springcard®

PMD2176-AB
DRAFT - PUBLIC

SpringCard PC/SC readers - H512 group

Developer's reference manual

www.springcard.com

Document Identification

Category	Developer's manual		
Family/Customer	PC/SC readers		
Reference	PMD2176	Version	AB
Status	draft	Classification	Public
Keywords	H512, PC/SC, NFC emulation, contactless cards, RFID labels, NFC tags		
Abstract			

File name	V:\Dossiers\notices\H512 Group\Developpement et integration\[PMD2176-AB] H512 Developer's Reference Manual.odt		
Date saved	10/10/12	Date printed	24/09/12

REVISION HISTORY

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	29/08/12	JDA				Created from PMD841P-FA
AB	09/10/12	JDA				First preliminary release

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 they 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),
- **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 within 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 time of writing, this document refers to all SpringCard PC/SC Readers in the **H512** group:

- The **H512S** : OEM module without antenna,
- The **H512-USB** : OEM module with antenna,
- The **NFC'Roll** : desktop reader with LEDs and buzzer.

1.3. Audience

This manual is written for use by application developers. It assumes that the reader has expert knowledge of computer development and a basic knowledge of PC/SC and NFC.

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

To get started with NFC, please visit NFC Forum (http://www.nfc-forum.org) and download their specifications.

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 available.

For technical support enquiries, please refer to SpringCard support page, on the web at

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
- NFC Forum: http://www.nfc-forum.org

1.6. GLOSSARY – USEFUL TERMS

The following list contains the terms that are directly related to the subject of this document. This is an excerpt 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) 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.
- **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 to PICC).

- **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: 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.
- **NDEF:** *NFC Data Exchange Format.* The format of the data on the NFC Tags specified by NFC Forum.
- **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 Micro Electronics 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 baud rates are possible: 106 kbit/s is mandatory, higher baud rates (212, 424 or 848 kbit/s) are optional.

- **ISO 18092** or **NFCIP-1:** This international standard defines a communication scheme (most of the time named "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 JIS:X6319-4 (aka Sony Felica protocol) at 212 and 424 kbit/s.
- **Initiator:** according to NFCIP-1, the NFC object that is the "master" of the communication with a peer known as target. A PCD is a sort of initiator.
- **Target:** according to NFCIP-1, the NFC object that is the "slave" in the communication with a peer known as initiator. A PICC is a sort of target.
- **NFC-DEP:** *NFC Data Exchange Protocol.* This is the name used by the NFC Forum for the ISO 18092 "high level" protocol. After an initial handshaking (ATR_REQ/ATR_RES), the initiator and the target exchanges transport-level blocks (DEP_REQ/DEP_RES).
- **LLCP:** *Logical Link Control Protocol.* A network protocol specified by NFC Forum on top of NFC-DEP.
- **SNEP:** *Simple NDEF Exchange Protocol.* An application protocol specified by NFC Forum to exchange NDEF messages on top of LLCP.
- **ISO 21481** or **NFCIP-2:** This international standard defines how a NFC object shall also be able to communicate using ISO 14443 and ISO 15693 standards.
- **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[2] stream cipher called CRYPTO1. Every sector is protected by 2 access keys called "key A" and "key B"[3]. 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.
- **Felica:** This trademark of Sony is the generic brand name of their PICC products. The underlying protocol has been standardized in Japan (JIS:X6319-4) and is used by ISO 18092 at 212 and 424 kbit/s.

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

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

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

2. The H512's four operating modes

2.1. Reader mode (PC/SC reader + NFC Initiator)

The **reader mode** is the default mode for the **H512**. Every time the device is started (power up, hardware reset or software reset), it enters this mode.

In this mode, the **H512** is a contactless smartcard reader (PCD), compliant with the PC/SC standard, and also a NFC Initiator:

- It detects and activates any ISO 14443-4 compliant contactless smartcard (micro-processor based, including NFC Forum type 4 Tag), and provides a direct communication channel between the application(s) running on the PC and the card,
- It detects and activates any supported contactless memory card (including Mifare family, and NFC Forum types 1, 2 and 3 Tags), and makes it possible for the application(s) to process them easily, thanks to the embedded APDU interpreter (see chapters 3. and 6.),
- It detects and activates any ISO 18092 compliant object, running as a target and using the passive communication scheme[4] (see chapter 7.),.

On the PC side, everything is implemented through exchanges of APDUs within ***SCardTransmit*** calls.

2.2. NFC Target mode

When this mode is selected, the **H512** is a NFC Target, compliant with ISO 18092. It is ready to be accessed by a remote NFC Initiator. Only the passive communication scheme is supported[5].

On the PC side, the application must implement the behaviour expected by the application running in the remote NFC Initiator.

Details are provided in chapter 8.

Note: NFC Target mode is not available in the early releases of the firmware (1.7x branch).

[4] The **H512** is active. The NFC Target remains passive during the communication.

[5] The remote NFC Initiator is active. The **H512** remains passive during the communication.

2.3. Card emulation mode

When this mode is selected, the **H512** emulates a PICC compliant with ISO 14443-4 type A (contactless smartcard). It is ready to be accessed by a remote PCD (contactless smartcard reader).

On the PC side, the application must implement the behaviour expected for the smartcard being emulated. For instance, this could be the processing of APDUs defined by ISO 7816-4.

Details are provided in chapter 9.

2.4. NFC Tag emulation mode

When this mode is selected, the **H512** emulates a NFC Forum Tag; it is ready to be accessed by a remote PCD (for instance a NFC object running in **reader mode**). On the PC side, it emulates the same kind of Tag, so the PC application may read/write the data into the Tag. As a consequence, the Tag's data are 'shared' between the PC application and the remote PCD.

This mode accepts two sub modes; the **H512** may emulate either

- a type 2 Tag, i.e. a memory card in the Mifare UltraLight family,
- a type 4 Tag, i.e. an ISO 7816-4 smartcard on top of the ISO 14443-4 type A protocol,

Refer to chapters 10. for type 4 and 11. for type 2.

2.5. Selecting one of the emulation modes

The selection of the mode has to be done with a ***SCardControl*** function call (see chapter 4. for reference).

- To enter the **NFC Target mode**, refer to § 8.3.
- To enter the **Card emulation mode**, refer to § 9.3.
- To start the **emulation of a NFC Forum type 4 Tag** (NFC Tag emulation mode, type 4 sub mode), refer to § 10.7.
- To start the **emulation of a NFC Forum type 2 Tag** (NFC Tag emulation mode, type 2 sub mode), refer to § 11.4.

2.6. Going back to Reader mode

To go back to the **reader mode**, send the following command within a ***SCardControl*** function call:

ENTER READER MODE command

Opcode		Parameters	
h83	h10	h00	h00

ENTER READER MODE response – success

Status
h00

ENTER READER MODE response – failure: an external RF field has been detected

Status
h1D

Error: the **H512** was unable to enter the **reader mode** (PCD) due to another PCD being in the nearby.

The **H512** is not in any **emulation mode** any more. Stop the other PCD or move it far from the **H512**, and send the same command again.

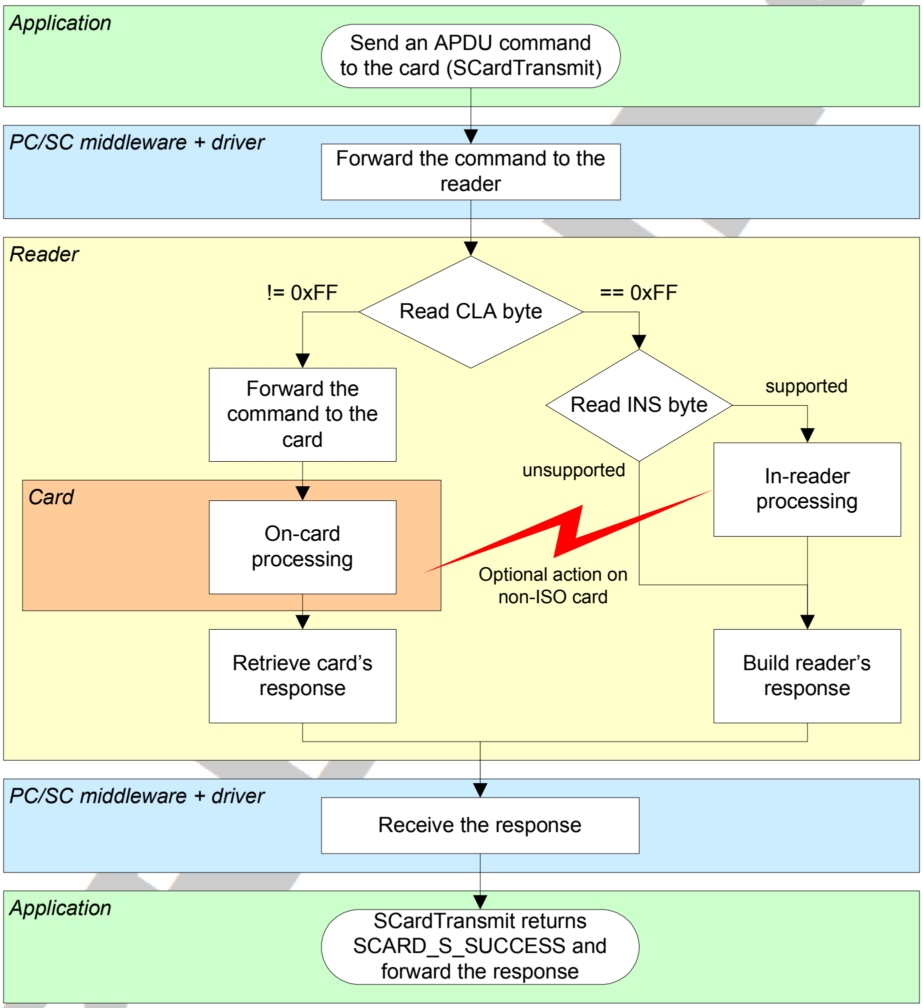
3. Embedded APDU interpreter

3.1. Basis

In PC/SC architecture, the ***SCardTransmit*** function implements the dialogue between an application and a smartcard, the reader being only a "passive" gateway. The reader transmits frames in both directions, without any specific processing. The dialogue 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 obeys 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.



3.1.1. CLA byte of the embedded APDU interpreter

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

a. Changing the CLA byte of the embedded APDU interpreter

The CLA byte of the embedded APDU interpreter is stored in register $_h$B2 of reader's non volatile memory (see § 5.1.2).

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

b. Disabling the embedded APDU interpreter

Define CLA byte = $_h$00 (register $_h$B2= $_h$00, see § 5.1.2) to disable the embedded APDU interpreter.

3.1.2. Status words returned by the embedded APDU interpreter

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

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

3.1.3. Embedded APDU interpreter instruction list

Instruction	INS	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
MIFARE CLASSIC READ	$_h$F3	D
MIFARE CLASSIC WRITE	$_h$F4	D
MIFARE CLASSIC VALUE	$_h$F5	D
CONTACTLESS SLOT CONTROL	$_h$FB	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

3.2. PC/SC standard instructions

3.2.1. GET DATA instruction

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

GET DATA command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hCA	See below	See below	-	-	h00

GET DATA command parameters

P1	P2	Action	
		Standard PC/SC-defined values	
h00	h00	Serial number of the PICC - ISO 14443-A : UID (4, 7 or 11 bytes) - ISO 14443-B : PUPI (4 bytes) - Innovatron : DIV (4 bytes) - Felica : IDm (8 bytes) - others: see chapter 6 for details	
		SpringCard specific values	
h01	h00	- ISO 14443-A : historical bytes from the ATS - ISO 14443-B : INF field in ATTRIB response - Felica : PMm (8 bytes) - ISO 18092 : G_T bytes in ATR_RES - others: see chapter 6 for details	
hF0	h00	Complete identifier of the PICC: - ISO 14443-A : ATQA (2 bytes) + SAK (1 byte) + UID - ISO 14443-B : complete ATQB (11 or 12 bytes)[6] - Innovatron : REPGEN - Felica : IDm and PMm (16 bytes) - ISO 18092 : complete ATR_RES - others: see chapter 6 for details	
hF1	h00	Type of the PICC, according to PC/SC part 3 supplemental document: PIX.SS (standard, 1 byte) + PIX.NN (card name, 2 bytes) See chapter 6.1 for details	

[6] 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.

