PMDE051-DD
15/12/14

SpringCard SpringProx Contactless Couplers

Developer's Guide

www.springcard.com

Document Identification

Category	Developer's Guide		
Family/Customer	SpringProx		
Reference	PMDE051	Version	DD
Status	draft	Classification	Public
Keywords	K531, K632, K663, CSB4, Legacy		
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\Commun\Developpement SpringProx\[PMDE051-DD] SpringProx developer's guide.odt		
Date saved	02/03/17	Date printed	

Revision History

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	15/03/04	JDA				Early draft
AB	25/05/04	JDA				Some errors corrected
BA	19/06/06	JDA				Added ISO 14443-B and low level interface
BB	22/03/07	JDA				Added listing of error codes
BC	28/02/08	LTC				Added ISO 15693
BD	25/03/09	LTC				Some errors corrected
CA	15/06/10	JDA				New SpringCard layout, major rewriting Added automated card discovery (version 1.54)
CB	18/05/11	JDA				Documented the Bus protocol Documented functions for Innovatron and ICODE1 mode Improved automated card discovery (version 1.56 build 25) Dropped fast/slow separation for ISO 15693 and ICODE1 (now decided in the coupler, not accessible to the application)
CC	11/07/11	JDA				Documented the change baudrate command
DA	15/01/14	JDA				Updated template Added K663 and derivates (Prox'N'Drive)
DB	04/06/14	JDA				Cleanup Documented LPCD
DC	11/12/14	JIZ				ISO 15693: modification of the Standard Frame Exchange (§10.2.1)
DD	15/12/14	JDA				Fixed a few typos and mistakes in chapter 11

Contents

1. Introduction

1.1. Abstract

This document provides all necessary information to operate a **SpringCard contactless coupler** (a.k.a. "**SpringProx Coupler**") through its serial interface, using the SpringCard "legacy" protocol.

This document covers both the protocol itself and the set of functions offered by the **SpringProx Coupler** and that have to be invoked by an application running in the host system in order to "process" the contactless cards or tags.

1.2. Important – Read me first

SpringCard has developed a comprehensive and complete software library that implements everything that is described in this document (communication protocols, host-side high-level API), and more[1].

This software library is called "**SpringProx API**" and is available free of charge in the SDK provided by **SpringCard**, both as source code portable portable on virtually every system (ANSI C) and as binary for Windows (`springprox.dll`).

The **SpringProx API** library facilitates the integration and limits software development effort, thanks to the samples supplied in the SDK that could be used immediately. It also provides full abstraction of the coupler actually in use. Il is documented online at

http://apidoc.springcard.com/springprox/

Using the SpringProx API is therefore the recommended solution to operate SpringProx Couplers.

When using the **SpringProx API**, you may keep this document as a reference, but it is advised whenever it is possible to invoke high-level functions and to avoid any direct access to the coupler.

1.3. Supported products

At the date of writing, this document refers to every SpringCard contactless product running a "legacy" firmware **version ≥ 1.51**[2] :

- **SpringProx-CF**, **SpringProx-RC**, etc,
- **K531**, **K632** and their derivatives, such as the **CSB4** family,
- **K663** and its derivatives, such as **Prox'N'Drive**,

[1] An important feature of the SpringProx API is its "knowledge" of the various software and hardware versions, and its ability to overcome silently (and most of the time efficiently) a function that is missing or that is known to have limitations in some versions, as the present document only covers the version that is currently shipping at the date of writing.

[2] For older products: either upgrade the firmware with an up-to-date version, or refer to earlier versions of this document.

- All other **SpringCard** products supporting **Legacy mode**, once they have been configured to operate in this mode.

Please refer to the product leaflet and the integration guide of each product for accurate specifications and a detailed list of features.

1.4. Product groups – compatibility matrixes

Note that **some card technologies are not available to all products**, due to limitations in hardware. For every card technology, the document contains a "compatibility matrix" that states which hardware groups do/don't support which technology.

Here's the list of the hardware groups:

Group	*Products in this group*	*Supported tech.*
K531 group	**K531, K531-TTL, K531-232** **CSB4.1, CSB4.2, CSB4.3** **SpringProx-CF** **SpringProx-CFUP**	ISO 14443 (NFC-A, NFC-B) and related
K632 group	**K632, K632-TTL, K632-232** **CSB4.4** **CSB6, CrazyWriter, CrazyWriter-LT (in Legacy mode)**	ISO 14443 (NFC-A, NFC-B) and related (but ASK CTS) ISO 15693 (NFC-V) ICODE1
K663 group	**K663, K663-TTL, K663-232** **CSB4.6** **Prox'N'Drive HF**	ISO 14443 (NFC-A, NFC-B) and related (but ASK CTS) ISO 15693 (NFC-V) Felica (NFC-F)

This documentation covers the 3 groups, but pay attention that some functions are missing from some hardware.

NB: the actual features supported by a coupler are exposed to the host thanks to the Get Reader Capabilities command (see 6.1.2 and Table 4).

1.5. 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 the 13.56 MHz RFID and contactless card standards (ISO 14443 and ISO 15693), 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.

1.6. Support and updates

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

www.springcard.com

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

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

www.springcard.com/support

2. RFID, contactless smartcards and NFC: quick introduction and glossary

2.1. Smartcard 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[3],
- **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 **SpringProx Coupler** could be seen as a gateway that implements this **APDU exchange** stuff, with a relative abstraction from the transport-level protocols.

2.2. Contactless cards that are not smartcards

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:

[3] 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.

- **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...).

In these cases, the card could only be access through its (more or less) proprietary transport protocol, and using its (generally fully) proprietary command set.

In this context, the **SpringProx Coupler** is no more a "dumb" gateway between the application and a smart card; it must embed in its firmware all the logic required to transform a high level "READ" or "WRITE" command into a set of action other the RF field that makes sense for the very contactless card that is there.

The logic embedded in the **SpringProx Coupler** hides the complexity of the cards to the host system, yet the host application remains responsible of selecting the protocol(s) it want to support, then invoking the functions that are relevant for the card it has recognized.

2.3. 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).

In the **SpringCard SpringProx Coupler** family, only the **K663 group** is partially compliant with NFCIP-1 (initiator role, passive communication mode only). The couplers in this group are also compliant with all NFC Forum Tag types (T1, T2, T3, T4A and T4B).

The K531 and K632 groups are not compliant with NFCIP-1 and support only T1, T2, T4A, T4B.

> *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**.*

2.4. 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 7816-4:** 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 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 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:

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

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 (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

[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.

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. Serial communication

3.1. Overview

The **SpringProx Coupler** uses a master-slave scheme, where the coupler is the slave of a 'host' system (either a computer or a microcontroller).

Therefore, the communication flow is driven by the host: a Command from the host is followed (after necessary processing time) by a Response from the coupler.

The communications follow a three-layer scheme:

- The Command-Response layer,
- The Transport layer,
- The Physical layer.

3.2. The Command-Response layer

The Command-Response layer handles and interprets the SpringProx commands and responses.

Chapters Erreur : source de la référence non trouvée to 12 fully describe the Command set and the Response format.

a. Format

Commands and Response are made of three fields:

- The **command code** (CMD) for a Command or the **status code** (STA) for a Response,
- The size of the data (LEN),
- The **data** itself, if some (DATA).

b. Status codes

The status returned by the coupler in its Response is explained on a single byte, from $_h00$ to $_h80$.

For coherence with the documentation of the SpringProx API, the status codes are **documented as negative decimal values**.

This is only a convention. Translation from one representation to another is trivial:

- $_h00$ ⇔ 0 means "Success",
- $_h01$ to $_h7F$ ⇔ -1 to -127 are warning and error codes (see chapter 12 for the complete list),
- $_h80$ ⇔ -128 means "time extension" (see paragraph d below).

c. Length

When size of data is lower than 128, then the LEN field is expressed as a single byte *(value between 0 and 127)*.

When size of data is between 128 and 255, the LEN field is expressed as 2 bytes:

- The 1st byte is fixed to $_h80$ *(128)*
- The 2nd byte is equal to LEN minus $_h80$ *(value between 0 and 127)*.

