SPRINGCARD PC/SC READERS - CSB6 GROUP

APDU interpreter and vendor-specific commands

Document Identification

Category	Developer's manual		
Family/Customer	PC/SC readers		
Reference	PMD841P	Version	FA
Status	draft	Classification	Public
Keywords	CSB6, PC/SC, contactless cards, RFID labels, NFC tags		
Abstract	Most contactless cards are not actually smartcards by wired-logic memory cards. The SpringCard PC/SC readers embed an APDU interpreter that makes it easy to work with contactless memory cards, RFID labels, NFC tags, as if they were smartcards. Also a few vendor-specific commands give access to reader's LED, buzzer, runtime configuration either through SCardTransmit or SCardControl functions. This document details all these features.		

File name	V:\Dossiers\notices\CSB6 Group\Developpement et integration\[PMD841P-FA] CSB6 PCSC APDU interpreter and specific commands.odt		
Date saved	29/08/12	Date printed	15/06/11

Revision History

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
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)
DB	14/12/10	JDA				Corrected bogus INS for General Authenticate (firmware >= 1.55) Listed new features of firmware >= 1.55 (GET DATA and SLOT CONTROL) Added paragraph 3.2, added explanation of MS' CCID driver behavior Added new values for P1,P2 in GET DATA APDU Fixed a few typos
EA	15/06/11	JDA				Change in title + a few comestic changes : this document now targets every SpringCard PC/SC readers (CSB7, CSB8, CSB9 families) and not only the CSB6 family as before.
EB	26/07/11	JDA				Added chapter 2 "getting started with PC/SC", chapter 6.2 "contactless smart cards"
EC	24/11/11	JDA				READ BINARY and UPDATE BINARY add support NFC Forum Type 2 tags with sector select feature (firmware >= 1.62) Added information regarding communication speeds in GET DATA (firmware >= 1.62)
ED	08/02/12	JDA				Added support for Kovio RF barcode and non-standard ISO 15693 tags (from ST) having the address on 2 bytes (firmware >= 1.64)
FA	29/08/12	JDA				Documented new behaviour of firmware >= 1.70 (MIFARE CLASSIC VALUE, key selector in MIFARE CLASSIC READ / WRITE) Detailed ISO 15693 commands Added the glossary Renumbering of chapter 3

Contents

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 PC/SC Readers** comply with this standard. This makes those products usable on most operating systems, using an high-level and standardized API.

Contactless microprocessor-based smartcards do comply with the ISO 7816-4 standard. This means that you only have to use the *SCardTransmit* function to exchange APDUs with the card, and it makes no difference whether the underlying layer is “contact” (ISO 7816-3 T=0 or T=1 as transport protocol) or “contactless” (using ISO 14443-4 “T=CL” as transport protocol).

Anyway, a lot of contactless cards are not actually “smartcards” because thery are not ISO 7816-4 compliant, and therefore they are not natively supported by the system's PC/SC stack. This is the case of

- **Wired-logic memory cards** (Mifare, CTS, SR... families),
- **RFID labels** (ISO 15693, ICODE, TagIT... families),
- **NFC tags** (type 1, type 2, type 3),
- Even some proprietary microprocessor cards that use a specific communication protocol (Desfire EV0...).

The role of the **embedded APDU interpreter** is to make the PC/SC stack and the application work with for those cards as if they were smartcards.

Also, some actions are to be performed on the reader itself, and not onto the card: driving LEDs or buzzer, getting reader’s serial number... **Vendor specific commands** that could be sent to the reader through *SCardControl* (or withing a custom APDU through *ScardTransmit*) are designed to address this need.

This document is the reference manual, both for the embedded APDU interpreter and the vendor specific commands.

1.2. Supported products

At the date of writing, this document refers to all SpringCard PC/SC Readers featuring an USB interface:

- CSB6 group: CSB6, Prox'N'Roll PC/SC, CrazyWriter, EasyFinger, TagPad (starting with firmware release 1.47),
- H663 group *(planned)*: H663, CSB-HSP, CrazyWriter-HSP,
- NFC Roll and H512 *(planned)* when running in reader mode.

Note that not all products support all the feature described in this document. Please review the datasheet of the product(s) you're working with, 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 and a basic knowledge of PC/SC.

To get started with PC/SC, please read our Introduction to PC/SC development and simplified documentation of the API, available online at

http://www.springcard.com/download/find.php?file=pmdz061

1.4. Support and updates

Useful 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 are 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 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. Glossary – useful terms

The following list contains the terms that are directly related to the subject of this document. This is an excerp from our technical glossary, available online at:

http://www.springcard.com/blog/technical-glossary/

- **ICC:** *integrated-circuit card*. This is the standard name for a plastic card holding a silicon chip (an integrated circuit) compliant with the ISO 7816 standards. A common name is *smartcard*.
- **CD:** *coupling device* or **coupler**. A device able to communicate with an ICC. This is what everybody calls a *smartcard reader.* Technically speaking it could be seen as a gateway between the computer and the card.
- **Microprocessor-based card:** an ICC (or a PICC) whose chip is a small computer. This is the case of high-end cards used in payment, transport, eID/passports, access control... Key features are security, ability to store a large amount of data and to run an application inside the chip. Most of the time they implement the command set defined by ISO 7816-4.
- **Memory card** or **wired logic card:** an ICC (or a PICC, or a VICC) whose chip is only able to store some data, and features a limited security scheme (or no security scheme at all). They are cheaper than microprocessor-based cards and therefore are widely used for RFID traceability, loyalty, access control...
- **PICC:** *proximity integrated-circuit card*. This is the standard name for any contactless card compliant with the ISO 14443 standards (proximity: less than 10cm). This could either be a smartcard or a memory card, or also any NFC object running in card emulation mode. Common names are *contactless card*, or *RFID card, NFC tag***.**
- **PCD:** *proximity coupling device.* A device able to communicate with a PICC, i.e. a contactless reader compliant with ISO 14443.
- **VICC:** *vicinity integrated circuit card*. This is the standard name for any contactless card compliant with the ISO 15693 standards (vicinity: less than 150cm). Common names are *RFID tag*, *RFID label*.
- **VCD:** *vicinity coupling device.* A device able to communicate with a VICC, i.e. a contactless reader compliant with ISO 15693.

- **RFID:** *radio-frequency identification*. This is the general name for any system using radio waves for M2M communication (machine to machine, in our case PCD/VCD to PICC/VICC).
- **NFC:** *near-field communication*. A subset of RFID, where the operating distance is much shorter than the wavelength of the radio waves involved. This is the case for both ISO 14443 and ISO 15693: the carrier frequency is 13.56MHz, leading to a wavelength of 22m. The proximity and vicinity ranges are shorter than this wavelength.
- **NFC Forum:** an international association that aims to standardize the applications of NFC in the 13.56MHz range. Their main contribution is the **NFC Tags**, which are nothing more than PICCs which data are formatted according to their specifications, so the information they contain is understandable by any compliant application.
- **ISO 7816-1** and **ISO 7816-2:** This international standard defines the hardware characteristics of the ICC. The standard smartcard format (86x54mm) is called **ID-1**. A smaller form-factor is used for SIM cards (used in mobile phone) or SAM (secure authentication module, used for payment or transport applications) and is called **ID-000**.
- **ISO 7816-3:** This international standard defines two communication protocols for ICCs: T=0 and T=1. A compliant reader must support both of them.
- **ISO 7816-4:** This international standard defines both a communication scheme and a command set. The communication scheme is made of APDUs. The command set assumes that the card is structured the same way as a computer disk drive: directories and files could be selected (SELECT instruction) and accessed for reading or writing (READ BINARY, UPDATE BINARY instructions). More than 40 instructions are defined by the standard, but most cards implement only a small subset, and often add their own (vendor-specific) instructions.
- **APDU:** *application protocol datagram unit*. These are the frames that are exchanged at application-level between an application running on the computer and a smartcard. The format of those frames is defined by ISO 7816-4 and checked by the system's PC/SC stack. The command (application to card) is called a C-APDU, the response (card to application) a R-APDU. Note that this is a request/response scheme: the smartcard has no way to send something to the application unless the application asks for it.
- **ISO 14443:** This international standard defines the PCD/PICC communication scheme. It is divided into 4 layers:
  1. Defines the hardware characteristics of the PICC,
  2. Defines the carrier frequency and the bit-level communication scheme,
  3. Defines the frame-level communication scheme and the session opening sequence (anti-collision),
  4. Defines the transport-level communication scheme (sometimes called “T=CL”).

  The application-level is out of the scope of ISO 14443. Most microprocessor-based PICCs implement ISO 7816-4 on top of ISO 14443-4.

A lot of wired logic PICCs (NXP Mifare family, ST MicroElectronics ST/SR families, to name a few) implements only a subset of ISO 14443, and have their own set of functions on top of either ISO 14443-2 or ISO 14443-3.

Note that ISO 14443-2 and ISO 14443-3 are divided into 2 protocols called 'A' and 'B'. A PCD shall implement both, but the PICCs implement only one of them[1]. Four communication baudrates are possible: 106 kbit/s is mandatory, higher baudrates (212, 424 or 848 kbit/s) are optional.

- **ISO 15693:** This international standard defines the VCD/VICC communication scheme. It is divided into 3 layers:

  1. Defines the hardware characteristics of the VICC,
  2. Defines the carrier frequency and the bit-level communication scheme,
  3. Defines the frame-level communication scheme, the session opening sequence (anti-collision/inventory), and the command set of the VICC.

  All VICCs are memory chips. Their data storage area is divided into blocks. The size of the blocks and the number of them depend on the VICC.

  Note that ISO 18000-3 mode 1 is the same as ISO 15693[2].

- **ISO 18092 or NFCIP-1:** This international standard defines a communication scheme (most of the time refered as “peer to peer mode”) where two peer “objects” are able to communicate together (and not only a PCD and a PICC). The underlying protocol is ISO 14443-A at 106 kbit/s and the Sony Felica protocol at 212 and 424 kbit/s. The **SpringCard PC/SC Readers** depicted in this document do not provide this feature.

- **ISO 21481 or NFCIP-2:** This international standard defines how an NFC object shall be able to emulate an ISO 14443 PICC (and maybe an ISO 15693 VICC). When NFC objects run in this “card emulation mode”, the **SpringCard PC/SC Readers** are fully able to communicate with them.

