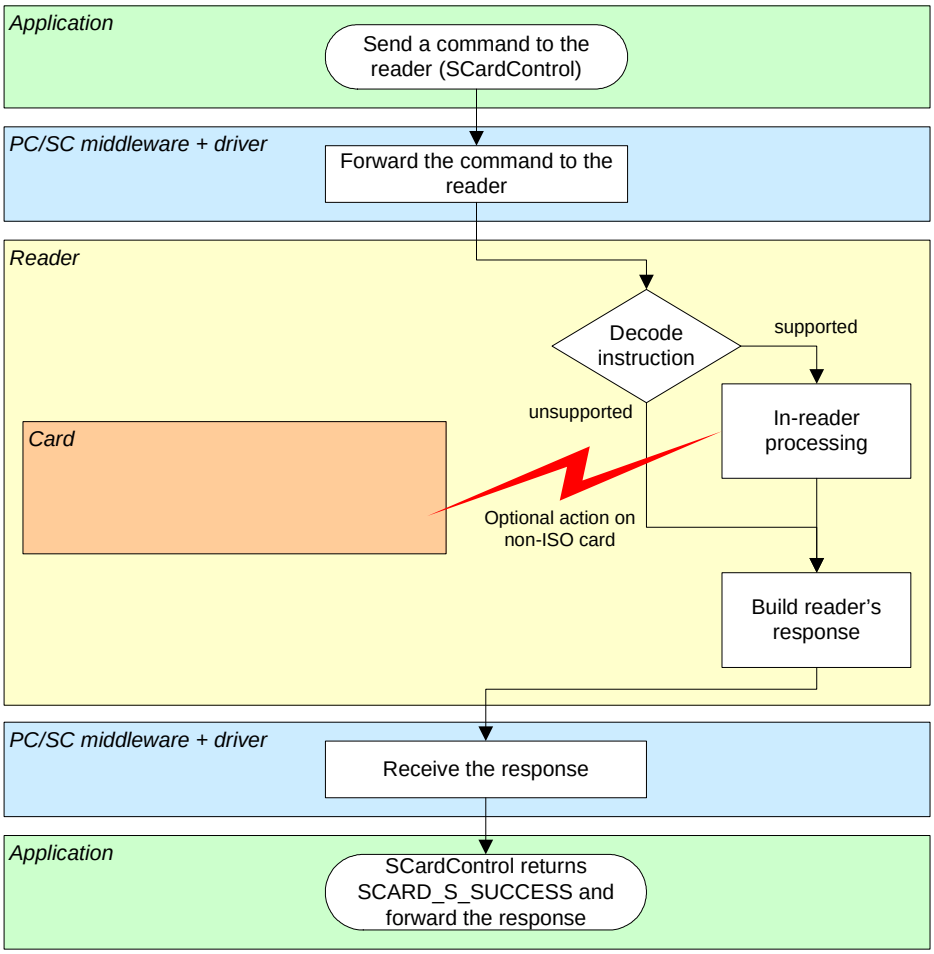
In PC/SC architecture, the **SCardControl** function implements the dialog between an application and the reader, even when there's no card in the slot.

Access to the reader must be gained using **SCardConnect**, specifying SCARD_SHARE_DIRECT as reader sharing mode.

If you CSB6 reader is a multi-slot device (contactless, contact, SAM…), calling SCardConnect with the SCARD_SHARE_DIRECT flag set gives the caller an exclusive and direct access to one slot only (a logical reader).

It doesn't prevent another application (or thread) to access the same physical reader, through another slot.

It is highly recommended to use a system-wide synchronisation object (mutex, critical section, …) to prevent any access to the same physical reader while one thread has taken direct access privilege.



