CSB6 FAMILY - PC/SC

Vendor specific attributes and commands

DOCUMENT INFORMATION

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

[pmd841p-da] csb6 pcsc vendor attributes and commands.doc
saved 24/03/10 - printed 17/05/10

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 explains 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 § 2.4
CA	22/01/09	LTC				Corrected P1 allowed values for LOAD KEY instruction Added ASK CTSB, ST SR176 support (firmware >= 1.50) Added ISO/IEC 15693 and ICODE1 support (firmware >= 1.50, RC632 chipset)
CB	18/03/09	ECL				Documentation of buzzer configuration register added
CC	04/05/09	ECL				New PIX.SS and PIX.NN values
CD	02/06/09	JDA				New SLOT CONTROL instruction (firmware >= 1.51)
CE	12/08/09	JDA				Added § 3.4 Details regarding memory card READ/UPDATE moved to chapter 5 Added support of Inside Contactless PicoPass and NXP Mifare Plus (firmware >= 1.52)
DA	09/02/10	JDA				Added ISO 15693 and "ciphered" Mifare frames in ENCAPSULATE (firmware >= 1.53) Added support of Innovision Jewel/Topaz (firmware >= 1.53)

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. It 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

1.6. HARDWARE VERSION WARNING

Note that most products in the CSB6 Family have been manufactured in two different hardware groups:

- Hardware having the "RC531 chipset" supports contactless cards that are compliant *(more or less)* with the ISO 14443 standard (Mifare, Desfire, ICAO passports and others travel documents, Calypso or equivalent cards for public transport, payment cards…),

- Hardware having the "RC632 chipset" adds support for vicinity cards and RFID tags, that are compliant with the ISO 15693 standard (ICODE, TagIT, …).

Nowadays, all manufactured products are (unless specified) equipped with "NXP RC632" chipset. Be aware when upgrading an old product that changing the firmware is not enough to add ISO 15693 support.

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.
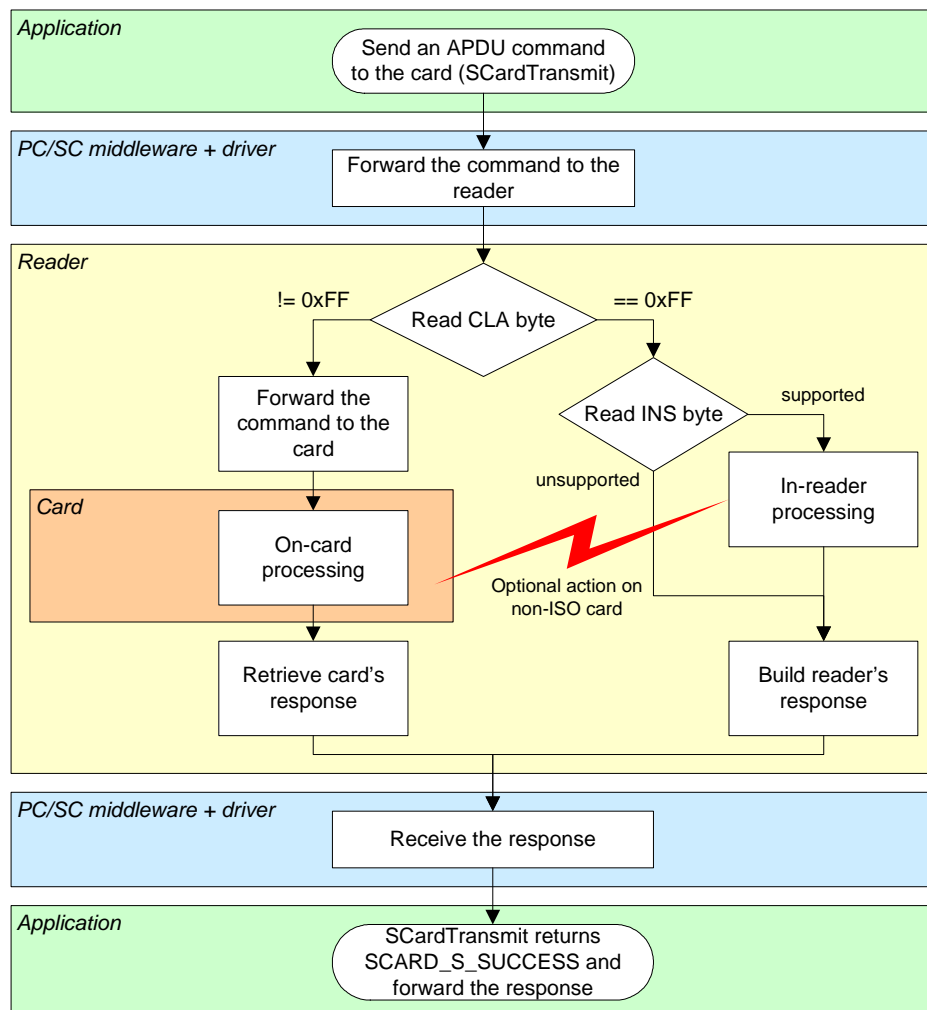
2.1.1. CLA byte of the embedded APDU interpreter

Default class is $_h$FF. This means that every APDU starting with CLA= $_h$FF 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 in register $_h$B2 of reader's non volatile memory (see § 3.4.3).

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

b. Disabling the embedded APDU interpreter

Define CLA byte = $_h$00 (register $_h$B2= $_h$00, see § 3.4.3) to disable the embedded APDU interpreter.

2.1.2. Status words returned by the embedded APDU interpreter

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$67	$_h$00	Wrong length (Lc incoherent with Data In)
$_h$68	$_h$00	CLA byte is not correct
$_h$6A	$_h$81	Function not supported (INS byte is not correct), or not available for the selected card
$_h$6B	$_h$00	Wrong parameter P1-P2
$_h$6F	$_h$01	Card mute (or removed)

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	$_h$82	✓	
GENERAL AUTHENTICATE	$_h$86	✓	
READ BINARY	$_h$B0	✓	
GET DATA	$_h$CA	✓	✓
UPDATE BINARY	$_h$D6	✓	
READER CONTROL	$_h$F0	✓	✓
RC CONTROL	$_h$F1	✓	
GEMCORE CONTROL	$_h$F1		✓
MIFARE CLASSIC READ	$_h$F3	✓	
MIFARE CLASSIC WRITE	$_h$F4	✓	
SET DATA	$_h$FA	✓	
SLOT CONTROL	$_h$FB	✓	
RFU (CALYPSO)	$_h$FC		
TEST	$_h$FD	✓	✓
ENCAPSULATE	$_h$FE	✓	✓

2.2. COMMANDS AVAILABLE ON ALL SLOTS

Those commands allow interacting 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 driving 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 <u>logical</u> reader) but may have a global impact to the whole <u>physical</u> reader.
>
> In other words, sending a READER CONTROL instruction in one card connection may have an impact on another card.
>
> It is <u>highly recommended</u> to use a synchronisation object (mutex, critical section, …) to prevent any concurrent access to the same physical reader when the READER CONTROL instruction is called.

