springcard®

SPRINGCARD E663/RDR, FUNKYGATE-IP NFC, E663/MIO, HANDYDRUMMER-IP

Network Integration and Configuration

www.springcard.com

DOCUMENT IDENTIFICATION

Category	Admin/Config Manual		
Family/Customer	Network Devices		
Reference	PMA14166	Version	AC
Status	Final	Classification	Public
Keywords			
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\RFID scanners et lecteurs\IWM2-Commun\[PMA14166-AC] E663-RDR, FunkyGate-IP, E663-MIO, HandyDrummer-IP Network Integration and Configuration.odt
Date saved	09/09/16
Date printed	03/06/14

REVISION HISTORY

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	30/07/14	JDA				Created from PMA13257-AC
AB	12/08/14	JDA				Defined the HTTP REST API for I/O Modules Improved the explanation in 5.3.6 and 5.6.1.b Fixed the definition of p in 5.6.2.b
AC	09/09/16	JDA				Added support for DNS client (resolv) in version 1.69 of firmware Removed HTTP REST API (moved to product's specific documentation)

CONTENTS

1. INTRODUCTION

1.1. ABSTRACT

SpringCard E663/RDR is SpringCard's network-attached, smart Reader core. In provides in the same device a versatile RFID (13.56MHz) and NFC interface, and the logic to fetch data from virtually any compliant card or tag, including secure authentication on popular NXP MIFARE Classic, Plus and DESFire chips.

The **SpringCard E663/RDR** core is notably present in these products:

- **SpringCard FunkyGate-IP NFC** is a wall-mount network-attached Reader, for access control applications,

- **SpringCard FunkyGate-IP+POE NFC** adds the "powered by the network" (POE) feature,

- **SpringCard TwistyWriter-IP/RDR** is an OEM Reader, suitable for integration in kiosks, gate controllers, etc.

SpringCard E663/MIO is SpringCard's network-attached, I/O controller core. It is present in one product:

- **SpringCard HandyDrummer-IP** is a I/O Module, featuring 8 ON/OFF input and 8 output (relays), and highly expandable. The **SpringCard HandyDrummer-IP+POE** version adds the "powered by the network" (POE) feature.

Both **E663/RDR** and **E663/MIO** share the same TCP/IP (IPv4) on top of Ethernet implementation. This document provides all necessary information to perform the network configuration of all the products based on these two cores, and to develop a software that will exchange data with them.

1.2. AUDIENCE

This manual is designed for use by application developers and system integrators. It assumes that the reader has a good knowledge of computer development and TCP/IP networks.

1.3. 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

1.4. RELATED DOCUMENTS

1.4.1. Products' specifications

You'll find the feature-list and the technical characteristics of every product in the corresponding leaflet.

Document ref.	Content
PFL13276	FunkyGate NFC family leaflet
PFL14164	HandyDrummer family leaflet

1.4.2. Products' specific technical documentations

Every product has its own Integration and Configuration Guide that details the parts not covered in this document.

Document ref.	Content
PMA13257	FunkyGate-IP NFC Integration and Configuration Guide
PMA14165	HandyDrummer-IP Integration and Configuration Guide

2. DEFINE THE DEVICE'S IP ADDRESS

The **Device** comes out of factory without an IP address. This means that you <u>must</u> assign it an IP address before being able to access it either through Telnet link (chapter 3) or using the TCP client/server protocol depicted in chapters 4 and 5.

Using **SpringCard Network Device Discovery Utility (NDDU)** is the preferred method to assign an IP address to the Device.

2.1. ASSIGN AN IP ADDRESS USING NDDU SOFTWARE

SpringCard Network Device Discovery Utility (NDDU) is a Windows-based software that discovers and configures SpringCard Device connected on same the Local Area Network (LAN) as the computer it is running on.

Please use a wired network connection, and make sure the Device(s) you want to configure are on the same LAN as your computer. NDDU makes use of broadcast UDP frames to discover and configure the Devices; therefore, it can't work behind a router or gateway.

2.1.1. Download and install the NDDU software

Make sure your Windows account has administrative privileges.

Download the installer from URL

<div align="center">

www.springcard.com/download/find/file/sn13210

</div>

Install the software.

This software relies on the .NET framework version 4. Please download and install this framework from Microsoft's in case it hasn't already been deployed onto your computer.