3.2. IMPLEMENTATION DETAILS

3.2.1. Sample code

```
#include <winscard.h>
#define IOCTL_CSB6_PCSC_ESCAPE     SCARD_CTL_CODE(2048)

// Use ScardListReaders to get reader_name

LONG csb6_control(const char *reader_name,
                  const BYTE in_buffer[],
                  DWORD      in_length,
                  BYTE       out_buffer[],
                  DWORD      max_out_length,
                  DWORD      *got_out_length)
{
  SCARDCONTEXT hContext;
  SCARDHANDLE  hCard;

  LONG rc;
  DWORD dwProtocol;

  rc = SCardEstablishContext(SCARD_SCOPE_SYSTEM,
                             NULL,
                             NULL,
                             &hContext);
  if (rc != SCARD_S_SUCCESS)
    return rc;

  rc = SCardConnect(hContext,
                    reader_name,
                    SCARD_SHARE_DIRECT,
                    0,
                    &hCard,
                    &dwProtocol);
  if (rc != SCARD_S_SUCCESS)
  {
    SCardReleaseContext(hContext);
    return rc;
  }

  rc = SCardControl(hCard,
                    IOCTL_CSB6_PCSC_ESCAPE,
                    in_buffer,
                    in_length,
                    out_buffer,
                    max_out_length,
                    got_out_length);

  SCardDisconnect(hCard, SCARD_LEAVE_CARD);
  SCardReleaseContext(hContext);

  return rc;
}
```

3.2.2. Link to K531/SpringProx/CSB legacy protocol

Sending an escape sequence through SCardControl (with vendor-specific parameter `IOCTL_CSB6_PCSC_ESCAPE` asserted) is exactly the same as sending a "legacy command" to a CSB6 running in legacy mode.

The detailed reference of all the command supported by our reader is available in CSB4 and/or K531 development kits. The paragraphs below depicts only a

subset of the whole function list, but the functions listed here are the most useful in the PC/SC context.

3.2.3. Format of response, return codes

When dialog with the reader has been performed successfully, SCardControl returns SCARD_S_SUCCESS, and at least one byte is returned in `out_buffer` (at position 0).

The value of this byte is the actual status code of the reader : 0x00 on success, a non-zero value upon error. The complete list of reader's error codes is given in chapter 6.

When there's some data available, the data is returned at position 1 in `out_buffer`.

3.3. LIST OF AVAILABLE CONTROL SEQUENCES

3.3.1. Human interface related sequences

a. Driving reader's LEDs

For a reader with only red and green LEDs, send the sequence :

```
58 1E <red> <green>
```

For a reader with red, green and yellow / blue LEDs, send the sequence :

```
58 1E <red> <green> <yellow/blue>
```

Choose values for red, green and yellow/blue in this table :

0x00	LED is switched OFF
0x01	LED is switched ON
0x02	LED blinks slowly
0x03	LED is driven automatically by reader's firmware *(default behaviour)*
0x04	LED blinks quickly
0x05	LED performs the "heart-beat" sequence

b. Driving reader's buzzer

Some hardware feature a single tone beeper. To start the buzzer, send the sequence :

```
58 1C <duration MSB> <duration LSB>
```

Where duration specifies the length of the tone, in milliseconds (max is 60000ms).

Set duration to 0 if you need to stop the buzzer before the duration started in a previous call.

3.3.2. Obtaining information on reader and slot

The sequences below are useful to retrieve textual information such as product name, slot name, etc. The numerical information (such as version, serial number) are returned as hexadecimal strings.

Remember that the returned value (if some) is prefixed by the status code (0x00 on success).

a. Reader "product-wide" information

Sequence	Will return…
58 20 01	Vendor name ("SpringCard")
58 20 02	Product name
58 20 03	Product serial number
58 20 04	USB vendor ID and product ID
58 20 05	Product version
58 20 10	NXP MfRCxxx product code
58 20 11	Gemalto GemCore product name and version

b. Slot related information

Sequence	Will return…
58 21	Name of the current slot
58 21 00	Name of slot 0
58 21 01	Name of slot 1
58 21 NN	Name of slot N

Slot naming obey to the following rule :

- The contactless slot is named "Contactless",
- The contact smartcard slot (when present) is named "Contact",
- The external SIM/SAM slot (when present) is named "SIM/SAM (Main)",
- The two internal SIM/SAM slots (when present) are named "SIM/SAM (Aux A)" and "SIM/SAM (Aux B)".

