PMD2271-BC
- PUBLIC

SpringCard PC/SC Couplers - H663 group

Developer's reference manual

www.springcard.com

Document Identification

Category	Developer's manual		
Family/Customer	PC/SC Couplers		
Reference	PMD2271	Version	BC
Status		Classification	Public
Keywords	H663, CrazyWriter HSP, TwistyWriter HSP, CSB HSP, Prox'N'Roll HSP, FunkyGate HSP, PC/SC, NFC P2P, contactless cards, RFID labels, NFC tags		
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\H663 Group\Developpement\[PMD2271-BC] H663 Developer's Reference Manual.odt		
Date saved	14/03/17	Date printed	05/12/12

Revision History

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	29/08/12	JDA				Created from PMD841P-FA and PMD2176-AD
AB	18/03/13	JDA				Separated chapter 2 from chapter 1 Renumbered the chapters (moved up “contactless hints” and “NFC initiator role”), named last chapters “annex”. New drawings Documented all the configuration registers Removed the preliminary Watermark
AC	21/05/13	JDA				Added the capability to stop the contactless slot when a contact smartcard is inserted
AD	23/07/13	JDA				Documented new behaviour of firmware ≥ 1.75 for NFC Forum Type 1 tags, including Innovision (now Broadcom) Topaz and alike
AE	17/06/14	JDA				Added time-out extensions for T=0 and T=1 Improved chapter 7
AF	27/08/14	JDA				A few rewordings Order codes and related documents added
AG	07/11/14	JDA				New features from FW 1.81: - 'RFID memory control' instruction (INS=F6) - Support of EM5134 - Recognition of Mifare Plus cards, updated the PIX.NN listing
BA	13/04/15	JDA				New features from FW 2.00: - buzzer - name of reader could be the serial number Added FreeRTOS licence information Added Prox'N'Roll HSP
BB	06/06/16	JDA				Description of register C8 was not correct
BC	14/03/17	LCO				Kovio → ThinFilm

Contents

1. Introduction

1.1. Abstract

SpringCard H663 is a PC/SC RFID and NFC coupler module, featuring 0 to 5 optional T=0/T=1 interfaces for contact smartcards or SIM/SAM. The **H663** module is the core of numerous PC/SC Couplers offered by **SpringCard**, and also of specific readers designed by OEMs.

This document provides all necessary information to develop software that will use the **H663** core.

1.2. Supported products

At the time of writing, this document refers to all **SpringCard PC/SC Couplers** in the **H663** group:

- The **H663S** and **H663A**: OEM modules without antenna,
- The **H663-USB**: OEM coupler with integrated antenna based on the **H663S**,
- The **CrazyWriter HSP**: multi-interface OEM coupler based on the **H663A**,
- The **TwistyWriter HSP**: OEM coupler with remote antenna based on the **H663S**,
- The **Prox'N'Roll HSP**: desktop coupler,
- The **CSB HSP LT**: large desktop coupler based on the **H663S**,
- The **CSB HSP**: multi-interface desktop coupler based on the **H663S.**

Product list with order codes

Product name	Order code	Description
H663S	SC14182	Contactless PC/SC OEM module for balanced antenna
H663SC	SC14183	Contact & contactless PC/SC OEM module for balanced antenna
H663A	SC14184	Contactless PC/SC OEM module for unbalanced antenna
H663AC	SC14185	Contact & contactless PC/SC OEM module for unbalanced antenna
H663-USB	SC3016	Contactless PC/SC OEM coupler with integrated antenna
CrazyWriter HSP	SC0168	Contact & contactless PC/SC OEM coupler with 1 x SAM + 1 x remote antenna (50Ω) *(expansion board for 3 more SAM available as an option)*
CrazyWriter HSP Dual	SC14148	Contact & contactless PC/SC OEM coupler with 1 x SAM + 2 x remote antennas (50Ω) *(expansion board for 3 more SAM available as an option)*
TwistyWriter HSP	SC14190	Contactless PC/SC OEM coupler with remote antenna (TP)
Prox'N'Roll HSP PC/SC	SC15131	Desktop contactless PC/SC coupler
CSB HSP	SC0177	Desktop contact & contactless PC/SC coupler with 3 x SAM + 1 x ID-1 slot
CSB HSP LT	SC14048	Desktop contactless PC/SC coupler

1.3. Related documents

a. End users

Editor	Doc #	Description
SpringCard	PMU14186	CSB HSP, CSB HSP LT QuickStart Guide
SpringCard	PMU14092	CrazyWriter HSP QuickStart Guide
SpringCard	PMU14189	TwistyWriter HSP QuickStart Guide

b. Integrators

Editor	Doc #	Description
SpringCard	PMD2236	H663 Hardware Integration guide
SpringCard	PNA14187	CrazyWriter HSP Hardware Integration guide
SpringCard	PNA14188	TwistyWriter HSP Hardware Integration guide

1.4. Audience

This manual is designed for use by application developers. It assumes that the reader has expert knowledge of computer development and a basic knowledge of PC/SC, of the ISO 7816-4 standard for smartcards, and of the NFC Forum's specifications.

Chapter 2 provides a quick introduction to those technologies and concepts, but can't cover all the aspects, as would a book or a training session.

1.5. 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

1.6. 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

2. PC/SC, smartcards and NFC: quick introduction and glossary

2.1. Smart-card and contactless smartcards standards

A **smartcard** is a microprocessor (running a software of course) mounted in a plastic card.

The **ISO 7816** family of standards defines everything for contact smartcards:

- **ISO 7816-1** and **ISO 7816-2** defines the form-factor and electrical characteristics,
- **ISO 7816-3** introduces two transport-level protocols between the coupler and the card: "T=0" and "T=1",
- **ISO 7816-4** mandates a common function set. This function set exposes the smartcard as a small file-system, with directories and files, where the data are stored. The application-level frames are called **APDU**s.

The **ISO 14443** family is the normative reference for contactless smartcards:

- **ISO 14443-1** and **ISO 14443-2** defines the form-factor, RF characteristics, and bit-level communication,
- **ISO 14443-3** specifies the byte- and frame-levels part of the communication[1],
- **ISO 14443-4** introduces a transport-level protocol that more-or-less looks like T=1, so it is often called "T=CL" (but this name never appears is the standard).

On top of T=CL, the **contactless smartcard** is supposed to have the same function set and APDUs formatting rules as **contact smartcard**, i.e. it should be "ISO 7816-4 on top of ISO 14443".

In this context, working with a smartcard (either contact or contactless) is as easy as sending a command (C-APDU) to the card, and receive its response (R-APDU). The "smartcard reader" is only a gateway that implements this **APDU exchange** stuff (with a relative abstraction from the transport-level protocols).

[1] ISO 14443-2 and -3 are divided into 2 technologies: ISO 14443 type A and ISO 14443 type B. They use different codings and low-level protocols, but the transport protocol defined in ISO 14443-4 is type-agnostic: it makes no difference whether the card is type A or type B.

2.2. PC/SC

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

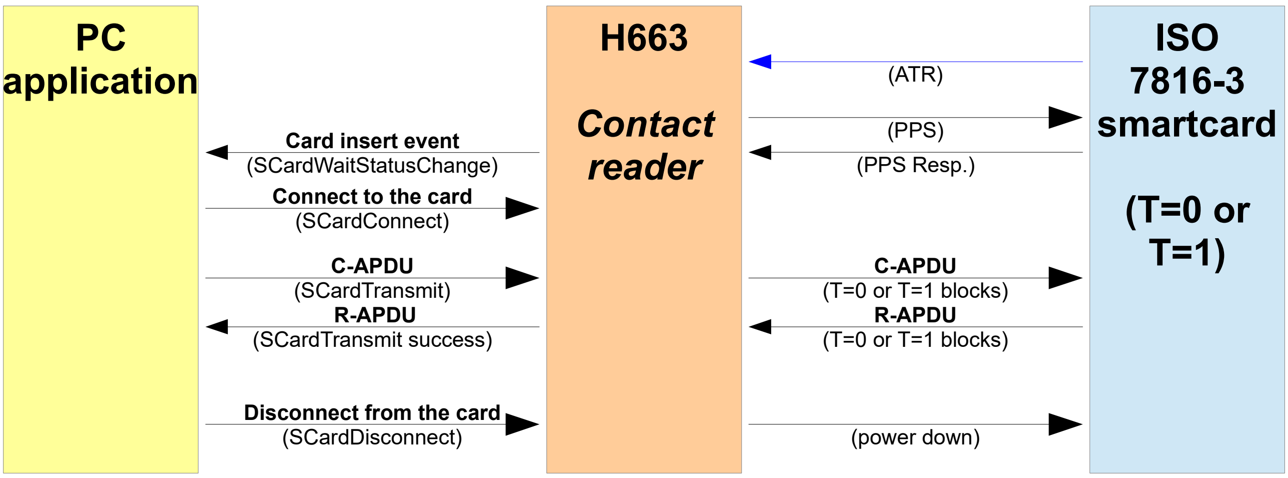
To get started with PC/SC, please read our Introduction to PC/SC development and simplified documentation of the API, available online at

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

The heart of PC/SC is the *SCardTransmit* function, that is the implementation in the computer of the **APDU exchange** stuff.

If the smartcard you are working with does comply with ISO 7816-4, there is nothing more to add! Refer to the ISO 7816-4 standard and/or to the documentation of the card[2] to know the APDUs you must send, and how to understand the responses. Then implement your card-aware process through a batch of SCardTransmit calls. Whether the smartcard is contactless or contact makes little to no difference[3].

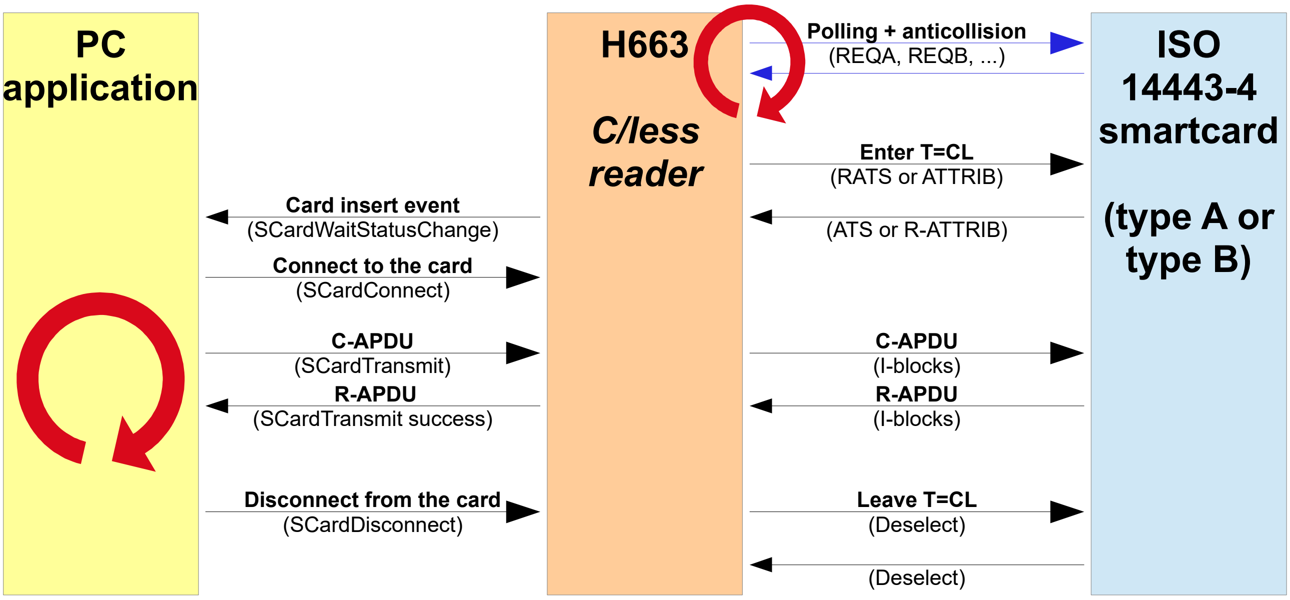
a. PC/SC and a contact smartcard



[2] Note that a microprocessor-based smartcard is a chip plus some application running in it. It could be a monolithic application, without an operating system, or an operating system (for instance JavaCard) with some applications added. You need the documentation of the application(s) and in some situations the documentation of the operating system, not the chip's.