P1	P2	Action	
hF1	h01	NFC Forum Tag[7] support: - h01 if the PICC is recognized as a NFC Forum Type 1 Tag - h02 if the PICC is recognized as a NFC Forum Type 2 Tag - h03 if the PICC is recognized as a NFC Forum Type 3 Tag - h00 otherwise	
hF2	h00	"Short" serial number of the PICC - ISO 14443-A : UID truncated to 4 bytes, in "classical" order - others: same as P1,P2=h00,h00	
hFA	h00	Card's ATR	
hFC	h00	PICC/PCD communication speeds on 2 bytes (DSI, DRI)	
hFF	h81	Vendor name in ASCII ("SpringCard")	
hFF	h82	Product name in ASCII	
hFF	h83	Product serial number in ASCII	
hFF	h84	Product USB identifier (VID/PID) in ASCII	
hFF	h85	Product version ("x.xx") in ASCII	

GET DATA response

Data Out	SW1	SW2
XX … XX	See below	

GET DATA status word

SW1	SW2	Meaning
h90	h00	Success
h62	h82	End of data reached before Le bytes (Le is greater than data length)
h6C	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.

3.2.2. 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
hFF	h82	Key location	Key index	h06	Key value	-

LOAD KEY command parameter P1 (key location)

P1	
h00	The key is to be loaded in reader's volatile memory
h20	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 = h00, 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 = h00 to P2 = h03 are "type A" keys,

P2 = h10 to P2 = h13 are "type B" keys.

When P1 = h20, 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 = h00 to P2 = h0F are "type A" keys,

P2 = h10 to P2 = h1F are "type B" keys.

Note there's no way to read-back 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 the **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

3.2.3. 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 is not a Mifare Classic!

GENERAL AUTHENTICATE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	h86	h00	h00	h05	See below	-

GENERAL AUTHENTICATE Data In bytes

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
h01	h00	Block number	Key location or Key type	Key index

The ***block number*** (byte 2) is the address on the Mifare card, where the application tries to be authenticated *(note: this is the block number, not the sector number)*.

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

- h60 for authentication using a CRYPTO1 "A" key *(standard PC/SC-defined value)*,
- h61 for authentication using a CRYPTO1 "B" key *(standard PC/SC-defined value),*
- Same value as the P1 parameter used in the LOAD KEY instruction: h00 or h20 *(SpringCard specific value)*.

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

- If *key type* (byte 3) is h60, use values h00 to h03 to select one of the "A" keys stored in reader's volatile memory, and values h20 to h2F to select one of the "A" keys stored in reader's non-volatile memory (if available),
- If *key type* (byte 3) is h61, use values h00 to h03 to select one of the "B" keys stored in reader's volatile memory, and values h20 to h2F to select one of the "B" keys stored in reader's non-volatile memory (if available),
- If *key type* (byte 3) is either h00 or h20 (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

3.2.4. READ BINARY instruction

The **READ BINARY** instruction retrieves data from a memory card (wired-logic PICC). Refer to chapter 6 for details.

For any PICC but Mifare Classic, this instruction is executed without any prerequisite.
For Mifare Classic, to be able to read the sector's data, the application must be authenticated on the card's sector. The application must therefore 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 (§ 3.3.1) 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 in its specific read command. Most PICC 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. 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 every time 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 supports 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
h90	h00	Success
h62	h82	End of data reached before Le bytes (Le is greater than data length)
h69	h81	Command incompatible
h69	h82	Security status not satisfied
h6A	h82	Wrong address (no such block or no such offset in the PICC)
h6C	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

3.2.5. UPDATE BINARY instruction

The **UPDATE BINARY** instruction writes data into a memory card (wired-logic PICC). Refer to chapter 6 for details.

For any PICC but Mifare Classic, this instruction is executed without any prerequisite.
For Mifare Classic, to be able to read the sector's data, the application must be authenticated on the card's sector. 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 (§ 3.3.2.) 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 in its specific write command. Most PICC 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. 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 every time 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
$_h$90	$_h$00	Success
$_h$69	$_h$82	Security status not satisfied
$_h$6A	$_h$82	Wrong address (no such block or no such offset in the PICC)
$_h$6A	$_h$84	Wrong length (trying to write too much data at once)

Important disclaimer

Most PICC have specific areas :
- that can 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 chip (lock bits),*
- and/or because writing an 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!

3.3. Vendor specific instructions for the contactless slot

3.3.1. 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 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 keys 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 (§ 3.2.4) for response and status words.

b. MIFARE CLASSIC READ selecting a key in the reader

In this mode, the application chooses one of 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
$_h$FF	$_h$F3	$_h$00	Block Number	$_h$02	Key Location or Type	Key Index	XX

The understanding and values for bytes ***Key location or Key type*** and ***Key index*** are documented in § 3.2.3 (GENERAL AUTHENTICATE instruction).

Refer to the READ BINARY instruction (§ 3.2.4) 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
$_h$FF	$_h$F3	$_h$00	Block Number	$_h$06	Key value (6 bytes)	XX

Refer to the READ BINARY instruction (§ 3.2.4) for response and status words.

3.3.2. 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 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 CLASSIC 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 (§ 3.2.5) 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 § 3.2.3 (GENERAL AUTHENTICATE instruction).

Refer to the UPDATE BINARY instruction (§ 3.2.5) 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 (§ 3.2.5) for response and status words.

3.3.3. MIFARE CLASSIC VALUE instruction

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 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$7FFFFFFF). The operand is subtracted from the current value of the source block, and the result is kept by the PICC in a register,
- RESTORE operation: the operand must be 0 ($_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 keys he knows (both permanent keys in E2PROM and temporary keys previously loaded in volatile memory) until one succeeds.

Because the reader must try all the keys, this method can 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 optionally 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 (§ 3.2.5) 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 optionally 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
hFF	hF5	Opcode	Source block	h07	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 § 3.2.3 (GENERAL AUTHENTICATE instruction).

Refer to the UPDATE BINARY instruction (§ 3.2.5) 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 optionally 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
hFF	hF5	Opcode	Source block	h0A	Operand (4B – MSB first)	Key value (6B)	-

MIFARE CLASSIC VALUE command APDU, key specified, with backup

CLA	INS	P1	P2	Lc	Data In			Le
hFF	hF5	Opcode	Source block	h0B	Operand (4B – MSB first)	Key value (6B)	Dest. block	-

Refer to the UPDATE BINARY instruction (§ 3.2.5) for response and status words.

3.3.4. 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 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
hFF	hFB	See below	See below	-	-	-

SLOT CONTROL command parameters

P1	P2	Action	
h00	h00	Resume the card tracking mechanism	
h01	h00	Suspend the card tracking mechanism	
h10	h00	Stop the RF field	
h10	h01	Start the RF field	
h10	h02	Reset the RF field (10ms pause)	
h20	h00	T=CL de-activation (DESELECT[10])	
h20	h01	T=CL activation of ISO 14443-A card (RATS)	
h20	h02	T=CL activation of ISO 14443-B card (Attrib)	
h20	h04	Disable the next T=CL activation[11]	
h20	h05	Disable every T=CL activation (until reset of the reader)	
h20	h06	Enable T=CL activation again	
h20	h07	Disable the next T=CL activation and force a RF reset	
hDE	hAD	Stop the slot NOTE: a stopped slot is not available to *SCardConnect* any more. It may be restarted only through an *SCardControl* command.	

[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

3.3.5. ENCAPSULATE instruction

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

ENCAPSULATE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FE	See below	See below	XX	Frame to send to the PICC	XX

ENCAPSULATE command parameter P1

P1	Standard communication protocols
$_h$00	For ISO 14443-4 (A or B) PICCs : send the frame in the T=CL stream[12]. Data In shall not include PCB, CID, NAD nor CRC fields For ISO 18092 targets : send the frame DEP_REQ/DEP_RES stream. Data In shall not include PFB, DID, NAD nor CRC fields
$_h$01	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.
$_h$02	Send the frame “as is” using the ISO 14443-3 B protocol. 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.

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.
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.

ENCAPSULATE command parameter P2

P2 encodes the frame timeout.

P2	Timeout value
$_h$-0	If P1 = $_h$00, use the default timeout defined by the PICC or the target (T=CL: card's FWT) If P1 ≠ $_h$00, 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

ENCAPSULATE response

Data Out	SW1	SW2
Frame received from the PICC or target	See below	

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 PICC
$_h$6C	XX	Le is shorter than actual response from PICC

3.4. Other vendor specific instructions

3.4.1. READER CONTROL instruction

The **READER CONTROL** instruction allows driving the global behaviour 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 4).

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 4.

Therefore, every command documented in § 4.4 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.

3.4.2. 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 analyses 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.

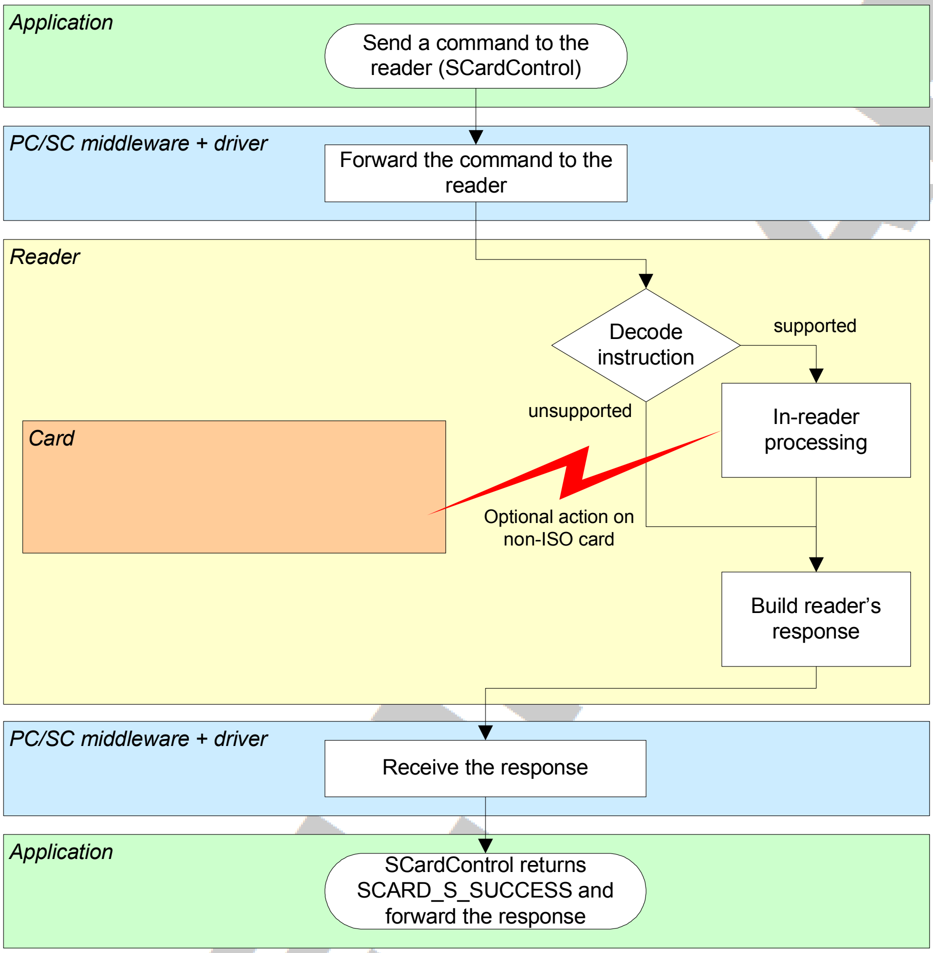
SW1	SW2	Meaning
$_h$90	$_h$00	Success, APDU correctly formatted
$_h$67	$_h$00	APDU is badly formatted (total length incoherent with Lc value)
$_h$6A	$_h$82	Le is greater than data length specified in P1
$_h$6C	P1	Le is shorter than data length specified in P1

4. Direct control of the reader

4.1. Basis

In PC/SC architecture, the ***SCardControl*** function implements the dialogue 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.