- **Mifare:** This trademark of NXP (formerly Philips Semiconductors) is the generic brand name of their PICC products. Billions of **Mifare Classic** cards have been deployed since the 90's. This is a family of wired-logic PICCs were data storage is divided into sectors and protected by a proprietary[3] stream cipher called **CRYPTO1**. Every sector is protected by 2 access keys called “key A” and “key B”[4]. NXP also offers another family of wired-logic PICCs called **Mifare UltraLight** (adopted by NFC Forum as Type 2 NFC Tags). **Mifare SmartMX** (and former Pro/ProX) is a family of microprocessor-based PICCs that may run virtually any smartcard application, typically on top a JavaCard operating system. **Mifare Desfire** is a particular microprocessor-based PICC that runs a single general-purpose application.

[1] Yet some NFC objects may emulate both an ISO 14443-A and an ISO 14443-B card.

[2] ISO 15693 has been written by the workgroup in charge of smartcards, and then copied by the workgroup in charge of RFID into ISO 18000, the large family of RFID standards.

[3] And totally broken. Do not rely on this scheme in security-sensitive applications!

[4] A typical formating would define key A as the key for reading, and key B as the key for reading+writing.

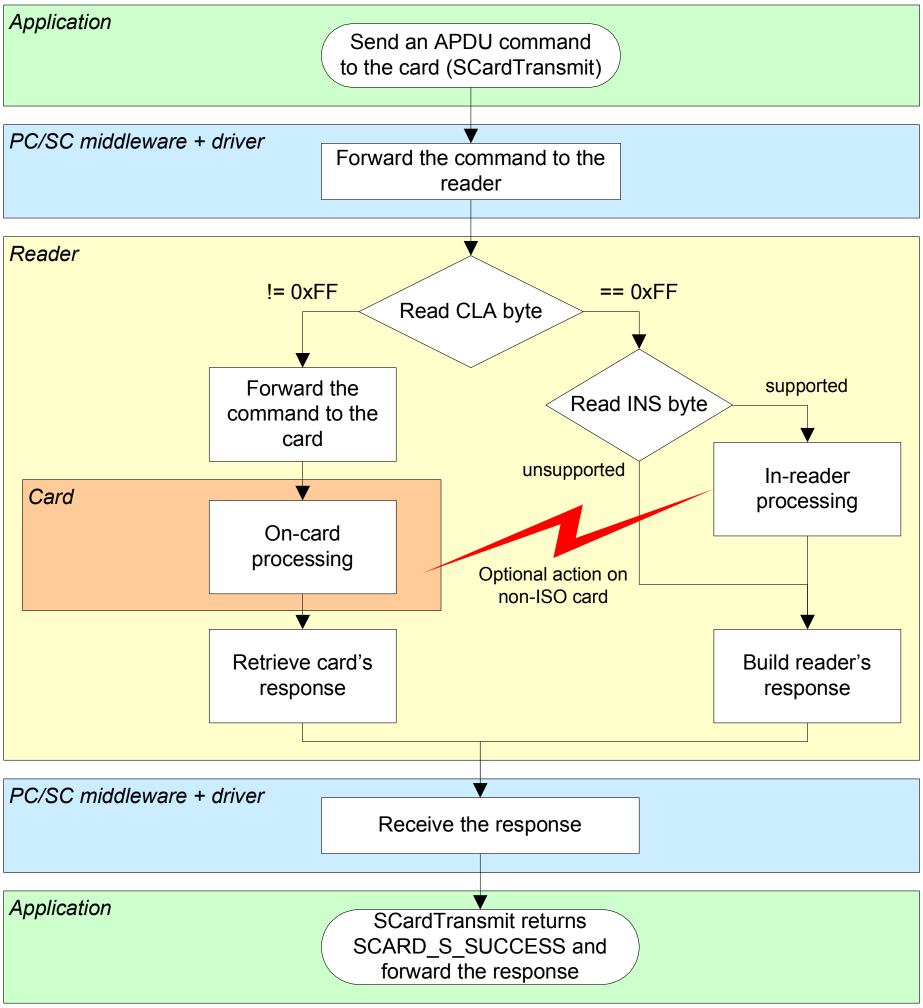
2. Embedded APDU interpreter

2.1. Basis

In PC/SC architecture, the ***SCardTransmit*** function implements the dialog between an application and a smartcard, through a "passive" gateway, the reader. The reader only transmits frames in both directions, without any specific processing. The dialog follows the ISO 7816-4 APDU rules:

- Application to smartcard **C-APDU** is *CLA, INS, P1, P2, Data In (optional)*
- Smartcard to application **R-APDU** is *Data Out (optional), SW1, SW2*

In order to work with non ISO 7816-4 cards as if they were smartcards, the embedded APDU interpreter obey to the same rules, offering its own list of instructions under the reserved class **CLA=$_h$FF**. It is therefore available through regular *ScardTransmit* calls.


Application
Send an APDU command to the card (SCardTransmit)
PC/SC middleware + driver
Forward the command to the reader
Reader
!= 0xFF
Read CLA byte
== 0xFF
Forward the command to the card
Read INS byte
supported
unsupported
In-reader processing
Card
On-card processing
Optional action on non-ISO card
Retrieve card's response
Build reader's response
PC/SC middleware + driver
Receive the response
Application
SCardTransmit returns SCARD_S_SUCCESS and forward the response

2.1.1. CLA byte of the embedded APDU interpreter

Default class is hFF. This means that every APDU starting with CLA= hFF 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 hB2 of reader’s non volatile memory (see § 3.5.3).

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

b. Disabling the embedded APDU interpreter

Define CLA byte = h00 (register hB2= h00, see § 3.5.3) to disable the embedded APDU interpreter.

2.1.2. Status words returned by the embedded APDU interpreter

SW1	SW2	Meaning
h90	h00	Success
h67	h00	Wrong length (Lc incoherent with Data In)
h68	h00	CLA byte is not correct
h6A	h81	Function not supported (INS byte is not correct), or not available for the selected card
h6B	h00	Wrong parameter P1-P2
h6F	h01	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. Embedded APDU interpreter instruction list

Instruction	INS	Contactless	Contact	Notes (see below)
LOAD KEY	$_h$82	✓		C
GENERAL AUTHENTICATE	$_h$86	✓		C
READ BINARY	$_h$B0	✓		A
ENVELOPE	$_h$C2	✓		B
GET DATA	$_h$CA	✓	✓	C
UPDATE BINARY	$_h$D6	✓		A
READER CONTROL	$_h$F0	✓	✓	D
RC CONTROL	$_h$F1	✓		D
GEMCORE CONTROL	$_h$F1		✓	D
MIFARE CLASSIC READ	$_h$F3	✓		D
MIFARE CLASSIC WRITE	$_h$F4	✓		D
MIFARE CLASSIC VALUE	$_h$F5	✓		D
CONTACTLESS SLOT CONTROL	$_h$FB	✓		D
CONFIGURE CALYPSO SAM	$_h$FC		✓	D
TEST	$_h$FD	✓	✓	D
ENCAPSULATE	$_h$FE	✓	✓	D

Notes:

A Function fully implemented according to PC/SC standard

B Function implemented according to PC/SC standard, but some feature are not supported

C Function implemented according to PC/SC standard, but also provides vendor-specific options

D Vendor-specific function

2.2. PC/SC standard instructions for the contactless slot

2.2.1. GET DATA instruction

The **GET DATA** instruction retrieves information regarding the inserted card. It can be used with any kind of PICC/VICC, 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	Action	Fw
Standard PC/SC-defined values			
$_h$00	$_h$00	Serial number of the PICC/VICC - 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	≥ 1.51
SpringCard specific values			
$_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	≥ 1.51
$_h$F0	$_h$00	Complete identifier of the PICC/VICC: - ISO 14443-A : ATQA (2 bytes) + SAK (1 byte) + UID - ISO 14443-B : complete ATQB (11 or 12 bytes)[5] - ISO 15693 : answer to GET SYSTEM INFORMATION command[6] - Innovatron : REPGEN - others: see chapter 5 for details	≥ 1.52
$_h$F1	$_h$00	Type of the PICC/VICC, 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	≥ 1.52

[5] SpringCard PC/SC Readers are ready to support the extended ATQB (12 bytes), but since a lot of PICC currently in circulation don't reply to the REQB command with the 'extended' bit set, this feature is not enabled by default.

[6] If the card doesn't support the GET SYSTEM INFORMATION COMMAND, a valid SYSTEM INFORMATION value is constructed, including the UID and the DSFID byte.

P1	P2	Action	Fw
$_h$F1	$_h$01	NFC Forum Tag[7] support: - $_h$01 if the PICC is recognized as a NFC Forum Type 1 Tag - $_h$02 if the PICC is recognized as a NFC Forum Type 2 Tag - $_h$00 otherwise	≥ 1.62
$_h$F2	$_h$00	"Short" serial number of the PICC/VICC - ISO 14443-A : UID truncated to 4 bytes, in "classical" order - others: same as P1,P2=$_h$00,$_h$00	≥ 1.52
$_h$FA	$_h$00	Card's ATR	≥ 1.53
$_h$FC	$_h$00	PICC/PCD communication speeds on 2 bytes (DSI, DRI)	≥ 1.62
$_h$FF	$_h$00	Reader's serial number (4-byte UID of the NXP RC chipset)	≥ 1.52
$_h$FF	$_h$01	Reader's hardware identifier (5-byte HWID of the NXP RC chipset)	≥ 1.55
$_h$FF	$_h$81	Vendor name in ASCII ("SpringCard")	≥ 1.55
$_h$FF	$_h$82	Product name in ASCII	≥ 1.55
$_h$FF	$_h$83	Product serial number in ASCII	≥ 1.55
$_h$FF	$_h$84	Product USB identifier (VID/PID) in ASCII	≥ 1.55
$_h$FF	$_h$85	Product version ("x.xx") in ASCII	≥ 1.55

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)

[7] Please refer to NFC Forum's specifications for details. Note that Type 4 Tags are 'standard' contactless smartcards; it is up to the application level to send the proper SELECT APPLICATION to recognize them. Type 3 Tags (Felica) are not supported by this hardware.

2.2.2. LOAD KEY instruction

The **LOAD KEY** instruction loads a 6-byte Mifare Classic access key (CRYPTO1) into 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 bytes (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 (secure E2PROM inside the RC chipset, if available[8])

LOAD KEY command parameter P2 (key index)

When P1 = $_h$00, P2 is the identifier of the key into reader's volatile memory. The memory has the capacity to store up to 4 keys of each type (A or B).