When size of data is 256 or greater, the LEN field is expressed as 3 bytes:

- The 1st byte is fixed to $_h80$ *(128)*
- The 2nd byte is fixed to $_h80$ *(128)*
- The 3rd byte is equal to LEN minus $_h100$ *(value between 0 and 127)*.

d. Processing timeout and Time Extension

The coupler shall always send its Response within 1000ms after receiving the Command[8].

If the processing is not terminated within 1000ms, the coupler sends a Time Extension frame. The Time Extension frame is repeated every 600ms to 800ms until processing is terminated; then the actual Response takes place.

The host shall not answer the Time Extension frame, nor send any new Command while the coupler is still processing the previous Command.

3.3. The Transport layer

The Transport layer handles message addressing, specifies the transmission type, and validates every transmission. The transport layer can use one out of four protocols.

- The ASCII protocol,
- The (modified) OSI3964R protocol,
- The “Fast” protocol,
- The “Bus” protocol.

The “Fast” protocol is the recommended choice.

NB: the “Bus” protocol is only available in the firmware of products based on an RS-485 physical interface (see Table 4: Reader’s capabilities).

a. Protocol selection

The choice of the protocol is up to the host. Upon start-up, the coupler is ready to accept all protocols, and will answer using the same protocol as used by the host.

[8] The returning Wait One Card (§ 4.2.1) and Wait Multiple Cards (4.2.2) are the two only exception to this rule.

b. Protocol change

The host may switch at any time from ASCII protocol to either OSI or "Fast" protocols. Reverting to ASCII protocol from another is not possible (until the coupler resets).

The host may switch at any time from OSI protocol to "Fast" protocol. Reverting to OSI protocol from "Fast" is not possible (until the coupler resets).

3.3.2. The ASCII protocol

This is a lightweight protocol, easy to implement in a host with small RAM capacity, or even manually under a terminal-emulation program (HyperTerminal for instance).

The transmission is made in ASCII Hexadecimal ($_{h}$12 $_{h}$34 $_{h}$56 is transmitted as string "123456"). The only valid characters are '0' to '9', 'A' to 'F' or 'a' to 'f', and the characters used for control ('$', '+', '-', {CR}, {LF}). Any other character would be discarded.

a. Host to coupler

$ <CMD_{ASC}> <LEN_{ASC}> [<DATA_{ASC}>] {CR} [{LF}]

The frame is prefixed by the ASCII dollar sign ('**$**') and terminated by a Carriage Return (CR). The Line Feed (LF) is optional.

The coupler immediately acknowledges the frame with an ASCII plus sign ('**+**').

If the coupler detects a communication error, it N-acknowledges with an ASCII minus sign ('**-**') followed by a NACK-code (see § 3.3.6).

b. Coupler to host

<abs(STA)$_{ASC}$> <LEN_{ASC}> [<DATA_{ASC}>] {CR} {LF}

c. Time Extension

The Time Extension frame is an ASCII plus sign ('**+**').

d. Communication timeouts

There's no communication timeout associated to this protocol.

3.3.3. The (modified) OSI3964R protocol

The OSI 3964R protocol is a feature-rich network protocol. The implementation in the coupler has strong limitations due to memory and code-size constraints.

NB: This protocol is implemented for compliance with legacy products. Using this protocol is not recommended in new projects.

a. Principles

At first, the sender sends the ASCII {STX} character ($_h$02). The receiver confirms it is present and listening, sending ASCII {DLE} character ($_h$10).

The payload starts by a Sequence number (SEQ), followed by the Command or Status (CMD or STA), then LEN and DATA. It is ended by a LRC, and terminated by the ASCII {DLE} {ETX} sequence ({ETX} = $_h$03).

The receiver acknowledges the frame by sending {DLE} again. When the receiver detects a communication error, instead of sending {DLE} it sends {NAK} ($_h$15) followed by an error code (see § 3.3.6).

In the payload itself (from <SEQ> to <LRC> included), every byte equal to one of the protocol-reserved values ({DLE}, {STX} and {ETX}) shall be prefixed by a {DLE} protocol byte.

b. Host to coupler

Host	*Coupler*
{STX}	
	{DLE}
<SEQ> <CMD> <LEN> [<DATA>] <LRC> {DLE} {ETX}	
	{DLE}

Where <LRC> = *XOR* (<SEQ> <CMD> <LEN> [<DATA>])

The host shall increment its Sequence number after each exchange.

c. Coupler to host

Coupler	*Host*

{STX}

{DLE}

<SEQ> <CMD> <LEN> [<DATA>] <LRC> {DLE} {ETX}

{DLE}

Where <LRC> = *XOR* (<SEQ> <CMD> <LEN> [<DATA>])

In its Response, the coupler always echoes the Sequence number received within the Command.

d. Time Extension

The Time Extension frame is a single ASCII {DLE} byte. The host shall not acknowledge this byte.

e. Communication timeouts

- Sender {STX} to receiver {DLE} : 20ms,
- Sender {DLE} {ETX} to receiver {DLE} : 20ms,
- Inter-byte timeout: 5ms.

3.3.4. The "Fast" protocol

This protocol is designed for **high-speed communication on a reliable physical layer**. It provides frame synchronization and checking but no software flow control.

Every frame starts with the ASCII {SYN} character ($_h$16), followed by a Sequence number (SEQ). The host shall increment its Sequence number after each exchange. In its Response, the coupler always echoes the Sequence number received within the Command.

The LRC field helps detecting communication errors.

a. Host to coupler

{SYN} <SEQ> <CMD> <LEN> [<DATA>] <LRC>

Where <LRC> = *XOR* (<SEQ> <CMD> <LEN> [<DATA>])

When the coupler detects a communication error, it sends {NAK} ($_h$15) followed by an error code (see § 3.3.6). When no error is detected, the frame is <u>not</u> acknowledged.

b. Coupler to host

{SYN} <SEQ> <STA> <LEN> [<DATA>] <LRC>

Where <LRC> = *XOR* (<SEQ> <CMD> <LEN> [<DATA>])

If the host detects a communication error, it may either send the same Command again, or ask the coupler to repeat its Response by sending a Repeat Command (CMD=$_h$80 and LEN= $_h$00) with the same Sequence number. When no error is detected, the frame is not acknowledged.

c. Time Extension

The Time Extension frame is formatted as a standard Response frame, having STA=$_h$80 and LEN= $_h$00 (and the current Sequence number).

d. Communication timeouts

The transmission timeout (total frame duration from {SYN} to <LRC>) is fixed to 400ms.

There's no inter-byte timeout associated to this protocol.

3.3.5. The "Bus" protocol

The "Bus" protocol is an extension to the "Fast" protocol providing an optionnal "address" feature. This makes it possible to install more than one coupler on a single communication line (typically, a RS-485 bus), and to communicate alternatively with all of them. The host is the master of the communication and is responsible to address every coupler one after the other so no collision may occur on the answers.

Every frame starts with the ASCII {SOH} character ($_h$01) followed by the address of the target ($_h$01 to $_h$FE for a coupler, $_h$00 for the host, $_h$FF being the broadcast address).

NB: the "Bus" protocol is only available in the firmware of products based on an RS-485 physical interface (see Table 4: Reader's capabilities).

a. Host to coupler

{SOH} <ADR> {ACK} <SEQ> <CMD> <LEN> [<DATA>] <LRC>

Where

- <ADR> = address of the coupler on the bus ($_h$01 to $_h$FE)
- <LRC> = *XOR* (<SEQ> <CMD> <LEN> [<DATA>])

When the coupler detects a communication error, it sends {NAK} ($_h$15) followed by an error code (see § 3.3.6).

When no error is detected, the coupler sends {ACK} ($_h$06), so the host knows that there's a coupler at this address (except for a broadcast frame).

b. Coupler to host

{SOH} <ADR> {ACK} <SEQ> <STA> <LEN> [<DATA>] <LRC>

Where

- <ADR> = address of the host on the bus, fixed to $_h$00
- <LRC> = *XOR* (<SEQ> <CMD> <LEN> [<DATA>])

If the host detects a communication error, it may either send the same Command again, or ask the coupler to repeat its Response by sending a Repeat Command (CMD=$_h$80 and LEN= $_h$00) with the same Sequence number.