Sending the 0x58 0x21 escape sequence is the only non-equivoque way of determining whose physical slot a PC/SC reader instance is attached to.

3.3.3. Stopping / starting a slot

When a slot is stopped, the reader

1. powers down the smartcard in the slot (if some),
2. disable the slot[3],
3. send the "card removed" event if there was a card in the slot.

When a slot is started again, the reader

1. enable the slot[4],
2. try to power up the smartcard in the slot (if some),
3. if a card has been found, send the "card inserted" event.

[3] On contactless slot, the antenna RF field is switched OFF

[4] On contactless slot, the antenna RF field is switched ON

a. Stopping a slot

Sequence	Will return…
58 22	Stop current slot
58 22 00	Stop slot 0
58 22 01	Stop slot 1
58 22 NN	Stop slot N

b. Starting a slot

Sequence	Will return…
58 23	Start current slot
58 23 00	Start slot 0
58 23 01	Start slot 1
58 23 NN	Start slot N

3.3.4. Accessing reader's non-volatile memory (FEED)

a. Reading reader's FEED

To read a non-volatile configuration data from FEED, send the sequence :

```
58 0E <index>
```

Remember that the returned value (if some) is prefixed by the status code (0x00 on success).

b. Writing reader's FEED

To store a non-volatile configuration data from FEED, send the sequence :

```
58 0E <index> <…data…>
```

4. VENDOR ATTRIBUTES

There’s currently no vendor attribute available in this reader.

5. TIPS FOR CONTACTLESS CARDS

5.1. RECOGNIZING AND IDENTIFYING CONTACTLESS CARDS IN PC/SC ENVIRONMENT

5.1.1. ATR of a contactless card

a. Smartcard

For a contactless smartcard (i.e. a card compliant with 14443 level 4 "**T=CL**"), the reader builds a pseudo-ATR using the normalized format described in PC/SC specification :

For ISO 14443-A :

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	0x3B	Direct convention
1	T0	0x8…	Higher nibble 8 means : no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (0 to 15)
2	TD1	0x80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	0x01	Higher nibble 8 means : no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means : protocol T=1
4	H1	…	Historical bytes from ATS response
…	…		
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

For ISO 14443-B :

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	0x3B	Direct convention
1	T0	0x88	Higher nibble 8 means : no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (8)
2	TD1	0x80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	0x01	Higher nibble 8 means : no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means : protocol T=1
4	H1	…	Application data from ATQB
5	H2		
6	H3		
7	H4		
8	H5	…	Protocol info byte from ATQB
9	H6		
10	H7		
11	H8	XX	MBLI from ATTRIB command
12	TCK	XX	Checksum (XOR of bytes 1 to 11)

Note regarding Calypso transportation cards :

When the non-standard Innovatron protocol is used by the contactless reader, a compliant Calypso card will answer with its "real" (contact) ATR.

In this case, the reader will use this information instead of having to build a pseudo-ATR. The host application will see no difference between the card on a contactless reader running the Innovatron radio protocol, and the same card in a contact slot running the T=0 protocol.

On the other hand, the reader makes no difference between a Calypso card that complies with the ISO 14443 standard, and any other 14443 compliant card. In this case, the pseudo-ATR will be returned.