4.2. 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.

4.2.1. 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).*

4.2.2. 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 unequivocably 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).*

4.2.3. 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 unequivocably 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
                                    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).*

4.2.4. Direct control using PCSC-Lite CCID

To be written.

4.3. IMPLEMENTATION DETAILS

4.3.1. 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;
}
```

4.3.2. 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.

4.3.3. Format of response, return codes

When dialogue 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 12.

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

4.3.4. Redirection to the Embedded APDU Interpreter

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

4.4. List of available control sequences

4.4.1. Action on the LEDs

a. Setting the reader's LEDs manually

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:

$_h$00	LED is switched OFF
$_h$01	LED is switched ON
$_h$02	LED blinks slowly
$_h$04	LED blinks quickly
$_h$05	LED performs the “heart-beat” sequence

Once such a command has been sent to the reader, the firmware no longer manages the LEDs automatically: the LEDs remain permanently in the last state specified by the application.

Use the above command to make the firmware drive the LEDs automatically again.

b. Going back to default (LEDs managed by the reader's firmware)

Send the sequence

```
58 1E
```

To go back to default mode.

4.4.2. Action on the buzzer

a. Starting/stopping the 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.

Once such a command has been sent to the reader, the firmware no longer manages the buzzer automatically.

Use the above command to make the firmware drive the buzzer automatically again.

b. Going back to default (buzzer managed by the reader's firmware)

Send the sequence

```
58 1C
```

To go back to default mode.

4.4.3. 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

b. *Slot related information*

Sequence	Will return…
58 21	Name of the current slot ("Contactless")

4.4.4. Stopping / starting a slot

When a slot is stopped, the reader

- powers down the smartcard in the slot (if some),
- disable the slot[14],
- send the "card removed" event if there was a card in the slot.

When a slot is started again, the reader

- enable the slot[15],
- 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

b. Starting a slot

Sequence	Will return…
58 23	Start current slot

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

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

4.4.5. Reading/writing reader's configuration registers

The **H512** features a non-volatile memory to store configuration registers.

See chapter 5. 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. In this case, default value will be used.

The non-volatile memory has a limited write/erase endurance.
Writing any configuration register more than 100 times may permanently damage your product.

The value of the configuration registers is loaded by the H512's firmware upon reset only. To apply the new configuration, you must reset the H512 (or cycle power).

Alternatively, you may load temporary configuration settings as explained in the next paragraph.

4.4.6. Pushing a new temporary configuration

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