When no error is detected, the host shall not acknowledge the frame.

c. Time Extension

The Time Extension frame is formatted as a standard Response frame, having STA=$_h$80 and LEN= $_h$00 (and the current Sequence number).

d. Communication timeouts

The transmission timeout (total frame duration from {SOH} to <LRC>) is fixed to 400ms.

There's no inter-byte timeout associated to this protocol.

3.3.6. Transport layer error codes

NACK code	Meaning
$_h$09	**Overrun:** at least one byte has been lost
$_h$0A	**Length error:** the length of the frame doesn't match the LEN value
$_h$0B	**LRC error:** computed LRC doesn't match the LRC value
$_h$0C	**Buffer overflow:** frame is longer than supported
$_h$0D	**Protocol error:** an invalid value has been received
$_h$0E	**Timeout error:** a communication timeout has expired while receiving
$_h$0F	**Hardware error:** fatal UART error (check physical line)

3.4. The Physical layer

The Physical layer handles the data transmission itself. The physical layer uses an asynchronous serial protocol. The actual implementation depends on the hardware associated to the K531/K632 module (RS-232, RS at TTL level, RS over USB, RS-485, …).

The default configuration is 38400bps, 8 data bits, 1 stop bit, no parity, and no flow control.

4. CARD LOOKUP - POLLING SEQUENCES

The *Automatic Card Discovery* functions offer an efficient way to detect cards on the RF interface, whatever their protocol.

4.1. FIND CARD

NB: this function was introduced in firmware 1.51 and is not available on earlier versions.

4.1.1. Single shot find

Command

CMD	LEN	0..1
h60	h02	*Protocols*

- ***Protocols*** is a 16-bit field (MSB first). Set the bit(s) corresponding to the protocol(s) to be looked for, according to **Table 1: Card lookup – Card protocols bitmap**.

Response *no card in the field*

STA	LEN
-1	h00

Response *OK*

STA	LEN	0..1	2..xx-1
0	*xx*	*Card protocol*	*Card UID / PUPI*

- ***Card protocol*** is a 16-bit field (MSB first) where only one bit will be set, telling which kind of card has been found, according to **Table 1: Card lookup – Card protocols bitmap** on next page.
- ***Card UID / PUPI*** gives the protocol-level identifier of the card. Its size depends on the actual protocol (4, 7 or 10 for ISO 14443 type A, 4 for ISO 14443 type B, 8 for ISO 15693, etc). Please refer to the standards for details.

Invoke one of the *Get Card Information* commands (§ 4.2) to retrieve other card data.

Table 1: Card lookup – Card protocols bitmap

Bit	Mask	Type of card	Comp.
0	$_h$0001	ISO 14443-A (including Mifare) ISO 18092 @ 106kbit/s NFC Forum Type 2 and Type 4-A Tags	1 2 3
1	$_h$0002	ISO 14443-B NFC Forum Type 4-B Tags	1 2 3
2	$_h$0004	ISO 15693	2 3
3	$_h$0008	NXP ICODE1	2
4	$_h$0010	Inside Contactless PicoPass (also HID iClass)	1 2 3
5	$_h$0020	ST MicroElectronics SRxxx	1 2 3
6	$_h$0040	ASK CTS256B or CTS512B	1 2
7	$_h$0080	Innovatron (legacy Calypso cards – sometimes called 14443-B')	1 2 3
8	$_h$0100	*RFU*	
9	$_h$0200	*RFU*	
10	$_h$0400	NFC Forum Type 1 Tags (Innovision/Broadcomm chips)	1 2 3
11	$_h$0800	Kovio RF barcode	2 3
12	$_h$1000	JIS:X6319-4 (Felica) ISO 18092 @ 212 kbit/s and 424 kbit/s NFC Forum Type 3 Tags	3
13	$_h$2000	*RFU*	
14	$_h$4000	*RFU*	
15	$_h$8000	*RFU*	

Compatibiliy matrix	
K531 Group	**1**
K632 Group	**2**
K663 Group	**3**

4.2. Polling loops

NB: those functions were introduced in firmware 1.54 and are not available on earlier versions.

4.2.1. Wait One Card

This function instructs the coupler to start waiting for a card, according to the specified protocol(s). The function exits when a card is found or a timeout occurs.

Command (complete)

CMD	LEN	0..1	2..3	4	5..6
$_h$60	$_h$07	*Protocols*	*Timeout (s)*	*Options and delay*	*Interval (ms)*

Command (shortcut : interval = 250ms)

CMD	LEN	0..1	2..3	4
$_h$60	$_h$05	*Protocols*	*Timeout (s)*	*Options and delay*

Command (shortcut : interval = 250ms, options and delay = $_h$00)

CMD	LEN	0..1	2..3
$_h$60	$_h$04	*Protocols*	*Timeout (s)*

- ***Protocols*** is a 16-bit field (MSB first). Set the bit(s) corresponding to the protocol(s) to be looked for, according to **Table 1: Card lookup – Card protocols bitmap**.
- ***Timeout*** (seconds): the coupler stops waiting when this timeout is expired. Set to $_h$FFFF for an endless waiting.
- ***Options and delay***: define the initial lookup delay. See **Table 2: Wait One Card – Options and delay** on next page. *This value could be omitted (shortcut with LEN = $_h$04) and defaults to $_h$00.*
- ***Interval*** (milliseconds): this is the period between two consecutive lookup shots. Increasing this value will reduce coupler's average consumption (as the RF field is switched OFF in-between), but the coupler will be less "reactive". 250ms is the recommended value. Value $_h$FFFF is forbidden. *This value could be omitted (shortcut with with LEN = $_h$04 or LEN = $_h$05) and defaults to 250 ($_h$00FA).*

Table 2: Wait One Card – Options and delay

BIT	Role	Values	
7-4	Initial lookup delay	$_b$0000	*No delay (polling starts immediately)*
		$_b$0001	*50ms delay*
		$_b$0010	*100ms delay*
		$_b$0011	*250ms delay*
		$_b$0100	*500ms delay*
		$_b$0101	*1s delay*
		$_b$0110	*1.5s delay*
		$_b$0111	*2s delay*
		$_b$1000	*2.5s delay*
		$_b$1001	*3s delay*
		$_b$1010	*4s delay*
		$_b$1011	*5s delay*
		$_b$1100	*10s delay*
		$_b$1101	*30s delay*
		$_b$1110	*60s delay*
		$_b$1111	*RFU, do not use*
3	RFU		*RFU, must be $_b$0*
2	RFU		*RFU, must be $_b$0*
1	RFU		*RFU, must be $_b$0*
0	RFU		*RFU, must be $_b$0*

***NB:** due to the tolerance on the internal timers of the devices, the actual time observed for timeout, delay, interval may be longer than specified.*

The *Wait One Card* function is the only one (apart is sister-function *Wait Multiple Cards*) that returns two Responses:

1st Response *polling started*

STA	LEN
-20	$_h$00

2nd Response *timeout expired or break, no card found*

STA	LEN
-30	$_h$00

2nd Response *card found*

STA	LEN	0..1	2..xx-1
0	*xx*	*Card protocol*	*Card UID / PUPI*

Refer to § 4.1 for details.

NB: to cancel a running *Wait One Card* function, invoke the *Wait Cancel* function (§ 4.2.3) or one of the *Get Card Information* functions (§ 4.3). Any other function will fail with STA=-20.

4.2.2. Wait Multiple Cards

This function instructs the coupler to start waiting for a card, according to the specified protocol(s). The function exits only when a timeout occurs. When a card is found, a Response is sent, but the function keeps on looking for another card.

Command

CMD	LEN	0..1	2..3	4	5..6	7..8
$_h$60	$_h$09	*Protocols*	*Timeout (s)*	*Options and delay*	*Interval A (ms)*	*Interval P (ms)*

- ***Protocols*** is a 16-bit field (MSB first). Set the bit(s) corresponding to the protocol(s) to be looked for, according to **Table 1: Card lookup – Card protocols bitmap**.
- ***Timeout*** (seconds): the coupler stops waiting when this timeout is expired. Set to $_h$FFFF for an endless waiting.
- ***Options and delay***: and define the initial lookup delay. See **Table 2: Wait One Card – Options and delay**. *This value could be omitted (shortcut with LEN = $_h$04) and defaults to $_h$00.*
- ***Interval A*** *(milliseconds):* this is the period between two consecutive lookup shots. Increasing this value will reduce coupler's average consumption (as the RF field is switched OFF in-between), but the coupler will be less "reactive". 250ms is the recommended value. Value $_h$FFFF is forbidden. This value could be omitted *(shortcut with with LEN = $_h$04 or LEN = $_h$05) and defaults to 250 ($_h$00FA).*
- ***Interval P*** *(milliseconds)*: this is the period between two consecutive lookup shots, when there's a card in the field. The "card found" Response will be repeated according to this period until the card is removed from the field. 50ms is the recommended value. Values $_h$0000 and $_h$FFFF are forbidden.