READER CONTROL command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_hFF$	$_hF0$	$_h00$	$_h00$	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 :

$_h00$	LED is switched OFF
$_h01$	LED is switched ON
$_h02$	LED blinks slowly
$_h03$	LED is driven automatically by reader's firmware *(default behaviour)*
$_h04$	LED blinks quickly
$_h05$	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 $_h58$ 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 applications access to cards that don't comply with ISO 7816-4[1].

ENCAPSULATE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
ₕFF	ₕFE	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	Standard communication protocols
ₕ00	Send the frame in the **T=CL** stream, using the ISO 14443-4 protocol[2]. Data In shall not include PCB nor CRC fields
ₕ01	Send the frame "as is" using the ISO 14443-3 A protocol. The standard parity bits are added (and checked in return) by the reader. The standard CRC is added (and checked in return) by the reader.
ₕ02	Send the frame "as is" using the ISO 14443-3 B protocol. The standard CRC is added (and checked in return) by the reader.
ₕ04	Send the frame "as is" using the ISO 15693 protocol[3]. The standard CRC is added (and checked in return) by the reader.

P1	Non-standard communication
ₕ09	Send the frame "as is" using the ISO 14443-3 A modulation. The standard parity bits are added (and checked in return) by the reader, but the CRC is <u>not</u> added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out.
ₕ0A	Send the frame "as is" using the ISO 14443-3 B modulation. The CRC is <u>not</u> added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out.
ₕ0C	Send the frame "as is" using the ISO 15693 modulation. The CRC is <u>not</u> added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out.

---

[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.

[2] For instance a legacy Desfire frame, that doesn't obey to ISO 7816-4 rules (see note 1).

[3] The 2-low order bits in first byte of the frame are always overwritten by the reader according to its communication parameters: one subcarrier, fast baudrate VICC to VCD.

P1	Mifare low level communication[4]
$_h$0F	Send the frame "as is" using the ISO 14443-3 A modulation. The CRC is <u>not</u> added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out. The parity bits are <u>not</u> added (and not checked) by the reader → the application must provide a valid stream, including the parity bits). The last byte is complete (8 bits will be sent)
$_h$1F	Same as $_h$0F, but only 1 bit of the last byte will be sent
$_h$2F	Same as $_h$0F, but only 2 bits of the last byte will be sent
$_h$3F	Same as $_h$0F, but only 3 bits of the last byte will be sent
$_h$4F	Same as $_h$0F, but only 4 bits of the last byte will be sent
$_h$5F	Same as $_h$0F, but only 5 bits of the last byte will be sent
$_h$6F	Same as $_h$0F, but only 6 bits of the last byte will be sent
$_h$7F	Same as $_h$0F, but only 7 bits of the last byte will be sent
P1	Redirection to another slot[5]
$_h$80	Redirection to the main contact slot (if present)
$_h$81	Redirection to the 1st SIM/SAM slot (if present)
$_h$82	Redirection to the 2nd SIM/SAM slot (if present)
$_h$83	Redirection to the 3rd SIM/SAM slot (if present)
$_h$84	Redirection to the 4th SIM/SAM slot (if present)

[4] The above values allow an application to transmit "ciphered" Mifare frames (the CRYPTO1 stream cipher makes a non-standard use of the parity bits and CRC). The number of valid bits in the last byte of card's answer will be reported in SW2.

[5] Those values allow an application to transmit APDUs to a SAM or an auxiliary card through the PC/SC handle of the main card.

ENCAPSULATE command parameter P1 for the contact slots

P1	
$_h00$	Send the frame in the T=0 or T=1 stream Other values are RFU

ENCAPSULATE command parameter P2 for the contactless slot

P2 encodes the frame timeout.

P2	
$_h{-}0$	If P1 = $_h00$, use default T=CL timeout defined by the card If P1 ≠ $_h00$, this value shall not be used
$_h{-}1$	Timeout = 106 ETU ≈ 1ms
$_h{-}2$	Timeout = 212 ETU ≈ 2ms
$_h{-}3$	Timeout = 424 ETU ≈ 4ms
$_h{-}4$	Timeout = 848 ETU ≈ 8ms
$_h{-}5$	Timeout = 1696 ETU ≈ 16ms
$_h{-}6$	Timeout = 3392 ETU ≈ 32ms
$_h{-}7$	Timeout = 6784 ETU ≈ 65ms
$_h{-}8$	Timeout = 13568 ETU ≈ 0,125s
$_h{-}9$	Timeout = 27136 ETU ≈ 0,250s
$_h{-}A$	Timeout = 54272 ETU ≈ 0,500s
$_h{-}B$	Timeout = 108544 ETU ≈ 1s
$_h{-}C$	Timeout = 217088 ETU ≈ 2s
$_h{-}D$	Timeout = 434176 ETU ≈ 4s
$_h0{-}$	Set status word = $_h6F$ XX , XX being the contactless specific error
$_h8{-}$	Set status word = $_h63$ 00 on any contactless specific error

ENCAPSULATE command parameter P2 for the contact slot

P2	
$_h00$	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
$_h90$	$_h00$	Success - last byte of Data Out has 8 valid bits
$_h90$	$_h01$	Success - last byte of Data Out has 1 valid bits
$_h90$	$_h02$	Success - last byte of Data Out has 2 valid bits
$_h90$	$_h03$	Success - last byte of Data Out has 3 valid bits
$_h90$	$_h04$	Success - last byte of Data Out has 4 valid bits
$_h90$	$_h05$	Success - last byte of Data Out has 5 valid bits
$_h90$	$_h06$	Success - last byte of Data Out has 6 valid bits
$_h90$	$_h07$	Success - last byte of Data Out has 7 valid bits
$_h6F$	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.
$_h63$	$_h00$	Error reported by the contactless interface (when high-order bit of P2 is 1).
$_h62$	$_h82$	Le is greater than actual response from card
$_h6C$	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
$_h$FF	$_h$FD	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 :

- $_h$00 : empty Data Out, only SW returned
- $_h$FF : 255 bytes of data + SW
- All values between $_h$00 and $_h$FF are allowed