P2 = $_h$00 to P2 = $_h$03 are "type A" keys,

P2 = $_h$10 to P2 = $_h$13 are "type B" keys.

When P1 = $_h$20, P2 is the identifier of the key into the reader's non-volatile memory (if available). This memory can store up to 16 keys of each type (A or B).

P2 = $_h$00 to P2 = $_h$0F are "type A" keys,

P2 = $_h$10 to P2 = $_h$1F are "type B" keys.

Note there's no way to readback the keys stored in either volatile or non-volatile memory.

LOAD KEY response

SW1	SW2
See below	

[8] This feature is available on the CSB6 and H663 groups, but not on the CSB7 and H512 groups

LOAD KEY status word

SW1	SW2	Meaning
h90	h00	Success
h69	h86	Volatile memory is not available
h69	h87	Non-volatile memory is not available
h69	h88	Key index (P2) is not in the allowed range
h69	h89	Key length (Lc) is not valid

2.2.3. GENERAL AUTHENTICATE instruction

The **GENERAL AUTHENTICATE** instruction performs a Mifare Classic authentication (CRYPTO1). The application must provide the index of the key to be used; this key must have been loaded into the reader through a previous LOAD KEY instruction.

Do not invoke this function if the currently activated PICC/VICC is not a Mifare Classic!

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

The ***block number*** (byte 2) is the address on the card, where we try to be authenticated *(note: this is the block number, <u>not the sector</u> number)*.

The ***key location or Key type*** (byte 3) must be either:

- $_{h}$60 for authentication using a CRYPTO1 "A" key *(standard PC/SC-defined value)*,
- $_{h}$61 for authentication using a CRYPTO1 "B" key *(standard PC/SC-defined value),*
- Same value as the P1 parameter used in the LOAD KEY instruction: $_{h}$00 or $_{h}$20 *(SpringCard specific value)*.

The ***key index*** (byte 4) is defined as follow:

- If *key type* (byte 3) is $_{h}$60, use values $_{h}$00 to $_{h}$03 to select one of the "A" keys stored in reader's volatile memory, and values $_{h}$20 to $_{h}$2F to select one of the "A" keys stored in reader's non-volatile memory (if available),
- If *key type* (byte 3) is $_{h}$61, use values $_{h}$00 to $_{h}$03 to select one of the "B" keys stored in reader's volatile memory, and values $_{h}$20 to $_{h}$2F to select one of the "B" keys stored in reader's non-volatile memory (if available),
- If *key type* (byte 3) is either $_{h}$00 or $_{h}$20 (same value as the P1 parameter used in the LOAD key instruction), choose one of the values allowed for the P2 parameter in the same LOAD key instruction *(SpringCard specific value).*

GENERAL AUTHENTICATE response

SW1	SW2
See below	

GENERAL AUTHENTICATE status word

SW1	SW2	Meaning
h90	h00	Success
h69	h82	CRYPTO1 authentication failed
h69	h86	Key location or type (byte 3) is not valid (or not available for this reader)
h69	h88	Key index (byte 4) is not in the allowed range

2.2.4. READ BINARY instruction

The **READ BINARY** instruction retrieves data from a memory card (wired-logic PICC or VICC). Refer to chapter 5 for a details.

> *For any PICC/VICC but Mifare Classic, this instruction is executed without any prerequisite.*
> *For Mifare Classic, the reader must have been authenticated by the card on a target sector, before being able to read the sector's data. Your application must always invoke GENERAL AUTHENTICATE instruction (with a valid key A or key B for the sector) before invoking the READ BINARY instruction. Using the MIFARE CLASSIC READ instruction instead (§ 2.3.1) could be easier and may shorten the transaction time.*

READ BINARY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$B0	Address MSB	Address LSB	-	-	XX

P1 and P2 form the **address** that will be sent to the PICC/VICC in its specific read command. Most PICC/VICC are divided into small blocks (sometimes called pages). The address is a block number, and not to an absolute byte offset in memory.

Both the allowed range for the **address** and the value for **Le** depend on the capabilities of the PICC/VICC. Please always refer to its datasheet for details. Note that Le = $_h$00 should always work, provided that the address is valid.

> *For Mifare Classic, P1,P2 is the address of the block ($_h$0000 to $_h$00FF), but remember that the authentication is made on a per-sector basis. A new authentication must be performed everytime you have to access another sector.*
> *For a NFC Forum-compliant Type 2 NFC Tag, P2 is the block number, and P1 the sector number if the PICC does support this feature. Set P1 to $_h$00 if it is not the case.*

READ BINARY response

Data Out	SW1	SW2
XX … XX	See below	

READ BINARY status word

SW1	SW2	Will return in Data Out
$_h$90	$_h$00	Success
$_h$62	$_h$82	End of data reached before Le bytes (Le is greater than data length)
$_h$69	$_h$81	Command incompatible
$_h$69	$_h$82	Security status not satisfied
$_h$6A	$_h$82	Wrong address (no such block or no such offset in the card)
$_h$6C	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

2.2.5. UPDATE BINARY instruction

The **UPDATE BINARY** instruction writes data into a memory card (wired-logic PICC or VICC). Refer to chapter 5 for details.

> *For any PICC/VICC but Mifare Classic, this instruction is executed without any prerequisite.*
> *For Mifare Classic, the reader must have been authenticated by the card on a target sector, before being able to write the sector's data. Your application must always invoke GENERAL AUTHENTICATE instruction (with a valid key A or key B for the sector) before invoking the UPDATE BINARY instruction. Using the MIFARE CLASSIC WRITE instruction instead (§ 2.3.2.) could be easier and may shorten the transaction time.*

UPDATE BINARY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$D6	Address MSB	Address LSB	XX	Data	-

P1 and P2 form the **address** that will be sent to the PICC/VICC in its specific write command. Most PICC/VICC are divided into small blocks (sometimes called pages). The address is a block number, and not to an absolute byte offset in memory.

Both the allowed range for the **address** and the value for **Lc** depend on the capabilities of the PICC/VICC. Please always refer to its datasheet for details.

> *For Mifare Classic, P1,P2 is the address of the block ($_h$0000 to $_h$00FF), but remember that the authentication is made on a per-sector basis. A new authentication must be performed everytime you have to access another sector. Lc must be $_h$10 (a block is 16-B long).*
> *For a NFC Forum-compliant Type 2 NFC Tag, P2 is the block number, and P1 the sector number if the PICC does support this feature. Set P1 to $_h$00 if it is not the case. Lc must be $_h$04 (a block is 4-B long).*

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)

Important disclaimer

Most PICC/VICC have specific areas that may be written ***only once*** *(OTP: one time programming or fuse bits), and/or that must be written* ***carefully*** *because they are involved in the security scheme of the card (lock bits), and/or because writing a invalid value will make the card unusable (sector trailer of a Mifare Classic for instance).*

Before invoking UPDATE BINARY, always double check where you're writing, and for the sensitive addresses, what you're writing!

2.3. Vendor specific instructions for the contactless slot

2.3.1. MIFARE CLASSIC READ instruction

The **MIFARE CLASSIC READ** instruction retrieves data from a Mifare Classic PICC (e.g. Mifare 1K or Mifare 4K, or Mifare Plus in level 1).

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.

Do not invoke this function if the currently activated PICC/VICC is not a Mifare Classic!

a. *MIFARE CLASSIC READ using reader's keys*

In this mode, the application doesn't specify anything. The reader tries every key he knows (both permanent keys in E2PROM and temporary keys previously loaded in volatile memory – use LOAD KEY to do so) until one succeeds.

> *Since the reader must try all the keys, this method may take up to 1000ms. The ordering of the keys in reader's memory is very important to speed-up the process: the upper the right key is in the reader's memory, the sooner the authentication will succeed.*
> *Note the the reader tries all "type A" keys first, and only afterwards all the "type B" keys. This behaviour has been chosen because in 95% of Mifare applications, the "type A" key is the preferred key for reading (where the "type B" key is used for writing).*

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 (§ 2.2.4) for response and status words.

b. *MIFARE CLASSIC READ selecting a key in the reader*

In this mode, the application chooses one the key previously loaded in the reader through the LOAD KEY instruction.

MIFARE CLASSIC READ command APDU, selecting a key

CLA	INS	P1	P2	Lc	Data In		Le
hFF	hF3	h00	Block Number	h02	Key Location or Type	Key Index	XX

The understanding and values for bytes ***Key location or Key type*** and ***Key index*** are documented in § 2.2.3 (GENERAL AUTHENTICATE instruction).

Refer to the READ BINARY instruction (§ 2.2.4) for response and status words.

c. *MIFARE CLASSIC READ with specified key*

In this mode, the application provides the 6-B value of the key to the reader.

The reader tries the key as a “type A” first, and only afterwards as a “type B”.

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 instruction (§ 2.2.4) for response and status words.

2.3.2. MIFARE CLASSIC WRITE instruction

The **MIFARE CLASSIC WRITE** instruction writes data into a Mifare Classic PICC (e.g. Mifare 1K or Mifare 4K, or Mifare Plus in level 1).

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.

Do not invoke this function if the currently activated PICC/VICC is not a Mifare Classic!

Important disclaimer

> *Writing sector trailers (security blocks) is possible as long as the sector's current access condition allows it, but Mifare sector trailers have to follow a specific formatting rule (mix-up of the access conditions bits) to be valid. Otherwise, the sector becomes permanently unusable.*
> *Before invoking MIFARE CLASIC WRITE, always double check that you're not writing a sector trailer, and if you really have to do so, make sure the new content is formatted as specified in the datasheet of the PICC.*

a. MIFARE CLASSIC WRITE using reader’s keys

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

> *Since the reader must try all the keys, this method may take up to 1000ms. The ordering of the keys in reader's memory is very important to speed-up the process: the upper the right key is in the reader's memory, the sooner the authentication will succeed.*
> *Note the the reader tries all “type B” keys first, and only afterwards all the “type A” keys. This behaviour has been chosen because in 95% of Mifare applications, the “type B” key is the preferred key for writing[9].*

[9] 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 UPDATE BINARY instruction (§ 2.2.5) for response and status words.

b. MIFARE CLASSIC WRITE selecting a key in the reader

In this mode, the application chooses one the key previously loaded in the reader through the LOAD KEY instruction.