***NB:** due to the tolerance on the internal timers of the devices, the actual time observed for timeout, delay, interval A, interval B may be longer than specified.*

This function may return more than two Responses.

1st Response *polling started*

STA	LEN
-20	h00

Xth Response *timeout expired or break, no card found*

STA	LEN
-30	h00

Xth Response *card found*

STA	LEN	0..1	2..xx-1
0	*xx*	*Card protocol*	*Card UID / PUPI*

Refer to § 4.1 for details.

NB: to cancel a running *Wait Multiple Cards* function, invoke the *Wait Cancel* function (§ 4.2.3) or one of the *Get Card Information* functions (§ 4.3). Any other function will fail with STA=-20.

4.2.3. Wait Cancel

Invoke this function to cancel a running *Wait One Card* or *Wait Multiple Cards* function.

Command

CMD	LEN	0..1
h60	h02	h0000

Response *OK*

STA	LEN
0	h00

4.3. Get Card Information

NB: these functions were introduced in firmware 1.51 and are not available on earlier versions.

4.3.1. Get Card UID/PUPI

Command

CMD	LEN	0
h61	h01	h01

Response *OK*

STA	LEN	0..1	2..xx-1
0	*xx*	*Card protocol*	*Card UID / PUPI*

- ***Card protocol*** is a 16-bit field (MSB first) where only one bit will be set, telling which kind of card has been found, according to **Table 1: Card lookup – Card protocols bitmap**.
- ***Card UID / PUPI*** gives the protocol-level identifier of the card. Its size depends on the actual protocol (4, 7 or 10 for ISO 14443 type A, 4 for ISO 14443 type B, 8 for ISO 15693, etc). Please refer to the standards for details.

NB: this is exactly the same Response as in 4.1.1, 4.2.1.

NB: as a side effect, this function also cancels a running *Wait Card* or *Wait Multiple Cards* command.

4.3.2. Get Card Protocol Bytes

Command

CMD	LEN	0
h61	h01	h02

Response *when card protocol = ISO 14443 type A*

STA	LEN	0..1	2
0	h03	*ATQ*	*SAK*

Response *when card protocol = ISO 14443 type B*

STA	LEN	0..10
0	h0B	*ATQ*

Response *when card protocol = Innovatron*

STA	LEN	0..xx-1
0	*xx*	*REPGEN*

Response *other card protocols*

STA	LEN
0	$_h$00

NB: as a side effect, this function also cancels a running *Wait Card* or *Wait Multiple Cards* command.

5. Low Power Card Detection

Compatibiliy matrix	
K531 Group	✗
K632 Group	✗
K663 Group	✓

The Low Power Card Detection system (LPCD) is an innovative features added in **K663** and **Prox'N'Drive** to dramatically decrease the overall power consumption while the coupler is waiting for a card.

In the a classical approach, the coupler uses a polling loop to discover cards, i.e. it sends periodically lookup frames to see whether a card answers, or not. On the other hand, a coupler that supports LPCD is able to go into deep sleep mode, and to wake up only if “there is a chance” to have a card in the nearby. The magic behind this wake-up is the ability for the coupler to monitor the impedence of its antenna; when a contactless card approaches the coupler, the impedence is slightly modified, hopefully enough for the coupler to notice.

The LPCD feature relies on electromagnetic coupling between the coupler's antenna and the card's antenna. It works well when both antennas are the same size, and at short distance.
No assumption could be made regarding the actual distance a particular card will be “seen” at. When the card's antenna is much smaller (or much bigger) than the coupler's antenna, it may be even impossible to wake up the coupler.

5.1. Wait One Card with LPCD

This function instructs the coupler to enter LPCD mode. When the coupler wakes up (probably because there's a card In the nearby), it performs the polling sequence, and exits if it finds a card. Otherwise, the coupler goes back into LPCD mode until next wakeup (or the timeout is reached).

Command

CMD	LEN	0..1	2..3	4
$_h$60	$_h$07	*Protocols*	*Timeout (s)*	*Options and delay*

- ***Protocols*** is a 16-bit field (MSB first). Set the bit(s) corresponding to the protocol(s) to be looked for, according to **Table 1: Card lookup – Card protocols bitmap**.
- ***Timeout*** (seconds): the coupler stops waiting when this timeout is expired. Set to $_h$FFFF for an endless waiting.

- ***Options and delay***: define the initial lookup delay and activate the LPCD. See **Table 3: Wait One Card with LPCD – Options and delay** below.

Table 3: Wait One Card with LPCD – Options and delay

BIT	Role	Values	
7-4	Initial lookup delay	$_b$0000	*No delay (the coupler enters LPCD immediately)*
		$_b$0001	*50ms delay*
		$_b$0010	*100ms delay*
		$_b$0011	*250ms delay*
		$_b$0100	*500ms delay*
		$_b$0101	*1s delay*
		$_b$0110	*1.5s delay*
		$_b$0111	*2s delay*
		$_b$1000	*2.5s delay*
		$_b$1001	*3s delay*
		$_b$1010	*4s delay*
		$_b$1011	*5s delay*
		$_b$1100	*10s delay*
		$_b$1101	*30s delay*
		$_b$1110	*60s delay*
		$_b$1111	*RFU, do not use*
3	RFU		*RFU, must be $_b$0*
2	RFU		*RFU, must be $_b$0*
1	LPCD + classical	$_b$0 $_b$1	*Use LPCD only* *Perform a classical lookup every 2.5s*
0	Enable LPCD	$_b$1	*Must be 1 to enable the LPCD mode*

When the "LPCD + classical" bit is set to 1, the coupler wakes up every 2500ms (approx.) to perform a polling sequence. This makes it possible to "see" cards that do not trigger the LPCD wake-up.

***NB:** due to the tolerance on the internal timers of the devices, the actual time observed for timeout, delay, interval may be longer than specified.*

6. Coupler control & configuration

6.1. Device information and control

6.1.1. Get Firmware Information

Command

CMD	LEN
h4F	h00

Response *OK*

STA	LEN	0..3	4	5	6	7..11	12..15
0	h10	*Product ID*	*Ver. MSB*	*Ver. LSB*	*Build no.*	*Chipset info*	*Serial number*

- Product ID: either "CSB4", "K531", "K632", "K663", "SPX2", etc,
- Version is expressed as three bytes: *MSB.LSB [Build No]* ,
- *Chipset info* is the data identifier of the NXP RC531 or RC632 chipset,
- *Serial number* is the serial number of the NXP RC531 or RC632 chipset. This is used as the serial number of the product itself.

6.1.2. Get Reader Capabilities

Command

CMD	LEN
h50	h00

Response *OK*

STA	LEN	0..3
0	h04	*Capabilities*

Capabilities is a 32-bit field (MSB-first). Bits that are set denote the actual features offered by the coupler. Refer to Table 4 on next page.