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

- $_h$00 : no delay, response comes immediately
- $_h$3F : 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
$_h$90	$_h$00	Success, APDU correctly formatted
$_h$67	$_h$00	APDU is badly formatted (total length incoherent with Lc value)
$_h$6A	$_h$82	Le is greater than data length specified in P1
$_h$6C	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
$_h$FF	$_h$CA	See below	See below	–	–	$_h$00

GET DATA command parameters

P1	P2	Will return in Data Out
$_h$00	$_h$00	Card's serial number - ISO 14443-A : UID (4, 7 or 11 bytes) - ISO 14443-B : PUPI (4 bytes) - ISO 15693 : UID (8 bytes) - Innovatron : DIV (4 bytes) - others : see chapter 5 for details
$_h$01	$_h$00	- ISO 14443-A : historical bytes from the ATS - ISO 14443-B : INF field in ATTRIB response - others : see chapter 5 for details
$_h$F0	$_h$00	Card's complete identifier : - ISO 14443-A : ATQ (2 bytes) + SAK (1 byte) + UID - ISO 14443-B : full REQB response (11 bytes) - Innovatron : REPGEN - others : see chapter 5 for details
$_h$F1	$_h$00	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
$_h$F2	$_h$00	Card's short serial number - ISO 14443-A : UID truncated to 4 bytes, in "classical" order - others : same as P1,P2=$_h$00,$_{rh}$00
$_h$FA	$_h$00	Card's ATR
$_h$FF	$_h$00	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
$_h$90	$_h$00	Success
$_h$62	$_h$82	End of data reached before Le bytes (Le is greater than data length)
$_h$6C	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
$_h$FF	$_h$82	Key location	Key index	$_h$06	Key value (6 bytes)	–

LOAD KEY command parameter P1 (key location)

P1	
$_h$00	The key is to be loaded in reader's volatile memory
$_h$20	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 = $_h$00, 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
$_h$00	A	0
…	(…)	(…)
$_h$03	A	3
$_h$10	B	0
…	(…)	(…)
$_h$13	B	3

When P1 = $_h$20, 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
$_h$00	A	0
…	(…)	(…)
$_h$0F	A	15
$_h$10	B	0
…	(…)	(…)
$_h$1F	B	15

LOAD KEY response

SW1	SW2
See below	

LOAD KEY status word

SW1	SW2	Will return in Data Out
$_h$90	$_h$00	Success
$_h$69	$_h$86	Volatile memory is not available
$_h$69	$_h$87	Non-volatile memory is not available
$_h$69	$_h$88	Key number is not valid
$_h$69	$_h$89	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 specified 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
$_h$FF	$_h$86	$_h$00	$_h$00	$_h$05	See below	–

GENERAL AUTHENTICATE Data In bytes

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
$_h$01	$_h$00	Block number	Key location or Key type	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)*.

SpringCard specific:

- Key location (byte 3) is the same as P1 parameter used in the LOAD KEY command ($_h$00 or $_h$20).

- Key index (byte 4) is the same as P2 parameter used in the LOAD KEY command.

PC/SC interoperability:

Firmware >= 1.52

- Key type (byte 3) must be set to $_h$60 for authentication using a Mifare 'A' key, or to $_h$61 for authentication using a Mifare 'B' key.

- Key index (byte 4) is defined as follow:

Key index	If Key Type = $_h$60 ('A')	If Key Type = $_h$61 ('B')
$_h$00	key A, index 0 in RAM	key B, index 0 in RAM
…	*(…)*	*(…)*
$_h$03	key A, index 3 in RAM	key B, index 3 in RAM
$_h$20	key A, index 0 in EEPROM	key B, index 0 in EEPROM
…	*(…)*	*(…)*
$_h$2F	key A, index 15 in EEPROM	key B, index 15 in EEPROM

GENERAL AUTHENTICATE response

SW1	SW2
See below	

GENERAL AUTHENTICATE status word

SW1	SW2	Meaning
$_h90$	$_h00$	Success
$_h69$	$_h82$	Authentication failed
$_h69$	$_h86$	Key type is not valid
$_h69$	$_h88$	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).
>
> *Tip : when working with a Mifare Classic card, using the MIFARE CLASSIC READ instruction (§ 2.3.5) is easier and may shorten the transaction time.*

READ BINARY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_hFF$	$_hB0$	Address MSB	Address LSB	–	–	XX

> The allowed values for Address and Le vary with the card type. The chapter 5 summarizes the typical values for most commonly used cards. Always refer to the card's datasheet as published by the card's manufacturer for accurate information.

READ BINARY response

Data Out	SW1	SW2
XX … XX	See below	

READ BINARY status word

SW1	SW2	Will return in Data Out
$_h90$	$_h00$	Success
$_h62$	$_h82$	End of data reached before Le bytes (Le is greater than data length)
$_h69$	$_h81$	Command incompatible
$_h69$	$_h82$	Security status not satisfied
$_h6A$	$_h82$	Wrong address (no such block or no such offset in the card)
$_h6C$	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

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
$_h$FF	$_h$F3	$_h$00	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
hFF	hF3	h00	Block Number	h06	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
hFF	hD6	Address MSB	Address LSB	XX	Data	–

> The allowed values for Address and Lc vary with the card type. The chapter 5 summarizes the typical values for most commonly used cards. Always refer to the card's datasheet as published by the card's manufacturer for accurate information.

> ☠ Pay attention that most cards have specific areas ("one time programming", "fuses", "security blocks", "sector trailers" …) that may be written only once, and/or that must be written carefully because only specific values are allowed (and setting an invalid value may lock the card).

UPDATE BINARY response

SW1	SW2
See below	

UPDATE BINARY status word

SW1	SW2	Will return in Data Out
h90	h00	Success
h69	h82	Security status not satisfied
h6A	h82	Wrong address (no such block or no such offset in the card)
h6A	h84	Wrong length (trying to write too much data at once)

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.

> ☠ Writing sector trailers (security blocks) is possible as long as the sector's current access conditions allow it, <u>but</u> Mifare sector trailers <u>must</u> 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 <u>permanently unusable</u> (writing a key you don't remember would have the same impact).
>
> Please read card's documentation carefully before writing into those blocks.

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 order 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[6].

---

[6] 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.

MIFARE CLASSIC WRITE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF4	h00	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.

> 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
hFF	hF4	h00	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.

2.3.8. SLOT CONTROL instruction

The **SLOT CONTROL** instruction allows to pause and resume the card tracking mechanism of the contactless slot.

This is useful because card tracking implies sending commands to the card periodically (and watch-out its answer). Such monitoring commands may have unwanted side-effects, such as breaking the atomicity between a pair of commands. Switching the card tracking mechanism OFF during the transaction with solve this problem.

