PMD15282-CA
DRAFT - PUBLIC

SpringCard PC/SC Couplers

Zero-driver - CCID low-level implementation

Document Identification

Category	Specification		
Family/Customer	CCID PC/SC Couplers		
Reference	PMD15282	Version	CA
Status	Draft	Classification	Public
Keywords	E663, K663, Ethernet, Serial, BLE, Smart Card, Integrated Circuit(s) Cards, PC/SC, CCID, RS-232, RS-TTL, BLE, GATT, CCID		
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\PCSC\CCID low-level implementation\[PMD15282-CA] Zero-driver - CCID low-level implementation.odt		
Date saved	17/03/17	Date printed	16/04/15

Revision History

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	29/07/15	JDA				Draft
AB	14/10/15	JDA				Fixed a few errors
BA	13/05/16	JDA				ASCII version added for serial reader
BB	24/02/17	JDA				Title changed from “CCID over Serial and CCID over TCP implementations” to “Zero-driver – CCID low-level implementation” A few typos corrected Preparing BLE implementation
CA	17/03/17	MBA	JDA			BLE implementation

Contents

1. Introduction

1.1. Abstract

SpringCard's product range could be divided into USB devices on the one hand, and non-USB devices (serial, TCP over Ethernet) on the other hand.

Since 2009, **SpringCard**'s USB couplers take benefit of a full PC/SC compliance. It is achieved by the mean of a PC/SC driver (available for Windows and for most UNIX systems, including Linux and MacOS X), and by compliance with the USB CCID (Chip Card Interface Device) specification.

Starting with firmware version 2.xx, **SpringCard**'s non-USB couplers also provide a new communication scheme which is consistent with the CCID specification. This new communication scheme supersedes the legacy "SpringProx" protocol[1].

Designed with simplicity and interoperability standards in mind, this CCID implementation aims to shorten the development and validation times, and paves the way for a new generation of PC/SC driver using another medium and not USB.

This document is the specification of **SpringCard's CCID low-level implementation** over **Serial**, **TCP** or **BLE** links. It provides all the information a developer would need to communicate with a **SpringCard** coupler from his application, without going through a complex driver stack.

1.2. Supported product

At the date of writing, the products covered by this document are

- **SpringCard K663** version 2.02 or newer, and all products based on **K663** core (**K663-232**, **K663-TTL**, **TwistyWriter-232**, **TwistyWriter-TTL**, **CSB4.6S**, **CSB4.6U**) for the CCID over Serial implementation, and the derived products feature a Bluetooth Low Energy (BLE) interface for the CCID over BLE implementation,
- **SpringCard E663** version 2.02 or newer, and all products based on **E663** core (**FunkyGate IP PC/SC**, **FunkyGate IP+POE PC/SC**, **TwistyWriter IP PC/SC**) for the CCID over TCP implementation.

1.3. Audience

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

[1] The SpringProx protocol remains fully available; upward compliance in therefore ensure with systems using the legacy SpringProx API to communicate with the couplers.

1.4. Support and updates

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

www.springcard.com

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

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

www.springcard.com/support

1.5. Related documents

1.5.1. CCID standard

Reference	Publisher	Title
[CCID]	USB Workgroup	Universal Serial Bus Device Class: Smart Card Specification for Integrated Circuit(s) Cards Interface Devices Rev 1.1 – 22/04/2005 Download link: http://www.usb.org/developers/docs/devclass_docs/DWG_Smart-Card_CCID_Rev110.pdf

1.5.2. Developer's guides

Reference	Publisher	Title
PMD17041	SpringCard	PC/SC Couplers – Developer's Handbook

2. CCID over Serial – Binary implementation

This chapters concerns the K663 OEM Couplers, and all products based on the K663 core, starting with firmware version 2.02.

2.1. Introduction

The **CCID over Serial binary implementation** is the preferred method for using a SpringCard Coupler. It provides great performance (in term of communication speed) and could be implemented with a low footprint on the host-side.

2.2. Physical layer

The physical layer uses a serial asynchronous protocol. The communication is done over a UART. Default communication parameter are 38400bps, 8 data bits, 1 stop bit, no parity. A different baudrate could be selected in the coupler's non-volatile configuration.

No flow control is involved, which means that the serial communication uses only 2 lines:

- Coupler's RX line, *aka* **PC_To_RDR**,
- Coupler's TX line, *aka* **RDR_To_PC**.

If the 2 communication lines are mixed in the underlying medium (**RS-485**), only the **half-duplex** operation mode is available.

If the 2 communication lines are physically separated (**RS-232**, **RS-TLL**), the **full-duplex** operation mode is also available – and should be preferred.

For more details regarding the operation mode, refer to § 2.5.2.

2.3. Transport layer

2.3.1. Block format

Every block transmitted in the channel is formatted as follow:

START BYTE	ENDPOINT	HEADER	DATA	CHECKSUM
1 byte	1 byte	10 byte	0 to 262 bytes	1 byte

2.3.2. Description of the fields

Field	Description
START BYTE	The START BYTE is the constant value $_h$CD
ENDPOINT	The ENDPOINT byte is used to convey the CCID HEADER and DATA to the appropriate target service (as the USB endpoint feature does).
HEADER	For Bulk-Out and Bulk-In messages, the 10-B HEADER field follows [CCID]. For all the other messages, a proprietary format is defined.
DATA	For Bulk-Out and Bulk-In messages, the DATA field follows [CCID]. For all the other messages, a proprietary format is defined.
CHECKSUM	The CHECKSUM field is a XOR computed over all the bytes in the ENDPOINT, HEADER and DATA fields.

2.3.3. Values for the ENDPOINT field

Value	Symbolic name	Understanding
$_h$00	CONTROL_TO_RDR	Control Endpoint (orders and queries from PC to RDR)
$_h$80	CONTROL_TO_PC	Control Endpoint (answers from RDR to PC)
$_h$81	BULK_TO_PC	Bulk-In Endpoint (RDR_to_PC responses)
$_h$02	BULK_TO_RDR	Bulk-Out Endpoint (PC_to_RDR commands)
$_h$83	INTERRUPT_TO_PC	Interrupt Endpoint (notifications from RDR to PC)

2.3.4. Size of the blocks

The size of every block can't be less than 13 bytes.

The size of every block can't exceed 275 bytes.

2.3.5. Timeout

When the Start Byte is received, the Coupler opens a 500ms time window. The host must transmit an entire block within this time window, otherwise the block is discarded.

2.4. Command layer

Chapter 6, 7, 8 and 9 contain the documentation of the Command Layer.

Chapter 10 and document [PMD15305] explain how to use the Coupler once the protocol is correctly implemented.

2.5. General communication flow

2.5.1. Session establishment

The PC tries to connect to the Coupler over a serial link by sending GET DESCRIPTOR commands (§ 6.3).

Once a Coupler has been found, the PC queries all the Coupler's descriptors, and, when ready, starts the Coupler using the SET CONFIGURATION command (§ 6.4).

No communication could occur on the Bulk-In, Bulk-Out or Interrupt endpoints before the SET CONFIGURATION command has been issued by the PC and acknowledged by the Coupler.

2.5.2. Operation mode: half-duplex or full-duplex?

Depending on the underlying hardware, the serial link could be either half-duplex (RS-485) or full-duplex (RS-232, RS-TTL).

On a half-duplex medium, collisions must be avoided. The protocol is of request/response type: a RDR_To_PC frame could occur only as a response to a PC_To_RDR frame. As a consequence, there's no way for the Coupler to notify the PC when a card is inserted or removed. The PC must therefore “poll” the Coupler, by sending the PC_To_RDR_GetSlotStatus (§ 7.3.3) command as often as possible.

On a full-duplex medium, there's no risk of collision. The request/response protocol is enhanced by so-called interrupt frames (§ 9.3.1) to notify the PC when a card is inserted or removed. This operation mode dramatically lowers down the workload on the PC.

It's the responsibility of the PC to select the proper operation mode when sending SET CONFIGURATION command (§ 6.4).

2.6. Error handling and recovery

2.6.1. For the Coupler

- **Malformed frame, Protocol violation:** in case it receives a block that doesn't obey to the block-formatting rules, the Coupler discard the block and remain silent,
- **Communication timeout:** if the PC takes more than 1000ms to send a block, the Coupler discard the block and remain silent.