MIFARE CLASSIC WRITE command APDU, selecting a key

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF4	h00	Block Number	XX	See below	-

MIFARE CLASSIC WRITE command APDU, selecting a key: Data In bytes

Bytes 0 to Lc-3	Byte Lc-2	Byte Lc-1
Data to be written (multiple of 16 bytes)	Key Location or Type	Key Index

The understanding and values for bytes ***Key location or Key type*** and ***Key index*** are documented in § 2.2.3 (GENERAL AUTHENTICATE instruction).

Refer to the UPDATE BINARY instruction (§ 2.2.5) for response and status words.

c. MIFARE CLASSIC WRITE with specified key

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

> *The reader tries the key as a "type B" first, and only afterwards as a "type A".*

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, with specified key: Data In Bytes

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 UPDATE BINARY instruction (§ 2.2.5) for response and status words.

2.3.3. MIFARE CLASSIC VALUE instruction

Firmware >= 1.70

The **MIFARE CLASSIC VALUE** instruction makes it possible to invoke the DECREMENT, INCREMENT, and RESTORE functions of a Mifare Classic PICC (e.g. Mifare 1K or Mifare 4K, or Mifare Plus in level 1), followed by a TRANSFER function.

The DECREMENT, INCREMENT, RESTORE (and TRANSFER) functions could be performed only on the blocks that have been formatted as VALUE block in the sector trailer (access condition bits). Do not invoke this function on DATA blocks, and do not invoke this function if the currently activated PICC/VICC is not a Mifare Classic!

MIFARE CLASSIC VALUE opcodes, operand, and transfer address

The P1 parameter in the MIFARE CLASSIC VALUE command APDU in the PICCs' operation code *(opcode)*, as defined in Mifare Classic specification. Allowed values are:

- $_h$C1 for INCREMENT
- $_h$C0 for DECREMENT
- $_h$C2 for RESTORE

All three operations requires an operand. The operand is a 4-byte signed integer.

- INCREMENT operation: the operand must be > 0 (between $_h$00000001 and $_h$7FFFFFFF). The operand is added to the current value of the source block, and the result is kept by the PICC in a register,
- DECREMENT operation: the operand must be > 0 (between $_h$00000001 and $_h$7FFFFFFF). The operand is substracted from the current value of the source block, and the result is kept by the PICC in a register,
- RESTORE operation: the operand must be 0 ($_h$00000000). The PICC copies the current value of the source block into a register.

After the INCREMENT, DECREMENT or RESTORE operation has been performed by the PICC, the reader invokes the TRANSFER operation: the value of the register is written into a target block.

- If the destination block number is not the same as the source block number, the original value remains unchanged in the source block (this is a sort of 'backup' feature),
- If the destination block number is the same as the source block number, or not destination block number is defined, then the source block is overwritten with the new value.

a. MIFARE CLASSIC VALUE using reader's keys

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

> *Since the reader must try all the keys, this method may take up to 1000ms. The ordering of the keys in reader's memory is very important to speed-up the process: the upper the right key is in the reader's memory, the sooner the authentication will succeed.*
> *For DECREMENT and RESTORE operations, the reader tries all "type A" keys first, and only afterwards all the "type B" keys.*
> *For INCREMENT operation, the reader tries all "type B" keys first, and only afterwards all the "type A" keys.*

The destination block could optionnaly be specified at the end of the command APDU. If not, the source block is overwritten by the TRANSFER operation.

MIFARE CLASSIC VALUE command APDU, using reader's key, without backup

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF5	Opcode	Source block	h04	Operand (4B – MSB first)	-

MIFARE CLASSIC VALUE command APDU, using reader's key, with backup

CLA	INS	P1	P2	Lc	Data In		Le
hFF	hF5	Opcode	Source block	h05	Operand (4B – MSB first)	Dest. block	-

Refer to the UPDATE BINARY instruction (§ 2.2.5) for response and status words.

b. MIFARE CLASSIC VALUE selecting a key in the reader

In this mode, the application chooses one the key previously loaded in the reader through the LOAD KEY instruction.

The destination block could optionnaly be specified at the end of the command APDU. If not, the source block is overwritten by the TRANSFER operation.

MIFARE CLASSIC VALUE command APDU, selecting a key, without backup

CLA	INS	P1	P2	Lc	Data In			Le
hFF	hF5	Opcode	Source block	h06	Operand (4B – MSB first)	Key location or Type	Key index	-

MIFARE CLASSIC VALUE command APDU, selecting a key, with backup

CLA	INS	P1	P2	Lc	Data In				Le
$_{h}$FF	$_{h}$F5	Opcode	Source block	$_{h}$07	Operand (4B – MSB first)	Key location or Type	Key index	Dest. block	-

The understanding and values for bytes ***Key location or Key type*** and ***Key index*** are documented in § 2.2.3 (GENERAL AUTHENTICATE instruction).

Refer to the UPDATE BINARY instruction (§ 2.2.5) for response and status words.

c. MIFARE CLASSIC VALUE with specified key

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

> *For DECREMENT and RESTORE operations, the reader tries the key as a "type A" first, and only afterwards as a "type B".*
> *For INCREMENT operation, the reader tries the key as a "type B" first, and only afterwards as a "type A".*

The destination block could optionnaly be specified at the end of the command APDU. If not, the source block is overwritten by the TRANSFER operation.

MIFARE CLASSIC VALUE command APDU, key specified, without backup

CLA	INS	P1	P2	Lc	Data In		Le
$_{h}$FF	$_{h}$F5	Opcode	Source block	$_{h}$0A	Operand (4B – MSB first)	Key value (6B)	-

MIFARE CLASSIC VALUE command APDU, key specified, with backup

CLA	INS	P1	P2	Lc	Data In			Le
$_{h}$FF	$_{h}$F5	Opcode	Source block	$_{h}$0B	Operand (4B – MSB first)	Key value (6B)	Dest. block	-

Refer to the UPDATE BINARY instruction (§ 2.2.5) for response and status words.

2.3.4. CONTACTLESS SLOT CONTROL instruction

The **CONTACTLESS SLOT CONTROL** instruction allows pausing and resuming the card tracking mechanism of the contactless slot.

This is useful because card tracking implies sending commands to the PICC/VICC periodically (and watch-out its answer). Such 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	Fw
$_h$00	$_h$00	Resume the card tracking mechanism	≥ 1.52
$_h$01	$_h$00	Suspend the card tracking mechanism	≥ 1.52
$_h$10	$_h$00	Stop the RF field	≥ 1.52
$_h$10	$_h$01	Start the RF field	≥ 1.52
$_h$10	$_h$02	Reset the RF field (10ms pause)	≥ 1.52
$_h$20	$_h$00	T=CL de-activation (DESELECT[10])	≥ 1.53
$_h$20	$_h$01	T=CL activation of ISO 14443-A card (RATS)	≥ 1.53
$_h$20	$_h$02	T=CL activation of ISO 14443-B card (Attrib)	≥ 1.53
$_h$20	$_h$04	Disable the next T=CL activation[11]	≥ 1.55
$_h$20	$_h$05	Disable every T=CL activation (until reset of the reader)	≥ 1.55
$_h$20	$_h$06	Enable T=CL activation again	≥ 1.55
$_h$20	$_h$07	Disable the next T=CL activation and force a RF reset	≥ 1.55
$_h$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.	≥ 1.52

[10] Or DISC for Innovatron cards. This makes it possible to operate ISO 14443-4 compliant cards at ISO 14443-3 level. No CARD INSERTED event is triggered, so the ATR of the card stays unchanged.

[11] Upon DISCONNECT, the CARD REMOVED event fires, then the CARD INSERTED event. A new ATR is computed, and reflects that the card runs at ISO 14443-3 level.

SLOT CONTROL response

Data Out	SW1	SW2
-	See below	

SLOT CONTROL status word

SW1	SW2	Meaning
h90	h00	Success

2.3.5. ENCAPSULATE instruction

The **ENCAPSULATE** instruction has been designed to help the applications access to PICC/VICC that don't comply with ISO 7816-4.

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.

a. *Contactless slot*

ENCAPSULATE command parameter P1 for the contactless slot

Firmware ≥ 1.51

P1	Standard communication protocols
h00	Send the frame in the **T=CL** stream, using the ISO 14443-4 protocol[12]. Data In shall not include PCB nor CRC fields

Firmware ≥ 1.53

P1	Standard communication protocols
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. The standard CRC is added (and checked in return) by the reader.
h05	Send the frame "as is" using the ISO 15693 protocol. The UID of the card is added to the frame. The standard CRC is added (and checked in return) by the reader.

…/…

[12] This is the only way to send commands to a T=CL PICC that doesn't comply with the ISO 7816-4 APDU formatting, for instance a Desfire 0.4.

Firmware ≥ 1.53 (cont.)

P1	Non-standard communication
$_h$09	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 not added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out.
$_h$0A	Send the frame "as is" using the ISO 14443-3 B modulation. The CRC is not added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out.
$_h$0C	Send the frame "as is" using the ISO 15693 modulation. The CRC is not added (and not checked) by the reader → the application must append the CRC to Data In and check it in Data Out.
P1	**Mifare low level communication[13]**
$_h$0F	Send the frame "as is" using the ISO 14443-3 A modulation. The CRC is not 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 not 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

[13] 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.

Firmware ≥ 1.54

P1	Redirection to another slot[14]
$_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)

ENCAPSULATE command parameter P2 for the contactless slot

P2 encodes the frame timeout.