SLOT CONTROL command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	See below	See below	–	–	–

SLOT CONTROL command parameters

P1	P2	Action
$_h$00	$_h$00	Resume the card tracking mechanism
$_h$01	$_h$00	Suspend the card tracking mechanism
$_h$10	$_h$00	Stop the RF field
$_h$10	$_h$01	Start the RF field
$_h$10	$_h$02	Reset the RF field
$_h$10	$_h$03	Reset the RF field and temporarily disable the T=CL activation
$_h$20	$_h$00	T=CL de-activation (DESELECT[7])
$_h$20	$_h$01	T=CL activation of ISO 14443-A card (RATS)
$_h$20	$_h$02	T=CL activation of ISO 14443-B card (Attrib)
$_h$20	$_h$10	Specific activation of Inside PicoPass (cf § 5.3.4)
$_h$DE	$_h$AD	Stop the slot NOTE: a stopped slot is not available to *SCardConnect* anymore. It may be restarted only through an *SCardControl* command.

SLOT CONTROL response

Data Out	SW1	SW2
–	See below	

SLOT CONTROL status word

SW1	SW2	Meaning
$_h$90	$_h$00	Success

---

[7] Or DISC for Innovatron cards

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
hFF	hFC	See below	See below	h00	–	–

CONFIGURE CALYPSO SAM command parameters

P1	P2	Will return in Data Out
h04	h00	Configure Calypso SAM for 9600 bps communication
h04	h01	Configure Calypso SAM for 115200 bps communication
h08	h00	Disable Calypso internal DigestUpdate mode
h08	h01	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
h90	h00	Success
h6B	h00	Wrong value for P1
h6F	hE7	SAM didn't answer with 9000 (maybe this is not a Calypso SAM !)
h6F	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.
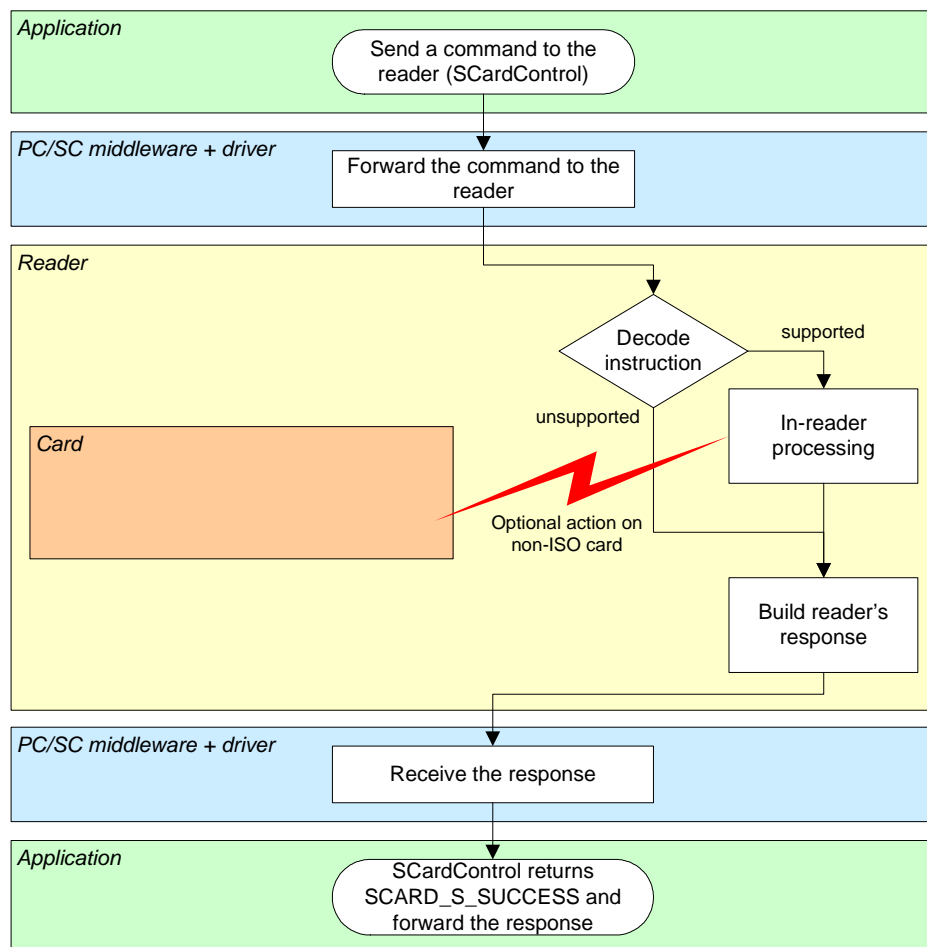
Access to the reader must be gained using **SCardConnect**, specifying SCARD_SHARE_DIRECT as reader sharing mode.

> 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 <u>logical</u> reader).
>
> It doesn't prevent another application (or thread) to access the same <u>physical</u> reader, through another slot.
>
> It is <u>highly recommended</u> 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.

SPRINGCARD, the SPRINGCARD logo, PRO ACTIVE and the PRO ACTIVE logo are registered trademarks of PRO ACTIVE SAS.
All other brand names, product names, or trademarks belong to their respective holders.
Information in this document is subject to change without notice. Reproduction without written permission of PRO ACTIVE is forbidden.

3.2. IMPLEMENTATION DETAILS

3.2.1. Sample code

```c
#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 depict 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 : $_h00$ 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`.

3.2.4. Redirection to the Embedded APDU Interpreter

SCardControl buffers starting by $_hFF$ (CLA byte of the Embedded APDU Interpreter) as processed as if they were received in a *SCardTransmit* stream.

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 :

$_h00$	LED is switched OFF
$_h01$	LED is switched ON
$_h02$	LED blinks slowly
$_h03$	LED is driven automatically by reader's firmware *(default behaviour)*
$_h04$	LED blinks quickly
$_h05$	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.

To control buzzer's behaviour when a card is detected, see 3.3.4.b

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 ($_h$00 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 $_h$58 $_h$21 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[8],
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[9],
2. try to power up the smartcard in the slot (if some),
3. if a card has been found, send the "card inserted" event.

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

---

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

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

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

Products in the **SpringCard CSB6 Family** features a non-volatile memory to store configuration registers.

See next paragraph for the list of these registers, and their allowed values.

a. Reading reader's registers

To read the value of the configuration register at <index>, send the sequence :

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

Remember that the returned value (if some) is prefixed by the status code ($_h00$ on success, $_h16$ if the value is not defined in the non-volatile memory).

b. Writing reader's registers

To define the value of the configuration register at <index>, send the sequence :