2.6.2. For the PC

- **Malformed frame, Protocol violation:** in case it receives a block that doesn't obey to the block-formatting rules, the PC shall wait 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 2.5.1),
- **Communication timeout:** if the Coupler takes more than 1000ms to send a block (time between the Start Byte and the CRC), the PC shall wait at least 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 2.5.1),
- **Processing timeout:** the Coupler starts its answer (Start Byte) within 500ms for the commands sent to the Control Endpoint (CONTROL_TO_RDR), and within 1500ms for the commands sent to the Bulk-Out Endpoint (BULK_TO_RDR). If the Coupler doesn't answer within the specified time, the PC shall wait at least 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 2.5.1).

2.6.3. Quick recovery

Instead of waiting 2000ms before running the Session establishment procedure again, the PC may

1. Reset the Coupler by setting the Coupler's /RESET pin to LOW level during at least 2ms,
2. Wait for the Coupler's startup string on the serial line (constant “K663”)[2],
3. Wait 50ms after the end of the startup string,
4. Run the Session establishment procedure.

[2] The time between the reset and the arrival of the “K663” startup string depends on how the bootloader is configured in the device. The host software shall be prepared to wait up to 1500ms to accommodate every configuration of the bootloader, but with the standard configuration the K663 starts in less than 75ms.

3. CCID over Serial – ASCII implementation

This chapters concerns the K663 OEM Couplers, and all products based on the K663 core, starting with firmware version 2.06.

3.1. Introduction

The **CCID over Serial binary implementation** is an alternative method for interfacing a SpringCard CCID Serial Coupler. It is less efficient than the binary implementation, but is easier to implement "manually" or through scripting.

3.2. Physical layer

The physical layer uses a serial asynchronous protocol. The communication is done over a UART. Default communication parameter are 38400bps, 8 data bits, 1 stop bit, no parity. A different baudrate could be selected in the coupler's non-volatile configuration.

No flow control is involved, which means that the serial communication uses only 2 lines:

- Coupler's RX line, *aka* **PC_To_RDR**,
- Coupler's TX line, *aka* **RDR_To_PC**.

If the 2 communication lines are mixed in the underlying medium (**RS-485**), only the **half-duplex** operation mode is available.

If the 2 communication lines are physically separated (**RS-232**, **RS-TLL**), the **full-duplex** operation mode is also available – and should be preferred.

For more details regarding the operation mode, refer to § 3.5.2.

3.3. Transport layer

3.3.1. Byte representation

Every byte is transmitted as two hexadecimal characters.

The Coupler transmits in upper case (digits are '0', '1', … 'A', … 'F').

The PC may transmit either in upper case ('0' … 'A' … 'F') or in lower case ('0' … 'a' … 'f').

3.3.2. Block format

To make the protocol less verbose (and as a consequence more easy to read or write for a human), unnecessary fields are suppressed:

- The ENDPOINT is not transmitted – it could be easily inferred from the Message Type
- There's no Length field in the HEADER – the frame is terminated by an explicit END MARK
- The HEADER itself is truncated to keep only the minimal amount of bytes that are required to understand the Request.

As a consequence, there's not a single block format but 3 block formats, corresponding to a group of Requests, and tailored for this very group.

a. Block format – CONTROL

This block format is applicable for both *CONTROL_TO_RDR* and *CONTROL_TO_PC* ENDPOINTs, i.e. to messages

- *GET STATUS*
- *GET DESCRIPTOR*
- *SET CONFIGURATION*

START MARK	Message Type	Control Header	Data	END MARK
1 char	1 byte (2 hex chars)	6 bytes (12 hex chars)	0 to 262 bytes (2 to 524 hex chars)	1 or 2 chars

The Control Header contains only the fields Value_L, Value_H, Index_L, Index_H and Option of the actual HEADER.

b. Block format – BULK

This block format is applicable for both *BULK_TO_RDR* and *BULK_TO_PC* ENDPOINTs, i.e. to messages

- *PC_To_RDR_IccPowerOn*
- *PC_To_RDR_IccPowerOff*
- *PC_To_RDR_GetSlotStatus*
- *PC_To_RDR_Escape*
- *PC_To_RDR_XfrBlock*
- *RDR_To_PC_DataBlock*
- *RDR_To_PC_SlotStatus*
- *RDR_To_PC_Escape*

START MARK	Message Type	Bulk Header	Data	END MARK
1 char	1 byte (2 hex chars)	1 byte (2 hex chars)	0 to 262 bytes (2 to 524 hex chars)	1 or 2 chars

For *PC_To_RDR…* messages, the Bulk Header contains only the field Slot Number of the actual HEADER.

For *RDR_To_PC…* messages, the Bulk Header contains only the field Slot Status of the actual HEADER.

c. Block format – INTERRUPT

This block format is applicable for the *INTERRUPT_TO_PC* ENDPOINT, i.e. only to message *PC_To_RDR_NotifySlotChange*.

START MARK	Message Type	Data	END MARK
1 char	1 byte (2 hex chars)	1 byte (2 hex chars)	1 or 2 chars

3.3.3. Start and End Marks

Field	Description
START MARK	The START MARK is the constant value '^' **(caret, $_h$5E)**
END MARK	For messages going from the PC to the Coupler, the END MARK may be either '\r' (carriage return, $_h$0D), '\n' (line feed, $_h$0A), or both. For messages going from the Coupler to the PC, the END MARK is '\r' ($_h$0D) followed by '\n' ($_h$0A).

3.3.4. Timeout

There's no timeout.

3.4. Command layer

Chapter 6, 7, 8 and 9 contain the documentation of the Command Layer.

Chapter 10 and document [PMD15305] explain how to use the Coupler once the protocol is correctly implemented.

3.5. General communication flow

3.5.1. Session establishment

The PC tries to connect to the Coupler over a serial link by sending GET DESCRIPTOR commands (§ 6.3).

Once a Coupler has been found, the PC queries all the Coupler's descriptors, and, when ready, starts the Coupler using the SET CONFIGURATION command (§ 6.4).

No communication could occur on the Bulk-In, Bulk-Out or Interrupt endpoints before the SET CONFIGURATION command has been issued by the PC and acknowledged by the Coupler.

3.5.2. Operation mode: half-duplex or full-duplex?

Depending on the underlying hardware, the serial link could be either half-duplex (RS-485) or full-duplex (RS-232, RS-TTL).

On a half-duplex medium, collisions must be avoided. The protocol is of request/response type: a RDR_To_PC frame could occur only as a response to a PC_To_RDR frame. As a consequence, there's no way for the Coupler to notify the PC when a card is inserted or removed. The PC must therefore “poll” the Coupler, by sending the PC_To_RDR_GetSlotStatus (§ 7.3.3) command as often as possible.

On a full-duplex medium, there's no risk of collision. The request/response protocol is enhanced by so-called interrupt frames (§ 9.3.1) to notify the PC when a card is inserted or removed. This operation mode dramatically lowers down the workload on the PC.

It's the responsibility of the PC to select the proper operation mode when sending SET CONFIGURATION command (§ 6.4).

3.6. Error handling and recovery

3.6.1. For the Coupler

- **Malformed frame, Protocol violation:** in case it receives a block that doesn't obey to the block-formatting rules, the Coupler sends a NAK char ($_h$15).

3.6.2. For the PC

- **Malformed frame, Protocol violation:** in case it receives a block that doesn't obey to the block-formatting rules, the PC shall wait 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 3.5.1),
- **Processing timeout:** the Coupler starts its answer (Start Byte) within 500ms for the commands sent to the Control Endpoint (CONTROL_TO_RDR), and within 1500ms for the commands sent to the Bulk-Out Endpoint (BULK_TO_RDR). If the Coupler doesn't answer within the specified time, the PC shall wait at least 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 3.5.1).

4. CCID over TCP

This chapters concerns the E663 OEM Couplers, and all products based on the E663 core.

4.1. TCP link

The **SpringCard** network-attached PC/SC Coupler is a TCP Server, and the Host is the Client.

Note that the Coupler is not able to accept more than one Client at the time. Trying to connect to the same Coupler from two different Host is not supported, and shall not be tried. An undefined behaviour may occur.