Table 4: Reader's capabilities

Bit	Mask	Meaning
0	h00000001	*The coupler supports ISO 14443*[9]
1	h00000002	*The coupler supports ISO 15693*[10]
2	h00000004	RFU
3	h00000008	RFU
4	h00000010	RFU
5	h00000020	*The coupler supports ISO 7816 (smartcard interface)*[11]
6	h00000040	Deprecated (CSB5-Bio / SpringProx-RC-Bio)
7	h00000080	RFU
8	h00000100	The coupler implements the ASCII protocol (§)
9	h00000200	The coupler implements the "Fast" protocol (§)
10	h00000400	RFU
11	h00000800	The coupler implements the "Bus" protocol (§)
12	h00001000	The coupler's USER pin drives a RS-485 line buffer
13	h00002000	The coupler supports the Repeat command
14	h00004000	The coupler supports 115200bps baudrate
15	h00008000	Deprecated (early CSB4-S)
16	h00010000	The coupler has an USB interface / CDC-ACM profile
17	h00020000	The coupler has an USB interface / CCID profile (PC/SC)
18	h00040000	The coupler has an USB interface / HID profile (keyboard)
19	h00080000	RFU
20	h00100000	The coupler has an EEPROM to store its configuration
21	h00200000	RFU
22	h00400000	RFU
23	h00800000	RFU
24	h01000000	*The coupler features a "console" (text command processor)*
25	h02000000	Deprecated (early CSB4-S)
26	h04000000	RFU
27	h08000000	RFU
28	h10000000	RFU
29	h20000000	RFU
30	h40000000	RFU
31	h80000000	RFU

[9] This bit is always set for a contactless coupler!

[10] This bit is set for K632 and K663 groups

[11] This bit is set for CSB5, SpringProx-RC, SpringWAP, CSB6, CrazyWriter, EasyFinger

6.1.3. Reset Reader

Command

CMD	LEN	0	1	2	3
$_h$9F	$_h$04	$_h$DE	$_h$AD	$_h$DE	$_h$AD

NB: the reader resets immediately, so it never answers this Command. Upon start-up, the reader sends its *Product ID* at 38400bps and waits for the first Command. ISO 14443-A mode is selected.

6.1.4. Change Baudrate

NB: the 115200bps baudrate is not supported by all readers. Before trying to choose this baudrate, use the Get Reader Capabilities command to verify that the connected reader supports it (see 6.1.2 and Table 4).

Command

CMD	LEN	0	1
$_h$58	$_h$02	$_h$0B	*Value*

Value must take one of the following values:

- $_h$09 ($_d$9) : 9600 bps,
- $_h$26 ($_d$38) : 38400 bps,
- $_h$73 ($_d$115) : 115200 bps.

The reader sends its answer (should be $_h$73 – OK – with 0 byte of data), and then configure its UART with the new baudrate. A dummy or invalid byte may be sent during the reset of the UART.

The host shall wait at least 20ms before sending the next command, at the new specified baudrate of couse.

6.2. RF interface configuration and control

6.2.1. Select RF protocol

Command

CMD	LEN	0	1
h58	h02	h0C	*Mode*

Mode must take one of the following values:

- h01 : ISO 14443-A protocol,
- h02 : ISO 14443-B protocol,
- h03 : Innovatron protocol (legacy radio protocol of Calypso cards),
- h04 : ISO 15693 protocol (RC632 and RC663 only),
- h05 : ICODE1 protocol (RC632 only),
- h06 : Felica protocol (RC663 only).

Response *OK*

STA	LEN
0	h00

6.2.2. RF Field ON/OFF

Command

CMD	LEN	0	0
h58	h02	h0A	*Flag*

Flag must take one of the following values:

- h00 : RF field is switched OFF,
- h01 : RF field is switched ON.

Response *OK*

STA	LEN
0	h00

6.2.3. Read RC Register

Command

CMD	LEN	0
hAB	h01	*Addr*

Response *OK*

STA	LEN	0
0	h01	*Value*

6.2.4. Write RC Register

Command

CMD	LEN	0	1
hAB	h02	*Addr*	*Value*

Response *OK*

STA	LEN
0	h00

6.2.5. Reset RF Interface

Command

CMD	LEN
h9F	h00

Response *OK*

STA	LEN
0	h00

NB: ISO 14443-A mode is selected when the RF interface resets.

6.3. Non-volatile configuration (EEPROM)

NB: not all readers have an EEPROM to store their non-volatile configuration. Before invoking the *Read Configuration Register / Write Configuration Register* functions, check that the connected reader actually supports them, using the *Get Reader Capabilities* command (6.1.2 and Table 4).

6.3.1. Read Configuration Register

Command

CMD	LEN	0	1
$_h$58	$_h$02	$_h$0E	*Addr*

Response *OK*

STA	LEN	0..xx-1
0	*xx*	*Value*

6.3.2. Write Configuration Register

Command

CMD	LEN	0	1	2..xx-1
$_h$58	*xx*	$_h$0D	*Addr*	*Value*

Response *OK*

STA	LEN
0	$_h$00

6.4. LEDs, buzzer and I/Os functions

6.4.1. Set LEDs

Command *reader with 2 LEDs*

CMD	LEN	0	1	2
$_h$58	$_h$03	$_h$1E	*Red*	*Green*

Command *reader with 3 LEDs*

CMD	LEN	0	1	2	3
$_h$58	$_h$04	$_h$1E	*Red*	*Green*	*Yellow / blue*

Every LED control byte (*Red*, *Green* or *Yellow/blue*) could take one of the following values:

- $_h$00 : switched OFF,
- $_h$01 : switched ON,
- $_h$02 : slow blinking,
- $_h$04 : fast blinking,
- $_h$05 : “heart beat”,
- $_h$06 : slow blinking, inverted,
- $_h$07 : fast blinking, inverted,
- $_h$08 : “heart beat”, inverted,
- $_h$09 : switched ON, half intensity.

Response *OK*

STA	LEN
0	$_h$00

6.4.2. Set Buzzer

Command

CMD	LEN	0	1..2
$_h$58	$_h$03	$_h$1C	*Duration (ms)*

Response *OK*

STA	LEN
0	$_h$00

6.4.3. Set USER

This function configures the USER pin as output and defines its state.

NB: the USER pin is not available on all hardware. Before invoking this function, check in device documentation whether it is available or not.

Command

CMD	LEN	0	1
$_h$58	$_h$02	$_h$1F	*Flag*

Flag could take one of the following values:

- $_h$00 : set USER pin at LOW level,
- $_h$01 : set USER pin at HIGH level.

Response *OK*

STA	LEN
0	$_h$00

6.4.4. Get USER

This function configures the USER pin as input and returns its current state.

NB: the USER pin is not available on all hardware. Before invoking this function, check in device documentation whether it is available or not.

Command

CMD	LEN	0
$_h$58	$_h$01	$_h$1F

Response *OK*

STA	LEN	0
0	$_h$01	*Flag*

Flag takes one of the following values:

- $_h$00 : USER pin is at LOW level,
- $_h$01 : USER pin is at HIGH level.

6.4.5. Get MODE

This function configures the MODE pin as input and returns its current state.

NB: the MODE pin is not available on all hardware. Before invoking this function, check in device documentation whether it is available or not.

Command

CMD	LEN	0
$_h$58	$_h$01	$_h$1D

Response *OK*

STA	LEN	0
0	$_h$01	*Flag*

Flag takes one of the following values:

- $_h$00 : MODE pin is at LOW level,
- $_h$01 : MODE pin is at HIGH level.

7. ISO 14443-A and Mifare

Compatibiliy matrix	
K531 Group	✓
K632 Group	✓
K663 Group	✓

The functions listed in this chapter are available when the reader's RF interface has been configured for ISO 14443-A mode (see § 6.2.1).

Note that this is the default mode upon start-up.

7.1. ISO 14443-A card control

Please refer to ISO 14443-3 standard (type A) for details on this chapter.

7.1.1. Activate Idle (REQA / Anticoll / Select)

This function performs the standard activation loop. Only cards in the IDLE state could be activated.

Thanks to the anti-collision scheme, only one card will be selected, even if there are numerous cards in front of the antenna.

Command

CMD	LEN
$_h$4D	$_h$00

Response *no card in the field*

STA	LEN
-1	$_h$00

Response *card found, 4-byte UID*

STA	LEN	0..3	4..6	5
0	$_h$06	*UID*	*ATQ*	*SAK*

Response *card found, 7-byte UID*

STA	LEN	0..6	7..8	9
0	$_{h}$0A	*UID*	*ATQ*	*SAK*

Response *card found, 10-byte UID*

STA	LEN	0..9	10..11	12
0	$_{h}$0D	*UID*	*ATQ*	*SAK*

7.1.2. Activate Any (WUPA /Anticoll / Select)

This function performs the standard ISO 14443-A level 3 activation loop. Cards either in the IDLE or the HALTED state could be activated.

Thanks to the anti-collision scheme, only one card will be selected, even if there are numerous cards in front of the antenna.