```
58 0D <index> <…data…>
```

Send an empty <data> (zero-length) to erase the current value.

The non-volatile memory has a limited write/erase endurance.

Writing any configuration register more than 100 times may permanently damage your product.

3.4. CONFIGURATION REGISTERS

3.4.1. Card lookup list

Firmware >= 1.52

This register defines the list of protocols activated by the reader. Any contactless card compliant with one of the activated protocols will be "seen", and the others ignored.

Address: $_h$B0 – Size: 2 bytes (MSB first)

	Bit	Activ. protocol (if set)	Note	RC531	RC632
msb	15	*RFU*			
	14	*RFU*			
	13	*RFU*			
	12	Innovision Topaz/Jewel (NFC Forum's type 1 tags)		✔	✔
	11	NXP ICODE1 (slow VCD to VICC baudrate)		-	✔
	10	ISO 15693 (slow VCD to VICC baudrate) [10]		-	✔
	9	*RFU*			
	8	*RFU*			
	7	Innovatron (legacy Calypso cards – sometimes called ISO 14443-**B'**)		✔	✔
	6	ASK CTS256B et CTS512B		✔	✔
	5	ST MicroElectronics SRxxx		✔	✔
	4	Inside Contactless PicoPass (also HID iClass)		✔	✔
	3	NXP ICODE1 (fast VCD to VICC baudrate)		-	✔
	2	ISO 15693 (fast VCD to VICC baudrate)		-	✔
	1	ISO 14443-B		✔	✔
lsb	0	ISO 14443-A		✔	✔

Default value: $_h$FFFF (all protocols are activated)

3.4.2. CCID slot mapping

Address: $_h$B1

RFU, leave undefined (unless instructed by SpringCard support team).

3.4.3. CLA byte of CCID interpreter

This register defines the CLA (class) byte affected to the APDU interpreter (see § 2.1.1).

To disable the APDU interpreter, define this register to $_h$00.

Address: $_h$B2 – Size: 1 byte

Default value: $_h$FF

---

[10] The VICC to VCD baudrate is always fast

3.4.4. Misc. T=CL options

This register defines the behaviour of the reader against ISO 14443-4 (T=CL) cards.

Address: $_h$B3 – Size: 1 byte

	Bit	Action if set	Note
msb	7	Innovatron : return the "real" T=0 ATR (as supplied in REPGEN) instead of building a pseudo ATR	Setting this bit breaks the compatibility with MS CCID driver, because the card is connected in T=1 where its ATR claims it is T=0 only[11]
	6	*RFU*	
	5	*RFU*	
	4	*RFU*	
	3	*RFU*	
	2	*RFU*	
	1	No T=CL activation over ISO 14443-B	Send SLOT CONTROL P1,P2=$_h$20,01 to activate the card manually
lsb	0	No T=CL activation over ISO 14443-A	Send SLOT CONTROL P1,P2=$_h$20,02 to activate the card manually

Default value: $_h$00 (T=CL active over 14443 A and B)

---

[11] Firmware < 1.52 returns the "real" T=0 ATR only. This prevents correct operation with Innovatron Calypso cards when Microsoft's CCID driver is used. Use SpringCard's CCID driver instead.

3.4.5. Firmware operating mode

This register defines how the product's firmware will be seen by the computer. It can be either PC/SC or Legacy. Note that this documentation is related to PC/SC mode only.

> ☠ Setting an inappropriate value in this register will make the reader permanently unusable.

Address: $_h$C0 – Size: 1 byte

Value	Operating mode
$_h$00	*RFU*
$_h$01	Legacy mode
$_h$02	PC/SC mode
$_h$03	*Not supported by this firmware*
$_h$80	*RFU*
$_h$81	Legacy mode without serial number in USB descriptor
$_h$82	PC/SC mode without serial number in USB descriptor
$_h$83	*Not supported by this firmware*

Default value: $_h$02 (PC/SC)

3.4.6. RC531/RC632 chipset configuration vector

Address: $_h$C1

RFU, leave undefined *(unless instructed by SpringCard support team)*.

3.4.7. Calypso compliance

Address: $_h$C2

RFU, leave undefined *(unless instructed by SpringCard support team)*.

3.4.8. T=CL speed limit

Firmware >= 1.52

This register defines the fastest speed that the reader will try to negotiate when a T=CL (ISO 14443-4) card enters its field.

> Every reader in the **SpringCard CSB6 Family** is theoretically able to communicate with contactless cards at 848kbps in both directions, but the actual maximum speed depends heavily on the card characteristics, and on the reader's environment.
>
> Therefore, it is generally speaking better to put the limit at 106kbps or 212kbps.
>
> Communication is slower yet more reliable, so the overall transaction time often appears shorter than when a faster baudrate (leading to frequent errors and retries) is selected.

Address: $_h$C4 – Size: 2 bytes (MSB first)

	Bit	Meaning (if set)
		ISO 14443-A DS
msb	15	*RFU, must be 0*
	14	Allow ISO 14443-A DS (card → reader) = 848kbps
	13	Allow ISO 14443-A DS (card → reader) = 424kbps
	12	Allow ISO 14443-A DS (card → reader) = 212kbps
		ISO 14443-A DR
	11	*RFU, must be 0*
	10	Allow ISO 14443-A DR (reader → card) = 848kbps
	9	Allow ISO 14443-A DR (reader → card) = 424kbps
	8	Allow ISO 14443-A DR (reader → card) = 212kbps
		ISO 14443-B DS
	7	*RFU, must be 0*
	6	Allow ISO 14443-B DS (card → reader) = 848kbps
	5	Allow ISO 14443-B DS (card → reader) = 424kbps
	4	Allow ISO 14443-B DS (card → reader) = 212kbps
		ISO 14443-B DR
	3	*RFU, must be 0*
	2	Allow ISO 14443-B DR (reader → card) = 848kbps
	1	Allow ISO 14443-B DR (reader → card) = 424kbps
lsb	0	Allow ISO 14443-B DR (reader → card) = 212kbps

Default value: $_h$1111 (212kbps)[12].

3.4.9. Buzzer settings

This register defines the duration or reader's beep when a card enters its field.

To disable the buzzer[13], define this register to $_h$00.

Address: $_h$CC – Size: 1 byte

Default value: $_h$08 (beep 80ms when a card is "seen").

---

[12] For firmware <=1.50, readers are limited to 106kbps in both direction.

[13] Buzzer may still be driven by PC software, see § 3.3.1.b .

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 smartcards

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 :

a. For ISO 14443-A :

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$8…	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	$_h$80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	$_h$01	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)

b. For ISO 14443-B :

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$88	Higher nibble 8 means : no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (8)
2	TD1	$_h$80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	$_h$01	Higher nibble 8 means : no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means : protocol T=1
4	H1		
5	H2		Application data from ATQB
6	H3	…	
7	H4		
8	H5		
9	H6	…	Protocol info byte from ATQB
10	H7		
11	H8	XX	MBLI from ATTRIB command
12	TCK	XX	Checksum (XOR of bytes 1 to 11)

c. For Innovatron (legacy Calypso cards)[14] :