This Transport Layer is designed to support the transmission of variable-length blocks. The session-establishment makes it possible for both partners to check they are running the same protocol.

4.2. Transport layer

4.2.1. Block format

Every block transmitted in the channel is formatted as follow:

ENDPOINT	HEADER	DATA
1 byte	10 byte	0 to 262 bytes

4.2.2. Description of the fields

Field	Description
ENDPOINT	The ENDPOINT byte is used to route the CCID HEADER and DATA to the appropriate target service (as the USB endpoint feature does).
HEADER	For Bulk-Out and Bulk-In messages, the 10-B HEADER field follows [CCID]. For all the other messages, a proprietary format is defined.
DATA	For Bulk-Out and Bulk-In messages, the DATA field follows [CCID]. For all the other messages, a proprietary format is defined.

4.2.3. Values for the ENDPOINT field

Value	Name	Understanding
h00	CONTROL_TO_RDR	Control Endpoint (orders and queries from PC to RDR)
h80	CONTROL_TO_PC	Control Endpoint (answers from RDR to PC)
h81	BULK_TO_PC	Bulk-In Endpoint (RDR_to_PC responses)
h02	BULK_TO_RDR	Bulk-Out Endpoint (PC_to_RDR commands)
h83	INTERRUPT_TO_PC	Interrupt Endpoint (notifications from RDR to PC)

4.2.4. Size of the blocks

The size of every block can't be less than 11 bytes.

The size of every block can't exceed 273 bytes in plain communication.

4.3. Command layer

Chapter 6, 7, 8 and 9 contain the documentation of the Command Layer.

Chapter 10 and document *[TBD]* explains how to use the Coupler once the protocol is correctly implemented.

4.4. General communication flow

4.4.1. Session establishment

The PC tries to connect to one (or many) Couplers.

When a connection is established on the Coupler, the PC queries the Coupler's descriptors, and, when ready, starts the Coupler using the SET CONFIGURATION command (§ 6.4).

No communication could occur on the Bulk-In, Bulk-Out or Interrupt endpoints before the SET CONFIGURATION command has been issued by the PC and acknowledged by the Coupler.

4.4.2. Nominal dialogue

The TCP channel is full-duplex; both the Coupler and the PC may send at any time, and therefore must be ready to receive at any time.

The PC may send GET STATUS commands to monitor the link (§ 6.2). The Coupler then sends a GET STATUS response within 500ms max.

4.5. Error handling and recovery

4.5.1. For the Coupler

- **Too many hosts:** when receiving a valid SET CONFIGURATION command, the Coupler drops any previous connection,
- **Malformed frame, Protocol violation:** in case it receives a frame that doesn't obey to the block-formatting rules, the Coupler drops the connection,
- **No activity error:** if the PC remains silent for 120s, the Coupler drops the connection.

4.5.2. For the PC

- **Malformed frame, Protocol violation:** in case it receives a frame that doesn't obey to the block-formatting rules, the PC shall drop the connection,
- **Timeout error:** if the PC doesn't receive an answer to a GET STATUS command within 1s + (estimated network time), the PC shall drop the connection.

4.5.3. Recovery

If the connection is dropped for any reason, the PC shall wait at least 5s before trying to connect again to the same Coupler.

5. CCID over BLE

This chapters concerns the products featuring a Bluetooth Low Energy (BLE) interface. The BLE interface shall be loaded with the firmware implementing the CCID GATT documented below.

5.1. Introduction

The **SpringCard** BLE PC/SC Coupler is a Bluetooth LE Device (it advertises its presence) and a GATT Server (it exposes its services through characteristics).The smartphone, tablet or computer that uses the **SpringCard** BLE PC/SC Coupler (in short, the Host) is a Bluetooth LE Central, and a GATT Client. The Central and the Device shall communicate in Bonded mode.

The CCID over BLE implementation has noticeable differences with the Serial and TCP counterparts (and with USB of course):

- Only the two BULK_TO_RDR and BULK_TO_PC endpoints are implemented “as is” through GATT characteristics;
- The endpoints CONTROL_TO_RDR and CONTROL_TO_PC have no equivalent, because the same features are already offered by other means in BLE:
  - The GET DESCRIPTOR command is replaced by the Device GAP and the GATT Device Information Service,
  - The SET CONFIGURATION command is replaced by the BLE protocol itself (the coupler starts running as soon as a BLE Client connects, and stops when the connection is closed or lost),
- The endpoint INTERRUPT_TO_PC is mapped to a specific CCID Status characteristics (5.3.2), which convey both the card presence information and the “BULK_TO_PC is ready” notification.

5.2. Advertisement and scan response

5.2.1. Advertisement frame

The advertisement frame is implemented as follow:

Descriptor #1			Descriptor #2		
Len	Type	Data	Len	Type	Data
$_h$02	$_h$01	$_h$05	$_h$11	$_h$16	$_H$6A BF 51 A3 07 8F 3A 9C 30 4C F2 86 AC 42 A4 BB
Flags Record		- LE Limited Discoverable Mode - No BR/EDR (BLE only)	**Incomplete list of 128-bit Service Class UUIDs** Record		UUID of the SpringCard CCID Service

5.2.2. Scan response frame

Descriptor #1		
Len	Type	Data
$_h$14	$_h$09	$_h$53 6F 63 6B 65 74 20 44 36 30 30 20 xx xx xx xx xx xx
Complete Local Name Record		"*Product Name* xxxxxx" where xx..xx are the last 3 bytes of the BT_ADDR, expressed in hexadecimal *Android uses this field to show the product to the user.*

5.3. GATT profile

The following pages detail the GATT profile of the product.

5.3.1. Standard services

UUID	Mnemonic	Access	Description
Generic Access Profile			
1800	org.bluetooth.service.generic_access		
2A00	org.bluetooth.characteristic.gap.device_name	Read	"*Product Name* XXXXXX" *where XXXXXX are the last 3 bytes of the BT_ADDR, expressed in hexadecimal* *IOS could use this field for display.*
2A01	org.bluetooth.characteristic.gap.appearance	Read	$_h$00 00 *(unknown)*

UUID	Mnemonic	Access	Description
Device Information			
180A	org.bluetooth.service.device_information		
2A29	org.bluetooth.characteristic.manufacturer_name_string	Read	"Springcard" *(not changeable)*
2A24	org.bluetooth.characteristic.model_number_string	Read	"*Product Name*" *(not changeable)*
2A25	org.bluetooth.characteristic.serial_number_string	Read	"xxxxxxxxxxxx" *(BT_ADDR in hex)*
2A26	org.bluetooth.characteristic.firmware_revision_string	Read	The version of the BlueGecko stack in the BGG113/BGM11 "xxxx yyyy zzzz" (major version, minor version, patch level)
2A28	org.bluetooth.characteristic.software_revision_string	Read	"MM.mm PCSC 01"
2A50	org.bluetooth.characteristic.pnp_id	Read	Vendor ID Source = $_h$02 Vendor ID = $_h$1C34 Product ID = $_h$ABD0 Product Version = $_h$*MMmm*

UUID	Mnemonic	Access	Description
Battery level			
180F	org.bluetooth.service.battery		
2A19	org.bluetooth.characteristic.battery_level	Read	Battery level, 0 to 100 %
Service changed			
1801	org.bluetooth.service.generic_attribute		
2A05	org.bluetooth.characteristic.gatt.service_changed	Read, Indicate	Tell the host to refresh its cache

5.3.2. CCID Status characteristic

UUID	Name	Type	Size	Access
7C334BC2-1812-4C7E-A81D-591F92933C37	CCID Status	byte	2	Read, indicate

The CCID Status characteristic is a **BYTE array** holding 3 informations:

- The number of installed slots (1 for a single contactless coupler),
- The fact that there's a card present in every slot, or not,
- The fact that the Host shall read a response from the CCID BULK_TO_PC characteristic.

The CCID Status characteristic is **indicated**, which means that the Host receives a notification from the Device everytime the value changes.