[3] Actually there's more differences between contact protocols T=0 and T=1 than between contactless protocol “T=CL” and T=1. When developing an application for a contactless smartcard, read the ISO 7816-4 standard and the documentation of the smartcard as if it were running T=1.

b. PC/SC and a contactless smartcard



2.3. Non-7816-4 contactless cards – introducing the embedded APDU interpreter

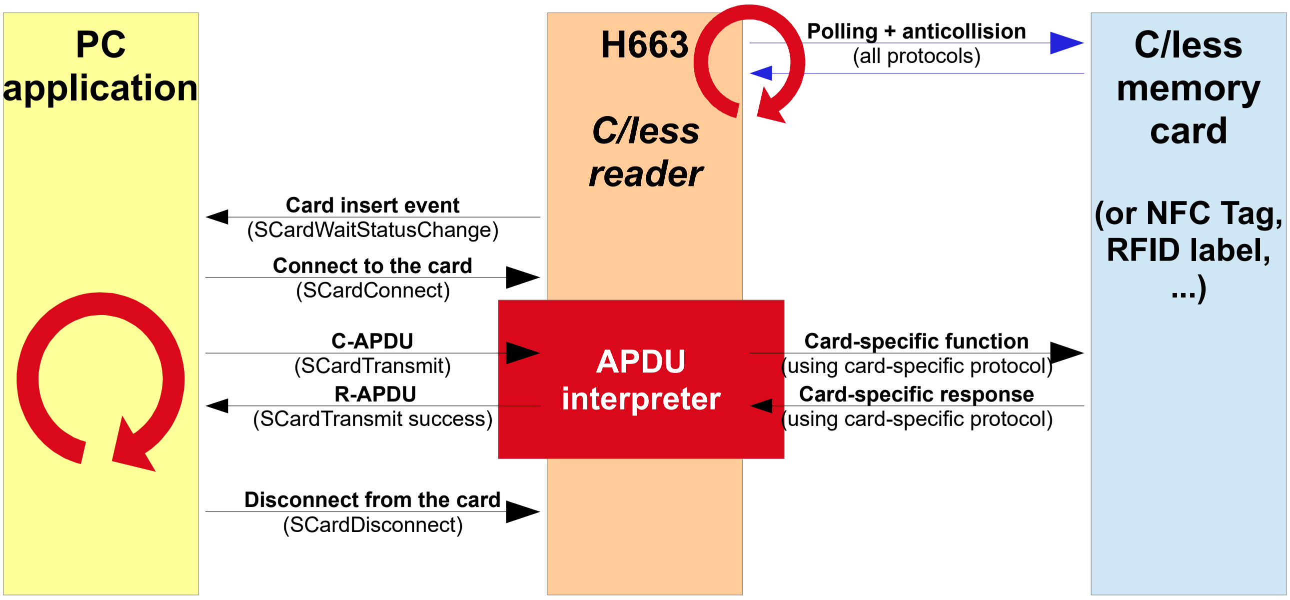
A lot of contactless cards are not actually “smartcards” because they are not ISO 7816-4 compliant. They don't comply with the ISO 14443-4 transport-level protocol, and their vendor-specific function set can't fit directly in a single “exchange” function. Therefore, they are not natively supported by 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-based cards that use a **specific communication protocol** with a frame format not compliant with ISO 7816-4 (Desfire EV0...).

The role of the **embedded APDU interpreter**, running in the **H663**, is to 'emulate' a standard smartcard in those cases. Doing so, the PC/SC stack (and as a consequence your application) doesn't have to deal with the underlying protocols and chip-specific commands.

Basically, the **embedded APDU interpreter** exposes the card as being a T=1 compliant smartcard, and provides two functions taken from ISO 7816-4: READ BINARY and UPDATE BINARY. In ISO 7816-4, those functions are intended to access data within a file (in the card's file-system), but on memory cards they give access to the “raw” storage, using a byte-, block- or page-based access depending on the card technology and features.

a. Role of the embedded APDU interpreter



2.4. NFC ?

NFC stands for **Near Field Communication**, which is the case of all communication systems using low frequencies or very short operating distance. But NFC is now understood as both

- **NFCIP-1** (Near Field Communication Interface and Protocol), i.e. the ISO 18092 standard, which defines a new transport-level protocol sometimes called “peer-to-peer” (but this name never appears is the standard),
- **NFC Forum**, an association that promotes the uses of NFC and publishes “application-level” standards (where ISO focuses on the technical levels).

SpringCard H663 and derived products are partially compliant with NFCIP-1 (initiator role, passive communication mode only). Please refer to chapter 5 for details. These products should also support NFC Forum's applications, but no compliance with NFC Forum's low level requirements is claimed.

> *Note that in NFC Forum's literature,*
> *- **ISO 14443 type A** and **ISO 18092 @ 106kbit/s** is called **NFC-A**,*
> *- **ISO 14443 type B** is called **NFC-B**,*
> *- **JIS:X6319-4** and **ISO 18092 @ 212/424kbit/s** is called **NFC-F**.*

2.5. Vendor-specific features – direct control of the coupler

PC/SC's *SCardTransmit* function implements a communication channel between your application and the card. But sometimes the application wants to access some features of the **H663** itself: driving the LEDs or buzzer, getting the serial number... In other words, the application wants to talk to the coupler and not to the card.

The PC/SC's *SCardControl* function has been designed to do so. Chapter 6 details the commands supported by the **H663** using this direct communication channel. But opening a *SCardControl* channel means getting a direct (and exclusive) access to the coupler, and as a consequence blocks the other communication channel(s).

To overcome this drawback, the **embedded APDU interpreter** could also be used to convey commands to the coupler with an existing card-channel and using *SCardTransmit* calls (see § 3.4.1 for details).

2.6. 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 coupler 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).
- **VICC:** *vicinity integrated circuit card*. This is the standard name for any contactless card compliant with the ISO 15693 standards (vicinity: less than 150cm). Common names are *RFID tag*, *RFID label*.
- **VCD:** *vicinity coupling device.* A device able to communicate with a VICC, i.e. a contactless coupler compliant with ISO 15693.
- **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 the 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 coupler 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) an 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[4]. Four

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

communication baud rates are possible: 106 kbit/s is mandatory, higher baud rates (212, 424 or 848 kbit/s) are optional.

- **ISO 15693:** This international standard defines the VCD/VICC communication scheme. It is divided into 3 layers:
  1. Defines the hardware characteristics of the VICC,
  2. Defines the carrier frequency and the bit-level communication scheme,
  3. Defines the frame-level communication scheme, the session opening sequence (anti-collision/inventory), and the command set of the VICC.

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

  Note that ISO 18000-3 mode 1 is the same as ISO 15693[5].

- **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 the NFC Forum on top of NFC-DEP.
- **SNEP:** *Simple NDEF Exchange Protocol.* An application protocol specified by the 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[6] stream cipher called CRYPTO1. Every sector is protected by 2 access keys called "key A" and "key B"[7]. NXP also offers another family of wired-logic PICCs called Mifare UltraLight (adopted by the NFC Forum as NFC Type 2 Tags). Mifare SmartMX

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

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

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

(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. The Felica standard includes a Sony-proprietary security scheme that is not implemented in SpringCard's products. Therefore, only the Felica chips configured to work without security (“Felica Lite”, “Felica Lite-S”, or NFC Type 3 Tags) are supported.

3. Embedded APDU interpreter

3.1. Basis

In PC/SC architecture, the ***SCardTransmit*** function implements the dialogue between an application and a smartcard, the coupler being only a “passive” gateway. The coupler 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)*
- Smart-card 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 **H663**, 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 **H663**’s non volatile memory (see § 7.4.2).

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 § 7.4.2) 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/VICC
$_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	Contactless	Contact	Notes (see below)
LOAD KEY	$_h$82	✓		C
GENERAL AUTHENTICATE	$_h$86	✓		C
READ BINARY	$_h$B0	✓		A
ENVELOPE	$_h$C2	✓		B
GET DATA	$_h$CA	✓	✓	C
UPDATE BINARY	$_h$D6	✓		A
READER CONTROL	$_h$F0	✓	✓	D
MICORE CONTROL	$_h$F1	✓		D
ABCCORE CONTROL	$_h$F1		✓	D
MIFARE CLASSIC READ	$_h$F3	✓		D
MIFARE CLASSIC WRITE	$_h$F4	✓		D
MIFARE CLASSIC VALUE	$_h$F5	✓		D
RFID MEMORY CONTROL	$_h$F6	✓		D – starting from FW 1.81
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. Instructions defined by the PC/SC standard (v2 part 3)

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
$_h$FF	$_h$CA	See below	See below	-	-	$_h$00

GET DATA command parameters

P1	P2	Action	Fw
		Standard PC/SC-defined values	
$_h$00	$_h$00	Serial number of the PICC - ISO 14443-A : UID (4, 7 or 11 bytes) - ISO 14443-B : PUPI (4 bytes) - ISO 15693: UID (8 bytes) - Innovatron : DIV (4 bytes) - JIS:X6319-4 : IDm (8 bytes) - others: see chapter 4 for details	
		SpringCard specific values	
$_h$01	$_h$00	- ISO 14443-A : historical bytes from the ATS - ISO 14443-B : INF field in ATTRIB response - JIS:X6319-4 : PMm (8 bytes) - ISO 18092 : G_T bytes in ATR_RES - others: see chapter 4 for details	
$_h$F0	$_h$00	Complete identifier of the PICC: - ISO 14443-A: ATQA (2 bytes) + SAK (1 byte) + UID - ISO 14443-B: complete ATQB (11 or 12 bytes)[8] - ISO 15693: answer to GET SYSTEM INFORMATION command[9] - Innovatron: REPGEN - Innovision/Broadcom/NFC Forum Type 1 Tag: HR0, HR1 - JIS:X6319-4 : IDm and PMm (16 bytes) - ISO 18092 : complete ATR_RES	≥ 1.75

[8] SpringCard PC/SC Couplers 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.