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$8...	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	$_h$80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	$_h$01	Higher nibble 8 means : no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means : protocol T=1
4	H1		Historical bytes from REPGEN. This is the last part of the card's T=0 ATR, including its serial number[15].
...	...	...	
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

> Most Calypso cards are able to communicate either according to ISO 14443-B or to Innovatron protocol. The choice between the two protocols is unpredictable.
>
> The same card will have two different ATR (one is ISO 14443-B is selected, the other if Innovatron protocol is selected). The host application must get and check the card's serial number[16] to make sure it will not start a new transaction on the same card as earlier.

---

[14] When bit 7 of register $_h$B3 is unset (and firmware version is ≥ 1.52). Otherwise, the "real" card ATR (found in REPGEN) is returned. This ATR reports that the card supports T=0 only, but the card behaves as it were T=1. This behaviour is not compliant with Microsoft's CCID driver.

[15] As a consequence, all the cards have a different ATR.

[16] Provided in the historical bytes of the ATR when the Innovatron protocol is selected, or available through the Calypso "Select Application" command.

5.1.2. ATR of a contactless memory cards

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	h3B	Direct convention
1	T0	h8F	Higher nibble 8 means : no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (15)
2	TD1	h80	Higher nibble 8 means : no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means : protocol T=0
3	TD2	h01	Higher nibble 8 means : no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means : protocol T=1
4	H1	h80	
5	H2	h4F	Application identifier presence indicator
6	H3	h0C	Length to follow (12 bytes)
7	H4	hA0	
8	H5	h00	Registered Application Provider Identifier
9	H6	h00	A0 00 00 03 06 is for PC/SC workgroup
10	H7	h03	
11	H8	h06	
12	H9	**PIX.SS**	Standard (see 5.1.4)
13	H10	**PIX.NN**	Card name (see 5.1.5)
14	H11		
15	H12	00	
16	H13	00	RFU
17	H14	00	
18	H15	00	
19	TCK	XX	Checksum (XOR of bytes 1 to 18)

5.1.3. 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, …).

5.1.4. 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	1	0	0	1	ICODE 1
0	0	0	0	1	0	1	1	ISO 15693

Note : PIX.SS is defined for both memory and micro-processor based cards, but available in the ATR for memory cards only. In the other case, use the GET DATA command APDU (with parameters P1,P2=$_h$F1,00) to get the underlying protocol used by the smartcard.

5.1.5. 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	
ₕ00 ₕ01	NXP Mifare Standard 1k
ₕ00 ₕ02	NXP Mifare Standard 4k
ₕ00 ₕ03	NXP Mifare UltraLight
ₕ00 ₕ06	ST MicroElectronics SR176
ₕ00 ₕ07	ST MicroElectronics SRIX4K
ₕ00 ₕ12	Texas Intruments TAG IT
ₕ00 ₕ13	ST MicroElectronics LRI512
ₕ00 ₕ14	NXP ICODE SLI
ₕ00 ₕ16	NXP ICODE1
ₕ00 ₕ21	ST MicroElectronics LRI64
ₕ00 ₕ24	ST MicroElectronics LR12
ₕ00 ₕ25	ST MicroElectronics LRI128
ₕ00 ₕ26	NXP Mifare Mini
ₕ00 ₕ2F	Innovision Jewel
ₕ00 ₕ30	Innovision Topaz (NFC Forum's type 1 tag)
ₕ00 ₕ35	NXP ICODE SL2
SpringCard proprietary extension	
ₕFF ₕA0	Generic/unknown 14443-A card
ₕFF ₕB0	Generic/unknown 14443-B card
ₕFF ₕB1	ASK CTS 256B
ₕFF ₕB2	ASK CTS 512B
ₕFF ₕB3	ST MicroElectronics SRI 4K
ₕFF ₕB4	ST MicroElectronics SRI X512
ₕFF ₕB5	ST MicroElectronics SRI 512
ₕFF ₕB6	ST MicroElectronics SRT 512
ₕFF ₕB7	Inside Contactless PICOTAG/PICOPASS
ₕFF ₕC0	Calypso card using the Innovatron protocol
ₕFF ₕD0	Unidentified 15693 card (unknown manufacturer)
ₕFF ₕD1	Unidentified LEGIC 15693 card
ₕFF ₕD2	Unidentified ST MicroElectronics 15693 card
ₕFF ₕFF	Virtual card (test only)

Note : PIX.NN is specified for memory cards only. Even if the GET DATA command APDU allows to retrieve PIX.NN even for micro-processor based cards (smartcards), the returned value is unspecified and shall not be used to identify the card.

5.2. WORKING WITH MEMORY CARDS, ISO 14443-A GROUP

Please download the datasheets of the cards at www.nxp.com. Useful information are also to be found at www.mifare.net.

5.2.1. Mifare Classic cards

The cards covered by this chapter are :

- Mifare 1k (NXP MF1ICS50),
- Mifare 4k (NXP MF1ICS70),
- Mifare Mini (NXP MF1ICS20),
- Mifare Plus (X or S) when used in level 1 (see § 5.2.2).

All these cards are divided into 16-byte blocks. The blocks are grouped in sectors. At the end of every sector a specific block ("sector trailer") is reserved for security parameters (access keys and access conditions).

Most ISO 14443-A compliant smartcards or NFC objects are able to emulate Mifare Classic cards, but since they are also ISO 14443-4 (T=CL) compliant, the reader will activate them at T=CL level, "hiding" the Mifare emulation mode.

There are 3 workarounds :

- Send the T=CL DESELECT command to the card (SLOT CONTROL with P1,P2=$_h$20,00),
- Reset the RF field and temporarily disable T=CL activation (SLOT CONTROL with P1,P2=$_h$10,03),
- Permanently disable T=CL activation through configuration register $_h$B3.

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,
- 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),
- As the size of every 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 MIFARE CLASSIC READ instruction (§ 2.3.5) is easier and may shorten the transaction time.

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,

- P2 is the address of the <u>first block to be written</u> (1 to 63 for a Mifare 1k, 1 to 255 for a Mifare 4k),

- As the size of every block is 16, <u>Lc must be a multiple of 16</u> (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, <u>but</u> Mifare sector trailers <u>must</u> 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 <u>permanently unusable</u> (writing a key you don't remember would have the same impact).
>
> Please read card's documentation carefully before writing into those 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 MIFARE CLASSIC WRITE instruction (§ 2.3.7) is easier and may shorten the transaction time.

5.2.2. Mifare Plus X and Mifare Plus S

The Mifare Plus cards implement 4 different security level. The behaviour of the card changes dramatically with the selected security level.

a. Level 0