Bits	Value	Meaning
Byte 0 – Number of slots and response bit		
7	$_b$1 $_b$0	There's a response to read in CCID BULK_TO_PC No response
6 - 4		RFU
3 - 0	$_b$0001	Number of slots present into the device
Byte 1 – Slots state		
7 - 0	One bit per slot	$_b$1 if there is a card on the slot else $_b$0

5.3.3. CCID BULK_TO_RDR characteristic

UUID	Name	Type	Size	Access
91ACE9FD-EDD6-40B1-BA77-050A78CF9BC0	BULK_TO_RDR	byte	Long write	Write

The BULK_TO_RDR characteristic is used to send a CCID command to the Device, exactly as this is the case of the BULK_TO_RDR endpoint in other implementations.

The endpoint byte (1st byte) shall not be transmitted (transmission starts with Message Type byte).

5.3.4. BULK_TO_PC characteristic

UUID	Name	Type	Size	Access
B4CA2D75-B855-4C1A-BF40-4A72AE46BD5A	BULK_TO_PC	byte	Long read	Read

The BULK_TO_PC characteristic is used to read a CCID response from the Device, exactly as this is the case of the BULK_TO_PC endpoint in other implementations.

The endpoint byte (1st byte) is not transmitted (transmission starts with Message Type byte).

The Host shall not poll this characteristic. The Device will set bit 7 in CCID Status (5.3.2) and notify the Host (BLE *indicate* feature) as soon as the response is ready.

6. Command Layer – Control Endpoint

The commands/responses described in this chapter are proprietary, yet inspired for the largest part by the USB Specification [USB].

6.1. List of Control message pairs

ID	Control request	See
h00	GET STATUS	6.2
h06	GET DESCRIPTOR	6.3
h09	SET CONFIGURATION	6.4

6.2. Get Status command/response

The GET STATUS command/response pair is used to monitor the link.

a. GET STATUS command format

GET STATUS command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h00	CONTROL_TO_RDR
1	Message Type	1	h00	GET_STATUS
2	Data Length	4	h00000000	*Data field is empty*
6	Value_L, Value_H	2	h0000	*Not used*
8	Index_L, Index_H	2	h0000	*Not used*
10	Option	1	h00	*Not used*

GET STATUS command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	h00	GET_STATUS
1	Value_L, Value_H	2	h0000	*Not used*
3	Index_L, Index_H	2	h0000	*Not used*
5	Option	1	h00	*Not used*

b. GET STATUS response format

GET STATUS response format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h80	CONTROL_TO_PC
1	Message Type	1	h00	GET_STATUS
2	Data Length	4	h00000000	*Data field is empty*
6	Value_L, Value_H	2	h0000	*Not used*
8	Index_L, Index_H	2	h0000	*Not used*
10	Status	1		Reader status or error code. See § c below.

GET STATUS response format – ASCII protocols

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$00	GET_STATUS
1	Value_L, Value_H	2	$_h$0000	*Not used*
3	Index_L, Index_H	2	$_h$0000	*Not used*
5	Status	1		Reader status or error code. See § c below.

c. Value of the Status byte in GET STATUS response

Status byte	Description	Possible cause
Success		
$_h$00	OK	(answer to GET_STATUS command)
Not-fatal errors (the link remains active)		
$_h$01	Error	Unsupported Control command
Fatal errors (the RDR closes the link)		
$_h$FC	Overrun	The PC sends a new Bulk In command while a Bulk In command is already pending
$_h$FD	Denied	The PC sends Bulk In command but is not the 'owner' of the RDR (SET_CONFIGURATION must be called before)
$_h$FE	Overflow	The Bulk In command is too long for the RDR's buffer
$_h$FF	Protocol error	- The PC sends a command to an invalid endpoint - The PC sends a mal-formed frame

6.3. Get Descriptor command/response

The GET STATUS command/response pair is used to detect the Coupler and retrieve its information.

6.3.1. Command/response format

a. GET DESCRIPTOR command format

GET DESCRIPTOR command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h00	CONTROL_TO_RDR
1	Message Type	1	h06	GET_DESCRIPTOR
2	Data Length	4	h00000000	*Data field is empty*
6	Value_L	1		Descriptor Type
7	Value_H	1		Descriptor Index
3	Index_L, Index_H	2	h0000	*Not used*
10	Option	1	h00	*Not used*

GET DESCRIPTOR command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	h06	GET_DESCRIPTOR
1	Value_L	1		Descriptor Type
2	Value_H	1		Descriptor Index
3	Index_L, Index_H	2	h0000	*Not used*
5	Option	1	h00	*Not used*

b. GET DESCRIPTOR response format

GET DESCRIPTOR response format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CONTROL_TO_PC
1	Message Type	1	$_h$06	GET_DESCRIPTOR
2	Data Length	4		The length of the Descriptor (L)
6	Value_L	1		Same as Descriptor Type specified by the PC
7	Value_H	1		Same as Descriptor Index specified by the PC
3	Index_L, Index_H	2	$_h$0000	*Not used*
10	Status	1	$_h$00	*Not used*
11	Data	L		The content of the Descriptor

GET DESCRIPTOR response format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$06	GET_DESCRIPTOR
1	Value_L	1		Same as Descriptor Type specified by the PC
2	Value_H	1		Same as Descriptor Index specified by the PC
3	Index_L, Index_H	2	$_h$0000	*Not used*
5	Status	1	$_h$00	*Not used*
6	Data	L		The content of the Descriptor

6.3.2. List of available descriptors

Value_L	Value_H	Retrieves	See
Descr. Type	Descr. Index		
h01	h00	The Device descriptor	6.3.2.a
h02	h00	The Configuration descriptor	6.3.2.b
h03	h01	The Vendor Name (“SpringCard”)	
h03	h02	The Product Name	
h03	h03	The Product Serial Number	

String descriptors (Vendor Name, Product Name, Product Serial Number) are returned in UTF16.

a. The Device descriptor

The Device descriptor is defined as follow:

Offset	Field	Size	Value	Description / remark
0	Size	1	h12	
1	Type	1	h01	This is a Device descriptor
2	USB version	2	h0200	*Don't care*
4	Class	1	h00	
5	SubClass	1	h00	
6	Protocol	1	h00	
7	MaxPacketSize0	1	h00	*Don't care*
8	Vendor ID	2	h1C34	Pro Active / SpringCard
10	Product ID	2		The Product ID
12	Version	2		The Firmware Version
14	iManufacturer	1	h01	Index of the Vendor Name string
15	iProduct	1	h02	Index of the Product Name string
16	iSerialNumber	1	h03	Index of the Product Serial Number string
17	Configurations	1	h01	Only one configuration supported

b. The Configuration descriptor

The Configuration descriptor is defined as follow:

Offset	Field	Size	Value	Description / remark
Configuration part				
0	Size	1	h09	
1	Type	1	h02	This is a Configuration descriptor
2	Total size	2		
4	Interfaces	1	h01	Only one interface
5	Configuration	1	h01	This is the first (and single) configuration
6	iConfiguration	1	h00	No string to describe the Configuration
7	Attributes	1	h00	*Don't care*
8	MaxPower	1	h00	*Don't care*
Interface part				
9	Size	1	h09	
10	Type	1	h04	This is an Interface descriptor
11	Interface	1	h00	Interface number = 0
12	AlternateSettings	1	h00	*Don't care*
13	Endpoints	1	h03	3 endpoints
14	Class	1	h0B	Class is CCID
15	SubClass	1	h00	
16	Protocol	1	h00	
17	iInterface	1	h00	No string to describe the Interface
CCID-specific part				
18	Size	1	h36	
19	Type	1	h21	Specific to CCID
20	Version	2	h0110	Version of CCID implementation (1.10)
22	MaxSlotIndex	1		The number of CCID slots, minus 1
23	VoltageSupport	1		
24	Protocols	4	h00000003	Supports T=0 and T=1
28	DefaultClock	4	h0000A00F	Default clock is 4MHz (dummy value)