```
58 8D <index> <…data…>
```

Send an empty <data> (zero-length) to reload the default value.

This value will be applied immediately, but on next reset the H512 will reload its configuration registers from the non-volatile memory.

5. Configuration registers

5.1. Reader mode related registers

5.1.1. Protocol list

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

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

	Bit	Activ. protocol (if set)	
msb	15	RFU	
	14	RFU	
	13	RFU	
	12	Felica (NFC Forum's type 3 tags)	
	11	Innovision Topaz/Jewel (NFC Forum's type 1 tags)	
	10	RFU	
	9	RFU	
	8	RFU	
	7	Innovatron (legacy Calypso cards – sometimes called ISO 14443-**B'**)	
	6	*Not available in the **H512***	
	5	*Not available in the **H512***	
	4	*Not available in the **H512***	
	3	*Not available in the **H512***	
	2	*Not available in the **H512***	
	1	ISO 14443-B	
lsb	0	ISO 14443-A	

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

5.1.2. CLA byte of CCID interpreter

This register defines the CLA (class) byte affected to the APDU interpreter (see § 3.1.1).

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

Address: $_h$B2 – Size: 1 byte

Default value: $_h$FF

5.1.3. Misc. T=CL options

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

Address: $_h$B3 – Size: 1 byte

	Bit	Action if set	Note
msb	7	Innovatron: return the "real" T=0 ATR (as supplied in REPGEN) instead of building a pseudo ATR	Setting this bit breaks the compatibility with MS CCID driver, because the card is connected in T=1 where its ATR claims it is T=0 only[16]
	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$20,01 to activate the PICC manually
lsb	0	No T=CL activation over ISO 14443-A	Send SLOT CONTROL P1,P2=$_h$20,02 to activate the PICC manually

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

[16] 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.

5.1.4. T=CL speed limit

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

*The **H512** is theoretically able to communicate with PICCs at 848kbps in both directions, but the actual maximum speed depends heavily on the characteristics of the PICC, and on the reader's actual antenna and environment.*
Therefore, generally speaking, it is 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 (PICC → reader) = 848kbps
	13	Allow ISO 14443-A DS (PICC → reader) = 424kbps
	12	Allow ISO 14443-A DS (PICC → reader) = 212kbps
		ISO 14443-A DR
	11	RFU, must be 0
	10	Allow ISO 14443-A DR (reader → PICC) = 848kbps
	9	Allow ISO 14443-A DR (reader → PICC) = 424kbps
	8	Allow ISO 14443-A DR (reader → PICC) = 212kbps
		ISO 14443-B DS
	7	RFU, must be 0
	6	Allow ISO 14443-B DS (PICC → reader) = 848kbps
	5	Allow ISO 14443-B DS (PICC → reader) = 424kbps
	4	Allow ISO 14443-B DS (PICC → reader) = 212kbps
		ISO 14443-B DR
	3	RFU, must be 0
	2	Allow ISO 14443-B DR (reader → PICC) = 848kbps
	1	Allow ISO 14443-B DR (reader → PICC) = 424kbps
lsb	0	Allow ISO 14443-B DR (reader → PICC) = 212kbps

Default value: $_h$1111 (212kbps)[17].

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

5.2. PICC emulation mode related registers

5.2.1. ATQ, UID and SAK in NFC type 2 Tag emulation mode

Address: $_h$D2 – Size: 3, 7 or 13 bytes

Byte	Data	Default value	Remark
0	ATQ MSB	$_h$00	
1	ATQ LSB	$_h$44	
2	SAK	$_h$00	Bits 2 and 5 must be cleared
3 ..	UID	*empty*	Length of UID must be either 0, 4 or 7 If length is 0 (default), a 4-B random ID is used

5.2.2. ATQ, UID and SAK in NFC type 4 Tag emulation mode

Address: $_h$D3 – Size: 3, 7 or 13 bytes

Byte	Data	Default value	Remark
0	ATQ MSB	$_h$03	
1	ATQ LSB	$_h$44	
2	SAK	$_h$20	Bit 2 must be cleared Bit 5 must be set to claim ISO 14443-4 compliance
3 ..	UID	*empty*	Length of UID must be either 0, 4 or 7 If length is 0 (default), a 4-B random ID is used

5.2.3. Historical Bytes of the ATS in NFC type 4 Tag emulation mode

Address: $_h$D4 – Size: 0 to 15 bytes

If this register remains empty, the ATS in this mode has no Historical Bytes.

5.2.4. ATQ, UID and SAK in Card emulation mode

Address: $_h$E3 – Size: 3, 7 or 13 bytes

Byte	Data	Default value	Remark
0	ATQ MSB	$_h$03	
1	ATQ LSB	$_h$44	
2	SAK	$_h$20	Bit 2 must be cleared Bit 5 must be set to claim ISO 14443-4 compliance
3 ..	UID	*empty*	Length of UID must be either 0, 4 or 7 If length is 0 (default), a 4-B random ID is used

5.2.5. Historical Bytes of the ATS in Card emulation mode

Address: $_h$E4 – Size: 0 to 15 bytes

If this register remains empty, the ATS in this mode has no Historical Bytes.

5.3. ISO 18092 related registers

5.3.1. SENS_RES, SEL_RES, NFCID2 in NFC target mode

Address: $_h$E0 – Size: 11 bytes

Byte	Data	Default value	Remark
0	SENS_RES MSB	$_h$03	
1	SENS_RES LSB	$_h$44	
2	SEL_RES	$_h$04	Bits 2 must be set to claim ISO 18092 compliance
3 .. 10	NFCID2	$_h$01 $_h$FE $_h$53 $_h$70 $_h$72 $_h$69 $_h$6E $_h$67	NFCID2 must start with 01 FE to claim NFC-DEP compliance

Note: NFCID1 can't be personalized. A 4-byte random value is used.

5.3.2. Global Bytes in ATR_RES and ATR_REQ

Address: $_h$E1 – Size: 0 to 15 bytes

This register defines the G_I bytes when running in **reader** (NFC initiator) **mode**, and the G_T bytes when running in NFC target mode.

If this register remains empty, the default value is:

46 66 6D	LLCP magic number
01 01 11	LLCP version 1.1
03 02 00 13	Services = LLC Link Management + SNEP (NDEF exchange protocol)
04 01 96	Link timeout = 1.5 seconds

5.4. Misc control registers

5.4.1. Buzzer and LEDs settings

If the reader has some LEDs, the reader shows its state (card present, card absent, error) by its LEDs. You may disable this feature by setting bit 7 of this register to 1 (the application is still able to control the LEDs as documented in § 4.4.1.a.).

If the reader has a buzzer, the buzzer sounds every time a PICC is activated. The 6 low-order bytes of this register define the duration or this beep, in 10ms interval. To disable the automatic beep on card arrival, set this value to 0 (the application is still able to control the buzzer as documented in § 4.4.2.).

Address: $_h$CC – Size: 1 byte

	Bit	Values / Meaning
msb	7	1 : the reader does signal its state on the LEDs 0 : the reader doesn't signal its state on the LEDs
	6	RFU, must be 0
lsb	5	Duration of the automatic beep on card arrival, x 10ms (0 to 630ms) Set to $_h$00 to disable the automatic beep

Default value: $_h$88 (80ms beep on PICC arrival + state on LEDs)

6. WORKING WITH CONTACTLESS CARDS – USEFUL HINTS

6.1. RECOGNIZING AND IDENTIFYING PICC IN PC/SC ENVIRONMENT

6.1.1. 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)[18]:*

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[19].
…	…		
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[20] to make sure it will not start a new transaction on the same card as earlier.

[18] 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.

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

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

6.1.2. ATR of a wired-logic PICC

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	Protocol (see 6.1.4)
13	H10	PIX.NN	Card name (see 6.1.5)
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)

6.1.3. Using the GET DATA instruction

With the **GET DATA** instruction (documented in § 3.2.1), the host application is able to retrieve every information needed to identify a PICC:

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

6.1.4. Contactless protocol

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) Also ISO 18092 NFC-A
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	*Not available in the **H512***
0	0	0	0	1	0	1	1	*Not available in the **H512***
0	0	0	1	0	0	0	1	Felica Also ISO 18092 NFC-F

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.

6.1.5. 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$00	Unrecognised card, or card not in the list	
$_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$26	NXP Mifare Mini	
$_h$00 $_h$2F	Innovision Jewel	
$_h$00 $_h$30	Innovision Topaz (NFC Forum type 1 tag)	
$_h$00 $_h$3A	NXP Mifare UltraLight C Other Type 2 NFC Tags *(NFC Forum)* with a capacity > 64 bytes	
$_h$00 $_h$3B	Felica	
	SpringCard proprietary extension[21]	
$_h$FF $_h$C0	Calypso card using the Innovatron protocol	
$_h$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.

[21] 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.

6.2. ISO 14443-4 PICCs

6.2.1. Desfire first version (0.4)

Since this PICC is not ISO 7816-4 compliant, the Desfire commands must be wrapped in an ENCAPSULATED instruction, with P1=$_h$00 (§ 3.3.5). The reader translates the C-APDU into a native Desfire command, retrieve the native Desfire answer, and translates it into a valid R-APDU.

6.2.2. Desfire EV0 (0.6) and EV1

This PICC 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.

6.2.3. 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.

6.3. Wired-logic PICCs based on ISO 14443-A

6.3.1. 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 § 6.3.2).

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 stay at Mifare level:

- Send the T=CL DESELECT command to the PICC (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 PICC).

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 CLASSIC WRITE, always double check that you're not writing a sector trailer. 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 (§ 3.3.2) 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 § 3.3.1,
- MIFARE CLASSIC WRITE, see § 3.3.2,
- MIFARE CLASSIC VALUE (implementing INCREMENT, DECREMENT and RESTORE followed by TRANSFER), see § 3.3.3.

6.3.2. Mifare Plus X and Mifare Plus S

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

The **Mifare Plus** implements 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 PICC is ISO 14443-4 (T=CL) compliant. The reader builds a smartcard ATR according to § 6.1.1. The historical bytes of the ATS are included in the ATR and help recognizing the card at this level.

As the PICC is not ISO 7816-4 compliant, the commands shall be sent wrapped in an ENCAPSULATED instruction with P1=$_h$00 (§ 3.3.5).

At the end of the personalisation process, the RF field must be reset (so the PICC will restart at Level 1 or more). Send the SLOT CONTROL instruction with P1,P2=$_h$10,02 to do so (§ 3.3.4)[22].

b. Level 1

At level 1, the PICC emulates a Mifare Classic (§ 6.3.1). The reader builds a memory card ATR according to § 6.1.1.

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

The PICC supports a new AES authentication Function. Use the ENCAPSULATE instruction with P1=$_h$01 (§ 3.3.5) 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 must be manually started, even if the PICC announces that is is not T=CL compliant. Send the SLOT CONTROL instruction with P1,P2=$_h$20,01 to do so (§ 3.3.4). Afterwards, process as documented for level 0.