At level 0, the card is ISO 14443-4 (T=CL) compliant. The reader builds a smartcard ATR according to § 5.1.1. The historical bytes of the ATS are included in the ATR and help recognizing the card at this level.

The application may send the card commands "as it" in the *SCardTransmit* stream, but to preserve interoperability with other readers it is better to wrap them in ENCAPSULATE commands APDU with P1=$_h$00 (§ 2.2.2).

At the end of the personalisation process, the RF field must be reset (so the card will restart at Level 1 or more). Send the SLOT CONTROL command APDU with P1,P2=$_h$10,02 to do so (§ 2.3.8)[17].

b. Level 1

At level 1, the card emulates a Mifare Classic card (§ 5.2.1). The reader builds a memory card ATR according to § 5.1.1.

The application shall use the MIFARE CLASSIC READ and MIFARE CLASSIC WRITE command APDUs to work with the card.

The card supports a new <u>AES authentication</u> Function. Use the ENCAPSULATE command APDU with P1=$_h$01 (§ 2.2.2) to implement this function.

In order to increase the security level of the card (going to level 2 or level 3), an ISO 14443-4 (T=CL) session opening must be forced onto the card[18]. Send the SLOT CONTROL command APDU with P1,P2=$_h$20,01 to do so (§ 2.3.8). Afterwards, process as documented for level 0.

c. Level 2

The level 2 <u>is not available</u> on Mifare Plus S cards.

Working with the Mifare Plus X at this level is possible thanks to the low level command calls (SLOT CONTROL, ENCAPSULATE, GENERAL AUTHENTICATE, READ BINARY, UPDATE BINARY).

Anyway, working with the card at this level through a PC/SC reader is not recommended[19].

---

[17] As a consequence, the card with be reported as REMOVED, then a new CARD INSERT event will be triggered (but with a different ATR as the security level is different).

[18] Because the card reports it is not 14443-4 compliant.

[19] Consider using the reader in Legacy mode if you really need to work with Mifare Plus X at security level 2.

d. Level 3

At level 4, the card is ISO 14443-4 (T=CL) compliant. The reader builds a smartcard ATR according to § 5.1.1. The historical bytes of the ATS are included in the ATR and help recognizing the card at this level.

The application may send the card commands "as it" in the *SCardTransmit* stream, but to preserve interoperability with other readers it is better to wrap them in ENCAPSULATE commands APDU with P1=$_h$00 (§ 2.2.2).

5.2.3. Mifare UltraLight or Mifare UltraLight C cards

The cards covered by this chapter are :

■ Mifare UL (NXP MF01CU1),

■ Mifare UL C (NXP MF01CU2).

These cards are divided into 4-byte pages.

a. READ BINARY

In the READ BINARY command APDU,

■ P1 must be $_h$00,

■ P2 is the address of the <u>first page to be read</u> (0 to 15),

■ As the size of every page is 4, <u>Le must be multiple of 4</u> (64 bytes for the full card).

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

■ P1 must be $_h$00,

■ P2 is the address of the <u>first page to be written</u> (2 to 15),

■ As the size of every page is 4, <u>Lc must be 4</u>, exactly.

> Some pages holds <u>lock</u> bits and OTP (one-time-programming) bits.
>
> Please read card's documentation carefully before writing into those pages.

c. Mifare UltraLight C 3-DES authentication

The Mifare UltraLight C supports a new <u>Triple-DES authentication</u> function.

Use the ENCAPSULATE command APDU with P1=$_h$01 (§ 2.2.2) to implement this function.

Tip : SpringCard PC/SC SDK for Windows provides a library (mifulc.dll) that fully implements the Mifare UltraLight C authentication scheme.

5.2.4. Innovision Topaz/Jewel cards (NFC Forum's type 1 tags)

a. READ BINARY (FULL CARD)

In the READ BINARY command APDU,

- P1 must be $_h00$,
- P2 must be $_h00$,
- Le can be $_h00$ or $_h80$.

The whole card content is returned as once.

b. READ BINARY (BYTE LEVEL)

In the READ BINARY command APDU,

- P1 must be $_h00$,
- P2 is the address of the <u>first byte to be read</u> (0 to 127),
- Le can be any length.

Tip : using the READ BINARY (FULL CARD) command is 10 times faster than this BYTE LEVEL version.

c. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h00$,
- P2 is the address of the <u>byte to be written</u> (0 to 127),
- <u>Lc must be 1</u>, exactly.

Some bytes hold <u>lock</u> bits and OTP (one-time-programming) bits.

Please read card's documentation carefully before writing into those bytes.

5.3. WORKING WITH MEMORY CARDS, ISO 14443-B GROUP

5.3.1. ASK CTS256B and CTS512B

These cards are divided into 2-byte areas.

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,

- P2 is the address of the <u>first area to be read</u> (0 to 15 for CTS256B, 0 to 31 for CTS512B),

- As the size of every area is 2, <u>Le must be multiple of 2</u> (32 bytes for the full CTS256B card, 64 bytes for the full CTS512B card).

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,

- P2 is the address of the area to be written,

- As the size of every area is 2, <u>Lc must be 2</u>, exactly.

> ☠ Some areas play a particular role in card's operation.
>
> Please read the card's documentation carefully before writing into those areas.

5.3.2. ST MicroElectronics ST176

These cards are divided into 2-byte blocks.

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>first block to be read</u> (0 to 15),
- As the size of every block is 2, <u>Le must be multiple of 2</u> (32 bytes for the full card).

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the block to be written,
- As the size of every block is 2, <u>Lc must be 2</u>, exactly.

> Some blocks play a particular role in card's operation.
>
> Please read the card's documentation carefully before writing into that block.

5.3.3. Inside Contactless PicoPass, ISO 14443-2 mode

This part applies to chips named either "PicoPass or PicoTag" <u>when the ISO 14443-3 compliance is NOT enabled</u> (see § 5.3.4 in the other case).

Those chips exist in two sizes (2K → 256 B, 16K → 2 kB), and in non-secure (2K, 16K) or secure (2KS, 16KS) versions.

SpringCard PC/SC readers may read/write the non-secure chips only (2K, 16K). The behaviour with the secure chips is undefined.

> Communication with these cards is not reliable. It may be necessary to repeat the READ BINARY or UPDATE BINARY commands twice or three time before getting the expected response.