This has two consequences :

- When a Calypso card is activated in ISO 14443 mode, the host application must issue the "Select Calypso Application" instruction and/or the "Calypso Get ATR" instruction to recognize the card for sure and have its serial number,
- When the reader alternates between ISO 14443 and Innovatron, the same card may be seen with two different ATR. The host application must add some processing to ensure it will not perform two subsequent transactions with the same card.

b. Memory card

For contactless memory cards (Mifare, CTS, etc), the reader builds a pseudo-ATR using the normalized format described in PC/SC specification :

Offset	Name	Value	
0	TS	0x3B	Direct convention
1	T0	0x8F	Higher nibble 8 means : no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (15)
2	TD1	0x80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	0x01	Higher nibble 8 means : no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means : protocol T=1
4	H1	0x80	
5	H2	0x4F	Application identifier presence indicator
6	H3	0x0C	Length to follow (12 bytes)
7	H4	0xA0	Registered Application Provider Identifier A0 00 00 03 06 is for PC/SC workgroup
8	H5	0x00	
9	H6	0x00	
10	H7	0x03	
11	H8	0x06	
12	H9	**PIX.SS**	Standard (see 5.1.3)
13	H10	**PIX.NN**	Card name (see 5.1.4)
14	H11		
15	H12	00	RFU
16	H13	00	
17	H14	00	
18	H15	00	
19	TCK	XX	Checksum (XOR of bytes 1 to 18)

5.1.2. Using the GET DATA command

With the GET DATA command (documented in 2.3.1), the host application is able to retrieve every information needed to identify a contactless card (either memory card or smartcard) :

- Serial number (UID or PUPI),
- Protocol related values (ATQA and SAKA or ATQB, …).

5.1.3. Contactless card standard

The **standard** byte (**PIX.SS** in PC/SC specification) is constructed as follow :

b7	b6	b5	b4	b3	b2	b1	b0	Description
0	0	0	0	0	0	0	0	No information given
0	0	0	0	0	0	0	1	ISO 14443 A, level 1
0	0	0	0	0	0	1	0	ISO 14443 A, level 2
0	0	0	0	0	0	1	1	ISO 14443 A, level 3 or 4 (and Mifare)
0	0	0	0	0	1	0	1	ISO 14443 B, level 1
0	0	0	0	0	1	1	0	ISO 14443 B, level 2
0	0	0	0	0	1	1	1	ISO 14443 B, level 3 or 4
0	0	0	0	0	1	0	0	RFU

5.1.4. Contactless card name bytes

The **name** bytes (**PIX.NN** in PC/SC specification) are specified as follow :

NN	Card name
Values specified by PC/SC	
0x00 0x01	NXP Mifare Standard 1k
0x00 0x02	NXP Mifare Standard 4k
0x00 0x03	NXP Mifare UltraLight
0x00 0x06	ST MicroElectronics SR176
0x00 0x07	ST MicroElectronics SRIX4K
SpringCard proprietary extension	
0xFF 0xA0	Unidentified 14443-A card
0xFF 0xB0	Unidentified 14443-B card
0xFF 0xB1	ASK CTS 256B
0xFF 0xB2	ASK CTS 512B
0xFF 0xC0	Calypso card using the Innovatron protocol
0xFF 0xFF	Virtual card (test only)

6. SPECIFIC ERROR CODES

When the APDU interpreter returns SW1 = 0x6F, the value of SW2 maps to one of the reader specific error codes listed below.