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

P1	P2	Action	
hF1	h00	Type of the PICC/VICC, according to PC/SC part 3 supplemental document: PIX.SS (standard, 1 byte) + PIX.NN (card name, 2 bytes) See chapter 4.1 for details	
hF1	h01	NFC Tag[10] compliance: - h01 if the PICC is recognized as a NFC Type 1 Tag - h02 if the PICC is recognized as a NFC Type 2 Tag - h03 if the PICC is recognized as a NFC 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	ISO 14443 communication indexes on 2 bytes (DSI, DRI)	
hFC	h01	PICC/VICC → H663 baudrate (DS in kbit/s, 2 bytes, MSB first)	
hFC	h02	H663 → PICC/VICC baudrate (DR in kbit/s, 2 bytes, MSB first)	
hFC	h03	Index of the active antenna on 1 byte	
hFF	h00	Product serial number (raw value on 4 bytes)	
hFF	h01	*Not available for H663*	
hFF	h02	Name of the RF interface component ("RC663")	
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)

[10] Please refer to NFC Forum's specifications for details. Note that NFC Forum 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 the **H663**'s memory.

LOAD KEY command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$82	Key location	Key index	$_h$06	Key value	-

LOAD KEY command parameter P1 (key location)

P1	
$_h$00	The key is to be loaded in H663's volatile memory
$_h$20	The key is to be loaded in H663's non-volatile memory (secure E2PROM inside the RC chipset)

LOAD KEY command parameter P2 (key index)

When P1 = $_h$00, P2 is the identifier of the key into **H663**'s volatile memory. The memory has the capacity to store up to 4 keys of each type (A or B).

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

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

When P1 = $_h$20, P2 is the identifier of the key into the **H663**'s non-volatile memory (if available). This memory can store up to 16 keys of each type (A or B).

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

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

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

LOAD KEY response

SW1	SW2
See below	

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 **H663** 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
$_h$FF	$_h$86	$_h$00	$_h$00	$_h$05	See below	-

GENERAL AUTHENTICATE Data In bytes

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

The ***block number*** (byte 2) is the address on the 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:

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

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

- If *key type* (byte 3) is $_h$60, use values $_h$00 to $_h$03 to select one of the “A” keys stored in the **H663**'s volatile memory, and values $_h$20 to $_h$2F to select one of the “A” keys stored in the **H663**'s non-volatile memory (if available),
- If *key type* (byte 3) is $_h$61, use values $_h$00 to $_h$03 to select one of the “B” keys stored in the **H663**'s volatile memory, and values $_h$20 to $_h$2F to select one of the “B” keys stored in the **H663**'s non-volatile memory (if available),
- If *key type* (byte 3) is either $_h$00 or $_h$20 (same value as the P1 parameter used in the LOAD key instruction), choose one of the values allowed for the P2 parameter in the same LOAD key instruction *(SpringCard specific value).*

GENERAL AUTHENTICATE response

SW1	SW2
See below	

GENERAL AUTHENTICATE status word

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$69	$_h$82	CRYPTO1 authentication failed
$_h$69	$_h$86	Key location or type (byte 3) is not valid (or not available for this coupler)
$_h$69	$_h$88	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 or VICC). Refer to chapter 4 for details.

For any PICC/VICC 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/VICC in its specific read command. Most PICC/VICC are divided into small blocks (sometimes called pages). The address is a block number, and not to an absolute byte offset in memory.

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

For Mifare Classic, P1,P2 is the address of the block ($_h$0000 to $_h$00FF), but remember that the authentication is made on a per-sector basis. A new authentication must be performed every time you have to access another sector.
For a NFC Type 2 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
$_h$90	$_h$00	Success
$_h$62	$_h$82	End of data reached before Le bytes (Le is greater than data length)
$_h$69	$_h$81	Command incompatible
$_h$69	$_h$82	Security status not satisfied
$_h$6A	$_h$82	Wrong address (no such block or no such offset in the PICC/VICC)
$_h$6C	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 or VICC). Refer to chapter 4 for details.

For any PICC/VICC 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/VICC in its specific write command. Most PICC/VICC are divided into small blocks (sometimes called pages). The address is a block number, and not to an absolute byte offset in memory.

Both the allowed range for the **address** and the value for **Lc** depend on the capabilities of the PICC. 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 Type 2 Tag, P2 is the block number, and P1 the sector number if the PICC does support this feature. Set P1 to $_h$00 if it is not the case. Lc must be $_h$04 (a block is 4-B long).

UPDATE BINARY response

SW1	SW2
See below	

UPDATE BINARY status word

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

Important disclaimer

Most PICC/VICC 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. SpringCard-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 **H663**, 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 coupler’s keys

In this mode, the application doesn't specify anything. The **H663** 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 coupler must try all the keys, this method may take up to 1000ms. The ordering of the keys in coupler's memory is very important to speed-up the process: the upper the right key is in the coupler's memory, the sooner the authentication will succeed.
Note that the coupler 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 coupler*

In this mode, the application chooses one of the key previously loaded in the **H663** 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 **H663**.

The coupler 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 **H663**, 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 coupler’s keys

In this mode, the application doesn't specify anything. The **H663** tries every key he knows (both permanent keys in E2PROM and temporary keys previously loaded in volatile memory) until one succeeds.

Since the coupler must try all the keys, this method may take up to 1000ms. The ordering of the keys in coupler's memory is very important to speed-up the process: the upper the right key is in the coupler's memory, the sooner the authentication will succeed.
Note that the coupler 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[11].

[11] 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
$_h$FF	$_h$F4	$_h$00	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 coupler

In this mode, the application chooses one the key previously loaded in the **H663** through the **LOAD KEY** instruction.

MIFARE CLASSIC WRITE command APDU, selecting a key

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F4	$_h$00	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 **H663**.

The coupler 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
$_h$FF	$_h$F4	$_h$00	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 opcode, 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 **H663** 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 coupler’s keys

In this mode, the application doesn't specify anything. The **H663** tries every keys he knows (both permanent keys in E2PROM and temporary keys previously loaded in volatile memory) until one succeeds.

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

In this mode, the application chooses one the key previously loaded in the **H663** 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 **H663**.

For DECREMENT and RESTORE operations, the coupler tries the key as a "type A" first, and only afterwards as a "type B".
For INCREMENT operation, the coupler 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. RFID MEMORY CONTROL instruction

The **RFID MEMORY CONTROL** instruction gives access to some functions of RFID wired-logic PICCs or VICCs that have no equivalent in the smartcard world.

For instance, Reading to / Writing from a RFID memory chip maps to READ BINARY / UPDATE BINARY which are “standards” instructions defined by ISO 7816. But ISO 7816 has no equivalent for a lot of functions defined in ISO 15693, such as “Write DSFID”, “Lock AFI”, and much more.

Therefore, the **RFID MEMORY CONTROL** instruction is a **SpringCard**-defined function that eases operating ISO 15693 and related VICCs, such as EM4134.

a. Read Single Block

This function is available for ISO 15693 and EM4134 VICCs.

This function is a low-level alternative to READ BINARY.

RFID MEMORY CONTROL : Read Single Block command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF6	h20	h00	h01	Address (1B)	h00

b. Write Single Block

This function is available for ISO 15693 and EM4134 VICCs.

This function is a low-level alternative to UPDATE BINARY.

RFID MEMORY CONTROL : Write Single Block command APDU

CLA	INS	P1	P2	Lc	Data In		Le
hFF	hF6	h21	h00	h01 +Len	Address (1B)	Data (Len)	-

c. Lock Block

This function is available for ISO 15693 and EM4134 VICCs.

RFID MEMORY CONTROL : Lock Block command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF6	h22	h00	h01	Address (1B)	-

d. Read Multiple Blocks

This function is available for ISO 15693 VICCs only.

This function is a low-level alternative to READ BINARY.

RFID MEMORY CONTROL : Read Multiple Blocks command APDU

CLA	INS	P1	P2	Lc	Data In		Le
hFF	hF6	h23	h00	h01	Address (1B)	Count (1B)	h00

e. Write Multiple Blocks

This function is available for ISO 15693 VICCs only.

This function is a low-level alternative to UPDATE BINARY.

RFID MEMORY CONTROL : Write Multiple Block command APDU

CLA	INS	P1	P2	Lc	Data In			Le
hFF	hF6	h24	h00	h02 +Len	Address (1B)	Count (1B)	Data (Len)	-

f. Write AFI

This function is available for ISO 15693 VICCs only.

RFID MEMORY CONTROL : Write AFI command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF6	h27	h00	h01	AFI (1B)	-

g. Lock AFI

This function is available for ISO 15693 VICCs only.

RFID MEMORY CONTROL : Lock AFI command APDU

CLA	INS	P1	P2	Lc	Le
$_h$FF	$_h$F6	$_h$28	$_h$00	$_h$00	-

h. Write DSFID

This function is available for ISO 15693 VICCs only.

RFID MEMORY CONTROL : Write DSFID command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F6	$_h$29	$_h$00	$_h$01	DSFID (1B)	-

i. Lock DSFID

This function is available for ISO 15693 VICCs only.

RFID MEMORY CONTROL : Lock DSFID command APDU

CLA	INS	P1	P2	Lc	Le
$_h$FF	$_h$F6	$_h$2A	$_h$00	$_h$00	-

j. Get System Information

This function is available for ISO 15693 VICCs only.

RFID MEMORY CONTROL : Get System Information command APDU

CLA	INS	P1	P2	Lc	Le
$_h$FF	$_h$F6	$_h$2A	$_h$00	-	$_h$00

Note: the **H663** always sends the *Get system information* command to the VICC, as part of the discovery process. Invoke the GET DATA instruction (§ 3.2.1) to retrieve the value already returned by the VICC to the **H663**.

k. Get Multiple Block Security

This function is available for ISO 15693 VICCs only.

RFID MEMORY CONTROL : Get Multiple Block Security command APDU

CLA	INS	P1	P2	Lc	Data In		Le
hFF	hF6	h24	h00	h02	Address (1B)	Count (1B)	-

3.3.5. 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
$_h$FF	$_h$FB	See below	See below	-	-	-

SLOT CONTROL command parameters

P1	P2	Action	
$_h$00	$_h$00	Resume the card tracking mechanism	
$_h$01	$_h$00	Suspend the card tracking mechanism	
$_h$10	$_h$00	Stop the RF field	
$_h$10	$_h$01	Start the RF field	
$_h$10	$_h$02	Reset the RF field (10ms pause)	
$_h$20	$_h$00	T=CL de-activation (DESELECT[12])	
$_h$20	$_h$01	T=CL activation of ISO 14443-A card (RATS)	
$_h$20	$_h$02	T=CL activation of ISO 14443-B card (Attrib)	
$_h$20	$_h$04	Disable the next T=CL activation[13]	
$_h$20	$_h$05	Disable every T=CL activation (until reset of the H663)	
$_h$20	$_h$06	Enable T=CL activation again	
$_h$20	$_h$07	Disable the next T=CL activation and force a RF reset	
$_h$FC	xx	Felica runtime parameters, see § 3.3.6 below	
$_h$DE	$_h$AD	Stop the slot NOTE: a stopped slot is not available to *SCardConnect* any more. It may be restarted only through an *SCardControl* command.	