c. Level 2

The level 2 is not available on Mifare Plus S.

[22] 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).

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

d. Level 3

At level 3, the PICC is ISO 14443-4 (T=CL) compliant. The reader builds a smartcard ATR according to § 6.1.1. 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 commands shall be sent wrapped in an ENCAPSULATED instruction, with P1=$_h$00 (§ 3.3.5).

6.3.3. 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 PICC 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 1 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 formatted according to NFC Forum specification, the capacity is stored among other data in the 1st OTP page (CC – capability container bytes).
In any other case, the application may find the number of pages by sending READ BINARY instruction, incrementing the address, until it fails.
Pay attention that unfortunately some PICCs do 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. Since pages 0 and 1 store the UID (serial number) of the PICC, compare pages 16, 17 to pages 0, 1 to see 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 hold
- OTP (one-time-programming) bits,
- and/or lock bits that are intended 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 (§ 3.3.5) 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

6.3.4. 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 hold:
- OTP (one-time-programming) bits,
- and/or lock bits that are intended to make the PICC memory read only.

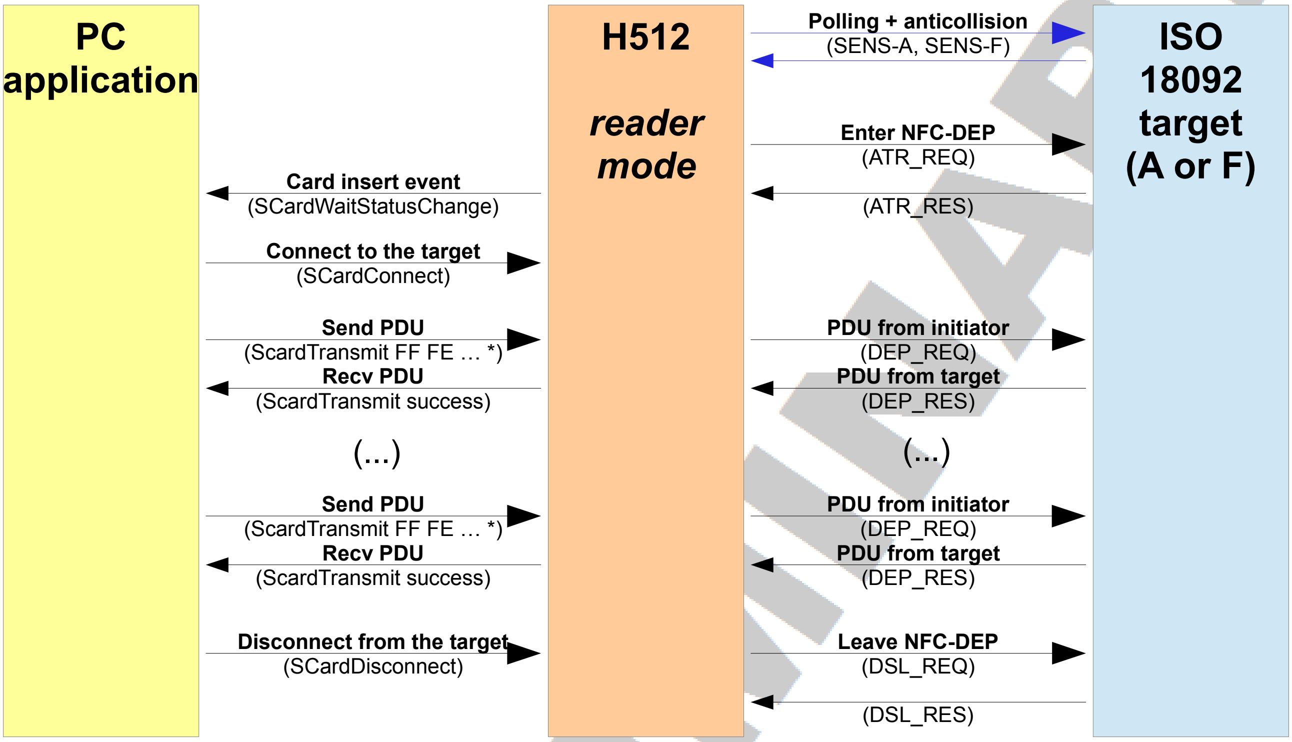
Do not write on those bytes without a good understanding of the consequences.

7. Reader mode: working with a P2P target

7.1. Introduction

When in **reader mode**, the **H512** is a NFC Initiator. It could activate a remote NFC Target (only the passive communication scheme is available).

The **H512** implements the ISO 18092 Transport Protocol (named NFC-DEP by NFC-Forum).



* The PDU must be ENCAPSULATEd if it doesn't meet ISO 7816-4 constraints.

7.1.1. Functions performed by the H512

The **H512** handles the NFC Transport Protocol internally:

- Transmission of *ATR_REQ* when a potential NFC Target has been detected, handing of *ATR_RES*,
- Initial exchange of parameters (*PSL_RES* / *PSL_RES*) if needed,
- Fragmentation of *DEP_REQ*, chaining of *DEP_RES*,
- Detection of transmission errors and recovery procedure,
- Detection of Target removal.

7.1.2. Functions to be implemented on the PC

In the NFC Forum architecture, NFC-DEP (ISO 18092) is seen as the low level transmission layer ("MAC") of an upper-level connection-oriented protocol, LLCP.

As the **H512** only implements ISO 18092, upper-level protocols and applications (for instance, LLCP and SNEP on top of LLCP) must be implemented by a PC application. **SpringCard SDK for PC/SC + NFC** provides various samples to do so. Please download this SDK from our web site.

Anyway, as support for LLCP must be claimed by the NFC initiator in its *ATR_REQ*, the **H512** has configurable G_I bytes, the default being the following value, compliant with LLCP:

46 66 6D 01 01 11 03 02 00 13 04 01 96

To change the G_I bytes, typically to disable LLCP, refer to § 7.3.1.

7.2. Mapping of the NFC functions into PC/SC functions

7.2.1. ATR of an ISO 18092 target

The reader builds a pseudo-ATR using the standard format defined in PC/SC specification:

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	h3B	Direct convention
1	T0	h8…	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	h80	Higher nibble 8 means: no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means: protocol T=0
3	TD2	h01	Higher nibble 8 means: no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means: protocol T=1
4	H1	…	G_T bytes from ATR_RES
…	…		
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

The target is LLCP compliant if ifs G_T bytes start with

46 66 6D

7.2.2. Using SCardTransmit (ENCAPSULATE) to exchange PDUs

ENCAPSULATE command APDU = DEP_REQ

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FE	$_h$00	$_h$00	XX	Transport data bytes	$_h$00

Up to 255 bytes of Transport data can be transmitted this way.

The **H512** adds the PFB (and the DID if required) and transmits a valid block. If the target's receive buffer is shorter than the actual size of the transport PDU, chained blocks are automatically. NAD is not supported.

During the reception of chained block, the **H512** re-assembles them and returns a single response. Up to 256 bytes of Transport data can be received.

ENCAPSULATE response = DEP_RES

Data Out	SW1	SW2
Transport data bytes	See below	

ENCAPSULATE status word

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$6F	XX	Error reported by the contactless interface. See chapter 6 for the list of possible values and their meaning.
$_h$62	$_h$82	Le is greater than actual response from target
$_h$6C	XX	Le is shorter than actual response from target

7.3. Advanced features

7.3.1. Changing the G_I bytes in the ATR_REQ

The General Bytes to be transmitted in the **H512**'s *ATR_REQ* (G_I bytes) when running in **reader mode** are stored in **register $_h$E1**.

*Note that the same value is used when the H512 is running in **target mode** for the G_T bytes in its ATR_RES (see chapter 8.).*

If this register remains empty, the default value is:

46 66 6D	LLCP magic number
01 01 11	LLCP version 1.1
03 02 00 13	Services = LLC Link Management + SNEP (NDEF exchange protocol)
04 01 96	Link timeout = 1.5 seconds

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new General Bytes before putting a new NFC target in front of the **H512**'s antenna.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

8. NFC Target mode

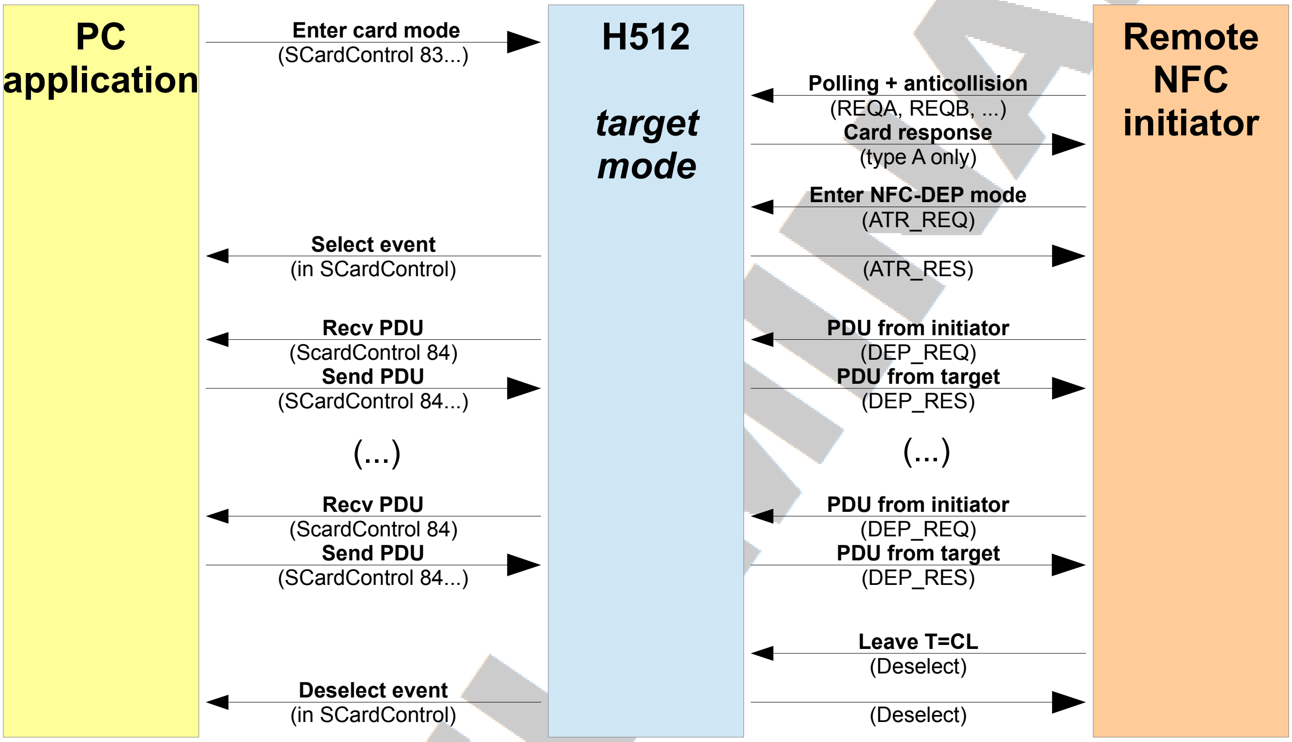
Note: NFC Target mode is not available in the early releases of the firmware (1.7x branch).

8.1. Introduction

In this mode, the **H512** is a NFC Target. It could be activated by a remote NFC Initiator (only the passive communication scheme is available).

The **H512** implements the ISO 18092 Transport Protocol (named NFC-DEP by NFC-Forum).

Upper-level protocols, such as NFC-Forum ***LLCP***, and upper-level services, for instance NFC-Forum ***SNEP***, shall be implemented by the PC-application.



8.1.1. Functions performed by the H512

The **H512** handles the NFC Transport Protocol internally:

- Transmission of *ATR_RES* when receiving *ATR_REQ*,
- Chaining of *DEP_REQ*, and fragmentation of *DEP_RES*,

- Handling of *PSL_REQ*,
- Handling of *DSL_REQ*,
- Detection of transmission errors and application of error-recovery rules.

8.1.2. Functions to be implemented on the PC

In the NFC Forum architecture, NFC-DEP (ISO 18092) is seen as the low level transmission layer ("MAC") of an upper-level connection-oriented protocol, LLCP.

As the **H512** only implements ISO 18092, upper-level protocols and applications (for instance, LLCP and SNEP on top of LLCP) must be implemented by a PC application. **SpringCard SDK for PC/SC + NFC** provides various samples to do so. Please download this SDK from our web site.

Anyway, as support for LLCP must be claimed by the NFC initiator in its *ATR_RES*, the **H512** has configurable G_T bytes, the default being the following value, compliant with LLCP:

46 66 6D 01 01 11 03 02 00 13 04 01 96

To change the G_T bytes, typically to disable LLCP, refer to § 7.3.1.

8.2. Technical data and limits

8.2.1. Characteristics of the contactless interface

Item	Value	Remark
Standard (Target)	ISO 18092 (*NFC-A* and *NFC-F*)	
DID support	yes	
NAD support	no	
Max size of incoming frame	64 bytes	
Max size of outgoing frame	64 bytes	
PDU buffer size	256 bytes	

8.2.2. Protocol data

a. NFC-A

Data	Value	Remark
SENS_RES	$_h$0344	Same as NXP Desfire
SEL_RES	$_h$04	ISO 18092 / NFC-DEP supported
UID	Random value on 4 bytes	

The SENS_RES and SEL_RES may be changed (see § 8.6.1.). Keeping the default values is highly recommended to remain compliant with ISO 18092.

b. NFC-F

Data	Value	Remark
NFCID2	$_h$01 $_h$FE + random value on 6 bytes	NFC-DEP supported
PAD0	$_h$FF $_h$FF	NFC Forum Digital specification, § 6.6.2.3
PAD1	$_H$00 $_h$00 $_h$00	
$MRTI_{CHECK}$	$_h$00	
$MRTI_{UPDATE}$	$_h$00	
PAD2	$_h$00	
RD	-	Not transmitted

The beginning of the NFCID2 may be changed (see § 8.6.1.). Keeping the default values is highly recommended to remain compliant with ISO 18092 and NFC Forum.

Other parameters can't be changed.

c. ATR_RES

Data	Value	Remark
NFCID3	$_h$01 $_h$FE $_h$53 $_h$70 $_h$72 $_h$69 $_h$6E $_h$67 $_h$00 $_h$00	The first 8 bytes are copied from NFCID2 The last 2 bytes are always $_h$00
BST	$_h$00	Active communication not supported
BRT	$_h$00	
TO	$_h$08	WT_{MAX} according to NFC Forum Digital specification, annex A.10
PP	$_h$00	LR = $_b$00 → maximum payload size is 64 bytes
G_T	$_h$46 $_h$66 $_h$6D $_h$01 $_h$01 $_h$11 $_h$03 $_h$02 $_h$00 $_h$13 $_h$04 $_h$01 $_h$96	NFC Forum LLCP "magic number" followed by LLCP parameters

The G_T bytes may be changed (see § 8.6.2.). Keeping the default values is highly recommended to remain compliant with ISO 18092 and NFC Forum.

Other parameters can't be changed.

8.3. Entering the NFC Target mode

Send the following command within a ***SCardControl*** function call:

ENTER NFC TARGET MODE command

Opcode		Parameters	
$_h$83	$_h$10	$_h$10	$_h$00

ENTER NFC TARGET MODE response

Status
$_h$00

8.4. Implementation on PC – basis

Every command depicted in this paragraphs goes through a ***SCardControl*** function call.

The model is event-driven: the PC application shall poll the **H512** to be notified of what is occurring on the NFC interface. 4 events are defined:

- ***Select:*** this event occurs when the **H512** has been selected by a remote NFC Initiator (*ATR_REQ* received, *ATR_RES* sent).
- ***PDU_REQ ready:*** this event occurs when the remote NFC Initiator has transmitted a PDU. The PC application shall retrieve the PDU, handle it, and provide its own PDU in response.
- ***PDU_RES done:*** this event notifies the PC application that its last Response has been transmitted to the remote NFC Initiator.
- ***Deselect:*** this event occurs either when the **H512** is deselected by the remote NFC Initiator (*DSL_REQ* received, *DSL_RES* sent) OR when the RF field disappears.

8.4.1. Polling the events of the Target

GET EVENT command

This is a 2-B command:

Opcode	
$_h$83	$_h$00

When this command is invoked, the **H512** sends an *EVENT response* immediately. The event code is $_h$00 (no event) in case there were no event in the queue.

WAIT EVENT control command

This is a 4-B command:

Opcode		Parameters	
$_h$83	$_h$00	*Time to wait*	
		MSB	LSB

Time to wait is a 2-B value in milliseconds. Value $_h$FFFF is RFU and shall not be used.

When this command is invoked,