2.1.2. Run the NDDU software
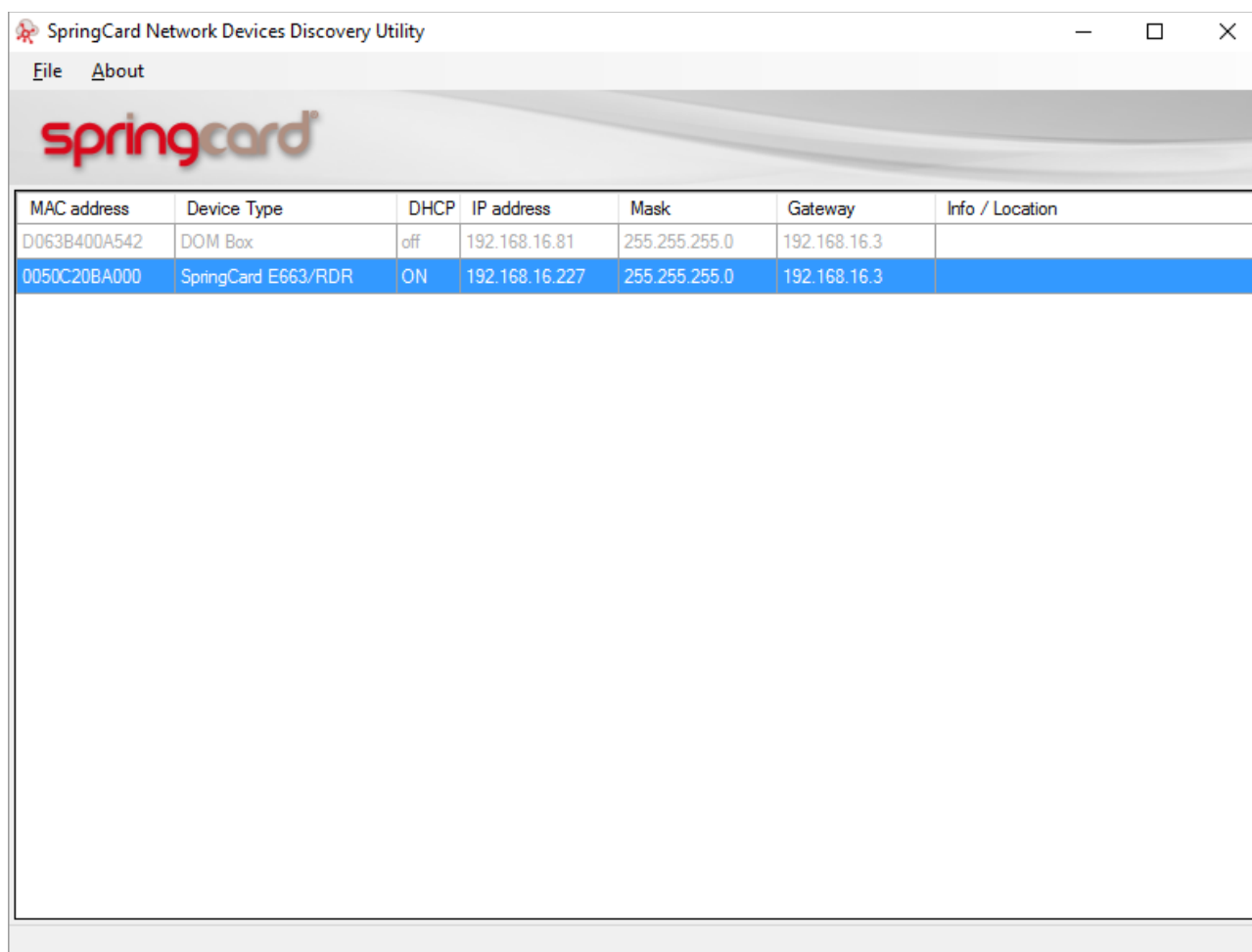
Make sure your Windows account has administrative privileges.

Launch the software: Start Menu → SpringCard → Network Discovery → Network Device Discovery Utility.

On first startup, you should be prompted by Windows Firewall whether you want to allow NDDU to access the network. Please confirm.