P2	Timeout value
$_h$-0	If P1 = $_h$00, use the default T=CL timeout defined by the card (card's FWT) If P1 = $_h$04 or P1 = $_h$05, use the default timeout allowed for ISO 15693 chips If P1 🕒$_h$00, P1 🕒$_h$04 and P1 🕒$_h$05, 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
$_h$0-	Set status word = $_h$6F XX , XX being the contactless specific error
$_h$8-	Set status word = $_h$63 00 on any contactless specific error

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

b. Contact slots

ENCAPSULATE command parameter P1 for the contact slots

P1	
$_{h}$00	Send the frame in the T=0 or T=1 stream Other values are RFU

ENCAPSULATE command parameter P2 for the contact slot

P2	
$_{h}$00	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
$_{h}$90	$_{h}$00	Success - last byte of Data Out has 8 valid bits
$_{h}$90	$_{h}$01	Success - last byte of Data Out has 1 valid bits
$_{h}$90	$_{h}$02	Success - last byte of Data Out has 2 valid bits
$_{h}$90	$_{h}$03	Success - last byte of Data Out has 3 valid bits
$_{h}$90	$_{h}$04	Success - last byte of Data Out has 4 valid bits
$_{h}$90	$_{h}$05	Success - last byte of Data Out has 5 valid bits
$_{h}$90	$_{h}$06	Success - last byte of Data Out has 6 valid bits
$_{h}$90	$_{h}$07	Success - last byte of Data Out has 7 valid bits
$_{h}$6F	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.
$_{h}$63	$_{h}$00	Error reported by the contactless interface (when high-order bit of P2 is 1).
$_{h}$62	$_{h}$82	Le is greater than actual response from card
$_{h}$6C	XX	Le is shorter than actual response from card

2.4. OTHER VENDOR SPECIFIC INSTRUCTIONS

2.4.1. READER CONTROL instruction

The **READER CONTROL** instruction allows driving the global behavior of the **SpringCard PC/SC Reader** (LEDs, buzzer, etc depending on product physical characteristics).

For advanced operation, or if you want to interact with the reader even when there's no card inserted, use *SCardControl* instead (see chapter 3).

> *If your reader is multi-slot (contactless + contact or SAM), the READER CONTROL instruction is sent to one slot (a logical reader), but is likely to have a global impact to the whole physical reader.*
> *In other words, sending a READER CONTROL instruction to one card channel may have an impact on another card channel.*
> *It is highly recommended 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

To go back to default (LEDs automatically driven by the reader), send the APDU:

```
FF F0 00 00 01 1E
```

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 0000 if you need to stop the buzzer before the duration started in a previous call.

To go back to default (buzzer automatically driven by the reader), send the APDU:

```
FF F0 00 00 01 1C
```

c. Others

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

Therefore, every command documented in § 3.4 and starting with code $_h$58 may be transmitted in the *SCardTransmit* link using the READER CONTROL instruction, exactly as if it were transmitted in a *SCardControl* link.

Do not use this feature unless you know exactly what you are doing.

2.4.2. 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

2.4.3. CONFIGURE CALYPSO SAM specific instruction

This instruction is only available on devices with 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

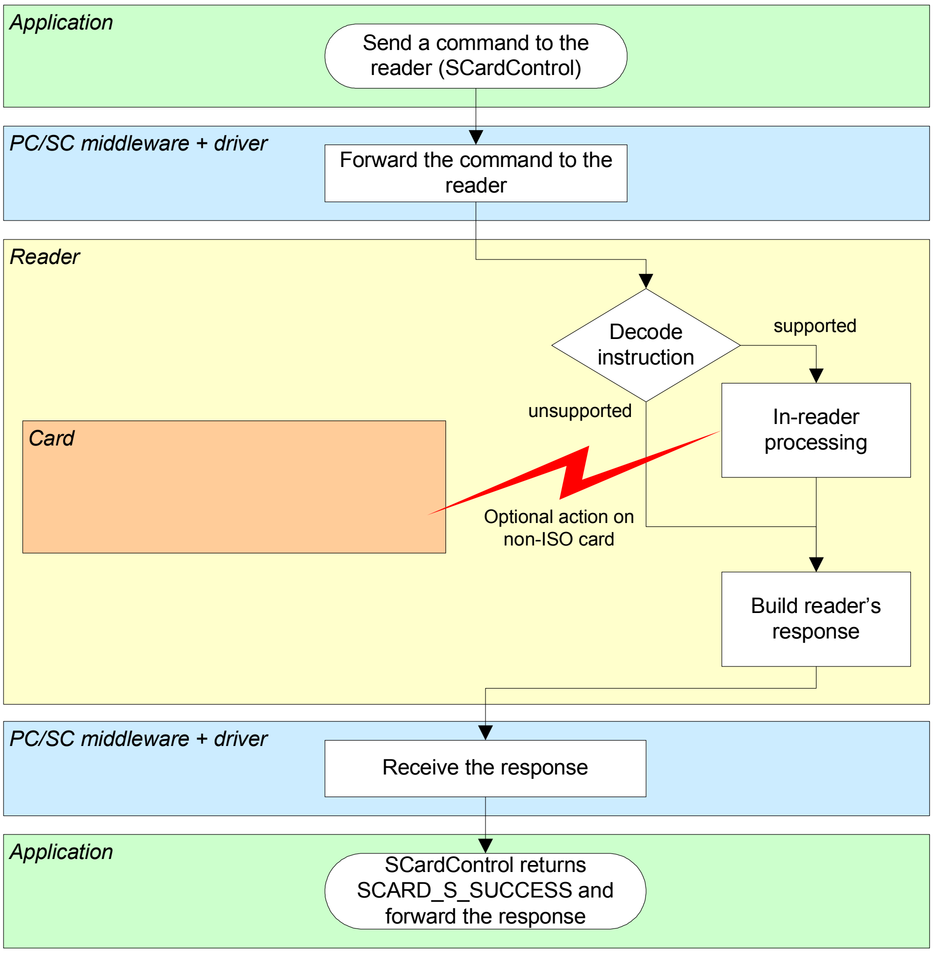
SW1	SW2	Meaning
h90	h00	Success
h6B	h00	Wrong value for P1
h6F	hE7	SAM didn't answer with 9000 (maybe this is not a Calypso SAM !)
h6F	XX	Error code returned by the Gemcore

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.


Application
Send a command to the reader (SCardControl)
PC/SC middleware + driver
Forward the command to the reader
Reader
Decode instruction
supported
In-reader processing
unsupported
Card
Optional action on non-ISO card
Build reader's response
PC/SC middleware + driver
Receive the response
Application
SCardControl returns SCARD_S_SUCCESS and forward the response

If your reader is multi-slot (contactless + contact and/or 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. CONFIGURING THE DRIVER TO ALLOW DIRECT CONTROL

Being compliant with the CCID specification, **SpringCard PC/SC Readers** are supported by (at least) 5 USB drivers:

- SpringCard CCID driver for Windows (ref. SDD480),
- Microsoft CCID kernel-mode driver (USBCCID) coming with Windows 2000/XP/Vista,
- Microsoft CCID user-mode driver (WUDFUsbccidDriver) coming with Windows 7,
- The open-source CCID driver from the PCSC-Lite package on Linux, MacOS X, and other UNIX operating systems.

3.2.1. Direct control using SpringCard SDD480

Direct control is always enabled in **SpringCard SDD480 driver**.

With this driver, in SCardControl function call, parameter dwControlCode shall be set to **SCARD_CTL_CODE(2048)**.

*SCARD_CTL_CODE is a macro defined in header winscard.h from Windows SDK. For non-C/C++ languages, replace SCARD_CTL_CODE(2048) by constant value $_h$**00241FE4** ($_d$3219456).*

3.2.2. Direct control using MS USBCCID

With **MS USBCCID** driver, direct control of the reader must be enabled on a per-reader basis : each reader has its own USB serial number, and the direct control has to be explicitly enabled for this serial number.

This is done by writing a value in registry, either using **regedit** or custom software. See for instance the command line tool **ms_ccid_escape_enable**, available with its source code in **SpringCard PC/SC SDK**.

The target key in registry is

```
HKEY_LOCAL_MACHINE
        SYSTEM
            CurrentControlSet
                Enum
                    USB
                        VID_1C34&PID_xxxx
                            yyyyyyyy
                                Device Parameters
```

Where *xxxx* is the reader's Product IDentifier (for instance, 7141 for Prox'N'Roll, 7113 for CrazyWriter, etc) and *yyyyyyyy* its serial number.

Under this registry key, create the registry entry **EscapeCommandEnabled**, of type **DWORD**, and set it to value **1**. Once the value has been written, unplug and plug the reader again (or restart the computer) so the driver will restart, taking the new parameter into account.

With this driver, in SCardControl function call, parameter dwControlCode shall be set to **SCARD_CTL_CODE(3050)**.

> *SCARD_CTL_CODE is a macro defined in header winscard.h from Windows SDK. For non-C/C++ languages, replace SCARD_CTL_CODE(3500) by constant value* $_h$***004074F8*** *(*$_d$*3225264).*

3.2.3. Direct control using MS WUDFUsbccidDriver

With **MS WUDFUsbccidDriver** (new user-mode driver introduced in Windows 7), direct control of the reader must also be enabled on a per-reader basis : each reader has its own USB serial number, and the direct control has to be explicitly enabled for this serial number.

This is done by writing a value in registry, either using **regedit** or custom software. See or instance the command line tool **ms_ccid_escape_enable**, available with its source code in **SpringCard PC/SC SDK**.

The target key in registry is

```
HKEY_LOCAL_MACHINE
        SYSTEM
            CurrentControlSet
                Enum
                    USB
                        VID_1C34&PID_xxxx
                            yyyyyyyy
                                Device Parameters
                                    WUDFUsbccidDriver
```

Where *xxxx* is the reader's Product IDentifier (for instance, 7141 for Prox'N'Roll, 7113 for CrazyWriter, etc) and *yyyyyyyy* its serial number.

Under this registry key, create the registry entry **EscapeCommandEnabled**, of type **DWORD**, and set it to value **1**. Once the value has been written, unplug and plug the reader again (or restart the computer) so the driver will restart, taking the new parameter into account.

With this driver, in SCardControl function call, parameter dwControlCode shall be set to **SCARD_CTL_CODE(3050)**.

> *SCARD_CTL_CODE is a macro defined in header winscard.h from Windows SDK. For non-C/C++ languages, replace SCARD_CTL_CODE(3500) by constant value* ${}_{h}$***004074F8*** *(*${}_{d}$*3225264).*

3.2.4. Direct control using PCSC-Lite CCID

To be written.