- If there's already an event in the queue, an *EVENT response* is sent immediately,
- If an event occurs before *Time to wait*, an *EVENT response* is sent as soon as the event occurs,
- If *Time to wait* is reached before an event occurs, an *EVENT response* with event code = h00 (no event).

GET EVENT or WAIT EVENT response

This is a 3-B response:

Status	Data	
$_h$00	Event code	Flags

If the Status byte is not $_h$00, refer to § 8.5.

The EVENT CODE byte

Value	Name	Explanation
$_h$00	No event	No event occurred since last query
$_h$01	**SELECT**	
$_h$02	**PDU_REQ READY**	
$_h$03	**PDU_RES DONE**	
$_h$04	**DESELECT**	

Other values are RFU (and will not be returned by the **H512**)

The FLAGS byte

This byte is RFU. The PC application shall not try to handle its content.

8.4.2. The SELECT event

The event is fired when a remote NFC Initiator is ready to send PDUs to the **H512**.

The application will typically reset its state machine and security status. Then the application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

8.4.3. The PDU_REQ READY event – receiving the PDU from the Initiator

The event is fired when the **H512** has successfully received a complete *DEP_REQ* sequence from the remote PCD.

The application shall retrieve the Initiator's PDU by sending the *PEEK COMMAND* command, and provide its own PDU using the *POKE RESPONSE* command.

PEEK COMMAND command

This is a 1-B command:

Opco.
$_h$84

PEEK COMMAND response

*The size of the response is 1 + the size of the command buffer received by the **H512**:*

Status	Data
$_h$00	PDU Initiator to Target

If the Status byte is not $_h$00, refer to § 8.5.

Since the **H512**'s internal PDU buffer size is 254-B long, the application must be ready to receive a response with a length up to 254 bytes. If the provided buffer is too small, the ***SCardControl*** function will fail and the PDU will be lost.

8.4.4. Sending the Target's PDU to the Initiator

Once the application receives and processes the Initiator's PDU, it shall provide a valid Target's PDU to be transmitted to the remote NFC Initiator using the *POKE RESPONSE* command.

Then the application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

POKE RESPONSE command

The size of the command is 1 + the size of the response buffer to be transmitted:

Opco.	Data
$_h$84	PDU Target to Initiator

Since the **H512**'s internal PDU buffer size is 254-B long, the application must not try to send more than 255 bytes, including the opcode.

POKE RESPONSE response

Status
$_h$00

If the Status byte is not $_h$00, refer to § 8.5.

8.4.5. Handling the PDU_RES DONE event

This event confirms that the Target's PDU has been successfully sent to the remote NFC Initiator.

This event is informative only. The application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

8.4.6. Handling the DESELECT event

The event is fired when the communication channel with the NFC Initiator has been closed for any reason:

- Explicit use of the *DSL_RES* command by the Initiator,
- Fatal protocol error,
- RF field lost.

The application shall typically cancel any pending transaction, and reset its state machine and security status. This event is informative only. Then the application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

8.5. Error codes and recovery actions

Status	Symbolic name[23]	Meaning / action
h00	MI_SUCCESS	
h03	MI_EMPTY	No PDU_REQ available for PEEK COMMAND. (this could occur if the H512 has been deselected, or the external RF field has disappeared, between the return of GET EVENT/WAIT EVENT and the call to PEEK COMMAND)
h13	MI_OVFLERR	**RC: FIFO overflow** The remote NFC Initiator is trying to send us more than 280 bytes. The communication has been terminated, and the **emulation mode** has been suspended. Activate the **Target mode** again using ENTER NFC TARGET MODE command.
h3B	MI_WRONG_MODE	**Command not available in this mode** The opcode is not available in the current mode / the **H512** is not running in the expected mode. Activate the **Target mode** again using ENTER NFC TARGET MODE command.
h3C	MI_WRONG_PARAMETER	**Reader: Invalid parameter** The parameters of the command are not valid. You may try again with different parameters.
h70	MI_BUFFER_OVERFLOW	**Reader: internal buffer overflow** The PC application is trying to send more than 280 bytes in a *POKE REPONSE* command. Try again with a shorter buffer.
h7D	MI_WRONG_LENGTH	**Reader: invalid length** The length of the command is not valid. You may try again with a valid length.

Any other status code (see chapter 12. for a complete list) shall be considered as a fatal error, and the PC application shall stop using the reader.

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

8.6. Advanced features

8.6.1. Changing SENS_RES, SEL_RES and NFCID2

The ISO 18092 data related to single-device detection (SENS_RES, SEL_RES, and NFCID2) to be used in NFC target mode are stored in **register $_h$E0**. This register must be 5-byte long.

Byte	Data	Default value	Remark
0	SENS_RES MSB	$_h$03	
1	SENS_RES LSB	$_h$44	
2	SEL_RES	$_h$04	Bits 2 must be set to claim ISO 18092 compliance
3 .. 4	Beginning of NFCID2	$_h$01 $_h$FE	NFCID2 must start with 01 FE to claim ISO 18092 compliance

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new SENS_RES, SEL_RES and NFCID2 before invoking *ENTER NFC TARGET MODE* command.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

8.6.2. Changing G_T bytes in the ATR_RES

The General Bytes to be transmitted in the **H512**'s *ATR_RES* (G_T bytes) when running in NFC target mode are stored in **register $_h$E1**.

Note that the same value is used when the H512 is running in ***reader mode*** *for the G_I bytes in its ATR_REQ (see chapter 7.).*

If this register remains empty, the default value is:

46 66 6D	LLCP magic number
01 01 11	LLCP version 1.1
03 02 00 13	Services = LLC Link Management + SNEP (NDEF exchange protocol)
04 01 96	Link timeout = 1.5 seconds

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new General Bytes before putting a new NFC target in front of the **H512**'s antenna.

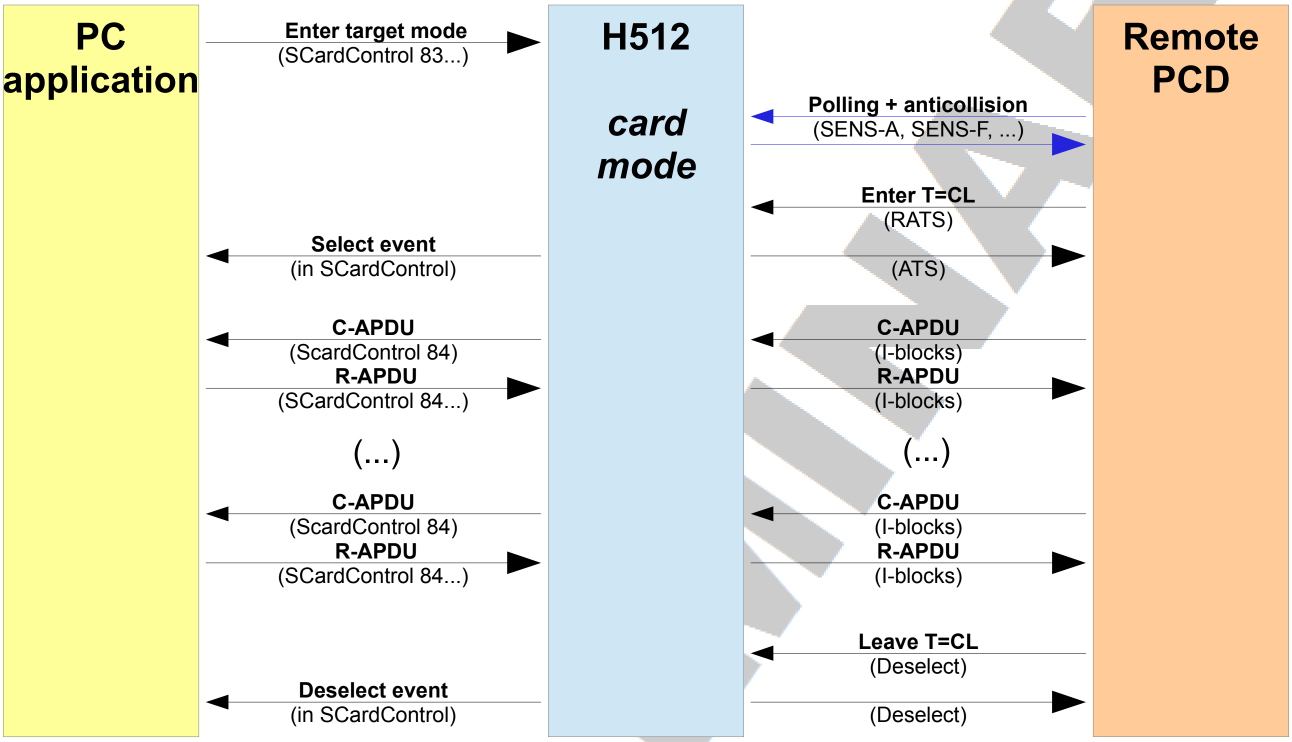
Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

9. Card emulation mode

9.1. Introduction

9.1.1. Basis

In this mode, the **H512** is a PICC. It could be activated by a remote PCD as if it were a contactless smartcard. The emulated Card application itself has to be implemented within a PC application.



9.1.2. Understanding the difference between NFC type 4 Tag emulation and Card emulation

When the NFC type 4 Tag emulation mode is selected, the **H512** emulates a very simple contactless smartcard with only 2 files. In this mode, all the layers (protocol, command set and data storage) are performed by the **H512**. The PC application may or may not be running, it makes no difference.

On the other hand, when this Card emulation mode is selected, the **H512** implements only the protocol (ISO 14443-4, type A). The upper layers (command set and data storage) must be handled by the PC application.

This architecture allows more flexibility, as virtually any smartcard application could be implemented that way, but due to the added complexity and to the the latency of the USB and PC/SC stacks, the speed of the transaction could be dramatically low.

9.1.3. Functions performed by the H512

The **H512** handles the T=CL protocol internally:

- Compliance with ISO 14443 type A PICC up to layer 4,
- Handling of Get ATS and PPS requests,
- Handling of DESELECT request,
- Reception and transmission of I-BLOCKs, with chaining up to 280 bytes of payload,
- Management of transmission errors through R-BLOCKs,
- Use of WTX *(Wait Time eXtension)* frames to keep the remote PCD waiting until the emulated Card application could provide an answer.

9.1.4. Functions to be implemented on the PC

The PC application shall behave as a smartcard application (or card operating system + applets), working on top of the ISO 14443-4 communication channel provided by the **H512**.

Generally speaking, the smartcard application will use the APDU format and the function that are both defined by ISO 7816-4, but it is also technically possible to implement virtually any other scheme not following those rules[24].

[24] For instance, the NXP Desfire EV0 is a contactless smartcard that communicates through the ISO 14443-4 protocol, but that has its a proprietary function set and a non-standard APDU format.

9.2. Technical data and limits

9.2.1. Characteristics of the contactless interface

Item	Value	Remark
Standard (PICC)	ISO 14443 type A up to layer 4	
CID support	yes	
NAD support	no	
Max size of incoming frame	64 bytes	ATS announces FSCI = 3
Max size of outgoing frame	64 bytes	
Max incoming baudrate	106kbit/s	ATS announces DRI = 1
Max outgoing baudrate	106kbit/s	ATS announces DSI = 1
Min SFGT	75ms	ATS announces SFGI = 8
Min FWT	75ms	ATS announces FWI = 8
APDU buffer size	280	

9.2.2. Protocol data

Data	Value	Remark
ATQ	$_h$0344	Same as NXP Desfire
SAK	$_h$20	Same as NXP Desfire
UID	Random value on 4 bytes	
ATS	$_h$75008802	FSCI = 5 TA1 = $_h$00 : only 106kpbs in both directions TB1 = $_h$88 : SFGI = FWI = 8 (≈ 77ms) TC1 = $_h$02 : CID supported, NAD not supported no Historical Bytes

The protocol data may be changed (see § 10.9.1. and § 10.9.2.). Keeping the default values is highly recommended to remain compliant with ISO 14443-4.

9.3. Activating the Card emulation mode

Send the following command within a ***SCardControl*** function call:

ACTIVATE EMULATION command

Opcode		Parameters	
h83	h10	h01	h00

ACTIVATE EMULATION response

Status
h00

9.4. Implementing the Card emulation on the PC

Every command depicted in this paragraph goes through a ***SCardControl*** function call.

The model is event-driven: the PC application shall poll the **H512** to be notified of what is occurring on the RF interface. 4 events are defined:

- ***Select:*** this event occurs when the emulated Card has been selected by a remote PCD, and that the dialogue has been established successfully.
- ***C-APDU ready:*** this event occurs when the remote PCD has transmitted a Command APDU to the emulated Card. The PC application shall retrieve the C-APDU, handle it, and provide a R-APDU.
- ***R-APDU done:*** this event notifies the PC application that its last Response has been transmitted to the remote PCD.
- ***Deselect:*** this event occurs either when the emulated Card is deselected by the remote PCD (ISO 14443-4 DESELECT command) OR when the RF field disappears.

9.4.1. Polling the events of the emulated Card

GET EVENT command

This is a 2-B command:

Opcode	
h83	h00

When this command is invoked, the **H512** sends an *EVENT response* immediately. The event code is h00 (no event) in case there were no event in the queue.