[12] 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.

[13] 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
$_h$90	$_h$00	Success

3.3.6. SET FELICA RUNTIME PARAMETERS instruction

Working with Felica (Lite) cards or NFC Type 3 Tags involves 4 parameters:

- The *SYSTEM CODE* is sent by the **H663** during the JIS:X6319-4 polling loop (SENSF_REQ) to specify which family of cards may answer. The value $_h$FFFF allows any card to answer,
- The *REQUEST CODE* is sent by the **H663** during the JIS:X6319-4 polling loop (SENSF_REQ) to get technical data from the cards, and not only their IDm/PPm. The value $_h$00 prevent the card from sending technical data,
- A first *SERVICE CODE* is a mandatory parameter used during read operations (READ BINARY instruction) to tell the card which "service" is accessed. The value $_h$000B has been assigned by the NFC Forum to give (read) access to a type 3 Tag's NDEF record,
- Another *SERVICE CODE* is a mandatory parameter used during write operations (UPDATE BINARY instruction) to tell the card which "service" is accessed. The value $_h$0009 has been assigned by the NFC Forum to give write access to a type 3 Tag's NDEF record.

The values emphasized in the above paragraph are the **H663**'s default values. They could be updated permanently thanks to the *WRITE REGISTER* command (§ 6.3.6) applied to the configuration registers $_h$B4 (§ 7.5.2) and $_h$CF (§ 7.6.1).

Alternatively, those values may be changed dynamically using a simple APDU command in the *SCardTransmit* stream, as depicted in the paragraphs below.

a. SERVICE CODE for the READ BINARY instruction

SET FELICA SERVICE READ command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	$_h$FC	$_h$01	$_h$02	Service Code to be used by the READ BINARY instruction (2 bytes, MSB first)	-

b. SERVICE CODE for the UPDATE BINARY instruction

SET FELICA SERVICE WRITE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hFB	hFC	h02	h02	Service Code to be used by the UPDATE BINARY instruction (2 bytes, MSB first)	-

c. SERVICE CODE for both READ BINARY and UPDATE BINARY instructions

SET FELICA SERVICES command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hFB	hFC	h03	h02	Service Code to be used both by the READ BINARY and UPDATE BINARY instructions (2 bytes, MSB first)	-

d. SYSTEM CODE and REQUEST code for Felica polling

SET FELICA SYSTEM CODE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hFB	hFC	h10	h02	System Code to be used during JIS:X6319-4 polling (SC in SENS_REQ) (2 bytes, MSB first)	-

SET FELICA REQUEST CODE command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hFB	hFC	h11	h01	Request Code to be used during JIS:X6319-4 polling (RC in SENS_REQ) (1 byte)	-

3.3.7. ENCAPSULATE instruction for the Contactless slot

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

ENCAPSULATE command APDU for the contactless slot

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

ENCAPSULATE command parameter P1 for the contactless slot

P1	Standard communication protocols
$_h$00	For ISO 14443-4 (A or B) PICCs : send the frame in the T=CL stream[14]. 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 @ 106 kbit/s. The standard parity bits are added (and checked in return) by the H663. The standard CRC is added (and checked in return) by the H663.
$_h$02	Send the frame "as is" using the ISO 14443-3 B protocol @ 106 kbit/s. The standard CRC is added (and checked in return) by the H663.
$_h$03	Send the frame "as is" using the JIS:X6319-4 protocol @ 212 kbit/s. The standard CRC is added (and checked in return) by the H663.
$_h$04	Send the frame "as is" using the ISO 15693 protocol. The standard CRC is added (and checked in return) by the H663.
$_h$05	Send the frame "as is" using the ISO 15693 protocol. The UID of the VICC is added to the frame *(unselected access mode)*. The standard CRC is added (and checked in return) by the H663.
$_h$07	Send the frame "as is" using the JIS:X6319-4 protocol @ 424 kbit/s. The standard CRC is added (and checked in return) by the H663.

…/…

[14] 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 @ 106 kbit/s. The standard parity bits are added (and checked in return) by the H663, but the CRC is not added (and not checked) by the H663 → 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 @ 106 kbit/s. The CRC is not added (and not checked) by the H663 → the application must append the CRC to Data In and check it in Data Out.
$_h$0C	Send the frame "as is" using the ISO 15693 modulation. The CRC is not added (and not checked) by the H663 → the application must append the CRC to Data In and check it in Data Out.
P1	**Mifare low level communication[15]**
$_h$0F	Send the frame "as is" using the ISO 14443-3 A modulation. The CRC is not added (and not checked) by the H663 → 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 H663 → 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

[15] 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.

P1	Redirection to another slot[16]
$_h$80	Redirection to the main contact slot (if present)
$_h$81	Redirection to the 1st SIM/SAM slot (if present)
$_h$82	Redirection to the 2nd SIM/SAM slot (if present)
$_h$83	Redirection to the 3rd SIM/SAM slot (if present)
$_h$84	Redirection to the 4th SIM/SAM slot (if present)

ENCAPSULATE command parameter P2 for the contactless slot

P2 encodes the frame time-out.

P2	Timeout value
$_h$-0	If P1 = $_h$00, use the default time-out 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

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

ENCAPSULATE response for the contactless slot

Data Out	SW1	SW2
Frame received from the PICC/VICC	See below	

ENCAPSULATE status word for the contactless slot

SW1	SW2	Meaning
h90	h00	Success – last byte of Data Out has 8 valid bits
h90	h01	Success – last byte of Data Out has 1 valid bits
h90	h02	Success – last byte of Data Out has 2 valid bits
h90	h03	Success – last byte of Data Out has 3 valid bits
h90	h04	Success – last byte of Data Out has 4 valid bits
h90	h05	Success – last byte of Data Out has 5 valid bits
h90	h06	Success – last byte of Data Out has 6 valid bits
h90	h07	Success – last byte of Data Out has 7 valid bits
h6F	XX	Error reported by the contactless interface (only allowed if high-order bit of P2 is 0). See chapter 8 for the list of possible values and their meaning.
h63	h00	Error reported by the contactless interface (when high-order bit of P2 is 1).
h62	h82	Le is greater than actual response from PICC/VICC
h6C	XX	Le is shorter than actual response from PICC/VICC

3.3.8. ENCAPSULATE instruction for one of the Contact slots

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

ENCAPSULATE command APDU for a contact slot

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FE	$_h$00	$_h$00	XX	Frame to send to the card	XX

ENCAPSULATE response for a contact slot

Data Out	SW1	SW2
Frame received from the card	See below	

ENCAPSULATE status word for a contact slot

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$6F	XX	Error reported by the contactless interface (only allowed if high-order bit of P2 is 0). See chapter 8 for the list of possible values and their meaning.
$_h$62	$_h$82	Le is greater than actual response from card
$_h$6C	XX	Le is shorter than actual response from card

3.4. Other SpringCard-specific instructions

3.4.1. READER CONTROL instruction

The **READER CONTROL** instruction allows driving the global behaviour of the **H663** (LEDs, buzzer, etc. depending on product physical characteristics).

For advanced operation, or if you want to interact with the **H663** even when there's no card inserted, use *SCardControl* instead (see chapter 6).

If your coupler is multi-slot (contactless + contact or SAM), the READER CONTROL instruction is sent to one slot (a logical coupler), but is likely to have a global impact to the whole physical coupler.
In other words, sending a READER CONTROL instruction to one card channel may have an impact on another card channel.
It is highly recommended to use a synchronisation object in your application(s) (mutex, critical section, …) to prevent any concurrent access to the same physical coupler when the READER CONTROL instruction is called.

READER CONTROL command APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF0	h00	h00	See below	See below	See below

a. Driving coupler's LEDs

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

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

For a coupler 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 the H663's firmware *(default behaviour)*
h04	LED blinks quickly
h05	LED performs the "heart-beat" sequence

To go back to default (LEDs driven by the **H663**'s firmware automatically), send the APDU:

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

b. Driving coupler'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 driven by the **H663**'s firmware automatically), 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 6.

Therefore, every command documented in § 6.3 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 **H663**:

$_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 **H663** version 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. Working with contactless cards – useful hints

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

4.1.1. ATR of an ISO 14443-4 compliant smartcard

If the PICC is with 14443 up to level 4 ("**T=CL**"), the **H663** 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)[17]:*

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[18].
…	…		
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[19] to make sure it will not start a new transaction on the same card as earlier.

[17] When bit 7 of register $_h$B3 is 0. 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.

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

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

4.1.2. ATR of a wired-logic PICC/VICC

For contactless memory cards and RFID tags (Mifare, CTS, etc.), the **H663** 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 4.1.4)
13	H10	PIX.NN	Card name (see 4.1.5)
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)

4.1.3. 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, …).

4.1.4. Contactless protocol

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

b7	b6	b5	b4	b3	b2	b1	b0	Value	Description
0	0	0	0	0	0	0	0	$_h$00	No information given
0	0	0	0	0	0	0	1	$_h$01	ISO 14443 A, level 1
0	0	0	0	0	0	1	0	$_h$02	ISO 14443 A, level 2
0	0	0	0	0	0	1	1	$_h$03	ISO 14443 A, level 3 or 4 (and Mifare) ISO 18092 @ 106 kbit/s “NFC-A”
0	0	0	0	0	1	0	1	$_h$05	ISO 14443 B, level 1
0	0	0	0	0	1	1	0	$_h$06	ISO 14443 B, level 2
0	0	0	0	0	1	1	1	$_h$07	ISO 14443 B, level 3 or 4
0	0	0	0	1	0	0	1	$_h$09	ICODE 1, EM4134
0	0	0	0	1	0	1	1	$_h$0B	ISO 15693
0	0	0	1	0	0	0	1	$_h$11	JIS:X6319-4 Felica cards ISO 18092 @ 212 or 424 kbit/s “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.