32	MaximumClock	4	$_h$0000A00F	Manimum clock is 4MHz (dummy value)
36	NumClockSupported	1	$_h$00	
37	DataRate	4		
41	MaxDataRate	4		
45	NumDataRateSupported	1		
46	MaxIFSD	4		
50	SynchProtocols	4		
54	Mechanical	4		
58	Features	4		
62	MaxCCIDMessageLength	4		
66	ClassGetResponse	1	$_h$FF	
67	ClassEnvelope	1	$_h$FF	
68	LcdLayout	2	$_h$0000	
70	PinSupport	1	$_h$00	
71	MaxCCIDBusySlot	1	$_h$01	
1st endpoint (Bulk In)				
72	Size	1	$_h$07	
73	Type	1	$_h$05	This is an Endpoint descriptor
74	Address	1	$_h$81	EP1, RDR to PC
75	Attributes	1	$_h$02	Bulk
76	MaxPacketSize	2	$_h$0118	Up to 280 bytes
78	Interval	1	$_h$00	*Don't care*
2nd endpoint (Bulk Out)				
72	Size	1	$_h$07	
73	Type	1	$_h$05	This is an Endpoint descriptor
74	Address	1	$_h$02	EP2, PC to RDR
75	Attributes	1	$_h$02	Bulk
76	MaxPacketSize	2	$_h$0118	Up to 280 bytes
78	Interval	1	$_h$00	*Don't care*
3rd endpoint (Interrupt In)				
72	Size	1	$_h$07	

73	Type	1	h05	This is an Endpoint descriptor
74	Address	1	h83	EP3, RDR to PC
75	Attributes	1	h03	Interrupt
76	MaxPacketSize	2	h0118	Up to 280 bytes
78	Interval	1	h01	*Don't care*

6.3.3. Response to a query for an unknown descriptor

If the Coupler receives a query for and unknown descriptor, it returns a GET DESCRIPTOR response with no data.

6.4. Set Configuration command/response

The SET CONFIGURATION command/response pair is used to start the Coupler (specifying the operation mode) or to stop it afterwards.

a. SET CONFIGURATION command format

SET CONFIGURATION command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$00	CONTROL_TO_RDR
1	Message Type	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value_L	1	$_h$00	
7	Value_H	1		$_h$01 to start the Coupler $_h$00 to stop the Coupler
8	Index	2	$_h$0000	*Not used*
10	Option	1		*Serial Coupler:* $_h$00 half-duplex operation mode $_h$01 full-duplex operation mode $_h$02 *RFU, do not use* $_h$03 full-duplex operation mode, LPCD *TCP Coupler:* $_h$00 all other values are RFU

SET CONFIGURATION command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$09	SET_CONFIGURATION
1	Value_L	1	$_h$00	
2	Value_H	1		$_h$01 to start the Coupler $_h$00 to stop the Coupler
3	Index_L, Index_H	2	$_h$0000	*Not used*
5	Option	1		*Serial RDR:* $_h$00 half-duplex operation mode $_h$01 full-duplex operation mode $_h$03 full-duplex operation mode, LPCD *TCP RDR: not available*

b. SET CONFIGURATION response format

SET CONFIGURATION response format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CONTROL_TO_PC
1	Message Type	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value_L	1	$_h$00	
7	Value_H	1		Same as the Value specified by the PC
8	Index_L, Index_H	2	$_h$0000	*Not used*
10	Status	1		$_h$00 Coupler is stopped $_h$01 Coupler is running $_h$FF An error has occurred

SET CONFIGURATION response format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$09	SET_CONFIGURATION
1	Value_L	1	$_h$00	
2	Value_H	1		Same as the Value specified by the PC
3	Index_L, Index_H	2	$_h$0000	Same as the Index specified by the PC
5	Status	1		$_h$00 Coupler is stopped $_h$01 Coupler is running $_h$FF An error has occurred

6.5. Answers to unsupported messages

Binary protocols

If the Coupler receives an unsupported command, or a command with invalid parameters, it sends a GET STATUS response claiming an error, as follow:

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CONTROL_TO_PC
1	Message Type	1	$_h$00	GET_STATUS
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value	2	$_h$0000	*Not used*
8	Index	2	$_h$0000	*Not used*
10	Status	1	$_h$FF	Unsupported command

ASCII protocols

The Coupler sends a NAK byte.

7. Command Layer – Bulk-Out Endpoint (PC to RDR messages)

All Bulk-Out commands described in this chapter are documented with more details in the CCID Specifications [CCID].

7.1. List of supported/unsupported Bulk-Out messages

ID	Command message	See	Response message	Supported
h61	PC_To_RDR_SetParameters			✗
h62	**PC_To_RDR_IccPowerOn**	7.3.1	**RDR_To_PC_DataBlock**	✓
h63	**PC_To_RDR_IccPowerOff**	7.3.2	**RDR_To_PC_SlotStatus**	✓
h65	**PC_To_RDR_GetSlotStatus**	7.3.3	**RDR_To_PC_SlotStatus**	✓
h69	PC_To_RDR_Secure			✗
h6A	PC_To_RDR_T0APDU			✗
h6B	**PC_To_RDR_Escape**	7.3.5	**RDR_To_PC_Escape**	✓
h6C	PC_To_RDR_GetParameters			✗
h6D	PC_To_RDR_ResetParameters			✗
h6E	PC_To_RDR_IccClock			✗
h6F	**PC_To_RDR_XfrBlock**	7.3.4	**RDR_To_PC_DataBlock**	✓
h71	PC_To_RDR_Mechanical			✗
h72	PC_To_RDR_Abort			✗
h73	PC_To_RDR_SetDataRateAnd ClockFrequency			✗

7.2. Binding to the Transport Layers

a. Binding to the Serial binary Transport

For a Serial communication using the binary protocol, the Bulk-Out message is sent by the PC to the Coupler in the following format:

START BYTE	ENDPOINT	CCID message	CHECKSUM
$_h$CD	$_h$02	10 + (0 to 262) bytes	1 byte

b. Binding to the Serial ASCII Transport

For a Serial communication using the ASCII protocol, the Bulk-Out message is sent by the PC to the Coupler in the following format:

START MARK	Modified CCID message	END MARK
'^'	2 to 264 bytes (4 to 528 hex chars)	'\r' or '\n' or "\r\n"

c. Binding to the TCP binary Transport

For a TCP communication, the Bulk-Out message is sent by the PC to the Coupler in the following format:

ENDPOINT	CCID message
$_h$02	10 + (0 to 262) bytes

7.3. PC to RDR messages

7.3.1. PC_To_RDR_IccPowerOn

The **PC_to_RDR_IccPowerOn** command allows to power up the card, and retrieve its ATR. This is the equivalent of the *SCardConnect* command in the PC/SC world.

If the card is present and has been correctly powered up, the response to this command is the **RDR_to_PC_DataBlock** message (§ 8.3.1); the Data returned is the card's ATR (Answer To Reset).

If there's no card in the slot, or the Coupler has failed to power up the card, the response to this command is the **RDR_to_PC_SlotStatus** message (§ 8.3.2).

PC_To_RDR_IccPowerOn command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$02	BULK_PC_TO_RDR
1	Message Type	1	$_h$62	PC_TO_RDR_ICCPOWERON
2	Data Length	4	$_h$00000000	
6	Slot Number	1		$_h$00 for contactless slot
7	Sequence	1		Sequence number assigned by the PC
8	Power Select	1	$_h$00	Automatic Voltage Selection only
9	RFU	2	$_h$0000	*Reserved for future use*

PC_To_RDR_IccPowerOn command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$62	PC_TO_RDR_ICCPOWERON
1	Slot Number	1		$_h$00 for contactless slot

7.3.2. PC_To_RDR_IccPowerOff

The **PC_to_RDR_IccPowerOn** command allows to power down the card. This is the equivalent of the *SCardDisconnect* command in the PC/SC world.

The response to this command is the **RDR_to_PC_SlotStatus** message (§ 8.3.2).

PC_To_RDR_IccPowerOff command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$02	BULK_PC_TO_RDR
1	Message Type	1	$_h$63	PC_TO_RDR_ICCPOWEROFF
2	Data Length	4	$_h$00000000	
6	Slot Number	1		$_h$00 for contactless slot
7	Sequence	1		Sequence number assigned by the PC
8	RFU	3	$_h$000000	*Reserved for future use*

PC_To_RDR_IccPowerOff command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$63	PC_TO_RDR_ICCPOWEROFF
1	Slot Number	1		$_h$00 for contactless slot

7.3.3. PC_To_RDR_GetSlotStatus

The **PC_to_RDR_GetSlotStatus** command allows to retrieve the status of a slot (whether a card is present, powered or unpowered, or absent). This is the equivalent of the *SCardStatus* command in the PC/SC world.