WAIT EVENT control command

This is a 4-B command:

Opcode		Parameters	
$_h$83	$_h$00	*Time to wait*	
		MSB	LSB

Time to wait is a 2-B value in milliseconds. Value $_h$FFFF is RFU and shall not be used.

When this command is invoked,

- If there's already an event in the queue, an *EVENT response* is sent immediately,
- If an event occurs before *Time to wait*, an *EVENT response* is sent as soon as the event occurs,
- If *Time to wait* is reached before an event occurs, an *EVENT response* with event code = h00 (no event).

GET EVENT or WAIT EVENT response

This is a 3-B response:

Status	Data	
$_h$00	Event code	Flags

If the Status byte is not $_h$00, refer to § 9.5.

The EVENT CODE byte

Value	Name	Explanation
$_h$00	No event	No event occurred since last query
$_h$01	**SELECT**	
$_h$02	**C-APDU READY**	
$_h$03	**R-APDU DONE**	
$_h$04	**UNSELECT**	

Other values are RFU (and will not be returned by the **H512**)

The FLAGS byte

This byte is RFU. The PC application shall not try to handle its content.

9.4.2. The SELECT event

The event is fired when a remote PCD is ready to send commands to the (emulated) Card application.

The application will typically reset its state machine and security status. Then the application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

9.4.3. The C-APDU READY event – receiving the C-APDU

The event is fired when the **H512** has successfully received a complete ISO 14443-4 sequence from the remote PCD.

The application shall retrieve the C-APDU (command buffer) by sending the *PEEK COMMAND* command, and provide a R-APDU (response buffer) using the *POKE RESPONSE* command.

PEEK COMMAND command

This is a 1-B command:

Opco.
$_h$84

PEEK COMMAND response

The size of the response is 1 + the size of the command buffer received by the ***H512****:*

Status	Data
$_h$00	C-APDU

If the Status byte is not $_h$00, refer to § 9.5.

Since the **H512**'s internal APDU buffer size is 280-B long, the application must be ready to receive a response with a length up to 281 bytes. If the provided buffer is too small, the ***SCardControl*** function will fail and the C-APDU will be lost.

9.4.4. Sending the R-APDU

Once the application receives and processes the C-APDU, it shall provide a valid R-APDU that will be transmitted to the remote PCD using the *POKE RESPONSE* command.

Then the application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

POKE RESPONSE command

The size of the command is 1 + the size of the response buffer to be transmitted:

Opco.	Data
$_h$84	R-APDU

Since the **H512**'s internal APDU buffer size is 280-B long, the application must not try to send more than 281 bytes, including the opcode.

POKE RESPONSE response

Status
$_h$00

If the Status byte is not $_h$00, refer to § 9.5.

9.4.5. Handling the R-APDU DONE event

This event confirms that the R-APDU has been successfully sent to the remote PCD.

This event is informative only. The application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

9.4.6. Handling the DESELECT event

The event is fired when the communication channel with the remote PCD has been close for any reason:

- Explicit use of the ISO 14443-4 DESELECT command,
- Fatal protocol error,
- RF field lost.

The application shall typically cancel any pending transaction, and reset its state machine and security status. This event is informative only. Then the application shall wait for the next event using either *WAIT EVENT* or *GET EVENT* commands.

9.5. Error codes and recovery actions

Status	Symbolic name[25]	Meaning / action
$_h$00	MI_SUCCESS	
$_h$03	MI_EMPTY	No C-APDU available for PEEK COMMAND. (this could occur if the H512 has been deselected, or the external RF field has disappeared, between the return of GET EVENT/WAIT EVENT and the call to PEEK COMMAND)
$_h$13	MI_OVFLERR	**RC: FIFO overflow** The remote PCD is trying to send us more than 280 bytes. The communication has been terminated, and the **emulation mode** has been suspended. Activate the Card emulation again using ACTIVATE EMULATION command.
$_h$3B	MI_WRONG_MODE	**Command not available in this mode** The opcode is not available in the current mode / the **H512** is not running in the expected mode. Activate the Card emulation again using ACTIVATE EMULATION command.
$_h$3C	MI_WRONG_PARAMETER	**Reader: Invalid parameter** The parameters of the command are not valid. You may try again with different parameters.
$_h$70	MI_BUFFER_OVERFLOW	**Reader: internal buffer overflow** The PC application is trying to send more than 280 bytes in a *POKE REPONSE* command. Try again with a shorter buffer.
$_h$7D	MI_WRONG_LENGTH	**Reader: invalid length** The length of the command is not valid. You may try again with a valid length.

Any other status code (see chapter 12. for a complete list) shall be considered as a fatal error, and the PC application shall stop using the reader.

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

9.6. Advanced features

9.6.1. Changing the ATQ, SAK

The ISO 14443-3 protocol data (ATQ and SAK) to be used in Card emulation mode are stored in **register $_h$E3**.

This register must be 3-byte long.

Byte	Data	Default value	Remark
0	ATQ MSB	$_h$03	
1	ATQ LSB	$_h$44	
2	SAK	$_h$20	Bit 2 must be cleared Bit 5 must be set to announce ISO 14443-4 compliance

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new ATQ and SAK before invoking *ACTIVATE CARD EMULATION*.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

9.6.2. Changing the Historical Bytes of the ATS

The **Historical Bytes** to be transmitted at the end of ISO 14443-4 Answer To Select (ATS) in Card emulation mode are stored in **register $_h$E4**.

If this register remains empty, the ATS has no Historical Bytes.

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new **Historical Bytes** before invoking *ACTIVATE CARD EMULATION*.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

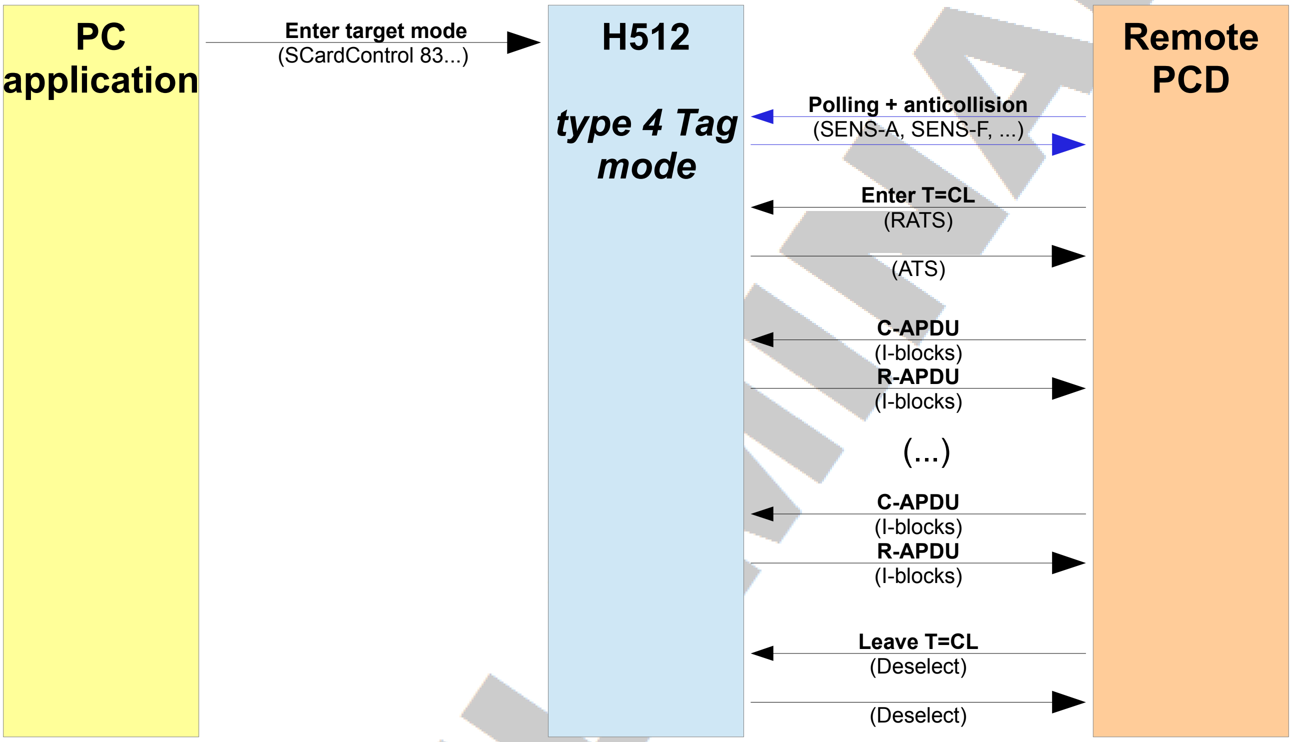
10. NFC type 4 tag emulation mode

10.1. Introduction

10.1.1. Basis

In this mode, the **H512** is a PICC. It could be activated by a remote PCD (a NFC reader typically), and be processed as a **NFC Forum type 4 Tag**.

The Tag's data are written by a PC application, and then the **H512** emulates the Tag without further action from the application.



10.1.2. The NFC Forum type 4 tag specification

A NFC Forum type 4 Tag is

- a contactless smartcard (microprocessor-based PICC),
- communicating through the ISO 14443 type A or B protocol up to level 4,
- implementing a few APDUs defined by ISO 7816-4,
- holding data that are formatted according to NFC Forum NDEF (NFC Data Exchange Format) specifications.

A NFC Forum type 4 tag is typically used to store a large Smart Poster or a vCard.

Please refer to the relevant NFC Forum document (NFCForum-TS-Type-4-Tag_2.0) for details.

10.1.3. Benefits of using the H512 in this mode

Running in this mode, the **H512** is seen as if it were a static Tag, with a fixed content, but is actually a dynamic Tag. This allow your application to deliver a variable content, following either a predefined event or an action from the user of the PC.

Your application could also be notified how many times the tag has been accessed.

Compared to NFC type 2 Tag emulation mode (chapter 11.), the NFC type 4 Tag emulation mode is a bit more complex for the developer of the PC application, but due to a more efficient communication scheme with the remote contactless reader, the transaction speed will be better.

10.2. Technical data and limits

10.2.1. Characteristics of the contactless interface

Item	Value	Remark
Standard (PICC)	ISO 14443 type A up to layer 4	
CID support	yes	
NAD support	no	
Max size of incoming frame	64 bytes	ATS announces FSCI = 3
Max size of outgoing frame	64 bytes	
Max incoming baudrate	106kbit/s	ATS announces DRI = 1
Max outgoing baudrate	106kbit/s	ATS announces DSI = 1
Min SFGT	75ms	ATS announces SFGI = 8
Min FWT	75ms	ATS announces FWI = 8

10.2.2. Protocol data

Data	Value	Remark
ATQ	$_h$0344	Same as NXP Desfire
SAK	$_h$20	Same as NXP Desfire
UID	Random value on 4 bytes	
ATS	$_h$75008802	FSCI = 5 TA1 = $_h$00 : only 106kpbs in both directions TB1 = $_h$88 : SFGI = FWI = 8 (≈ 77ms) TC1 = $_h$02 : CID supported, NAD not supported no Historical Bytes

The protocol data may be changed (see § 10.9.1. and § 10.9.2.). Keeping the default values is highly recommended to remain compliant with ISO 14443-4.

10.2.3. Command set

The **H512** implements those commands according to ISO 7816-4:

- SELECT APPLICATION / SELECT FILE (INS = $_h$A4)
- READ BINARY (INS = $_h$B0)
- UPDATE BINARY (INS = $_h$D6)

10.3. THE NDEF APPLICATION

The AID is $_h$D2760000850101.

The NDEF application is selected by default.

10.4. CC FILE

The Capability Container (CC) file stores the information that are required to identify the type of Tag.

10.4.1. File identifier

The identifier of the NDEF file is $_h$E103.

10.4.2. Size

The size of the CC file is 15 bytes.

10.4.3. Initial content

The initial content is loaded into the CC file every time the *First activation* command (§ 10.7.1.) is invoked.

Byte	Name	Initial value	Explanation
0	CCLEN	$_h$00	Size of CC file is 15 bytes
1		$_h$0F	
2	Version	$_h$20	Mapping Version 2.0
3	ML_E	$_h$00	Maximum R-APDU data size = 58
4		$_h$3A	
5	ML_C	$_h$00	Maximum C-APDU data size = 51
6		$_h$33	
7	NDEF File Control T	$_h$04	
8	NDEF File Control L	$_h$06	
9	NDEF File Control V	$_h$E1	The identifier of the NDEF file is $_h$E104
10		$_h$04	
11		$_h$10	The size of the NDEF file is 4096 bytes
12		$_h$00	
13		$_h$00	Read access granted without any security
14		$_h$FF or $_h$00	This value depends of the parameter to the *First activation* command (see § 10.7.1.) $_H$FF: No write access granted at all (read-only)[26] $_H$00: Write access granted without any security

10.5. NDEF file

10.5.1. File identifier

The identifier of the NDEF file is $_h$E104.

10.5.2. Size

The size of the CC file is 4096 bytes (4kB).

[26] On the contactless interface. The PC/SC interface has always a full access to the Tag's memory.

10.5.3. Initial content

Every time the *First activation* command (§ 10.7.1.) is invoked, all bytes of the NDEF file are initialized to $_h$00.

10.6. Access conditions

The byte 14 in the CC file is used to protect the Tag's content from being overwritten by a remote application. Allowed values are $_h$00 (Tag not protected) and $_h$FF (Tag is read-only). Any other value for this byte is RFU and shall not be used.

The PC application always has a read/write access on the Tag's files.

File	Access	PC application (PC/SC interface)	Remote application (contactless interface)	
			CC.14 = $_h$00	CC.14 = $_h$FF
CC	Read	yes	yes	yes
	Write	yes	yes	**no**
NDEF	Read	yes	yes	yes
	Write	yes	yes	**no**