4.1.5. Contactless card name bytes

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

NN	Card name	From FW
	Values specified by PC/SC	
$_h00$ $_h01$	NXP Mifare Classic 1k	
$_h00$ $_h02$	NXP Mifare Classic 4k	
$_h00$ $_h03$	NXP Mifare UltraLight NFC Forum Type 2 Tag with a capacity <= 64 bytes	
$_h00$ $_h06$	ST Micro Electronics SR176	
$_h00$ $_h07$	ST Micro Electronics SRI4K, SRIX4K, SRIX512, SRI512, SRT512	1.70
$_h00$ $_h0A$	Atmel AT88SC0808CRF	
$_h00$ $_h0B$	Atmel AT88SC1616CRF	
$_h00$ $_h0C$	Atmel AT88SC3216CRF	
$_h00$ $_h0D$	Atmel AT88SC6416CRF	
$_h00$ $_h12$	Texas Instruments TAG IT	
$_h00$ $_h13$	ST Micro Electronics LRI512	
$_h00$ $_h14$	NXP ICODE SLI	
$_h00$ $_h16$	*Not available in this product (NXP ICODE1)*	
$_h00$ $_h21$	ST Micro Electronics LRI64	
$_h00$ $_h24$	ST Micro Electronics LR12	
$_h00$ $_h25$	ST Micro Electronics LRI128	
$_h00$ $_h26$	NXP Mifare Mini	
$_h00$ $_h2F$	Innovision/Broadcom Jewel	
$_h00$ $_h30$	Innovision/Broadcom Topaz NFC Forum Type 1 Tag	
$_h00$ $_h34$	Atmel AT88RF04C	
$_h00$ $_h35$	NXP ICODE SL2	
$_h00$ $_h36$	NXP Mifare Plus 2K SL1	1.81
$_h00$ $_h37$	NXP Mifare Plus 4K SL1	1.81
$_h00$ $_h38$	NXP Mifare Plus 2K SL2	1.81
$_h00$ $_h39$	NXP Mifare Plus 4K SL2	1.81
$_h00$ $_h3A$	NXP Mifare UltraLight C, NXP NTAG203 NFC Forum Type 2 Tag with a capacity > 64 bytes	
$_h00$ $_h3A$	Felica NFC Forum Type 3 Tag	

NN	Card name	
	SpringCard proprietary extension [20]	
$_h$FF $_h$A0	Generic/unknown 14443-A card	
$_h$FF $_h$A1	ThinFilm	
$_h$FF $_h$B0	Generic/unknown 14443-B card	
$_h$FF $_h$B1	*Not available in this product (ASK CTS 256B)*	
$_h$FF $_h$B2	*Not available in this product (ASK CTS 512B)*	
$_h$FF $_h$B3	Pre-standard ST Micro Electronics SRI 4K	
$_h$FF $_h$B4	Pre-standard ST Micro Electronics SRI X512	
$_h$FF $_h$B5	Pre-standard ST Micro Electronics SRI 512	
$_h$FF $_h$B6	Pre-standard ST Micro Electronics SRT 512	
$_h$FF $_h$B7	Inside Contactless PICOTAG/PICOPASS	
$_h$FF $_h$B8	Generic Atmel AT88SC / AT88RF card	
$_h$FF $_h$C0	Calypso card using the Innovatron protocol	
$_h$FF $_h$D0	Generic ISO 15693 from unknown manufacturer	
$_h$FF $_h$D1	Generic ISO 15693 from EM Marin (or Legic)	
$_h$FF $_h$D2	Generic ISO 15693 from ST Micro Electronics, block number on 8 bits	
$_h$FF $_h$D3	Generic ISO 15693 from ST Micro Electronics, block number on 16 bits	
$_h$FF $_h$D5	Generic ISO 15693 from Infineon	
$_h$FF $_h$D6	EM MicroElectronic Marin EM4134 chip	1.81
$_h$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.

[20] 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. Set bit 6 of configuration register $_h$B3 (§ 7.4.3) to force PIX.NN = $_h$00 $_h$00 instead of using those proprietary values.

4.2. ISO 14443-4 PICCs

4.2.1. 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 (§ 3.3.7). The **H663** translates the C-APDU into a native Desfire command, retrieve the native Desfire answer, and translates it into a valid R-APDU.

4.2.2. 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.

4.2.3. 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 coupler.

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

4.3.1. 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 § 4.3.2).

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 **H663** will "hide" their Mifare **emulation mode** and make them appear as high-level smartcards.

There are 3 ways to force the **H663** 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 <u>first block to be read</u> (0 to 63 for a Mifare 1k, 0 to 255 for a Mifare 4k),

Since the size of every block is 16, <u>**Le** must be a multiple of 16</u>,

- 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.

4.3.2. 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 **H663** builds a smartcard ATR according to § 4.1.1. 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 (§ 3.3.7).

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 (§ 3.3.5)[21].

b. Level 1

At level 1, the PICC emulates a Mifare Classic (§ 4.3.1). The **H663** builds a memory card ATR according to § 4.1.1.

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 (§ 3.3.7) 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 (§ 3.3.5). Afterwards, process as documented for level 0.

c. Level 2

The level 2 is not available on Mifare Plus S.

[21] 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 **H663** (and not supported by our software library).

d. Level 3

At level 3, the PICC is ISO 14443-4 (T=CL) compliant. The **H663** builds a smartcard ATR according to § 4.1.1. 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 (§ 3.3.7).

4.3.3. NFC Forum Type 2 Tags – Mifare UltraLight and UltraLight C, NTAG203...

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 the specification of the NFC Forum Type 2 Tag.

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

Please visit www.nfcforum.org to get the specification of the Type 2 Tag.

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 the specification of the NFC Forum Type 2 Tag, 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 is the sector number. It must be $_h$00 for PICCs that have only one sector,
- P2 is the address of the first page to be read. Please refer to the chip's datasheet to know how many pages could be addressed.

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 is the sector number. It must be $_h$00 for PICCs that have only one sector,
- P2 is the address of the (single) page to be written. Please refer to the chip's datasheet to know how many pages could be addressed.

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

Some pages may 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 (§ 3.3.7) 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

4.3.4. NFC Forum Type 1 Tags – Innovision/Broadcom chips

Firmware ≥ 1.75

The PICCs covered by this chapter are:

- Innovision/Broadcom Topaz (**PIX.NN = $_h$002F**),
- Innovision/Broadcom Jewel (**PIX.NN = $_h$0030**),
- Any PICC compliant with NFC Forum Type 1 Tag specification.

Please visit www.nfcforum.org to get the Type 1 Tag specification.

a. Memory Structures

There are 2 groups of PICCs in this specification:

- PICCs with a **Static Memory Structure** provide 120 bytes of data. They do support only the RALL, READ, WRITE-E and WRITE-NE functions.
- PICCs with a **Dynamic Memory Structure** provide more than 120 bytes of data. They are divided into 8-bytes *blocks*. A *segment* is a group of 16 blocks (i.e. 128 bytes of data). New functions are provided to address *blocks* and *segments*: READ8, RSEG, WRITE-E8 and WRITE-NE8.

Those PICCs have 2 hardware information bytes called HR0 and HR1.

- HR0 = $_h$11 denotes a Static Memory Structure,
- HR0 = $_h$1*y*, where *y* ≠ 1, denotes a Dynamic Memory Structure,
- Other values for HR0 are RFU, HR1 is ignored.

Prior to read/write PICC's data, the application shall fetch HR0 to know whether the PICC has a Static or a Dynamic Memory Structure. To do so, the application may either:

- Invoke the READ BINARY instruction, specifying it wants to use the PICC's RALL function and expects 122 bytes of data (FF B0 00 00 7A). HR0 is the first byte in the response.
- Invoke the GET DATA instruction, specifying it wants to get the PICC's complete identifier (FF CA F0 00 00). HR0 is the first byte in the response.

b. READ BINARY instruction

L_E		P1	P2	PICC function	Description
Both Static and Dynamic Structures					
$_h$00 $_h$78	0 120	$_h$00	$_h$00	RALL	The coupler returns the 120 bytes of data returned by the PICC in response to RALL. The HR0 and HR1 bytes are dropped.
$_h$7A	122	$_h$00	$_h$00	RALL	The coupler returns the complete frame returned by the PICC in response to RALL, i.e. HR0 and HR1 followed by 120 bytes of data.
$_h$01	1	$_h$00, $_h$00 to $_h$00, $_h$7F		READ	P2 specify the <u>byte address</u> within the card from 0 to 127. One byte is returned.
Dynamic Memory Structure only					
$_h$80	128	$_h$00, $_h$00 $_h$00, $_h$80 $_h$01, $_h$00 ...		RSEG	P1, P2 specify the <u>byte address</u> within the card. A complete segment (128 bytes of data) is returned. Therefore, P1, P2 must be aligned to a segment boundary *(≡ 0 mod 128)*.
$_h$08	8	$_h$00, $_h$00 $_h$00, $_h$08 $_h$00, $_h$10 ...		READ8	P1, P2 specify the <u>byte address</u> within the card. A complete block (8 bytes of data) is returned. Therefore, P1, P2 must be aligned to a block boundary *(≡ 0 mod 8)*.

Using the RALL or RSEG functions is a lot faster than using READ/READ8 in a loop.

c. UPDATE BINARY instruction

<table>
<tr><th colspan="2">L_C</th><th>P1</th><th>P2</th><th>PICC function</th><th>Description</th></tr>
<tr><td colspan="6" align="center">Both Static and Dynamic Structures</td></tr>
<tr><td>$_h$01</td><td>1</td><td colspan="2">$_h$00, $_h$00
to
$_h$00, $_h$7F</td><td>WRITE-E</td><td>The coupler writes 1 byte of data into the Tag.
P2 specify the <u>byte address</u> (from 0 to 127)</td></tr>
<tr><td>$_h$01</td><td>1</td><td colspan="2">$_h$80, $_h$00
to
$_h$80, $_h$7F</td><td>WRITE-NE</td><td>The coupler updates 1 byte of data to the Tag.
The actual operation is a XOR between the current content of the card and the specified value.
P2 specify the <u>byte address</u> (from 0 to 127)</td></tr>
<tr><td colspan="6" align="center">Dynamic Memory Structure only</td></tr>
<tr><td>$_h$01</td><td>1</td><td colspan="2">$_h$00, $_h$00
$_h$00, $_h$08
$_h$00, $_h$10
...</td><td>WRITE-E8</td><td>The coupler writes 8 byte of data into the Tag.
P1, P2 specify the <u>byte address</u> within the card.
Therefore, P1, P2 must be aligned to a block boundary (≡ 0 mod 8).</td></tr>
<tr><td>$_h$01</td><td>1</td><td colspan="2">$_h$80, $_h$00
$_h$80, $_h$08
$_h$80, $_h$10
...</td><td>WRITE-NE8</td><td>The coupler updates 8 bytes of data to the Tag.
The actual operation is a XOR between the current content of the card and the specified value.
$P1_{0..6}$, P2 specify the <u>byte address</u> within the card. Therefore, $P1_{0..6}$, P2 must be aligned to a block boundary (≡ 0 mod 8).</td></tr>
</table>

Some blocks holds 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.

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

4.4.1. ST Micro Electronics SR176

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

They are divided into 2-byte *blocks*.

a. READ BINARY instruction

In the READ BINARY command APDU,

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

Since the size of every block is 2, Le must be multiple of 2 (up to 32 bytes),

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

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

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

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

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

4.4.2. ST Micro Electronics SRI4K, SRIX4K, SRI512, SRX512, SRT512

These PICCs are identified by **PIX.NN = $_h$0007**.