Command

CMD	LEN
$_{h}$40	$_{h}$00

Responses: same as 7.1.1

7.1.3. Activate Again (WUPA / Select)

This function tries to wake-up and select again a card whose UID is already known.

Command *activate last card again*

CMD	LEN
$_{h}$44	$_{h}$00

Command *activate a specific card again*

CMD	LEN	0..xx-1
$_{h}$44	*xx*	*UID*

xx (size of UID) could be either $_{h}$04, $_{h}$07 or $_{h}$0A.

Response *OK*

STA	LEN
0	$_{h}$00

Response *no answer*

STA	LEN
-1	$_H$00

7.1.4. Halt

This command puts the currently activated card into the HALT state.

Command

CMD	LEN
$_h$45	$_h$00

Response *OK*

STA	LEN
0	$_h$00

7.2. ISO 14443-A frame exchange

7.2.1. Standard Frame Exchange

Command

CMD	LEN	0..1	2..3	4	5..6	6..xx-1
$_h$94	$_h$xx	*Send Len*	*Recv Len*	$_h$01	*Timeout*	*Card's command*

Response *OK*

STA	LEN	0..1	2..xx-1
0	$_h$xx	*Recv Len*	*Card's Response*

7.2.2. Advanced Frame Exchange

To be written

7.3. Mifare Classic operation, keys stored in the reader

The functions listed here are related to NXP Mifare Classic cards (1K, 4K, Mini, Mifare Plus running in Level 1).

A good understanding of Mifare Classic memory mapping and authentication scheme is required to work with those cards. Please refer to relevant NXP's documentations for details on function parameters and usages.

7.3.1. Load Key in Secure EEPROM

The reader is able to store permanently 16 'A' keys and 16 'B' keys in a secure EEPROM (inside the NXP RC531 or RC632 chipset) to get automatically authenticated onto Mifare cards.

Command

CMD	LEN	0	1	2..7
$_h$A8	$_h$08	*Key type*	*Key index*	*Key value*

Key type could take one of the following values:

- $_h$00 : type 'A' key,
- $_h$01 : type 'B' key.

Key index could take one of the following values:

- $_h$00 : 1st key,
- $_h$01 : 2nd key,
- …
- $_h$0F : 16th key.

(A total of 32 keys -16 'A' and 16 'B'- could be stored).

Response *OK*

STA	LEN
0	$_h$00

7.3.2. Load Key in RAM

The reader is able to store 4 'A' keys and 4 'B' keys in its RAM to get automatically authenticated onto Mifare cards. Those keys are lost upon reset.

Command

CMD	LEN	0	1..7	8..13
$_h$4C	$_h$0E	*Key ID*	*(dummy)*	*Key value*

Key ID could take one of the following values:

- $_h$00 : 1st key A in RAM,
- $_h$01 : 2nd key A in RAM,
- $_h$02 : 3rd key A in RAM,
- $_h$03 : 4th key A in RAM,
- $_h$04 : 1st key B in RAM,
- $_h$05 : 2nd key B in RAM,
- $_h$06 : 3rd key B in RAM,
- $_h$07 : 4th key A in RAM.

Response *OK*

STA	LEN
$_h$00	$_h$00

7.3.3. Read Block

This function reads one block. All Mifare keys known by the readers (RAM and EEPROM) are tried in sequence until an authentication succeeds.

NB: depending on the location of the correct key in the list, this could take a long time (up to 1s per sector). Whenever this is possible, let the host specify the key (see 7.4.1).

Command

CMD	LEN	0
$_h$49	$_h$01	*Block*

Response *OK*

STA	LEN	0..15
0	h10	*Block data*

7.3.4. Write Block

This function writes one block. All Mifare keys known by the readers (RAM and EEPROM) are tried in sequence until an authentication succeeds.

NB: depending on the location of the correct key in the list, this could take a long time (up to 1s per sector). Whenever this is possible, let the host specify the key (see 7.4.2).

This function makes it possible to write sector's trailer (special blocks holding the access conditions and the authentication keys of the sector).

Always format block's content according to Mifare documentation when working with such blocks. Be aware that writing invalid data in a sector's trailer is likely to make the sector permanently unusable.

Command

CMD	LEN	0	1..16
h4B	h11	*Block*	*Block data*

Response *OK*

STA	LEN
0	h00

7.3.5. Read Sector

This function reads all the blocks of the specified sector (but sector's trailer). All Mifare keys known by the readers (RAM and EEPROM) are tried in sequence until an authentication succeeded.

NB: depending on the place of the correct key in the list, this could take a long time (up to 1s per sector). Whenever this is possible, let the host specify the key (see 7.4.3).

Command

CMD	LEN	0
h48	h01	*Sector*

Response *OK, 3-block sector*

STA	LEN	0..47
0	h30	*Sector data*

Response *OK, 15-block sector*

STA	LEN	0..239
0	hF0	*Sector data*

7.3.6. Write Sector

This function writes one sector. All Mifare keys known by the readers (RAM and EEPROM) are tried in sequence until an authentication succeeds.

NB: depending on the location of the correct key in the list, this could take a long time (up to 1s per sector). Whenever this is possible, let the host specify the key (see 7.4.2).

Command *3-block sector*

CMD	LEN	0	1..48
h4A	h31	*Sector*	*Sector data*

Command *15-block sector*

CMD	LEN	0	1..240
h4A	hF1	*Sector*	*Sector data*

Response *OK*

STA	LEN
0	h00

7.3.7. Get Authentication Information

Command

CMD	LEN	0
h58	h01	h33

Response *OK*

STA	LEN	0
0	h01	*Info*

7.4. Mifare Classic operation, keys provided by the host

The functions listed here are related to NXP Mifare Classic cards (1K, 4K, Mini, Mifare Plus running in Level 1).

A good understanding of Mifare Classic memory mapping and authentication scheme is required to work with those cards. Please refer to relevant NXP's documentations for details on function parameters and usages.

7.4.1. Read Block

This function reads one block, using the key provided by the host.

NB: the supplied key is first tried as a type 'A' key, and as a type 'B' key only upon authentication error. Therefore, reading a block using its 'B' key takes longer than reading the same block using its 'A' key.

Command

CMD	LEN	0	1..6
h49	h07	*Block*	*Key value*

Response *OK*

STA	LEN	0..15
0	h10	*Block data*

7.4.2. Write Block

This function writes one block, using the key provided by the host.

NB: the supplied key is first tried as a type 'B' key, and as a type 'A' key only upon authentication error. Therefore, writing a block using its 'A' key takes longer than writing the same block using its 'B' key.

> This function makes it possible to write sector's trailer (special blocks holding the access conditions and the authentication keys of the sector).
>
> Always format block's content according to Mifare documentation when working with such blocks. Be aware that writing invalid data in a sector's trailer is likely to make the sector permanently unusable.

Command

CMD	LEN	0	1..16	17..22
$_h$4B	$_h$17	*Block*	*Block data*	*Key value*

Response *OK*

STA	LEN
0	$_h$00

7.4.3. Read Sector

This function reads one sector, using the key provided by the host.

NB: the supplied key is first tried as a type 'A' key, and as a type 'B' key only upon authentication error. Therefore, reading a sector using its 'B' key takes longer than reading the same sector using its 'A' key.

Command

CMD	LEN	0	1..6
$_h$48	$_h$07	*Sector*	*Key value*

Response *OK, 3-block sector*

STA	LEN	0..47
0	$_h$30	*Sector data*

Response *OK, 15-block sector*

STA	LEN	0..239
0	hF0	*Sector data*

7.4.4. Write Sector

This function writes one sector, using the key provided by the host.

NB: the supplied key is first tried as a type ‘B’ key, and as a type ‘A’ key only upon authentication error. Therefore, writing a block using its ‘A’ key takes longer than writing the same block using its ‘B’ key.

Command *3-block sector*

CMD	LEN	0	1..48	49..54
h4A	h37	*Sector*	*Sector data*	*Key value*

Command *15-block sector*

CMD	LEN	0	1..240	241..246
h4A	hF7	*Sector*	*Sector data*	*Key value*

Response *OK*

STA	LEN
0	h00

7.5. Mifare UltraLight operation

The functions listed here are related to NXP Mifare UltraLight cards (and UltraLight C).