3.3. IMPLEMENTATION DETAILS

3.3.1. Sample code

```
#include <winscard.h>

// dwControlCode for SpringCard SDD480 driver
#define IOCTL_SC_PCSC_ESCAPE       SCARD_CTL_CODE(2048)
// dwControlCode for Microsoft CCID drivers
#define IOCTL_MS_PCSC_ESCAPE       SCARD_CTL_CODE(3050)

// This function is a wrapper around SCardControl
// It creates its own PC/SC context for convenience, but you
// may remain into a previously open context

// Note: Use ScardListReaders to get reader_name

LONG reader_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;

  // get a direct connection to the reader
  // this must succeed even when there's no card

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

  // direct control through SCardControl
  // dwControlCode for SpringCard SDD480 driver

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

  if ((rc == ERROR_INVALID_FUNCTION)
   || (rc == ERROR_NOT_SUPPORTED)
   || (rc == RPC_X_BAD_STUB_DATA))
  {
    // direct control through SCardControl
    // dwControlCode for Microsoft CCID drivers

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

  // close the connection
  // the dwDisposition parameter is coherent with the fact
  // that we didn't do anything with the card (or that there's
  // no card in the reader)

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

  return rc;
}
```

3.3.2. Link to K531/K632/SpringProx/CSB legacy protocol

Sending an escape sequence through *SCardControl* (with appropriate value for *dwControlCode*) is exactly the same as sending a “legacy command” to a SpringCard reader running in **legacy** mode.

The detailed reference of all the command supported by our reader is available in CSB4 and/or K531/K632 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.

3.3.3. 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 : $_h$00 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.

3.3.4. 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.

3.4. LIST OF AVAILABLE CONTROL SEQUENCES

3.4.1. 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 b

3.4.2. 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)".

3.4.3. Stopping / starting a slot

When a slot is stopped, the reader

- powers down the smartcard in the slot (if some),
- disable the slot[15],
- send the "card removed" event if there was a card in the slot.

When a slot is started again, the reader

- enable the slot[16],
- try to power up the smartcard in the slot (if some),
- 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

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

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

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

Most **SpringCard PC/SC Readers** feature 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 ($_h$00 on success, $_h$16 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.

3.5. Configuration registers

3.5.1. 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)	Support
msb	15	RFU	
	14	RFU	
	13	RFU	
	12	Kovio RF Barcode	Fw >= 1.64 A B
	11	Innovision Topaz/Jewel (NFC Forum's type 1 tags)	Fw >= 1.60 A B C D
	10	RFU	
	9	RFU	
	8	RFU	
	7	Innovatron (legacy Calypso cards – sometimes called ISO 14443-**B'**)	A B C D
	6	ASK CTS256B et CTS512B	A B C D
	5	ST MicroElectronics SRxxx	A B C D
	4	Inside Contactless PicoPass (also HID iClass)	Fw >= 1.55 A B C D
	3	NXP ICODE1	B
	2	ISO 15693	B D
	1	ISO 14443-B	A B C D
lsb	0	ISO 14443-A	A B C D

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

Hardware support

A	Supported by RC531-based hardware (some custom versions of CSB6)
B	Supported by RC632-based hardware (CSB6 mainstream)
C	Supported by RC523/PN512-based hardware (NFC'Roll, H512)
D	Supported by RC633-based hardware (H663)

3.5.2. CCID slot mapping

Address: $_h$B1

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

3.5.3. 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

3.5.4. Misc. T=CL options

Firmware ≥ 1.52

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[17]
	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)

[17] 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.

3.5.5. 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: ₕC0 – Size: 1 byte

Value	Operating mode
ₕ00	RFU
ₕ01	Legacy mode
ₕ02	PC/SC mode
ₕ03	Not supported by this firmware
ₕ80	RFU
ₕ81	Legacy mode without serial number in USB descriptor
ₕ82	PC/SC mode without serial number in USB descriptor
ₕ83	Not supported by this firmware

Default value: ₕ02 (PC/SC)

3.5.6. Advanced RF configuration

Address: ₕC1

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

Address: ₕC6

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

Address: ₕC7

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

3.5.7. Calypso compliance

Address: ₕC2

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

3.5.8. 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) PICC enters its field.

> ***SpringCard PC/SC Readers** are theoretically able to communicate with PICCs 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. Most readers ship with a factory configuration limiting them at 212kbps for ISO 14443-A and 106kbps for ISO 14443-B.*
> *Communication is slower yet more reliable, so the overall transaction time often appears faster because there are fewer errors and retries than with a higher baudrate.*

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)[18].

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

3.5.9. Buzzer settings

If the reader features a buzzer, it beeps everytime a card enters its field. This register defines the duration or the beep. To disable the beep, set this register to $_h$00.

Address: $_h$CC – Size: 1 byte

Default value: $_h$08 (80ms beep on card arrival).

4. VENDOR ATTRIBUTES

There's currently no documented vendor attribute for this reader.

5. Working with contactless cards – useful hints

5.1. Recognizing and identifying PICC/VICC in PC/SC environment

5.1.1. ATR of an ISO 14443-4 compliant smartcard

If the PICC is with 14443 up to level 4 ("**T=CL**"), the reader builds a pseudo-ATR using the standard format defined 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	…	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)

c. For Innovatron (legacy Calypso cards)[19]:

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[20].
…	…		
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

Most Calypso cards are able to communicate both 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[21] to make sure it will not start a new transaction on the same card as earlier.

[19] 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.

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

[21] 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 wired-logic PICC/VICC

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

Offset	Name	Value	
0	TS	$_h$3B	Direct convention
1	T0	$_h$8F	Higher nibble 8 means: no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (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	$_h$80	
5	H2	$_h$4F	Application identifier presence indicator
6	H3	$_h$0C	Length to follow (12 bytes)
7	H4	$_h$A0	Registered Application Provider Identifier **A0 00 00 03 06** is for PC/SC workgroup
8	H5	$_h$00	
9	H6	$_h$00	
10	H7	$_h$03	
11	H8	$_h$06	
12	H9	PIX.SS	Standard (see 5.1.4)
13	H10	PIX.NN	Card name (see 5.1.5)
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)

5.1.3. Using the GET DATA instruction

With the GET DATA instruction (documented in § 2.2.1), the host application is able to retrieve every information needed to identify a PICC/VICC:

- 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 instruction (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	Fw
Values specified by PC/SC		
$_h$00 $_h$01	NXP Mifare Standard 1k	
$_h$00 $_h$02	NXP Mifare Standard 4k	
$_h$00 $_h$03	NXP Mifare UltraLight Other Type 2 NFC Tags *(NFC Forum)* with a capacity <= 64 bytes	
$_h$00 $_h$06	ST MicroElectronics SR176	
$_h$00 $_h$07	ST MicroElectronics SRI4K, SRIX4K, SRIX512, SRI512, SRT512	≥ 1.55
$_h$00 $_h$0A	Atmel AT88SC0808CRF	
$_h$00 $_h$0B	Atmel AT88SC1616CRF	
$_h$00 $_h$0C	Atmel AT88SC3216CRF	
$_h$00 $_h$0D	Atmel AT88SC6416CRF	
$_h$00 $_h$12	Texas Intruments TAG IT	
$_h$00 $_h$13	ST MicroElectronics LRI512	
$_h$00 $_h$14	NXP ICODE SLI	
$_h$00 $_h$16	NXP ICODE1	
$_h$00 $_h$21	ST MicroElectronics LRI64	
$_h$00 $_h$24	ST MicroElectronics LR12	
$_h$00 $_h$25	ST MicroElectronics LRI128	
$_h$00 $_h$26	NXP Mifare Mini	
$_h$00 $_h$2F	Innovision Jewel	
$_h$00 $_h$30	Innovision Topaz (NFC Forum type 1 tag)	
$_h$00 $_h$34	Atmel AT88RF04C	
$_h$00 $_h$35	NXP ICODE SL2	
$_h$00 $_h$3A	NXP Mifare UltraLight C Other Type 2 NFC Tags *(NFC Forum)* with a capacity > 64 bytes	≥ 1.62

NN	Card name	Fw
SpringCard proprietary extension[22]		
$_h$FF $_h$A0	Generic/unknown 14443-A card	
$_h$FF $_h$A1	Kovio RF barcode	≥ 1.63
$_h$FF $_h$B0	Generic/unknown 14443-B card	
$_h$FF $_h$B1	ASK CTS 256B	
$_h$FF $_h$B2	ASK CTS 512B	
$_h$FF $_h$B3	Pre-standard ST MicroElectronics SRI 4K	< 1.55
$_h$FF $_h$B4	Pre-standard ST MicroElectronics SRI X512	< 1.55
$_h$FF $_h$B5	Pre-standard ST MicroElectronics SRI 512	< 1.55
$_h$FF $_h$B6	Pre-standard ST MicroElectronics SRT 512	< 1.55
$_h$FF $_h$B7	Inside Contactless PICOTAG/PICOPASS	
$_h$FF $_h$B8	Generic Atmel AT88SC / AT88RF card	
$_h$FF $_h$C0	Calypso card using the Innovatron protocol	
$_h$FF $_h$D0	Generic ISO 15693 from unknown manufacturer	
$_h$FF $_h$D1	Generic ISO 15693 from EMMarin (or Legic)	
$_h$FF $_h$D2	Generic ISO 15693 from ST MicroElectronics, block number on 8 bits	
$_h$FF $_h$D3	Generic ISO 15693 from ST MicroElectronics, block number on 16 bits	
$_h$FF $_h$FF	Virtual card (test only)	

Note: PIX.NN is specified for memory cards only. Even if the GET DATA instruction 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.

[22] The cards in this list are not referenced by PC/SC specification at the date of writing. In case they are added to the specification, the future firmware versions will have to use the new value. It is therefore advised **not to check those values** in the applications, as they are likely to be removed in the future.

5.2. ISO 14443-4 PICCs

5.2.1. Desfire first version (0.4)

Since the card is not ISO 7816-4 compliant, the Desfire commands must be wrapped in an ENCAPSULATED instruction, with P1=$_h$00 (§ 2.3.5). The reader translates the C-APDU into a native Desfire command, retrieve the native Desfire answer, and translates it into a valid R-APDU.

5.2.2. Desfire EV0 (0.6) and EV1

The card is ISO 7816-4 compliant. Native commands are wrapped into ISO 7816-4 APDUs with a card-specific CLA = $_h$90. Please refer to the card's datasheet for details.

5.2.3. Calypso cards

A Calypso card is ISO 7816-4 compliant. You may work with a contactless Calypso card as if it were inserted in a contact smartcard reader.

5.3. WIRED-LOGIC PICCS BASED ON ISO 14443-A

5.3.1. Mifare Classic

The PICCs covered by this chapter are:

- Mifare 1k (NXP MF1ICS50, **PIX.NN = $_h$0001**),
- Mifare 4k (NXP MF1ICS70, **PIX.NN = $_h$0002**),
- Mifare Mini (NXP MF1ICS20, **PIX.NN = $_h$0026**),
- Mifare Plus (X or S) when used in level 1 (see § 5.3.2).