The response to this command is the **RDR_to_PC_SlotStatus** message (§ 8.3.2).

> *When the half-duplex operation mode is used, the PC must send **PC_To_RDR_GetSlotStatus** as often as possible to know whether a card has been inserted or removed.*
> *When the full-duplex operation mode is used, the PC is notified of the insertions and removals by the **RDR_To_PC_NotifySlotChange** message (§ 9.3.1).*

PC_To_RDR_GetSlotStatus command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$02	BULK_PC_TO_RDR
1	Message Type	1	$_h$65	PC_TO_RDR_GETSLOTSTATUS
2	Data Length	4	$_h$00000000	
6	Slot Number	1		$_h$00 for contactless slot
7	Sequence	1		Sequence number assigned by the PC
8	RFU	3	$_h$000000	*Reserved for future use*

PC_To_RDR_GetSlotStatus command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$65	PC_TO_RDR_GETSLOTSTATUS
1	Slot Number	1		$_h$00 for contactless slot

7.3.4. PC_To_RDR_XfrBlock

The **PC_to_RDR_XfrBlock** command allows to send a C-APDU to a card – or to the Coupler's APDU interpreter, and receive its R-APDU. This is the equivalent of the *SCardTransmit* command in the PC/SC world.

If a valid R-APDU is returned by the card – or by the Coupler's APDU interpreter, the response to this command is the **RDR_to_PC_DataBlock** message (§ 8.3.1); the Data returned is the R-APDU.

If there's no card in the slot, or the Coupler has failed to communicate with the card, the response to this command is the **RDR_to_PC_SlotStatus** message (§ 8.3.2).

PC_To_RDR_XfrBlock command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$02	BULK_PC_TO_RDR
1	Message Type	1	$_h$6F	PC_TO_RDR_XFRBLOCK
2	Data Length	4		Size of the Data field of this message
6	Slot Number	1		$_h$00 for contactless slot
7	Sequence	1		Sequence number assigned by the PC
8	BWI	1	$_h$00	*Not used*
9	Level Parameter	2	$_h$0000	*Not used*
11	Data	Var.		C-APDU to send to the card. Only a maximum length of 262 bytes is supported

PC_To_RDR_XfrBlock command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$6F	PC_TO_RDR_XFRBLOCK
1	Slot Number	1		$_h$00 for contactless slot
2	Data	Var.		C-APDU to send to the card. Only a maximum length of 262 bytes is supported

7.3.5. PC_To_RDR_Escape

The **PC_to_RDR_Escape** command allows to send a Control command directly to the Coupler. This is the equivalent of the *SCardControl* command in the PC/SC world.

The response to this command is the **RDR_to_PC_Escape** message (§ 8.3.3); the Data returned is the response from the Coupler.

PC_To_RDR_Escape command format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$02	BULK_PC_TO_RDR
1	Message Type	1	$_h$6B	PC_TO_RDR_ESCAPE
2	Data Length	4		Size of the Data field of this message
6	Slot Number	1		$_h$00 for contactless slot
7	Sequence	1		Sequence number assigned by the PC
8	RFU	3	$_h$000000	*Reserved for future use*
11	Data	Var.		Data block sent to the RDR. Only a maximum length of 262 bytes is supported

PC_To_RDR_Escape command format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$6B	PC_TO_RDR_ESCAPE
1	Slot Number	1		$_h$00 for contactless slot
2	Data	Var.		Data block sent to the RDR. Only a maximum length of 262 bytes is supported

8. Command Layer – Bulk-In Endpoint (RDR to PC messages)

All Bulk-In responses described in this chapter are documented with more details in the CCID Specifications [CCID].

8.1. List of supported/unsupported Bulk-In messages

ID	Response message	See	In answer to these commands	Supported
h80	**PC_To_RDR_DataBlock**	8.3.1	**PC_To_RDR_IccPowerOn** **RDR_To_PC_XfrBlock**	✓
h81	**RDR_To_PC_SlotStatus**	8.3.2	**PC_To_RDR_IccPowerOff** **PC_To_RDR_GetSlotStatus** PC_To_RDR_Abort	✓ ✓ ✗
h82	RDR_To_PC_Parameters			✗
h83	**RDR_To_PC_Escape**	8.3.3	**PC_To_RDR_Escape**	✓
h84	RDR_To_PC_DataRateAnd ClockFrequency			✗

8.2. Binding to the Transport Layers

a. Binding to the Serial binary Transport

For a Serial communication using the binary protocol, the Bulk-In message is sent by the Coupler to the PC in the following format:

START BYTE	ENDPOINT	CCID message	CHECKSUM
$_h$CD	$_h$81	10 + (0 to 262) bytes	1 byte

b. Binding to the Serial ASCII Transport

For a Serial communication using the ASCII protocol, the Bulk-In message is sent by the Coupler to the PC in the following format:

START MARK	Modified CCID message	END MARK
'^'	2 to 264 bytes (4 to 528 hex chars)	“\r\n”

c. Binding to the TCP binary Transport

For a TCP communication, the Bulk-In message is sent by the Coupler to the PC in the following format:

ENDPOINT	CCID message
$_h$81	10 + (0 to 262) bytes

8.3. RDR to PC messages

8.3.1. RDR_To_PC_DataBlock

The **RDR_To_PC_Datablock** message comes as a response to:

- The **PC_To_RDR_IccPowerOn** command (§ 7.3.1); in this case, the Data field is the card's ATR,
- The **PC_To_RDR_XfrBlock** command (§ 7.3.4); in this case, the Data field is the R-APDU returned by the card – or by the Coupler's APDU interpreter.

RDR_To_PC_DataBlock response format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h81	BULK_RDR_TO_PC
1	Message Type	1	h80	RDR_TO_PC_DATABLOCK
2	Data Length	4		Size of the Data field of this message
6	Slot	1		Same as last PC_TO_RDR message
7	Sequence	1		
8	Slot Status	1		See § 8.4.1
9	Slot Error	1		See § 8.4.2
10	RFU	1	h00	*Reserved for future use*
11	Data	Var.		ATR or R-APDU. Only a maximum length of 262 bytes is supported

RDR_To_PC_DataBlock response format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	h80	RDR_TO_PC_DATABLOCK
1	Slot Status	1		See § 8.4.1
2	Data	Var.		ATR or R-APDU. Only a maximum length of 262 bytes is supported

8.3.2. RDR_To_PC_SlotStatus

The **RDR_To_PC_SlotStatus** message comes as a response to:

- The **PC_To_RDR_GetSlotStatus** command (§ 7.3.3),
- The **PC_To_RDR_IccPowerOff** command (§ 7.3.2),
- The **PC_To_RDR_IccPowerOn** command (§ 7.3.1) if there's no card in the slot or the Coupler has failed to power up the card,
- The **PC_To_RDR_XfrBlock** command (§ 7.3.4) if there's no card in the slot or the Coupler has failed to communicate with the card.

RDR_To_PC_SlotStatus response format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$81	BULK_RDR_TO_PC
1	Message Type	1	$_h$81	RDR_TO_PC_SLOTSTATUS
2	Data Length	4	$_h$00000000	
6	Slot	1		Same as last PC_TO_RDR message
7	Sequence	1		
8	Slot Status	1		See § 8.4.1
9	Slot Error	1		See § 8.4.2
10	ClockStatus	1	$_h$00	Other values are not supported

RDR_To_PC_SlotStatus response format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$81	RDR_TO_PC_SLOTSTATUS
1	Slot Status	1		See § 8.4.1

8.3.3. RDR_To_PC_Escape

The **RDR_To_PC_Escape** message comes as a response to the **PC_To_RDR_Escape** command (§ 7.3.5).

RDR_To_PC_Escape response format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h81	BULK_RDR_TO_PC
1	Message Type	1	h83	RDR_TO_PC_ESCAPE
2	Data Length	4		Size of the Data field of this message
6	Slot	1		Same as last PC_TO_RDR message
7	Sequence	1		
8	Slot Status	1		See § 8.4.1
9	Slot Error	1		See § 8.4.2
10	RFU	1	h00	*Reserved for future use*
11	Data	Var.		Data block sent by the RDR. Only a maximum length of 262 bytes is supported

RDR_To_PC_Escape response format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	h83	RDR_TO_PC_ESCAPE
1	Slot Status	1		See § 8.4.1
2	Data	Var.		Data block sent by the RDR. Only a maximum length of 262 bytes is supported

8.4. Values of the Status and Error fields

Each **RDR_To_PC** message contains a Slot Status and a Slot Error field.