Please refer to relevant NXP’s documentations for details on function parameters and usages.

7.5.1. Read 4 Pages

Command

CMD	LEN	0
h46	h01	*1st Page*

Response *OK*

STA	LEN	0..15
0	$_h$10	*Data*

7.5.2. Write Page

Command

CMD	LEN	0	1..4
$_h$47	$_h$05	*Page*	*Data*

Response *OK*

STA	LEN
0	$_h$00

8. ISO 14443-B

Compatibiliy matrix	
K531 Group	✓
K632 Group	✓
K663 Group	✓

The functions listed in this chapter are available when the reader's RF interface has been configured for ISO 14443-B mode (see § 6.2.1).

8.1. ISO 14443-B card control

Please refer to ISO 14443-3 standard (type B) for details on this chapter.

8.1.1. Activate Idle (REQB)

This function implements REQB. Only cards in the IDLE state could be activated. There's no anti-collision in this mode.

Command

CMD	LEN
$_h$4D	$_h$00

Response *no card in the field*

STA	LEN
-1	$_h$00

Response *card found*

STA	LEN	0..10
0	$_h$0B	*ATQ*

NB: the four first bytes of the ATQ are the card's PUPI.

8.1.2. Activate Any (WUPB)

This function implements WUPB. Cards either in the IDLE or the HALTED state could be activated. There's no anti-collision in this mode.

Command

CMD	LEN
$_h$4D	$_h$00

Responses: same as 8.1.1

8.1.3. Activate Again (WUPB / Select)

This function tries to wake-up and select again a card whose UID is already known.

Command *activate last card again*

CMD	LEN
$_h$44	$_h$00

Response *OK*

STA	LEN
0	$_h$00

Response *no answer*

STA	LEN
-1	$_h$00

8.1.4. Halt

This command puts the currently activated card into the HALT state.

Command

CMD	LEN
$_h$45	$_h$00

Response *OK*

STA	LEN
0	h00

8.2. ISO 14443-B FRAME EXCHANGE

8.2.1. Standard Frame Exchange

Command

CMD	LEN	0..1	2..3	4	5..6	6..xx-1
h97	hxx	*Send Len*	*Recv Len*	h01	*Timeout*	*Card's command*

Response *OK*

STA	LEN	0..1	2..xx-1
0	hxx	*Recv Len*	*Card's Response*

8.2.2. Advanced Frame Exchange

To be written

9. “T=CL” (ISO 14443-4)

Compatibiliy matrix	
K531 Group	✓
K632 Group	✓
K663 Group	✓

ISO 14443 level 4 (a.k.a. “T=CL”) is a high-level protocol to exchange application-level buffers (APDU) between the host and the card.

As the reader fully implements this protocol in its firmware, the complexity of the protocol is totally masked to the host.

ISO 14443 level 4 allows a single reader to communicate with more than one card at once (alternatively), thanks to an address field called CID (Card IDentifier). The reader is able to manage up to 14 active CIDs at the same time.

Please refer to ISO 14443-4 standard for details on this chapter.

9.1. T=CL activation – ISO 14443-A

9.1.1. R-ATS

This function asks the currently selected ISO 14443-A card to enter T=CL level. The card answers with its ATS (Answer To Select).

Command

CMD	LEN	0	1
$_h$81	$_h$02	$_h$10	*CID*

Value for CID could be either:

- $_h$FF : CID not used (only one card active at a time),
- $_h$00 to $_h$0E : the card takes the specified CID.

Response *OK*

STA	LEN	0..xx-1
0	*xx*	*ATS*

9.1.2. PPS

This function asks the card to change its communication parameters.

Command

CMD	LEN	0	1	2	3
$_h$81	$_h$04	$_h$11	*CID*	*DSI*	*DRI*

CID shall be the same as in *R-ATS*

DSI is the card-to-reader baudrate

DRI is the reader-to-card baudrate

Allowed values for both DSI and DRI are:

- $_h$00 : 106kbps (default),
- $_h$01 : 212kbps,
- $_h$02 : 424kbps,
- $_h$03 : 847kbps.

NB: the host application must ensure that the specified parameters are actually supported by both the reader and the card.

Response *OK*

STA	LEN
0	$_h$00

9.2. T=CL activation – ISO 14443-B

9.2.1. ATTRIB – Stay at 106kpbs

This function asks the currently selected ISO 14443-B card to enter T=CL level. The card answers with an ATTRIB RESPONSE (may be empty).

Command

CMD	LEN	0	1
$_h$81	$_h$02	$_h$10	*CID*

Value for CID could be either:

- $_h$FF : CID not used (only one card active at a time),
- $_h$00 to $_h$0E : the card takes the specified CID.

Response *OK*

STA	LEN	0..xx-1
0	*xx*	*R-ATTRIB*

9.2.2. ATTRIB + Set Baudrate

This function asks the currently selected ISO 14443-B card to enter T=CL level and changes its communication parameters. The card answers with an ATTRIB RESPONSE (may be empty).

Command

CMD	LEN	0	1	2	3
$_h$81	$_h$04	$_h$10	*CID*	*DSI*	*DRI*

Value for CID could be either:

- $_h$FF : CID not used (only one card active at a time),
- $_h$00 to $_h$0E : the card takes the specified CID.

DSI is the card-to-reader baudrate

DRI is the reader-to-card baudrate

Allowed values for both DSI and DRI are:

- $_h$00 : 106kbps (default),
- $_h$01 : 212kbps,
- $_h$02 : 424kbps,
- $_h$03 : 847kbps.

NB: the host application must ensure that the specified parameters are actually supported by both the reader and the card.

Response *OK*

STA	LEN	0..xx-1
0	*xx*	*R-ATTRIB*

9.3. T=CL APDU exchange

Command

CMD	LEN	0	1..xx-1
$_h$82	*xx*	*CID*	*Command to the card (C-APDU)*

CID shall be the same as in *R-ATS* or *ATTRIB*

Response *OK*

STA	LEN	0..xx-1
0	*xx*	*Response from the card (R-APDU)*

9.4. T=CL DESELECT

This functions asks T=CL card to enter the HALT state.

Command

CMD	LEN	0	1
$_h$81	$_h$02	$_h$12	*CID*

CID shall be the same as in *R-ATS* or *ATTRIB*

Response *OK*

STA	LEN
0	$_h$00

10. ISO 15693

NB: not all readers support ISO 15693 (and ICODE1). Before invoking one of the functions listed here, check that the connected reader actually supports them, using the *Get Reader Capabilities* command (6.1.2 and Table 4).

Compatibiliy matrix	
K531 Group	✗
K632 Group	✓
K663 Group	✓

The functions listed in this chapter are available when the reader's RF interface has been configured for ISO 15693 mode (see § 6.2.1).

10.1. ISO 15693 card control

Please refer to ISO 15693-2 and -3 standard for details on this chapter.

10.1.1. Select Any

This function tries to select a compliant card.

Command

CMD	LEN	00
$_h$40	$_h$01	*AFI*

Response *OK*

CMD	LEN	0..7
$_h$00	$_h$08	*UID*

NB: for every standard ISO 15693 card, UID[0] = $_h$E0

10.1.2. Select Again

This function tries to select again a card whose UID is already known.

Command *select last card again*

CMD	LEN
$_h$44	$_h$00

Command *select a specific card again*

CMD	LEN	0..7
$_h$44	$_h$08	*UID*

Response *OK*

STA	LEN
0	$_h$00

Response *no answer*

STA	LEN
-1	$_h$00

10.1.3. Halt

This command puts the currently selected card into the HALT state.

Command

CMD	LEN
$_h$45	$_h$00

Response *OK*

STA	LEN
0	$_h$00

10.1.4. Get System Information

This function is **optional** according to the ISO 15693-3 standard. It may be available or not, depending on the card.

Command

CMD	LEN
$_h$39	$_h$00

Response *OK*

STA	LEN	1	1..8	9..xx-1
0	$_h$xx	*Flags*	*UID*	*More Data*

The fields available in *More Data* are indicated by the *Flags*

- if (*Flags* & $_h$01) ➔ DSFID field present (1 byte),
- if (*Flags* & $_h$02) ➔ AFI field present (1 byte),
- if (*Flags* & $_h$04) ➔ Memory Size field present (2 bytes),
- if (*Flags* & $_h$08) ➔ IC reference field present (1 byte).