Please download the datasheets of the cards at www.nxp.com. Useful information are available at www.mifare.net.

All these PICCs 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).

Operating multi-standard PICCs as Mifare Classic

Some ISO 14443-4 compliant smartcards or NFC objects are also able to emulate Mifare Classic cards, but due to the ISO 14443-4 (T=CL) compliance, the reader will “hide” their Mifare emulation mode and make them appear as high-level smartcards.

There are 3 ways to force the reader to staty at Mifare level:

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

a. READ BINARY instruction

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),

Since the size of every block is 16, Le must be a multiple of 16,

- When Le=$_h$00 and the address is aligned on a sector boundary, all the data blocks of the sector are returned (48 or 240 bytes),

- When Le=$_h$00 and the address is not aligned, a single block is returned (16 bytes).

Note that when a sector trailer (security block) is read, the keys are not readable (they are 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.

> *Using the MIFARE CLASSIC READ instruction (§ 3.3.5) is easier and may shorten the transaction time.*

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- 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 every block is 16, Lc must be a multiple of 16 (48 bytes for standard sectors, 240 bytes for the largest sectors in Mifare 4k).

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

Important disclaimer

> *Writing sector trailers (security blocks) is possible as long as the sector's current access condition allows it, but Mifare sector trailers have to follow a specific formatting rule (mix-up of the access conditions bits) to be valid. Otherwise, the sector becomes permanently unusable.*
> *Before invoking MIFARE CLASIC WRITE, always double check that you're not writing a sector trailer, and if you really have to do so, make sure the new content is formatted as specified in the datasheet of the PICC.*

> *Using the MIFARE CLASSIC WRITE instruction (§ 2.3.2) is easier and may shorten the transaction time.*

c. Specific instructions for Mifare Classic

3 specific instructions exist to work with Mifare Classic PICCs:

- MIFARE CLASSIC READ, see § 2.3.1,
- MIFARE CLASSIC WRITE, see § 2.3.2,
- MIFARE CLASSIC VALUE (implementing INCREMENT, DECREMENT and RESTORE followed by TRANSFER), see § 2.3.3.

5.3.2. Mifare Plus X and Mifare Plus S

Please download the datasheets of the cards at www.nxp.com.

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

> ***SpringCard*** *has developed the PCSC_MIFPLUS software library (available as source code and as pre-compiled DLL in the SDK) to help working with Mifare Plus cards without going down at the APDU level and without the need to implement the security scheme by yourself.*
> *For the documentation of this API, go to*
> *http://www.springcard.com/support/apidoc/pcsc_mifplus/index.html*

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.

As the card is not ISO 7816-4 compliant, the card commands shall be sent wrapped in an ENCAPSULATED instruction with P1=$_h$00 (§ 2.3.5).

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 instruction with P1,P2=$_h$10,02 to do so (§ 2.3.4)[23].

b. Level 1

At level 1, the card emulates a Mifare Classic card (§ 5.3.1). The reader builds a memory card ATR according to § 5.1.1.

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

The card supports a new AES authentication Function. Use the ENCAPSULATE instruction with P1=$_h$01 (§ 2.3.5) 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[24]. Send the SLOT CONTROL instruction with P1,P2=$_h$20,01 to do so (§ 2.3.4). Afterwards, process as documented for level 0.

c. Level 2

The level 2 is not available on Mifare Plus S cards.

[23] 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).

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

Working with the Mifare Plus X at this level is possible thanks to the low level instruction calls (SLOT CONTROL, ENCAPSULATE) but is not implemented in the reader (not supported by our software library).

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.

Since the card is not ISO 7816-4 compliant, the card commands shall be sent wrapped in an ENCAPSULATED instruction, with P1=$_h$00 (§ 2.3.5).

5.3.3. Type 2 NFC Tags (NFC Forum) - Mifare UltraLight and UltraLight C

The cards covered by this chapter are:

- Mifare UL - NXP MF01CU1 (**PIX.NN = $_h$0003**),
- Mifare UL C - NXP MF01CU2 (**PIX.NN = $_h$003A**),
- Any card compliant with NFC Forum Type 2 tag specification.

Please download the datasheets of the cards at www.nxp.com. Please visit www.nfcforum.org for the Type 2 tag specification.

All these cards are divided into 4-byte *pages*. It is possible to write only one page at once, but reading is generally done 4 pages by 4 pages (16 bytes). A NFC Forum Type 2 tag could also be optionally divided into sectors of 256 pages (1024 bytes).

> *It isn't possible to discover the actual capacity of a compliant PICC at protocol level.*
> *If the PICC is already formated according to NFC Forum specification, the capacity is stored among other data in the 1st OTP page (CC – capability container bytes).*
> *If any other cases, the application may find the number of pages by sending READ BINARY instruction, incrementing the address, until it fails.*
> *Pay attention that some of those PICCs will unfortunately not fail but truncate the address; for instance a PICC with only 16 pages (0 to 15) may return the content of pages 0, 1, 2 and 3 when the address 16 is read. But as pages 0 and 1 store the UID (serial number) of the PICC, comparing pages 16, 17 to pages 0, 1 is enough to understand that the end of the memory space has been reached.*

a. *READ BINARY instruction*

In the READ BINARY command APDU,

- P1 must be $_h$00 for Mifare UL and Mifare UL C. For other NFC Forum Type 2 tags that have more than one sector, P1 is the sector number.
- P2 is the address of the first page to be read (0 to 15 for Mifare UltraLight, 0 to 40 for Mifare UltraLight C; for other NFC Forum Type 2 tags, refer to the datasheet).

Since the size of a page is 4 bytes, Le must be multiple of 4. When Le=$_h$00, 4 pages are returned (16 bytes).

It is possible to read the complete data area of a Mifare UL in a single call by setting Le to $_h$40 (64 bytes). For Mifare UL C, the same result is achieved by setting Le to $_h$90 (144 bytes).

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

- P1 must be $_h$00 for Mifare UL and Mifare UL C. For other NFC Forum Type 2 tags that have more than one sector, P1 is the sector number,
- P2 is the address of the (single) page to be written (2 to 15 for Mifare UltraLight, 2 to 40 for Mifare UltraLight C; for other NFC Forum Type 2 tags, refer to the datasheet).

Since the size of a page is 4 bytes, Lc must be 4, exactly.

> *Some pages holds OTP (one-time-programming) bits, and/or lock bits that are intented to make the PICC memory read only. Do not write on those pages without a good understanding of the consequences.*

c. Mifare UltraLight C 3-DES authentication

The Mifare UltraLight C supports a 3-pass Triple-DES authentication feature.

Use the ENCAPSULATE instruction with P1=$_h$01 (§ 2.3.5) to implement this function.

> ***SpringCard** has developed the PCSC_MIFULC software library (available as source code and as pre-compiled DLL in the SDK) to help working with Mifare UltraLight C cards without the need to implement the security scheme by yourself.*
> *For the documentation of this API, go to*
> *http://www.springcard.com/support/apidoc/pcsc_mifulc/index.html*

5.3.4. NFC Forum Type 1 tags - Innovision Topaz/Jewel

The PICCs covered by this chapter are:

- Innovision Topaz (**PIX.NN = $_h$002F**),
- Innovision Jewel (**PIX.NN = $_h$0030**).

a. READ BINARY instruction (full card)

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 must be $_h$00,

Set Le=$_h$00. The whole card content is returned as once.

b. READ BINARY instruction (single byte)

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first byte to be read (0 to 127),

Le can be any length but $_h$01.

> *Using the above READ BINARY (FULL CARD) instruction is 10 times faster than this BYTE LEVEL version.*

c. UPDATE BINARY instruction (single byte)

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the byte to be written (0 to 127),

Lc must be 1, exactly.

> *Some bytes holds OTP (one-time-programming) bits, and/or lock bits that are intented to make the PICC memory read only. Do not write on those bytes without a good understanding of the consequences.*

5.4. Wired-logic PICCs based on ISO 14443-B

5.4.1. ASK CTS256B and CTS512B

The PICCs covered by this chapter are:

- ASK CTS256B (**PIX.NN = $_h$FFB1**),
- ASK CTS512B or CTM512B (**PIX.NN = $_h$FFB2**).

These PICCs are divided into 2-byte *areas*.

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first area to be read (0 to 15 for CTS256B, 0 to 31 for CTS512B),

Since the size of every area is 2, Le must be multiple of 2 (32 bytes for the full CTS256B card, 64 bytes for the full CTS512B card),

When Le=$_h$00, a single area is returned (2 bytes).

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the area to be written,

Since the size of every area is 2, Lc must be 2, exactly.

Some areas play a particular role in the configuration of the PICC. Do not write on those areas without a good understanding of the consequences.

5.4.2. ST MicroElectronics SR176

These PICCs are identified by **PIX.NN = $_h$0006**.

They are divided into 2-byte *blocks*.

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first block to be read (0 to 15),

Since the size of every block is 2, Le must be multiple of 2 (32 bytes for the full card),

When Le=$_h$00, a single block is returned (2 bytes).

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the block to be written,

Since the size of every block is 2, Lc must be 2, exactly.

Some blocks play a particular role in the configuration of the PICC. Do not write on those blocks without a good understanding of the consequences.

5.4.3. ST MicroElectronics SRI4K, SRIX4K, SRI512, SRX512, SRT512

These PICCs are identified by **PIX.NN = $_{h}$0007**.

They are divided into 4-byte *blocks*.

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_{h}$00,
- P2 is the address of the first block to be read,

Since the size of every block is 2, Le must be multiple of 4,

When Le=$_{h}$00, a single block is returned (4 bytes).

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

- P1 must be $_{h}$00,
- P2 is the address of the block to be written,

Since the size of every block is 4, Lc must be 4, exactly.

> *Some blocks play a particular role in the configuration of the PICC. Do not write on those blocks without a good understanding of the consequences.*

5.4.4. Inside Contactless PicoPass, ISO 14443-2 mode

This part applies to chips named either "PicoPass or PicoTag" when the ISO 14443-3 compliance is NOT enabled in the card (see § 5.4.5 in the other case).

Those PICCs exist in two sizes (2K → 256 B, 16K → 2 kB), and in non-secure (2K, 16K) or secure (2KS, 16KS) versions. They are divided into 8-byte blocks.