They are divided into 4-byte *blocks*.

a. READ BINARY instruction

In the READ BINARY command APDU,

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

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

When Le=$_h$00, a single block is returned (4 bytes).

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

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

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

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

4.4.3. Inside Contactless PicoPass, ISO 14443-2 mode

This part applies to chips named either "PicoPass or PicoTag" when the ISO 14443-3 compliance is NOT enabled in the card (see § 4.4.4 in the other case).

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

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

The **H663** may read/write the non-secure chips only (2K, 16K). The behaviour with the secure chips is undefined.

a. READ BINARY instruction

In the READ BINARY command APDU,

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

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

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

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

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

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

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

c. Page select

The Inside specific Page select function is not implemented in the **H663**. Use the ENCAPSULATE instruction to send it directly to the PICC.

4.4.4. Inside Contactless PicoPass, ISO 14443-3 mode

This part applies to chips named either "PicoPass or PicoTag" when the ISO 14443-3 compliance IS enabled in the card (see § 4.4.3 in the other case).

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

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

The **H663** may read/write the non-secure chips only (2K, 16K). The behaviour with the secure chips is undefined.

a. READ BINARY instruction

In the READ BINARY command APDU,

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

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

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

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

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

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

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

4.4.5. Atmel CryptoRF

The PICCs covered by this chapter are:

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

The **H663** implements the read and write functions in non-authenticated mode. Advanced functions and authenticated communication has to be implemented by the application within an ENCAPSULATE instruction.

> *The coupler always activates this PICC with CID=$_h$01. Use this CID to build the actual command to be sent through the ENCAPSULATE instruction.*

a. READ BINARY instruction

In the READ BINARY command APDU,

P1,P2 is the first address to be read,

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

Note: the READ BINARY instruction maps to the “Read User Zone” low-level command. The “Read System Zone” command is not implemented in the **H663**, and therefore must be encapsulated.

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

P1,P2 is the first address to be written,

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

Note: the UPDATE BINARY instruction maps to the “Write User Zone” low-level command. The “Write System Zone” command is not implemented in the **H663**, and therefore must be encapsulated.

4.5. ISO 15693 VICCs

4.5.1. ISO 15693-3 read/write commands

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

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

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

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

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first block to be read; please read documentation of your VICC to know its number of blocks,

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

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

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

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the block to be written, please read documentation of your VICC to know its number of blocks,

Lc must be the size of the block, exactly.

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

4.5.2. Read/write commands for ST Micro Electronics chips with a 2-B block address

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

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

4.5.3. Complete ISO 15693 command set

The ISO 15693 standard defines numerous commands with or without an 'option' flag, and leaves the chip manufacturers free to implement virtually any custom or proprietary commands.

Starting with firmware version 1.81, the basic commands, in their basic implementation, are available through the **RFID MEMORY CONTROL instruction** (§ 3.3.4), but it remain impossible to implement all commands and all variations in a reader.

The **ENCAPSULATE instruction** (INS = $_h$FE, see § 3.3.7) for ISO 15693 has therefore been introduced; this instruction allows to send any arbitrary command to a 15693 chip.

Since the **H663** operates the ISO 15693 chip in addressed mode (the VICC is never put into *quiet state*), the chip's UID shall be provided within every command frame. The insertion of the UID is performed automatically by the ENCAPSULATE instruction when parameter P1 is set to $_h$05.

The APDU shall be build as follow:

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

Note: Le could be omitted.

Allowed values for the 'command flags' byte

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

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

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

4.5.4. Implementation of basic ISO 15693 commands

Starting with firmware version 1.81, the below commands are available through the RFID MEMORY CONTROL instruction (§ 3.3.4)

a. *Read single block*

ISO 15693 command code: $_h$20

The APDU is

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

b. *Write single block*

ISO 15693 command code: $_h$21

The APDU is

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

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

c. Lock block

ISO 15693 command code: $_h$22

The APDU is

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

Locking a block makes it permanently read-only. This operation can't be cancelled. Do not perform this operation without a good understanding of the consequence.

d. Write AFI

ISO 15693 command code: $_h$27

The APDU is

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

e. Lock AFI

ISO 15693 command code: $_h$28

The APDU is

`FF FE 05 00 02 22 28`

Locking the AFI can't be cancelled. Do not perform this operation without a good understanding of the consequence.

f. Write DSFID

ISO 15693 command code: $_h$29

The APDU is

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

g. Lock DSFID

ISO 15693 command code: $_h$2A

The APDU is

`FF FE 05 00 02 22 2A`

Locking the DSFID can't be cancelled. Do not perform this operation without a good understanding of the consequence.

h. Get system information

ISO 15693 command code: $_h$2B

The APDU is

`FF FE 05 00 02 22 2B`

Note: the **H663** always sends the *Get system information* command to the VICC, as part of the discovery process. Invoke the GET DATA instruction (§ 3.2.1) to retrieve the value already returned by the VICC to the **H663**.

4.6. Other non-ISO PICCs

4.6.1. NFC Forum Type 3 Tags / Felica

The PICCs covered by this chapter are:

- Felica Lite, Felica Lite-S (**PIX.NN = $_h$003B**),
- Any PICC compliant with the specification of the NFC Forum Type 3 Tag.

Please visit www.nfcforum.org to get the Type 3 Tag specification.

a. READ BINARY instruction

In the READ BINARY command APDU,

- P1 must be $_h$00,
- P2 is the address of the first block to read.

Since the size of a block is 16 bytes, **Le** must be multiple of 16 ($_h$10). When Le=$_h$00, a single block is returned (16 bytes).

It is possible to read up to 8 blocks at once.

The READ BINARY instruction is translated into the Felica "CHECK" command, using the current *SERVICE CODE for READ BINARY* value as the "Service Code" parameter to the command. The default value for this parameter is $_h$000B. See § 3.3.6 if you need to change value.

b. UPDATE BINARY instruction (single byte)

In the UPDATE BINARY command APDU,

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

Since the size of a block is 16 bytes, **Lc** must be 16 ($_h$10), exactly.

The UPDATE BINARY instruction is translated into the Felica "UPDATE" command, using the current *SERVICE CODE for UPDATE BINARY* value as the "Service Code" parameter to the command. The default value for this parameter is $_h$0009. See § 3.3.6 if you need to change value.

4.7. Other non-ISO VICCs

4.7.1. EM4134

These VICCs use the ISO 15693 bit modulation, but a vendor-specific frame format and command set. They are recognized by **PIX.NN = hFF D6**. They are divided into 16 words, each word being 32-bit (4-Byte) wide.

a. READ BINARY instruction

In the READ BINARY command APDU,

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

Since the size of a word is 4 bytes, **Le** must be multiple of 4 ($_h$04). When Le=$_h$00, a single word is returned (4 bytes).

It is possible to read up the complete card's content (16 words) at once.

b. UPDATE BINARY instruction

In the UPDATE BINARY command APDU,

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

Since the size of a word is 4 bytes, **Lc** must be 4 ($_h$04), exactly.

c. Lock

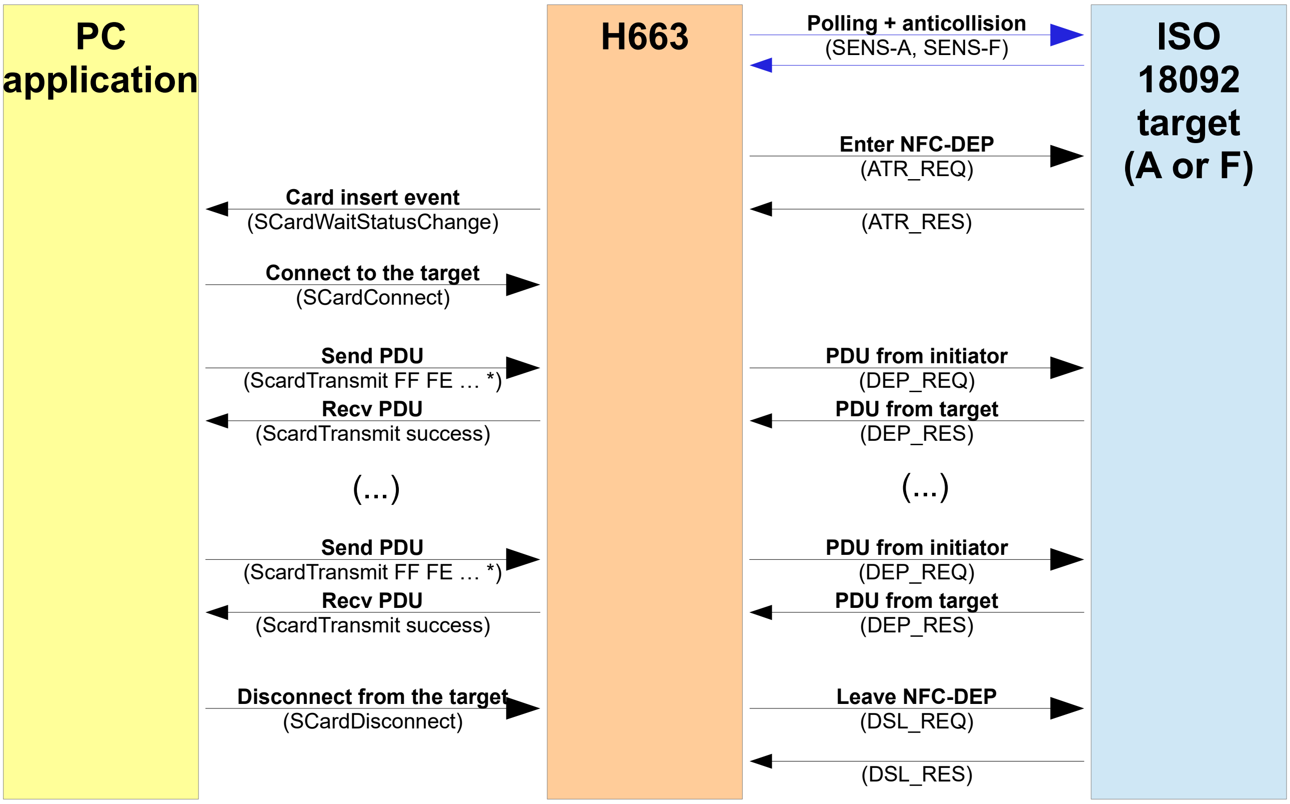
Locking a word is implemented through the **RFID MEMORY CONTROL instruction**, using the **Lock Block** function code (§ 3.3.4.c).

5. Using the H663 with a NFCIP-1 target

5.1. Introduction

The **H663** is a NFC Initiator. It could activate a remote NFC Target (only the passive communication scheme is available).

The **H663** implements the ISO 18092 "NFCIP-1" Transport Protocol, also named NFC-DEP by the NFC Forum.



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

5.1.1. Functions performed by the coupler

The **H663** 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.

5.1.2. Functions to be implemented on the PC

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

As the **H663** 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 **H663** 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 § 5.3.1

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

5.2.1. ATR of an ISO 18092 target

The **H663** builds a pseudo-ATR using the standard format defined in PC/SC specification:

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	…	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

5.2.2. 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 **H663** 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 **H663** 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

5.3. Advanced features

5.3.1. Changing the G_I bytes in the ATR_REQ

The General Bytes to be transmitted in the **H663**'s *ATR_REQ* (G_I bytes) are stored in **register $_h$E1**.