8.4.1. Slot Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Command Status		*Reserved for future use*				**Card Status**	
See below		0	0	0	0	*See below*	

a. Card Status field

Bit 1	Bit 0	Description
Card Status		
0	0	A Card is present and active (powered ON)
0	1	A Card is present and inactive (powered OFF or hardware error)
1	0	No card present (slot is empty)
1	1	*Reserved for future use*

b. Command Status field

Bit 7	Bit 6	Description
Command Status		
0	0	Command processed without error
0	1	Command failed (error code is provided in the Slot Error field)
1	0	Time Extension is requested
1	1	*Reserved for future use*

8.4.2. Slot Error

This field is valid only if Command Status = $_b$01 in the Slot Status field.

Error code	Error name	Possible cause
$_h$FF	CMD_ABORTED	The PC has sent an ABORT command
$_h$FE	ICC_MUTE	Time out in Card communication
$_h$FD	XFR_PARITY_ERROR	Parity error in Card communication
$_h$FC	XFR_OVERRUN	Overrun error in Card communication
$_h$FB	HW_ERROR	Hardware error on Card side (over-current?)
$_h$F8	BAD_ATR_TS	Invalid ATR format
$_h$F7	BAD_ATR_TCK	Invalid ATR checksum
$_h$F6	ICC_PROTOCOL_NOT_SUPPORTED	Card's protocol is not supported
$_h$F5	ICC_CLASS_NOT_SUPPORTED	Card's power class is not supported
$_h$F4	PROCEDURE_BYTE_CONFLICT	Error in T=0 protocol
$_h$F3	DEACTIVATED_PROTOCOL	Specified protocol is not allowed
$_h$F2	BUSY_WITH_AUTO_SEQUENCE	RDR is currently busy activating a Card
$_h$E0	CMD_SLOT_BUSY	RDR is already running a command
$_h$00		Command not supported

NB: this field is not available under the ASCII protocol.

9. Command Layer – Interrupt Enpoint (RDR to PC notifications)

9.1. List of supported Interrupt messages

ID	Response message	See	Supported
h50	RDR_To_PC_NotifySlotChange	9.3.1	✓
h51	RDR_To_PC_HardwareError		✗

9.2. Binding to the Transport Layers

a. Binding to the Serial binary Transport

In order to ease the implementation of the receiving logic, the CCID Interrupt frame is extended to reach the same length as the bulk messages.

For a Serial communication using the binary protocol, the Interrupt-In message is sent by the Coupler to the PC in the following format:

START BYTE	ENDPOINT	CCID message	CHECKSUM
hCD	h83	10 + (0 to 5) bytes	1 byte

b. Binding to the Serial ASCII Transport

For a Serial communication using the ASCII protocol, the Interrupt-In message is sent by the Coupler to the PC in the following format:

START MARK	Modified CCID message	END MARK
'^'	2 to 6 bytes (4 to 12 hex chars)	“\r\n”

c. Binding to the TCP binary Transport

In order to ease the implementation of the receiving logic, the CCID Interrupt frame is extended to reach the same length as the bulk messages.

For a TCP communication, the Interrupt-In message is sent by the Coupler to the PC in the following format:

ENDPOINT	CCID message
h83	10 + (0 to 262) bytes

9.3. Details

9.3.1. RDR_To_PC_NotifySlotChange

The **RDR_To_PC_NotifySlotChange** message comes every-time a card is inserted or removed, provided that the full-duplex operation mode is active.

Card insertion notifications are repeated with a period of about 1s until the PC issues the **PC_To_RDR_IccPowerOn** command to the slot.

Card removal notifications are sent only one. There's no need for the PC to issue the **PC_To_RDR_IccPowerOff** command after a removal.

RDR_To_PC_NotifySlotChange notification format – binary protocols

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$83	INTERRUPT_TO_PC
1	Message Type	1	$_h$50	
2	Data Length	4		Size of the SlotICCState array
6	RFU	5	$_H$00 … $_h$00	*Reserved for future use*
11	SlotICCState	Var.		This field is reported on byte granularity. The size is (2 bits * number of slots) rounded up to the nearest byte. Each slot has 2 bits. The least significant bit reports the current state of the slot (0 = no ICC present, 1 = ICC present). The most significant bit reports whether the slot has changed state since the last RDR_to_PC_NotifySlotChange message was sent (0 = no change, 1 = change). If no slot exists for a given location, the field returns 00 in those 2 bits. Example: A 1 slot CCID reports a single byte with the following format: Bit 0 = Slot 0 card present (1) / absent (0) Bit 1 = Slot 0 status changed (1) / unchanged (0) All other bits will be 0

RDR_To_PC_NotifySlotChange notification format – ASCII protocol

Offset	Field	Size	Value	Description / remark
0	Message Type	1	$_h$50	
1	SlotICCState	Var.		This field is reported on byte granularity. The size is (2 bits * number of slots) rounded up to the nearest byte. Each slot has 2 bits. The least significant bit reports the current state of the slot (0 = no ICC present, 1 = ICC present). The most significant bit reports whether the slot has changed state since the last RDR_to_PC_NotifySlotChange message was sent (0 = no change, 1 = change). If no slot exists for a given location, the field returns 00 in those 2 bits. Example: A 1 slot CCID reports a single byte with the following format: Bit 0 = Slot 0 card present (1) / absent (0) Bit 1 = Slot 0 status changed (1) / unchanged (0) All other bits will be 0

10. Mapping PC/SC calls to PC_To_RDR / RDR_To_PC messages

The PC/SC specification and Microsoft's reference implementation define a short set of functions to work with PC/SC Couplers (and with Cards through a PC/SC Coupler).

This chapter explains how the very basic PC/SC functions shall be implemented.

10.1.1. SCardStatus

In PC/SC, the **SCardStatus** function has two roles:

- Check whether there's a Card in a Coupler or not,
- If a Card is present in the Coupler, return its ATR.

The first role is directly available from CCID, using the PC_To_RDR_GetSlotStatus message:

1. Send the **PC_To_RDR_GetSlotStatus** command (§ 7.3.3),
2. The Coupler returns a **RDR_To_PC_SlotStatus** response (§ 8.3.2),
3. Observe the **Card Status field** in the response (§ 8.4.1.a) to know whether a Card is present and powered, present and not powered, or absent.

To retrieve the Card's ATR, the PC must activate the Card explicitely – this is done using the PC_To_RDR_IccPowerOn message when a Card is present:

4. Send the **PC_To_RDR_IccPowerOn** command (§ 7.3.1),
5. The Coupler returns a **RDR_To_PC_Datablock** response (§ 8.3.1),
6. Observe the **Status and Error fields** in the response (§ 8.4.1). If both fields are zero, communication with the Card is OK, and **the Data field contains the Card's ATR** (Answer To Reset)[3].

10.1.2. SCardConnect

In PC/SC, the **SCardConnect** function has two roles:

- Open a handle to communicate with the Card – the computer's PC/SC subsystem manages concurrent or exclusive access, and take care to release the handle if the application stops for any reason,

[3] For a Contactless Card (PICC/VICC), the ATR is constructed according to PC/SC v2 chapter 3 rules.

- Tells the Coupler which protocol shall be used to communicate with the Card (T=0 or T=1).

None of these two roles are significant for “low level” communication with a CCID Coupler[4]. The actual “Card power up” role is provided by PC_To_RDR_IccPowerOn, which has already been invoked by **SCardStatus** to retrieve the ATR.

10.1.3. SCardTransmit

In PC/SC, the **SCardTransmit** function is the very function that implements the exchange of APDUs (Application Protocol Datagram Units) with a contact or contactless Smart Card.