They are currently identified by **PIX.NN = $_h$FFB7** and **PIX.SS = $_h$06** (ISO 14443-B level 2). Pay attention that this may change in future versions since PC/SC has registered new PIX.NN for these PICCs.

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

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first block to be read (2K: 0 to 31; 16K: 0 to 255),

Since the size of every block is 8, Le must be multiple of 8,

When Le=$_h$00, a single block is returned (8 bytes).

b. UPDATE BINARY instruction

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),

Since the size of every block is 8, Lc must be 8, exactly.

Some blocks play a particular role in the configuration of the PICC. Do not write on those blocks without a good understanding of the consequences.

c. Page select

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

5.4.5. Inside Contactless PicoPass, ISO 14443-3 mode

This part applies to chips named either "PicoPass or PicoTag" when the ISO 14443-3 compliance IS enabled in the card (see § 5.4.4 in the other case).

Those PICCs exist in two sizes (2K → 256 B, 16K → 2 kB), and in non-secure (2K, 16K) or secure (2KS, 16KS) versions. They are divided into 8-byte blocks.

They are currently identified by **PIX.NN = $_h$FFB7** and **PIX.SS = $_h$07** (ISO 14443-B level 3 or 4). Pay attention that this may change in future versions since PC/SC has registered new PIX.NN for these PICCs.

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

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first block to be read (2K: 0 to 31; 16K: 0 to 255),

Since the size of every block is 8, Le must be multiple of 8,

When Le=$_h$00, a single block is returned (8 bytes).

b. UPDATE BINARY instruction

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),

Since the size of every block is 8, Lc must be 8, exactly.

Some blocks play a particular role in the configuration of the PICC. Do not write on those blocks without a good understanding of the consequences.

5.4.6. Atmel CryptoRF

The PICCs covered by this chapter are:

- AT88SC0808CRF (**PIX.NN = $_h$000A**),
- AT88SC1616CRF (**PIX.NN = $_h$000B**),
- AT88SC3216CRF (**PIX.NN = $_h$000C**),
- AT88SC6416CRF (**PIX.NN = $_h$000D**),
- AT88SCRF04C (**PIX.NN = $_h$0034**).

SpringCard PC/SC readers implement the read and write functions in non-authenticated mode. Advanced functions and authenticated communication has to be implemented by the application within an ENCAPSULATE instruction.

> *The card is always activated with CID=$_h$01. Use this CID to build the actual command to be sent through the ENCAPSULATE instruction.*

a. READ BINARY instruction

In the READ BINARY command APDU,

P1,P2 is the first address to be read,

Le is the length to be read (1 to 32 bytes).

Note: the READ BINARY instruction maps to the “Read User Zone” low-level command. The “Read System Zone” command is not implemented in the reader, and therefore must be encapsulated.

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

P1,P2 is the first address to be written,

Lc is the length to be written (1 to 32 bytes).

Note: the UPDATE BINARY instruction maps to the “Write User Zone” low-level command. The “Write System Zone” command is not implemented in the reader, and therefore must be encapsulated.

5.5. ISO 15693 VICCs

Only the readers based on RC632 or RC663 do implement the VCD mode.

5.5.1. ISO 15693-3 read/write commands

The size of the blocks depend on the card. Known sizes are

- 1 byte for ST MicroElectronics LRI64 (**PIX.NN = $_h$0021**),
- 4 bytes for NXP ICODE-SLI (**PIX.NN = $_h$0014**) and Texas Instrument TagIT cards (**PIX.NN = $_h$0012**),
- 8 bytes for EM MicroElectronics cards (**PIX.NN = $_h$FFD1**).

Please read the documentation of the card you're working with to know the actual size of its blocks, and the number of existing blocks.

> *Some VICCs feature special blocks called either OTP (one-time-programming), WORM (write one, read many) that can't be overwritten nor erased after a first write operation. Do not write on those blocks without a good understanding of the consequences.*

a. READ BINARY instruction

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 VICC to know its number of blocks,

Le must be a multiple of the size of the blocks,

When Le=$_h$00, a single block is returned (length depending on the card).

Note: ISO 15693 defines 2 functions to read date: READ SINGLE BLOCK and READ MULTIPLE BLOCKS. The reader's READ BINARY instruction tries both of them until one succeed.

b. UPDATE BINARY instruction

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 VICC to know its number of blocks,

Lc must be the size of the block, exactly.

Note: ISO 15693 defines 2 functions to read date: WRITE SINGLE BLOCK and WRITE MULTIPLE BLOCKS. The reader's UPDATE BINARY instruction tries both of them until one succeed.

5.5.2. Read/write commands for ST MicroElectronics chips with a 2-B block address

Firmware >= 1.70.

ST MicroElectronics' M24LR16E (**PIX.NN = $_h$FFD3**) implements an extented version of ISO 15693's commands, where the address are on 2 bytes instead of one.

Proceed as with other ISO 15693 chips with this difference: in READ BINARY and UPDATE BINARY instructions, P1 is the high-order byte of the address and could be non-zero.

5.5.3. Other ISO 15693 commands

The ISO 15693 standard defines numerous optional commands, and allows chip manufacturer to implement and huge number of custom or proprietary commands. It is therefore not possible to implement all of them in the readers. Hopefully, the ENCAPSULATE instruction (INS = $_h$FE, see § 2.3.5.) makes it easy to send any command to the 15693 chip currently activated by the reader.

Since the reader operates the ISO 15693 chip in addressed mode (the VICC is never put into *quiet state*), the UID of the chip shall be provided within every command frame. The insertion of the UID is performed automatically by the ENCAPSULATE instruction when parameter P1 is set to $_h$05.

The APDU shall be build as follow:

CLA	INS	P1	P2	Lc	Data In			Le
$_h$FF	$_h$FE	$_h$05	$_h$00	XX	Command flags	Command code	Command data (optional)	$_h$00

Note: Le could be omitted.

Allowed values for the 'command flags' byte

Bit		Value	Description
7	RFU	0	
6	Option	0/1	Meaning is defined by the command description. Please refer to the ISO 15693:3 standard and/or to the datasheet of the VICC for details
5	Address	1	The UID of the VICC is included in the command frame
4	Select	0	Not using the VICC quiet state
3	Protocol extension	0/1	Must be 0 for standard commands Some VICC may implement vendor-specific commands that require to have this bit set to 1
2	Inventory	0	It is not allowed to invoke the INVENTORY command through an ENCAPSULATE APDU
1	Data rate	1	High data rate shall be used
0	Sub carrier	0	A single sub-carrier shall be used

As a summary, typical values for the 'command flags' byte are:

- $_h$22 when the option flag is not set
- $_h$62 when the option flag is required by the PICC or the command

a. Read single block

ISO 15693 command code : $_h$20

The APDU is

```
FF FE 05 00 03 22 20 <block number>
```

b. Write single block

ISO 15693 command code : $_h$21

The APDU is

```
FF FE 05 00 <3 + data length > 22 21 <block number> <...data...>
```

The length of the data must match the size of the block. Please refer to the VICC's datasheet to know the size of its block.

c. Lock block

ISO 15693 command code : $_h$22

The APDU is

```
FF FE 05 00 03 22 22 <block number>
```

Locking a block makes it permanently read-only. This is a non-cancelable operation. Do not perform this operation without a good understanding of the consequence.

d. Write AFI

ISO 15693 command code : $_h$27

The APDU is

```
FF FE 05 00 03 22 27 <new AFI>
```

e. Lock AFI

ISO 15693 command code : $_h$28

The APDU is

```
FF FE 05 00 02 22 28
```

Locking the AFI is a non-cancelable operation. Do not perform this operation without a good understanding of the consequence.

f. Write DSFID

ISO 15693 command code : $_h$29

The APDU is

```
FF FE 05 00 03 22 29 <new DSFID>
```

g. Lock DSFID

ISO 15693 command code : $_h$2A

The APDU is

```
FF FE 05 00 02 22 2A
```

Locking the DSFID is a non-cancelable operation. Do not perform this operation without a good understanding of the consequence.

h. Get system information

ISO 15693 command code : $_h$2B

The APDU is

```
FF FE 05 00 02 22 2B
```

Note: the reader always sends the *Get system information* command to the VICC, as part of the discovery process. Invoke the GET DATA instruction (§ 2.2.1) to retrieve the value already returned by the VICC to the reader.

5.5.4. NXP ICODE1

Only the readers based on RC632 do support NXP ICODE1.

These VICCs are identified by **PIX.NN = $_h$0016**.

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first block to be read (0 to 15),

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

b. UPDATE BINARY instruction

This function is not implemented. The reader is not able to write into 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[25]	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$07	MI_CASCLEVEX	Bad anticollision sequence
${}_h$08	MI_SERNRERR	Card's serial number is invalid
${}_h$09	MI_LOCKED	Card or block locked
${}_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$0D	MI_VALUEERR	Card counter error
${}_h$0E	MI_TRANSERR	Card transaction error
${}_h$0F	MI_WRITEERR	Card write error
${}_h$10	MI_INCRERR	Card counter increment error
${}_h$11	MI_DECRERR	Card counter decrement 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
${}_h$17	MI_UNKNOWN_COMMAND	RC: unknown opcode
${}_h$18	MI_COLLERR	A collision has occurred
${}_h$19	MI_COMMAND_FAILED	RC: command execution failed
${}_h$1A	MI_INTERFACEERR	RC: hardware failure
${}_h$1B	MI_ACCESSTIMEOUT	RC: timeout
${}_h$1C	MI_NOBITWISEANTICOLL	Anticollision not supported by the card(s)
${}_h$1F	MI_CODINGERR	Bad card status
${}_h$20	MI_CUSTERR	Card: vendor specific error
${}_h$21	MI_CMDSUPERR	Card: command not supported

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

$_{h}$22	MI_CMDFMTERR	Card: format of command invalid
$_{h}$23	MI_CMDOPTERR	Card: option of command invalid
$_{h}$24	MI_OTHERERR	Card: other error
$_{h}$3C	MI_WRONG_PARAMETER	Reader: invalid parameter
$_{h}$64	MI_UNKNOWN_FUNCTION	Reader: invalid opcode
$_{h}$70	MI_BUFFER_OVERFLOW	Reader: internal buffer overflow
$_{h}$7D	MI_WRONG_LENGTH	Reader: invalid length

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 doesn't 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

Contact information

For more information and to locate our sales office or distributor in your country or area, please visit

www.springcard.com