10.2. ISO 15693 FRAME EXCHANGE

10.2.1. Standard Frame Exchange

Command

CMD	LEN	0..1	2..3	4	5..6	6..xx-3	xx-2	xx-1
$_h$98	$_h$xx	*Send Len*	*Recv Len*	$_h$01	*Timeout*	*Card's command (without CRC)*	$_h$*CC*	$_h$*CC*

Response *OK*

STA	LEN	0..1	2..xx-1
0	$_h$xx	*Recv Len*	*Card's Response + $_h$CCCC*

10.2.2. Advanced Frame Exchange

To be written

10.3. ISO 15693 Read

The functions listed here are **optional** according to the ISO 15693-3 standard. They may be available or not, depending on the card.

Please refer to the documentation of the card for details.

10.3.1. Read Block

Command

CMD	LEN	0
$_h$46	$_h$01	*Addr.*

Response *OK*

STA	LEN	0..xx-1
0	$_h$xx	Data

10.3.2. Read Multiple Blocks

Command

CMD	LEN	0	1	2
$_h$46	$_h$03	Addr.	$_h$00	Count

Response *OK*

STA	LEN	0..x-1
0	$_h$xx	*Data*

10.3.3. Read Bytes

Command

CMD	LEN	0	1	2
$_h$46	$_h$03	*Addr.*	$_h$01	*Count*

Response *OK*

STA	LEN	0..x-1
0	hxx	*Data*

Note: in this case LEN = *Count*

10.4. ISO 15693 Write and Lock

The functions listed here are **optional** according to the ISO 15693-3 standard. They may be available or not, depending on the card.

Please refer to the documentation of the card for details.

10.4.1. Write Block

Command

CMD	LEN	0	1..xx-1
h47	hxx	*Addr.*	*Data*

Response *OK*

STA	LEN
0	h00

10.4.2. Lock Block

Command

CMD	LEN	0
h59	h01	*Addr.*

Response *OK*

STA	LEN
0	h00

11. Other RF protocols

11.1. Innovatron Radio Protocol (Calypso)

Compatibiliy matrix	
K531 Group	✓
K632 Group	✓
K663 Group	✓

The functions listed in this chapter are available when the reader's RF interface has been configured for Innovatron mode (see § 6.2.1).

11.1.1. APGEN

Command

CMD	LEN
$_h$40	$_h$00

Response *no card in the field*

STA	LEN
-1	$_h$00

Response *card found, 4-byte UID*

STA	LEN	0..xx-1
0	*xx*	*Response from the card (REPGEN)*

11.1.2. ATTRIB

Command

CMD	LEN	0
$_h$81	$_h$01	$_h$10

Response *OK*

STA	LEN
0	h00

11.1.3. COM_R (APDU exchange)

Command

CMD	LEN	0	1..xx-1
h82	*xx*	hFF	*Command to the card (C-APDU)*

Response *OK*

STA	LEN	0..xx-1
0	*xx*	*Response from the card (R-APDU)*

11.1.4. DISC

Command

CMD	LEN	0
h81	h01	h12

Response *OK*

STA	LEN
0	h00

11.2. NXP ICODE1

Compatibiliy matrix	
K531 Group	✗
K632 Group	✓
K663 Group	✗

The functions listed in this chapter are available when the reader's RF interface has been configured for NXP ICODE1 mode (see § 6.2.1).

11.2.1. Select Any

Command

CMD	LEN	00
$_h$40	$_h$01	*AFI*

Response *OK*

CMD	LEN	0..7
$_h$00	$_h$08	*UID*

11.2.2. Halt

Command

CMD	LEN
$_h$45	$_h$00

Response *OK*

STA	LEN
0	$_h$00

11.2.3. Read Block

Command

CMD	LEN	0
$_h$46	$_h$01	*Addr.*

Response *OK*

STA	LEN	0..3
0	$_h$04	*Data*

11.2.4. Read Multiple Blocks

Command

CMD	LEN	0	1	2
$_h$46	$_h$03	*Addr.*	$_h$00	*Count*

Response *OK*

STA	LEN	0..(4*count)-1
0	4*count	*Data*

11.2.5. Write Block

Command

CMD	LEN	0	1..5
$_h$47	$_h$05	*Addr.*	*Data*

Response *OK*

STA	LEN
0	$_h$00

12. Status and error codes

The *symbolic names* are the one used in SpringProx API.

12.1. Success and special status

STA	Symbolic name	Meaning
0	MI_OK	Success
-20	MI_POLLING	Polling mode pending
-30	MI_QUIT	Polling terminated (timeout or break[12])
-128	MI_TIME_EXTENSION	Reader is still processing the Command

12.2. Errors in RF communication or protocol

STA	Symbolic name	Meaning
-1	MI_NOTAGERR	No answer (no card / card is mute)
-2	MI_CRCERR	Invalid CRC in card's response
-3	MI_EMPTY	No frame received (NFC mode)
-5	MI_PARITYERR	Invalid parity bit(s) in card's response
-7	MI_CASCLEVEX	Too many anti-collision loops
-8	MI_SERNRERR	Wrong LRC in card's serial number
-11	MI_BITCOUNTERR	Wrong number of bits in card's answer
-12	MI_BYTECOUNTERR	Wrong number of bytes in card's answer
-21	MI_FRAMINGERR	Invalid framing in card's response
-24	MI_COLLERR	A collision has occurred
-28	MI_NOBITWISEANTICOLL	More than one card found, but at least one does not support anti-collision
-29	MI_EXTERNAL_FIELD	An external RF field has been detected
-31	MI_CODING_ERR	Bogus status in card’s response

[12] Polling is terminated by a serial BREAK (more than 12 ETU at level 0), when receiving 15 empty bytes ($_h$00) or by the ASCII {ESCAPE} character ($_h$1B)

12.3. Errors reported by the card

STA	Symbolic name	Meaning
-4	MI_AUTHERR	Authentication failed or access denied
-6	MI_CODEERR	NACK or status indicating error
-9	MI_LOCKED	Card or block locked
-10	MI_NOTAUTHERR	Authentication must be performed first
-13	MI_VALUERR	Counter is invalid
-14	MI_TRANSERR	Transaction error
-15	MI_WRITEERR	Write failed
-16	MI_INCRERR	Counter increase failed
-17	MI_DECRERR	Counter decrease failed
-18	MI_READERR	Read failed
-22	MI_ACCESSERR	Access error (bad address or denied)
-32	MI_CUSTERR	Vendor specific error
-33	MI_CMDSUPERR	Command not supported
-34	MI_CMDFMTERR	Format of command invalid
-35	MI_CMDOPTERR	Option(s) of command invalid
-36	MI_OTHERERR	Other card error

12.4. Errors at T=CL level

STA	Symbolic name	Meaning
-71	MI_CID_NOT_ACTIVE	No active card with this CID
-75	MI_BAD_ATS_LENGTH	Length error in card’s ATS
-76	MI_ATTRIB_ERROR	Error in card’s response to ATTRIB
-77	MI_BAD_ATS_FORMAT	Format error in card’s ATS
-78	MI_TCL_PROTOCOL	Protocol error in card’s response
-87	MI_BAD_PPS_FORMAT	Format error in card’s PPS response
-88	MI_PPS_ERROR	Other error in card’s PPS response
-93	MI_CID_ALREADY_ACTIVE	A card is already active with this CID

12.5. Other errors

STA	Symbolic name	Meaning
-19	MI_OVFLERR	RC FIFO overflow
-23	MI_UNKNOWN_COMMAND	Unknown RC command
-25	MI_COMMAND_FAILED	Command execution failed
-26	MI_INTERFACEERR	Hardware error
-27	MI_ACCESSTIMEOUT	RC timeout
-59	MI_WRONG_MODE	Command not available in this mode
-60	MI_WRONG_PARAMETER	Wrong parameter for the command
-100	MI_UNKNOWN_FUNCTION	Command not supported by the reader
-112	MI_BUFFER_OVERFLOW	Internal buffer overflow
-125	MI_WRONG_LENGTH	Wrong data length for the command

Editor's information

PRO ACTIVE 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