10.7. Activating the NFC Forum type 4 Tag emulation mode

10.7.1. First activation – Tag's content is initialized

Send the following command within a ***SCardControl*** function call:

FIRST ACTIVATION TYPE 4 command, CC.14 is set to $_h$00

Opcode		Parameters	
$_h$83	$_h$10	$_h$04	$_h$11

FIRST ACTIVATION TYPE 4 command, CC.14 is set to $_h$FF

Opcode		Parameters	
$_h$83	$_h$10	$_h$04	$_h$12

FIRST ACTIVATION TYPE 4 response

Status
$_h$00

10.7.2. Following activation – Tag's content is preserved

Send the following command within a ***SCardControl*** function call:

NEXT ACTIVATION TYPE 4 command

Opcode		Parameters	
h83	h10	h04	h10

NEXT ACTIVATION TYPE 4 response

Status
h00

Note: invoking the *Following activation type 4* command without having a previously invoked *First activation type 4* is an error. The content of the memory is not defined in this case.

10.8. Accessing the emulated Tag's content through PC/SC

10.8.1. ATR

The ATR is:

ATR
H3B 80 01 81

(T=1 protocol only, no historical bytes)

10.8.2. Selecting the application and files, reading and writing data

The emulated Tag behaves as a ISO 7816-4 smartcard. Standard *SELECT*, *READ BINARY* and *UPDATE BINARY* commands (with CLA = h00) shall be used.

Please refer to ISO 7816-4 or to the NFC Forum document for details.

a. Selecting the NFC Forum NDEF application

Use *SCardTransmit* to send the APDU:

CLA	INS	P1	P2	Lc	Data In	Le
h00	hA4	h04	h00	h07	HD2 76 00 00 85 01 01	h00

b. Selecting the CC file

Use *SCardTransmit* to send the APDU:

CLA	INS	P1	P2	Lc	Data In	Le
$_h$00	$_h$A4	$_h$00	$_h$0C	$_h$02	$_H$E1 03	-

c. Selecting the NDEF file

Use *SCardTransmit* to send the APDU:

CLA	INS	P1	P2	Lc	Data In	Le
$_h$00	$_h$A4	$_h$00	$_h$0C	$_h$02	$_H$E1 04	-

d. Reading from the currently select file

Use *SCardTransmit* to send the APDU:

CLA	INS	P1	P2	Lc	Data In	Le
$_H$00	$_h$B0	Offset MSB	Offset LSB	-	-	Length

Note: L_E shall be ≤ 58.

e. Writing into the currently select file

Use *SCardTransmit* to send the APDU:

CLA	INS	P1	P2	Lc	Data In	Le
$_H$00	$_h$D6	Offset MSB	Offset LSB	Length	Data to be written	-

Note: L_C shall be ≤ 51.

10.9. Advanced features

10.9.1. Changing the ATQ and SAK

The ISO 14443-3 protocol data (ATQ and SAK) to be used in NFC type 4 Tag emulation mode is stored in **register $_h$D3**.

This register must be 3-byte long.

Byte	Data	Default value	Remark
0	ATQ MSB	$_h$03	
1	ATQ LSB	$_h$44	
2	SAK	$_h$20	Bit 2 must be cleared Bit 5 must be set to announce ISO 14443-4 compliance

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new ATQ and SAK <u>before</u> invoking either *FIRST ACTIVATION TYPE 4* or *NEXT ACTIVATION TYPE 4*.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

10.9.2. Changing the ATS

The **Historical Bytes** to be transmitted at the end of ISO 14443-4 Answer To Select (ATS) in NFC type 4 Tag emulation mode are stored in **register $_h$D4**.

If this register remains empty, the ATS has no **Historical Bytes**.

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new **Historical Bytes** <u>before</u> invoking *ACTIVATE CARD EMULATION*.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

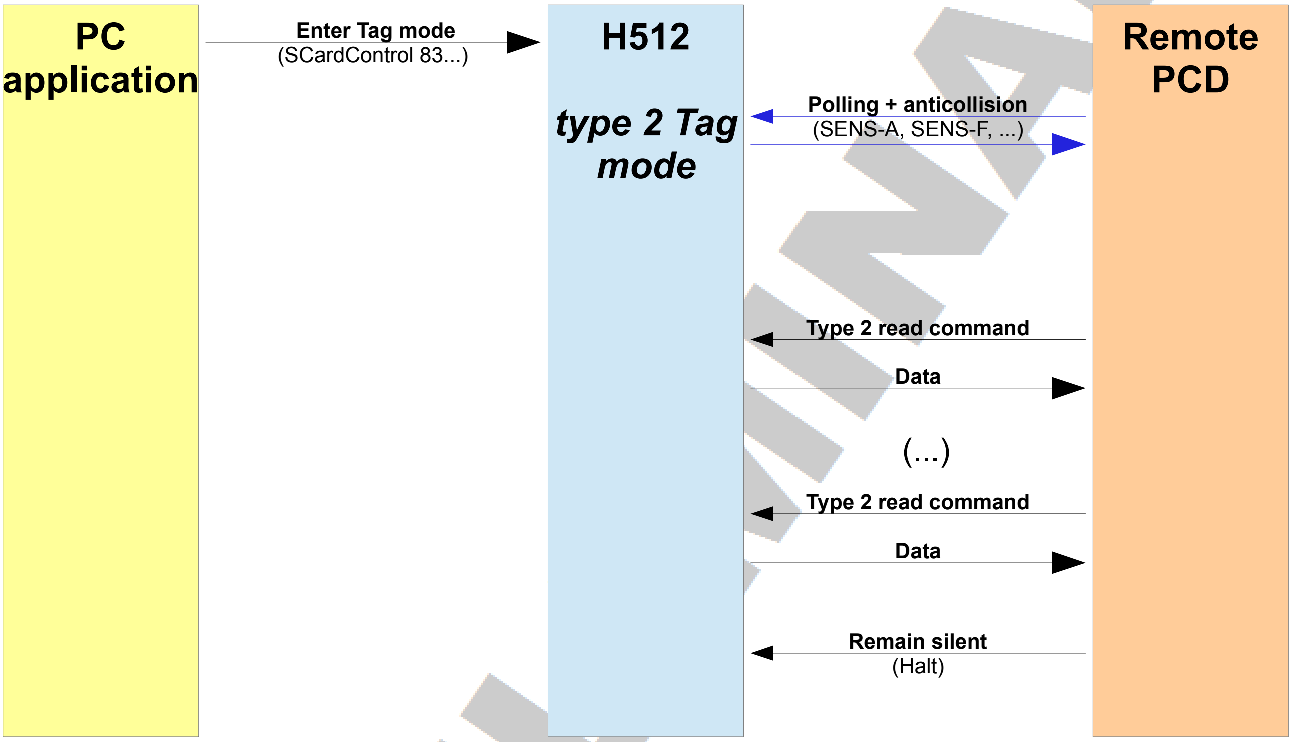
11. NFC type 2 Tag emulation mode

11.1. Introduction

11.1.1. Basis

In this mode, the **H512** is a PICC. It could be activated by a remote PCD (a NFC reader typically), and be processed as a **NFC Forum type 2 Tag**.

The Tag's data are written by a PC application, and then the **H512** emulates the Tag without further action from the application.



11.1.2. The NFC Forum type 2 Tag specification

A NFC Forum type 2 Tag is

- a contactless memory card (wired-logic PICC),
- communicating through the ISO 14443 type A protocol up to level 3,
- implementing the command set defined by NXP for the Mifare UltraLight family,
- holding data that are formatted according to NFC Forum NDEF (NFC Data Exchange Format) specifications.

A NFC Forum type 2 tag is typically used to store an URL, or an URL plus some related information (Smart Poster format).

Please refer to the NFC Forum relevant document (NFCForum-TS-Type-2-Tag_1.1) for details.

11.1.3. Benefits of using the H512 in this mode

Running in this mode, the **H512** behaves as a static Tag, with a fixed content. But it is actually a dynamic Tag. This allow your application to deliver a variable content, following either a predefined event or an action from the user of the PC.

Compared to NFC type 4 Tag emulation mode (chapter 10.), the NFC type 2 Tag emulation mode is easier for the developer of the PC application, but the remote PCD will read the data slower. This is not an issue for small amount of data (256 bytes or less), but for large amount of data and if transaction time is a concern, NFC type 4 emulation mode shall be preferred.

11.2. Technical data and limiting values

11.2.1. Characteristics of the contactless interface

Item	Value
Standard (PICC)	ISO 14443 type A up to layer 3
Max size of incoming frame	64 bytes
Max size of outgoing frame	64 bytes

11.2.2. Protocol data

Data	Value	Remark
ATQ	$_h$0044	Same as NXP Mifare UltraLight / NTAG203
SAK	$_h$00	Same as NXP Mifare UltraLight / NTAG203
UID	Random value on 4 bytes	

The protocol data may be changed (see § 11.6.1.). Keeping the default values is highly recommended to remain compliant with third-party NFC Tag reading applications.

11.2.3. Capacity

The capacity of the emulated Tag is 256 blocks x 4 bytes per block, i.e. a total of 1024 bytes.

The NDEF content starts on block 4, so the available size for the NDEF is 1008 bytes.

11.3. Memory mapping

11.3.1. Structure

Block #		Usage	Byte 0	Byte 1	Byte 2	Byte 3
0	h00	Not used	*??*	*??*	*??*	*??*
1	h01	Not used	*??*	*??*	*??*	*??*
2	h02	Internal / Lock	*Counter*	*Counter*	h00	h00
3	h03	Capability Container (CC)	CC0	CC1	CC2	CC3
4	h04	NDEF storage	data	data	data	data
5	h05		data	data	data	data
...			...			
255	hFF		data	data	data	data

11.3.2. Access conditions – role of CC3

CC3 is used to protect the Tag's content from being overwritten by a remote application. Allowed values are $_{h}$00 (Tag not protected) and $_{h}$0F (Tag is read-only). Any other value for the CC3 byte is RFU and shall not be used.

The PC application always has a read/write access on the whole structure.

Blocks	Access	PC application (PC/SC interface)	Remote application (contactless interface)	
			CC3 = $_{h}$00	CC3 = $_{h}$0F
0, 1, 2	Read	yes	yes	yes
	Write	yes	**no**	**no**
3	Read	yes	yes	yes
	Write	yes	yes	**no**
4 to 255	Read	yes	yes	yes
	Write	yes	yes	**no**

11.3.3. Initial content

The initial content is loaded into the memory structure every time the *First activation* command (§ 11.4.1.) is invoked.

a. Blocks 0 and 1

The initial content of these blocks is not specified and should not be used.

b. Block 2

All bytes are initialized to $_h$00.

c. CC block (block 3)

The CC block is initialized as follow, according to the specifications:

Byte	Initial value	Explanation
CC0	$_h$E1	NFC Forum "Magic Number"
CC1	$_h$10	Version of the specification: 1.0
CC2	$_h$7E	Memory size = 1008 bytes
CC3	$_h$0F or $_h$00	This value depends of the parameter to the *First activation* command (see § 11.4.1.) Read access granted without any security $_h$0F: No write access granted at all[27] $_h$00: Write access granted without any security

d. NDEF data area (blocks 4 to 255)

All bytes are initialized to $_h$00 (blank Tag).

11.4. Activating the NFC Forum type 2 Tag emulation mode

11.4.1. First activation – Tag's content is initialized

Send the following command within a ***SCardControl*** function call:

[27] On the contactless interface. The PC/SC interface has always a full access to the Tag's memory.

FIRST ACTIVATION TYPE 2 command, CC3 is set to $_h$00

Opcode		Parameters	
$_h$83	$_h$10	$_h$02	$_h$11

FIRST ACTIVATION TYPE 2 command, CC3 is set to $_h$0F

Opcode		Parameters	
$_h$83	$_h$10	$_h$02	$_h$12

FIRST ACTIVATION TYPE 2 response

Status
$_h$00

11.4.2. Following activation – Tag's content is preserved

Send the following command within a ***SCardControl*** function call:

NEXT ACTIVATION TYPE 2 command

Opcode		Parameters	
$_h$83	$_h$10	$_h$02	$_h$10

NEXT ACTIVATION TYPE 2 response

Status
$_h$00

Note: invoking the *Following activation type 2* command without having a previously invoked *First activation type 2* is an error. The content of the memory is not defined in this case.

11.5. Accessing the emulated Tag's content through PC/SC

Once the Tag is activated (*First activation* or *Following activation*), the **H512** reports that a card has been inserted in its contactless interface. This is a virtual card that gives access to the emulated Tag through ***SCardConnect*** and ***SCardTransmit*** functions.

11.5.1. ATR

The ATR is the same as the one of a NXP Mifare UltraLight C PICC:

ATR
$_H$3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 3A 00 00 00 00 51

11.5.2. Reading and writing data

The emulated Tag is accessed through the Embedded APDU Interpreter's *READ BINARY* and *UPDATE BINARY* commands (CLA = $_h$FF) as if it were a Mifare UltraLight PICC.

Refer to § 6.3.3. for details.

11.6. Advanced features

11.6.1. Changing the ATQ and SAK

The protocol data to be used in NFC type 2 Tag emulation mode are stored in **register $_h$D2**.

This register must be 3-byte long.

Byte	Data	Default value	Remark
0	ATQ MSB	$_h$00	
1	ATQ LSB	$_h$44	
2	SAK	$_h$00	Bits 2 and 5 must be cleared

Use the *PUSH REGISTER* command (§ 4.4.6.) to set the new ATQ and SAK before invoking either *FIRST ACTIVATION TYPE 2* or *NEXT ACTIVATION TYPE 2*.

Alternatively, use the *WRITE REGISTER* command (§ 4.4.5.) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

12. 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[28]	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$03	MI_EMPTY	No data available
$_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 anti-collision 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	Anti-collision not supported by the card(s)
$_h$1D	MI_EXTERNAL_FIELD	An external RF field is already present, unable to activate the reader's RF field

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

h1F	MI_CODINGERR	Bad card status
h20	MI_CUSTERR	Card: vendor specific error
h21	MI_CMDSUPERR	Card: command not supported
h22	MI_CMDFMTERR	Card: format of command invalid
h23	MI_CMDOPTERR	Card: option of command invalid
h24	MI_OTHERERR	Card: other error
h3C	MI_WRONG_PARAMETER	Reader: invalid parameter
h64	MI_UNKNOWN_FUNCTION	Reader: invalid opcode
h70	MI_BUFFER_OVERFLOW	Reader: internal buffer overflow
h7D	MI_WRONG_LENGTH	Reader: invalid length

13. VENDOR ATTRIBUTES

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

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.

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