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 time-out = 1.5 seconds

Use the *PUSH REGISTER* command (§ 6.3.7) to set the new General Bytes before putting a new NFC target in front of the **H663**'s antenna.

Alternatively, use the *WRITE REGISTER* command (§ 6.3.6) if you want the new configuration to be permanent. Pay attention that the non-volatile memory has a limited write endurance.

6. Direct control of the H663

6.1. Basis

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

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

*Not all PC/SC drivers allow the application to gain direct access to the coupler. If you're using **SpringCard SDD480 PC/SC driver for Windows**, there's nothing specific to do, but for other drivers, a specific configuration of the driver has to be performed. Please refer to chapter 9: Annex B – activating SCardControl with the different drivers.*

6.2. Implementation details

6.2.1. 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;
}
```

6.2.2. Link to SpringProx 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** coupler running in **legacy mode**.

The detailed reference of all the command supported by our readers is available in **SpringCard CSB4**, **K531**, **K632** or **K663** 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.

6.2.3. Format of response, return codes

When the dialogue with the **H663** 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 coupler's status code: $_h$00 on success, a non-zero value upon error. The complete list of the **H663**'s error codes is given in chapter 8: Annex A – Specific error codes.

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

6.2.4. 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.

6.3. List of available control sequences

6.3.1. Action on the LEDs

a. Setting the coupler's LEDs manually

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

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

For a coupler 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 **H663**, 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 coupler's firmware)

Send the sequence