SW2	Symbolic name[5]	Meaning
0x01	MI_NOTAGERR	No answer received (no card in the field, or card is mute)
0x02	MI_CRCERR	CRC error in card's answer
0x04	MI_AUTHERR	Card authentication failed
0x05	MI_PARITYERR	Parity error in card's answer
0x06	MI_CODEERR	Invalid card response opcode
0x08	MI_SERNRERR	CRC error in card's serial number
0x0A	MI_NOTAUTHERR	Card operation denied, must be authenticated first
0x0B	MI_BITCOUNTERR	Wrong number of bits in card's answer
0x0C	MI_BYTECOUNTERR	Wrong number of bytes in card's answer
0x0F	MI_WRITEERR	Card reported write operation error
0x12	MI_READERR	Card reported read operation error
0x13	MI_OVFLERR	RC FIFO overflow
0x15	MI_FRAMINGERR	Framing error in card's answer
0x17	MI_UNKNOWN_COMMAND	RC unknown opcode
0x18	MI_COLLERR	A collision between 2 (or more) cards has occured
0x1A	MI_INITERR	RC hardware failure
0x1B	MI_INTERFACEERR	Wrong status on RC interface
0x1C	MI_ACCESSTIMEOUT	Timeout on RC interface
0x1D	MI_NOBITWISEANTICOLL	2 (or more) cards have been found, but at least 1 does not support anticollision
0x34	MI_CASCLEVEX	Card serial number longer than specified
0x36	MI_BAUDRATE_NOT_SUPPORTED	RC doesn't support this baudrate
0x3C	MI_WRONG_PARAMETER_VALUE	RC doesn’t support this value
0x3D	TCL_NOTAGERR	A card has been found, but isn't T=CL compliant
0x47	TCL_CID_NOT_ACTIVE	The specified CID isn't assigned yet
0x4B	TCL_ATSLEN	Invalid length for card's ATS
0x4D	TCL_ATS_ERROR	Invalid data in card's ATS
0x4E	TCL_FATAL_PROTOCOL	Card's response doesn't meet T=CL specifications
0x4F	TCL_RECBUF_OVERFLOW	Card's response is longer than our receive buffer
0x52	TCL_TRANSMERR_NOTAG	Card not responding anymore (maybe it has been removed)
0x53	TCL_BAUDRATE_NOT _SUPPORTED_PICC	Card doesn't support the specified baudrate
0x57	TCL_PPS_FORMAT	Card's answer to our PPS request is invalid
0x58	TCL_PPS_ERROR	Card didn't accepted our PPS request
0x63	MI_BREAK	Command execution has been interrupted by another input

[5] As used in SpringProx API (defines in springprox.h)

0x64	MI_NY_IMPLEMENTED	Unknown or un-implemented command
0x6C	TCL_CID_ACTIVE	The specified CID is already in use
0x6E	MI_WRONG_ADDR	Invalid address (sector or block number)
0x70	MI_RECBUF_OVERFLOW	Internal buffer overflow
0x7B	MI_WRONG_VALUE	An invalid value has been specified in command input
0x7D	MI_WRONG_LENGTH	Length of input command is invalid

DISCLAIMER

This document is provided for informational purposes only and shall not be construed as a commercial offer, a license, an advisory, fiduciary or professional relationship between PRO ACTIVE and you. No information provided in this document shall be considered a substitute for your independent investigation.

The information provided in document may be related to products or services that are not available in your country.

This document is provided "as is" and without warranty of any kind to the extent allowed by the applicable law. While PRO ACTIVE will use reasonable efforts to provide reliable information, we don't warrant that this document is free of inaccuracies, errors and/or omissions, or that its content is appropriate for your particular use or up to date. PRO ACTIVE reserves the right to change the information at any time without notice.

PRO ACTIVE does not warrant any results derived from the use of the products described in this document. PRO ACTIVE will not be liable for any indirect, consequential or incidental damages, including but not limited to lost profits or revenues, business interruption, loss of data arising out of or in connection with the use, inability to use or reliance on any product (either hardware or software) described in this document.

These products are not designed for use in life support appliances, devices, or systems where malfunction of these product may result in personal injury. PRO ACTIVE customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify PRO ACTIVE for any damages resulting from such improper use or sale.

COPYRIGHT NOTICE

All information in this document is either public information or is the intellectual property of PRO ACTIVE and/or its suppliers or partners.

You are free to view and print this document for your own use only. Those rights granted to you constitute a license and not a transfer of title : you may not remove this copyright notice nor the proprietary notices contained in this documents, and you are not allowed to publish or reproduce this document, either on the web or by any mean, without written permission of PRO ACTIVE.

EDITOR'S INFORMATION

PRO ACTIVE SAS company with a capital of 227 000 €
RCS EVRY B 429 665 482
Parc Gutenberg, 13 voie La Cardon
91120 Palaiseau – France