1. Send your **C-APDU** (application-level Command) in the **Data field** of a **PC_To_RDR_XfrBlock** command (§ 7.3.4),
2. The Coupler returns a **RDR_To_PC_Datablock** response (§ 8.3.1),
3. Observe the **Status and Error fields** in the response (§ 8.4.1). If both fields are zero, communication with the Card is OK, and **the Data field contains the Card's R-APDU** (application-level Response)

> *Note that for a wired logic contactless Card (PICC not compliant with ISO 14443-4 or VICC), the SCardTransmit function doesn't actually communicate with the Card, but with the Coupler's APDU Interpreter which is responsible for translating the “standard” APDU into a low-level, proprietary command specific to the Card. This is the same here, the PC_To_RDR_XfrBlock command goes through the Coupler's APDU Interpreter and not directly to a wired-logic PICC/VICC.*

10.1.4. SCardDisconnect

In PC/SC, the **SCardDisconnect** function has two roles:

- Close the handle that has been opened by **SCardConnect**,
- Tells the Coupler to power down the Card.

The first role is not relevant here.

The second role is taken by the PC_To_RDR_IccPowerOff message:

1. Send the **PC_To_RDR_IccPowerOff** command (§ 7.3.2),
2. The Coupler returns a **RDR_To_PC_SlotStatus** response (§ 8.3.2),
3. Observe the **Card Status field** in the response (§ 8.4.1.a) to know whether the Card has already been removed, or is still present in the Coupler.

[4] All SpringCard couplers feature automatic protocol activation, which mean that the protocol is always selected by the Coupler, not by the application.

10.1.5. SCardControl

In PC/SC, the **SCardControl** function is used to send “low level” commands to the Coupler, to the Coupler's driver, or to the PC/SC subsystem.

In our case, only sending “low level” commands to the Coupler makes sense. This is done using the PC_To_RDR_Escape message.

1. Send the low level command in the **Data field** of a **PC_To_RDR_Escape** command (§ 7.3.5),
2. The Coupler returns its answer in a **RDR_To_PC_Escape** response (§ 8.3.3).

> *Sending an escape sequence through PC_To_RDR_Escape is exactly the same as sending a “legacy command” to a **SpringCard** coupler running in **legacy (SpringProx) mode**.*

11. Configuration registers for a Serial Coupler

There are 3 ways to define the other configuration registers:

- Manually, using any terminal application to communicate with the Coupler's over its serial line,
- Using **SpringCard MultiConf** application,
- Using the **PC_To_RDR_Escape** command, sending the configuration stream within the command's data.

This chapter shows only the few register that are relevant for the CCID-over-serial communication with the Coupler. Please refer to the Coupler's Developers' Guide for all details regarding the configuration methods and the other registers.

11.1. Operating mode

Name	Address	Description	Size
MOD	$_h$C0	Coupler operating mode and options. See table below	2

Coupler operating mode and option bits

Bits	Value	Meaning
Byte 0 : operating mode		
7-0	00000000	The PC selects the operating mode (Legacy or CCID)
	00000001	Legacy mode enforced
	00000010	CCID mode enforced
		All other values are RFU and shall not be set
Byte 1 : operating options		
7-4		*RFU (set to 0000)*
3-2		**CCID auto-start**
	00	do not start CCID operation upon reset
	01	start CCID operation using the binary protocol
	10	*RFU, do not use*
	11	start CCID operation using the ASCII protocol
1-0		**CCID operating mode**
	00	half-duplex operation mode
	01	full-duplex operation mode
	10	*RFU, do not use*
	11	full-duplex operation mode, LPCD

Default value: $_b$00000000 $_b$10000000

11.2. UART CONFIGURATION

Name	Address	Description	Size
SER	$_h$67	UART configuration bits. See table below	1

UART configuration bits

Bits	Value	Meaning
7	0	Echo is ON during console communication
	1	Echo is OFF during console communication
6-3		*RFU (set to 0000)*
2-0		**Baudrate**
	101	38400bps
	111	115200bps
		All other values are RFU and shall not be set

Default value: $_b$00000101

12. Configuration registers for a TCP Coupler

The Network configuration (address, net-mask, gateway, info/location and password for Telnet access) could be defined using **SpringCard NDDU** application (Network Devices Discovery Utility).

There are 3 ways to define all the other configuration registers:

- Manually, using a Telnet access to the Coupler,
- Using **SpringCard MultiConf** application once the Coupler is installed as a PC/SC Reader under Windows,
- Using the **PC_To_RDR_Escape** command, sending the configuration stream within the command's data.

This chapter shows only the few register that are relevant for the CCID-over-TCP communication with the Coupler. Please refer to the Coupler's Developers' Guide for all details regarding the configuration methods and the other registers.

12.1. Security options

Name	Address	Description	Size
SEC	$_h$6E	Security option bits. See table below	1

Security option bits

Bits	Value	Meaning
7	0	Telnet access is disabled
	1	Telnet access is enabled
6-0		*RFU (set to 0000000)*

Default value: $_b$10000000

12.2. Network configuration

12.2.1. IPv4 address, mask, and gateway

Name	Address	Description	Size
IPA	$_h$80	IPv4 configuration bytes, see table below	4, 8 or 12

IPv4 configuration bytes

Bytes	Contains	Remark
0	Address, 1st byte	Device's IPv4 Address.
1	Address, 2nd byte	
2	Address, 3rd byte	If these bytes are missing, the default IP Address $_h$C0 A8 00 FA (192.168.0.250) is taken.
3	Address, 4th byte	
4	Mask, 1st byte	Network Mask.
5	Mask, 2nd byte	
6	Mask, 3rd byte	If these bytes are missing, the default Mask $_h$FF FF FF FF (255.255.255.0) is taken.
7	Mask, 4th byte	
8	Gateway, 1st byte	Default Gateway.
9	Gateway, 2nd byte	
10	Gateway, 3rd byte	If these bytes are missing, the value $_h$00 00 00 00 (0.0.0.0) is taken, meaning that there's no Gateway.
11	Gateway, 4th byte	

Default value: $_h$C0 A8 00 FA FF FF FF 00 00 00 00 00

(address = 192.168.0.250, mask = 255.255.255.0, no gateway)

12.2.2. TCP server port

Name	Address	Description	Size
IPP	$_h$81	Listen TCP port for the server (2 bytes, MSB-first)	2

Default value: $_h$0F 9F (server TCP port = 3999)

12.2.3. Ethernet configuration

Name	Address	Description	Size
ETC	$_h$8D	Ethernet configuration bits. See table below	1

Ethernet configuration bits

Bits	Value	Meaning
7-1		*RFU (set to 0000000)*
0	0	Use auto-configuration (10/100Mbps, half or full-duplex)
	1	Force bitrate = 10Mbps, half-duplex

Default value: $_b$00000000

12.2.4. Info / Location

Name	Address	Description	Size
ILI	$_h$8E	Info / Location string	Var. 0-30
		Default value: empty	

The **Info / Location** string is a text value (ASCII) that appears

- When someone tries to connect on Telnet,
- In the NDDU software.

Use this string as a reminder of where your RDR is installed, or what is its role in your access-control system.

12.2.5. Password for Telnet access

Name	Address	Description	Size
ITP	$_h$8F	Password for Telnet access string	Var. 0-16
		Default value: “springcard”	

The **Password for Telnet access** string is a text value (ASCII) that protects the access to the RDR using Telnet protocol.

The password is mandatory. If this registry is not set, default value “springcard” is used.

Disclaimer

This document is provided for informational purposes only and shall not be construed as a commercial offer, a license, an advisory, fiduciary or professional relationship between SPRINGCARD and you. No information provided in this document shall be considered a substitute for your independent investigation.

The information provided in document may be related to products or services that are not available in your country.

This document is provided "as is" and without warranty of any kind to the extent allowed by the applicable law. While SPRINGCARD will use reasonable efforts to provide reliable information, we don't warrant that this document is free of inaccuracies, errors and/or omissions, or that its content is appropriate for your particular use or up to date. SPRINGCARD reserves the right to change the information at any time without notice.

SPRINGCARD doesn't warrant any results derived from the use of the products described in this document. SPRINGCARD will not be liable for any indirect, consequential or incidental damages, including but not limited to lost profits or revenues, business interruption, loss of data arising out of or in connection with the use, inability to use or reliance on any product (either hardware or software) described in this document.

These products are not designed for use in life support appliances, devices, or systems where malfunction of these product may result in personal injury. SPRINGCARD customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify SPRINGCARD for any damages resulting from such improper use or sale.

Copyright notice

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