```
58 1E
```

To go back to default mode.

6.3.2. 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 **H663**, 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 coupler's firmware)

Send the sequence

`58 1C`

To go back to default mode.

6.3.3. Obtaining information on coupler and slots

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. *Coupler "product-wide" information*

Sequence	Will return…
58 20 01	Vendor name ("SpringCard")
58 20 02	Product name
58 20 03	Product serial number (in ASCII)
58 20 04	USB vendor ID and product ID (in ASCII)
58 20 05	Product version (in ASCII)
58 20 80	Number of slots (raw value on 1 byte)
58 20 83	Product serial number (raw value on 4 bytes)
58 20 84	USB vendor ID and product ID (raw value on 4 bytes)
58 20 85	Product version (raw value on 3 bytes: major/minor/build)

b. *Slot related information*

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

Slot naming obey to the following rule:

- The contactless slot is named "Contactless",
- When a contact smartcard slot is present, its name is "Contact",
- When only one SIM/SAM slot is present, its name is either "SAM A" or "SAM" depending on the configuration set in factory,
- When more than one SIM/SAM slots are present, they are named "SIM/SAM A", "SIM/SAM B", "SIM/SAM C" and "SIM/SAM D".

> ***SpringCard** CCID driver for Windows (ref. SDD480) uses those names to construct the list that is returned to SCardListReaders. Other drivers are likely to implement a different naming convention.*

6.3.4. Stopping / starting a slot

When a slot is stopped, the **H663**

- powers down the smartcard in the slot (if some),
- disable the slot[22],
- send the "card removed" event if there was a card in the slot.

When a slot is started again, the **H663**

- enable the slot[23],
- 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	Action
58 22	Stop current slot

b. Starting a slot

Sequence	Action
58 23	Start current slot

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

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

6.3.5. Forced insert/remove sequences

Use these sequences to emulate card insertion or removal. This is useful for hardware where no "card presence" switch is available.

Pay attention that if no card is actually present when the Card insertion is performed, the **H663** will be busy a long time (for nothing), before eventually giving up and reporting the "card mute" status.

DO NOT use this feature unless explicitly advised by SpringCard's support team.

a. Card insertion

Sequence	Action
58 40 01 01	Simulate a card insertion in the 1st card slot (ID-1 slot if existing)
58 40 01	Simulate a card insertion in the current slot

b. Card remove

Sequence	Action
58 40 01 00	Simulate a card removal from the 1st card slot (ID-1 slot if existing)
58 40 00	Simulate a card removal from the current slot

6.3.6. Reading/writing H663's configuration registers

The **H663** features a non-volatile memory to store configuration registers.

See chapter 7 for the list of these registers, and their allowed values.

a. Reading coupler'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 coupler'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 a different value in a configuration register more than 100 times may permanently damage your product.

The configuration is loaded upon reset. To apply a new configuration, you must reset the H663 (or cycle power).

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

6.3.7. 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 **H663** will reload its configuration registers from the non-volatile memory.

7. Configuration registers

The **H663** features a non-volatile memory to store its configuration.

The memory is divided into “registers”. Every register is identified by its address (a 1-B value) and is documented in this chapter.

Warning 1
Some registers are not listed in this chapter, yet they may have been defined in factory, or should use the default value for correct operation. Do not write or erase any register that is not listed in this chapter.

Warning 2
The non-volatile memory has a limited write/erase endurance.
Writing a different value in a configuration register more than 100 times may permanently damage your product.

7.1. Editing coupler's configuration

7.1.1. By software

The coupler's configuration registers are made available through a **SCardControl** function call. Refer to § 6.3.6 for details.

The configuration is loaded upon reset. To apply the new configuration, the software shall prompt the user to reset or unplug/plug the H663.

7.1.2. Using SpringCard MultiConf software

SpringCard has developed a versatile configuration software[24] that covers most products, including the **H663** and all readers in the **H663 family**.

Download **SpringCard MultiConf** at http://www.springcard.com/en/download/find/file/sn14007

[24] Available for the Windows platform only.

7.2. List of the configuration registers available to the end-user or integrator

Address	Section	Name	See §
$_h$B0	Contactless	Enabled protocols	7.5.1
$_h$B2	PC/SC	CLA of the APDU interpreter	7.4.2
$_h$B3	PC/SC	RF behaviour in PC/SC mode	7.4.3
$_h$B4	Contactless	Parameters for polling	7.5.2
$_h$C3	7816	Options for the smartcard slots	7.8.1
$_h$C4	Contactless	Allowed baudrates in T=CL	7.5.4
$_h$C5	Contactless	Options for T=CL	7.5.5
$_h$C8	Contactless	Number of antennas + compatibility settings	7.5.6
$_h$C9	Contactless	Options for polling	7.5.3
$_h$CA	Core	Configuration of the LEDs	7.3.1
$_h$CB	Core	Options for the LEDs and GPIOs	7.3.2
$_h$CC	Core	Behaviour of the LEDs and buzzer	7.3.3
$_h$CF	Felica	Service Codes for Felica read/write	7.6.1
$_h$E1	NFC P2P	Global Bytes bytes in ATR_REQ	7.7.1

Do not write or erase any register that is not listed in this chapter.

7.3. Core configuration

7.3.1. Configuration of the LEDs

Address: $_h$CA – Size: 2 bytes

	Bit	Action if set	Note
msb	15 - 12	LED 1 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	
	11 - 8	LED 2 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	
	7 - 4	LED 3 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	
lsb	3 - 0	LED 4 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	

Default value: $_h$0000

7.3.2. Options for the LEDs and GPIOs

Address: $_h$C9 – Size: 1 byte1

	Bit	Action if set	Note
msb	7	Use PWM for buzzer	
	6	RFU	
	5	RFU	
	4	RFU	
	3	Invert logic for LED 4	
	2	Invert logic for LED 3	
	1	Invert logic for LED 2	
lsb	0	Invert logic for LED 1	

Default value: $_h$00

7.3.3. Behaviour of the LEDs and buzzer

If the coupler has some LEDs, the coupler 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 § 6.3.1.a and 3.4.1.a).

If the coupler 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 § 6.3.2 and 3.4.1.b).

Address: $_h$CC – Size: 1 byte

	Bit	Values / Meaning
msb	7	1 : the H663 does signal its state on the LEDs 0 : the H663 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)

7.4. PC/SC CONFIGURATION

7.4.1. Slot naming and startup mode

Address: ₕB1 – Size: 1 byte

	Bit	Action if set	Note
msb	7	Force a letter in the name of the SAM slots	Even if there's only one slot, it will be named “SAM A”
	6	Force a letter in the name of the ID-1 slots	Even if there's only one slot, it will be named “Contact A”
	5	RFU	
	4	Prefix the slot name using the product's serial number (in hex)	This is useful for computers with numerous products attached
	3	Start with SAM slot(s) OFF	All the SAM slot(s) will not run until resumed by a Control command
	2	Start with Contact slot OFF	The Contact slot will not run until resumed by a Control command
	1	Start with Contactless slot OFF	The Contactless slot will not run until resumed by a Control command
msb	0	No contactless slot	The Contactless slot will not be enumerated (and will never run)

Default value: $_h$00

7.4.2. CLA byte of APDU 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

7.4.3. Behaviour of the contactless slot in PC/SC mode

This register defines the behaviour of the **H663**'s contactless slot in PC/SC mode.

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
	6	Use only standard values for PIX.NN in the ATR	Numerous contactless PICCs/VICCs have not been registered by their vendor in the PC/SC specification to get a standard PIX.NN. SpringCard has defined vendor-specific values for those cards (see 4.1.5). If this bit is set, these non-standard values will not be used, and PIX.NN will be fixed to $_h$0000 for all PICCs/VICCs that are not in the standard.
	5	Disable the pause in RF field after the PICC/VICC has been removed	When the PICC/VICC stops responding, the H663 pauses its RF field for 10 to 20ms. Setting this bit disable this behaviour.
	4	Disable the pause in RF field after the PICC/VICC during the polling	During the polling sequence, the H663 pauses its RF field for 10 to 20ms between the polling loops. Setting this bit disable this behaviour.
	3	No NFC-DEP activation over Felica (ISO 18092 @ 212 or 424 kbit/s)	
	2	No NFC-DEP activation over ISO 14443-A (ISO 18092 @ 106 kbit/s)	
	1	No T=CL (ISO-DEP) activation over ISO 14443-B	Send SLOT CONTROL P1,P2=$_h$20,01 to activate the PICC manually
lsb	0	No T=CL (ISO-DEP) activation over ISO 14443-A	Send SLOT CONTROL P1,P2=$_h$20,02 to activate the PICC manually

Default value: $_h$00 (T=CL active over 14443 A and B, NFC-DEP active over 14443 A and Felica)

7.5. Contactless configuration

7.5.1. Enabled protocols

This register defines the list of protocols the **H663** will look for during its polling loop. Any PICC/VICC compliant with one of the active protocols will be "seen", and the others ignored.

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

	Bit	Active. protocol (if set)	Version
msb	15	*RFU*	
	14	*RFU*	
	13	*RFU*	
	12	JIS:X6319-4 (Felica) ISO 18092 @ 212 kbit/s and 424 kbit/s NFC Forum Type 3 Tags	
	11	ThinFilm	
	10	NFC Forum Type 1 Tags (Innovision/Broadcom chips)	
	9	*RFU*	
	8	EM 4134	≥ 1.81
	7	Innovatron (legacy Calypso cards – sometimes called 14443-B')	
	6	*RFU*	
	5	ST Micro Electronics SRxxx	
	4	Inside Contactless PicoPass (also HID iClass)	
	3	*RFU*	
	2	ISO 15693	
	1	ISO 14443-B NFC Forum Type 4-B Tags	
lsb	0	ISO 14443-A ISO 18092 @ 106kbit/s NFC Forum Type 2 and Type 4-A Tags	

Default value: $_h$F7FF (all supported protocols but ThinFilm are activated)

7.5.2. Parameters for polling

This register defines the parameters used by the **H663** for the PICC/VICC polling.

Address: $_h$B4 – Size: 5 bytes

Byte	Data	Default value	Remark
0	AFI for ISO 14443-B	$_h$00	Specify the *Application Family Identifier* to be used during ISO 14443-B polling. $_h$00 means that all PICCs shall answer.
1	AFI for ISO 15693	$_h$00	Specify the *Application Family Identifier* to be used during ISO 15693 polling. $_h$00 means that all VICCs shall answer.
2 - 3	SC for JIS:X6319-4 and ISO 18092 @ 212 and 424 kbit/s	$_h$FFFF	Specify the *System Code* to used during Felica polling (SENSF_REQ). The value is stored MSB first. $_h$FFFF means that all targets shall answer.
4	RC for JIS:X6319-4 and ISO 18092 @ 212 and 424 kbit/s	$_h$00	Specify the *Request Code* to used during Felica polling (SENSF_REQ). This value shall be $_h$00 to accept both NFC Type 3 Tags and NFC devices running in P2P mode (NFC-DEP), or $_h$01 to accept only NFC Type 3 Tags

7.5.3. Options for polling

Use this register to configure the extended ATQB support for ISO 14443-B cards, and to disable JIS:X6319-4 / ISO 18092 @ 424 kbit/s.

Address: $_h$C9 – Size: 1 byte

	Bit	Action if set	Note
msb	7	RFU	
	6	RFU	
	5	RFU	
	4	Activate extended ATQB	If this bit is set, the H663 will ask for an extended ATQB from ISO 14443-B. Not all cards do support this feature.
	3	Disable JIS:X6319-4 / ISO 18092 @ 424 kbit/s	If this bit is set, the H663 will communicate with Felica cards and NFC P2P targets up to 212 kbit/s only
	2	RFU	
	1	RFU	
lsb	0	RFU	

Default value: $_h$00 (normal ATQB, allow 424kbit/s for JIS:X6319-4)

7.5.4. Allowed baudrates in T=CL (ISO 14443-4)

Use this register to let the **H663** negotiate a baudrate greater than 106 kbit/s with ISO 14443-4 PICCs (DSI, DRI defined in PPS for ISO 14443 A, in ATTRIB for ISO 14443 B).

*The **H663** is theoretically able to communicate with PICCs at 848 kbit/s in both directions, but the actual maximum speed depends heavily on the characteristics of the PICC, and on the coupler's actual antenna and environment.*

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 PICC → H663 @ 848 kbit/s (DSI = 3 in PPS)
	13	Allow ISO 14443 A PICC → H663 @ 424 kbit/s (DSI = 2 in PPS)
	12	Allow ISO 14443 A PICC → H663 @ 212 kbit/s (DSI = 1 in PPS)
		ISO 14443-A DR
	11	RFU, must be 0
	10	Allow ISO 14443 A H663 → PICC @ 848 kbit/s (DRI = 3 in PPS)
	9	Allow ISO 14443 A H663 → PICC @ 424 kbit/s (DRI = 2 in PPS)
	8	Allow ISO 14443 A H663 → PICC @ 212 kbit/s (DRI = 1 in PPS)
		ISO 14443-B DS
	7	RFU, must be 0
	6	Allow ISO 14443 B PICC → H663 @ 848 kbit/s (DSI = 3 in ATTRIB)
	5	Allow ISO 14443 B PICC → H663 @ 424 kbit/s (DSI = 2 in ATTRIB)
	4	Allow ISO 14443 B PICC → H663 @ 212 kbit/s (DSI = 1 in ATTRIB)
		ISO 14443-B DR
	3	RFU, must be 0
	2	Allow ISO 14443 B H663 → PICC @ 848 kbit/s (DRI = 3 in ATTRIB)
	1	Allow ISO 14443 B H663 → PICC @ 424 kbit/s (DRI = 2 in ATTRIB)
lsb	0	Allow ISO 14443 B H663 → PICC @ 212 kbit/s (DRI = 1 in ATTRIB)

Default value: $_h$3333 (up to 424 kbit/s).

You must lower-down the allowed baudrates to 106kbps ($_h$0000) if your antenna is not capable to handle the higher baudrates without communication errors.

7.5.5. Options for T=CL (ISO 14443-4)

This register defines the behaviour of the ISO 14443-4 subsystem.

Address: $_h$C5 – Size: 4 bytes

Byte	Data	Default value	Remark
0	Extra guard time	$_h$00	Guard time (specified in ms) to add before sending a frame to the PICC.
1	Retries on card mute	$_h$03	Number of retries before giving up when the PICC does not answer (communication timeout, and no other error detected)
2	Retries on comm. error	$_h$03	Number of retries before giving up when the PCC does not understand the PICC's response (CRC, parity, framing errors...)
3	RFU	$_h$00	*This byte must be $_h$00*

7.5.6. Number of antennas + auto-stop

Address: $_h$C8 – Size: 1 byte

	Bit	Action if set	Note
msb	7	RFU	
	6	RFU	
	5	RFU	
	4	Be compliant with RCTIF	This feature changes the polling sequence and the behaviour against Calypso Innovatron cards
	3	RFU	
	2	RFU	
	1	RFU	
lsb	0	Activate the secondary antenna	

Default value: $_h$00 (only one antenna)

Please refer to doc. PNA2236 “H663 integration guide” for information regarding the second antenna. This feature is available for H663S (unbalanced) only. Note that both antennas must have exactly the same RF characteristics.

7.6. Felica configuration

7.6.1. Service Codes for Felica read/write

Use this register to define how the **H663** processes Felica cards and NFC Type 3 Tags.

Address: $_h$CF – Size: 4 bytes

Byte	Data	Default value	Remark
0 - 1	Read Service Code	$_h$000B	Service Code used when the READ BINARY instruction is invoked (MSB first) The value $_h$000B is mandated by the specification of the NFC Forum Type 3 Tag
2 - 3	Update Service Code	$_h$0009	Service Code used when the UPDATE BINARY instruction is invoked (MSB first) The value $_h$0009 is mandated by the specification of the NFC Forum Type 3 Tag

Those values may be temporarily overwritten right into the *SCardTransmit* stream using the **SET FELICA RUNTIME PARAMETERS** instruction (§ 3.3.6).

7.7. ISO 18092 / NFC-DEP configuration

7.7.1. Global Bytes in ATR_REQ

Address: $_h$E1 – Size: 0 to 15 bytes

This register defines the G_I bytes sent in ATR_REQ.

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

7.8. ISO 7816 CONFIGURATION

7.8.1. Options for the smartcard slots

This register defines the parameters used by the **H663** for the smartcard and SIM/SAM slots.

Address: $_h$C3 – Size: 5 bytes

Byte	Data	Default value	Remark
0	Configuration of the ID-1 slot	$_h$B3	- Contact if the coupler has a ID-1 slot - not used otherwise
1	Configuration of the SAM1 slot	$_h$B3	- “SAM A” if the coupler has 1 or 4 SAMs - not used otherwise
2	Configuration of the SAM2 slot	$_h$B3	- “SAM A” if the coupler has 3 SAMs - “SAM B” if the coupler has 4 SAMs
2	Configuration of the SAM3 slot	$_h$B3	- “SAM B” if the coupler has 3 SAMs - “SAM C” if the coupler has 4 SAMs
4	Configuration of the SAM4 slot	$_h$B3	- “SAM C” if the coupler has 3 SAMs - “SAM D” if the coupler has 3 SAMs

Every byte's bits are defined as follow:

	Bit	Action if set	Note
msb	7	Enable automatic PPS	
	6	Enable HSP	Use this only with Calypso SAMs
	5	Enable EMV power on	EMV mode is tried before standard mode
	4	Enable non-EMV power on	Non-EMV cards will be rejected
	3	RFU	
	2	Enable class C (1.8V)	The coupler tries the lower voltage class first, and then increments until one matches.
	1	Enable class B (3V)	
lsb	0	Enable class A (5V)	

$_h$B3 stands for

- PPS automatic
- HSP disabled
- EMV power on tried before standard power on
- Class = AB (3V then 5V)

8. Annex A – Specific error codes

When the APDU interpreter returns SW1 = $_h$6F, the value of SW2 maps to one of the **H663** specific error codes listed below.

SW2	Symbolic name[25]	Meaning
$_h$01	MI_NOTAGERR	No answer received (no card in the field, or card is mute)
$_h$02	MI_CRCERR	CRC error in card's answer
$_h$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 coupler's RF field

[25] 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	Coupler: invalid parameter
h64	MI_UNKNOWN_FUNCTION	Coupler: invalid opcode
h70	MI_BUFFER_OVERFLOW	Coupler: internal buffer overflow
h7D	MI_WRONG_LENGTH	Coupler: invalid length

9. Annex B – activating SCardControl with the different drivers

Being compliant with the CCID specification, the **H663** is supported by (at least) 4 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.

9.1. 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).*

9.2. Direct control using MS USBCCID

With **MS USBCCID** driver, direct control of the coupler must be enabled on a per-coupler basis: each coupler has its own USB serial number, and the direct control has to be unequivocally 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_91B1
                            yyyyyyyy
                                Device Parameters
```

where *yyyyyyyy* is the coupler's 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 coupler 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).*

9.3. Direct control using MS WUDFUsbccidDriver

With **MS WUDFUsbccidDriver** (new user-mode driver introduced in Windows 7), direct control of the coupler must also be enabled on a per-coupler basis: each coupler has its own USB serial number, and the direct control has to be unequivocally 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_91B1
                                   yyyyyyyy
                                        Device Parameters
                                             WUDFUsbccidDriver
```

where *yyyyyyyy* is the coupler's 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 coupler 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).*

9.4. DIRECT CONTROL USING PCSC-LITE CCID

To be written.

10. 3rd-party licenses

SpringCard H663 uses one 3-rd party open-source software component.

10.1. FreeRTOS

FreeRTOS is a market leading real time operating system (or RTOS) from Real Time Engineers Ltd.

Starting from firmware version 2.00, **SpringCard H663** runs on FreeRTOS v8.2.0.

FreeRTOS is distributed under a modified GNU General Public License (GPL) that allows to use it in commercial, closed-source products.

For more information, or to download the source code of FreeRTOS, please visit

www.freertos.org

Editor's information

SPRINGCARD SAS company with a capital of 227 000 €

RCS EVRY B 429 665 482

Parc Gutenberg, 2 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