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>first block to be read</u> (2K: 0 to 31; 16K: 0 to 255),
- As the size of every block is 8, <u>Le must be multiple of 8</u>.

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the block to be written (2K: 0 to 31; 16K: 0 to 255),
- As the size of every block is 8, <u>Lc must be 8</u>, exactly.

> Some blocks play a particular role in card's operation.
>
> Please read the card's documentation carefully before writing into those blocks.

c. Page select

The Inside specific Page select function is not implemented in the reader. Use the ENCAPSULATE command APDU to send it directly to the card.

5.3.4. Inside Contactless PicoPass, ISO 14443-3 mode

This part applies to chips named either "PicoPass or PicoTag" <u>when the ISO 14443-3 compliance IS enabled</u> (see § 5.3.3 in the other case).

Those chips exist in two sizes (2K → 256 B, 16K → 2 kB), and in non-secure (2K, 16K) or secure (2KS, 16KS) versions.

SpringCard PC/SC readers may read/write the non-secure chips only (2K, 16K). The behaviour with the secure chips is undefined.

> Communication with these cards is not reliable. It may be necessary to repeat the READ BINARY or UPDATE BINARY commands twice or three time before getting the expected response.

a. Card activation

The card answers to standard ISO 14443-3 activation commands, but doesn't give any information that may help distinguishing it from another card.

Therefore, a specific activation sequence must be performed by the application : send the SLOT CONTROL command APDU with P1,P2=$_h$20,10 to activate the card (§ 2.3.8).

b. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>first block to be read</u> (2K: 0 to 31; 16K: 0 to 255),
- As the size of every block is 8, <u>Le must be multiple of 8</u>.

c. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the block to be written (2K: 0 to 31; 16K: 0 to 255),
- As the size of every block is 8, <u>Lc must be 8</u>, exactly.

> Some blocks play a particular role in card's operation.
>
> Please read the card's documentation carefully before writing into those blocks.

d. Page select

The Inside specific Page select function is not implemented in the reader. Use the ENCAPSULATE command APDU to send it directly to the card.

5.4. WORKING WITH MEMORY CARDS, ISO 15693 GROUP

RC632 chipset only

5.4.1. "Standard" cards

This applies to NXP ICODE SLI or Texas Instrument TAG-IT typically.

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>first block to be read</u>, please read documentation of your 15693 card to know the actual number of blocks,
- As the size of every is 4, <u>Le must be multiple of 4</u>.

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>block to be written</u>, please read documentation of your 15693 card to know the number of blocks,
- Since the size of each block is 4, <u>Lc must be 4</u>, exactly.

5.4.2. ST MicroElectronics LRI64

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>first block to be read</u> (0 to 14),
- The size of every block is 1. Le may be chosen freely.

b. UPDATE BINARY

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the <u>first block to be written</u> (8 to 14),
- As the size of every block is 1, <u>Lc must be 1</u>, 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.

5.4.3. NXP ICODE1

a. READ BINARY

In the READ BINARY command APDU,

- P1 must be $_h$00,

- P2 is the address of the <u>first block to be read</u> (0 to 15),

- As the size of every block is 4, <u>Le must be multiple of 4</u> (64 bytes for the full card).

b. UPDATE BINARY

The reader is not able to write the NXP ICODE1 cards.

6. SPECIFIC ERROR CODES

When the APDU interpreter returns SW1 = $_h$6F, the value of SW2 maps to one of the reader specific error codes listed below.

SW2	Symbolic name[20]	Meaning
$_h$01	MI_NOTAGERR	No answer received (no card in the field, or card is mute)
$_h$02	MI_CRCERR	CRC error in card's answer
$_h$04	MI_AUTHERR	Card authentication failed
$_h$05	MI_PARITYERR	Parity error in card's answer
$_h$06	MI_CODEERR	Invalid card response opcode
$_h$08	MI_SERNRERR	Card's serial number is invalid
$_h$0A	MI_NOTAUTHERR	Card operation denied, must be authenticated first
$_h$0B	MI_BITCOUNTERR	Wrong number of bits in card's answer
$_h$0C	MI_BYTECOUNTERR	Wrong number of bytes in card's answer
$_h$0E	MI_TRANSERR	Card transaction error (NACK received)
$_h$0F	MI_WRITEERR	Card write error
$_h$12	MI_READERR	Card read error
$_h$13	MI_OVFLERR	RC FIFO overflow
$_h$15	MI_FRAMINGERR	Framing error in card's answer
$_h$16	MI_ACCESSERR	Card access error (NACK received)
$_h$17	MI_UNKNOWN_COMMAND	RC unknown opcode
$_h$18	MI_COLLERR	A collision between 2 (or more) cards has occured
$_h$1A	MI_INITERR	RC hardware failure
$_h$1B	MI_INTERFACEERR	Wrong status on RC interface
$_h$1C	MI_ACCESSTIMEOUT	Timeout on RC interface
$_h$1D	MI_NOBITWISEANTICOLL	2 (or more) cards have been found, but at least 1 does not support anticollision
$_h$34	MI_CASCLEVEX	Card serial number longer than specified
$_h$36	MI_BAUDRATE_NOT_SUPPORTED	RC doesn't support this baudrate
$_h$3C	MI_WRONG_PARAMETER_VALUE	RC doesn't support this value
$_h$3D	TCL_NOTAGERR	A card has been found, but isn't T=CL compliant
$_h$47	TCL_CID_NOT_ACTIVE	The specified CID isn't assigned yet
$_h$4B	TCL_ATSLEN	Invalid length for card's ATS
$_h$4D	TCL_ATS_ERROR	Invalid data in card's ATS
$_h$4E	TCL_FATAL_PROTOCOL	Card's response doesn't meet T=CL specifications
$_h$4F	TCL_RECBUF_OVERFLOW	Card's response is longer than our receive buffer
$_h$52	TCL_TRANSMERR_NOTAG	Card not responding anymore (maybe it has been removed)
$_h$53	TCL_BAUDRATE_NOT _SUPPORTED_PICC	Card doesn't support the specified baudrate
$_h$57	TCL_PPS_FORMAT	Card's answer to our PPS request is invalid
$_h$58	TCL_PPS_ERROR	Card didn't accepted our PPS request

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

$_h63$	MI_BREAK	Command execution has been interrupted by another input
$_h64$	MI_NY_IMPLEMENTED	Unknown or un-implemented command
$_h6C$	TCL_CID_ACTIVE	The specified CID is already in use
$_h6E$	MI_WRONG_ADDR	Invalid address (sector or block number)
$_h70$	MI_RECBUF_OVERFLOW	Internal buffer overflow
$_h7B$	MI_WRONG_VALUE	An invalid value has been specified in command input
$_h7D$	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

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