2.1.3. Discovered devices

After a few seconds, NDDU displays the list of devices it has found on the LAN.
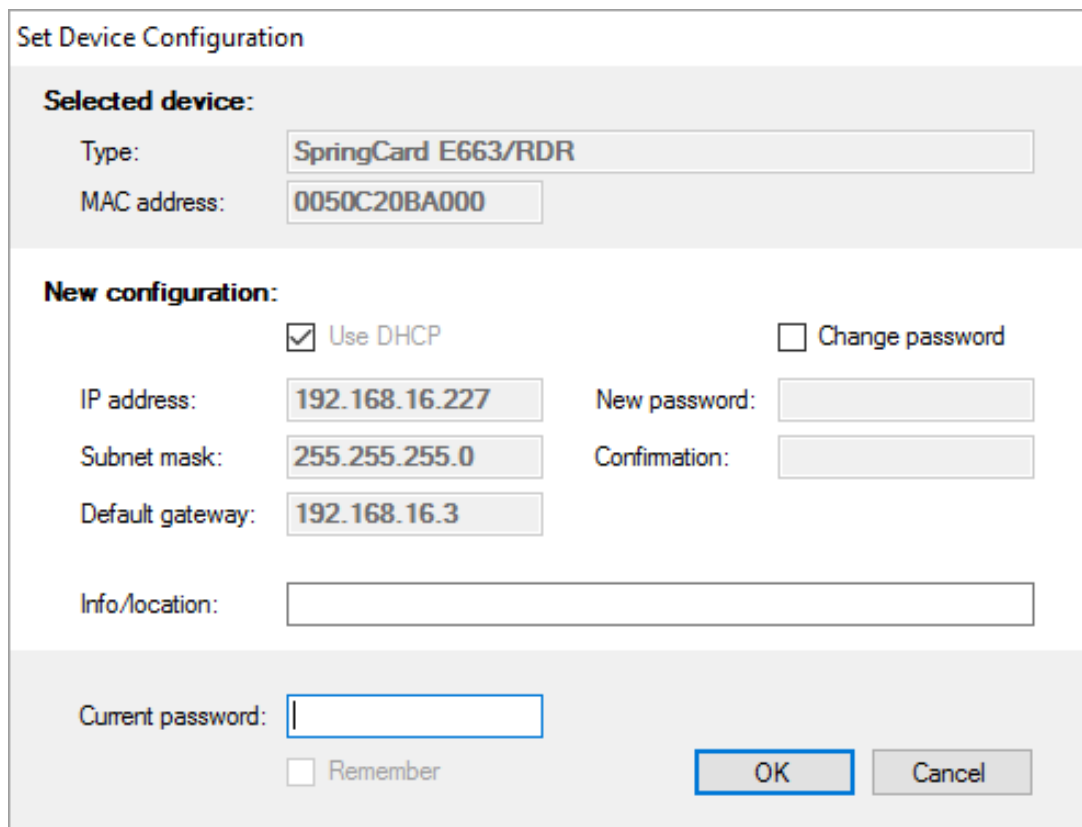
The software's main screen shows 7 columns:

- The MAC address (Ethernet address and also serial number) of every SpringCard Device found on the LAN,

- The device type:

    - code name **SpringCard E663/RDR** for FunkyGate-IP NFC, FunkyGate-IP+POE NFC, TwistyWriter-IP/RDR, and any future product based on E663/RDR core,

    - code name **SpringCard E663/MIO** for HandyDrummer-IP, HandyDrummer-IP+POE, and any future product based on E663/MIO core,

- Whether DHCP is enabled or not (DHCP is not supported on early firmware versions),

- The device's current IP address, local network mask, and default gateway. Until the device has been properly configured, those entries show has "0.0.0.0",

■ A user-defined string named "Info / location", which will be used as a hint to identify the device in your own system.

2.1.4.   Configure a Device

Double-click one of the devices in the list. The configuration form appears:

The form shows the device's current configuration. Enter the new configuration.

a.   Use DHCP?

DHCP stands for Dynamic Host Configuration Protocol. Enable DHCP on the device only if there's a DHCP server running on the network.

Note: in most situations, software that use the device will connect as a client to a server service running in the device. If the device uses DHCP, its address is likely to change frequently, and so the client software must be reconfigured accordingly to know the actual address of its server. It is recommended to reserve a permanent lease on the DHCP server for the device to suppress this annoyance.

b.   Static configuration

IPv4 address and subnet mask are mandatory data and couldn't be left empty. The default gateway is optional; if the devices won't need to use a gateway, leave this field to "0.0.0.0".

c.   Info/Location

In the "info/location" field, enter a short string (less than 32 characters) as a reminder of the device's location or role.

d.   Password

Check the box "change password" and enter a new password twice if you want to change the device's password.

Terminate by entering the device's current password to confirm that your allowed to change this device's configuration.

*The default password for all devices is **springcard**.*

When ready, click "OK".

2.1.5. Verify the new configuration

If everything is OK, including the current password, the NDDU software is able to configure the device. The following message confirms that the new configuration has been accepted:
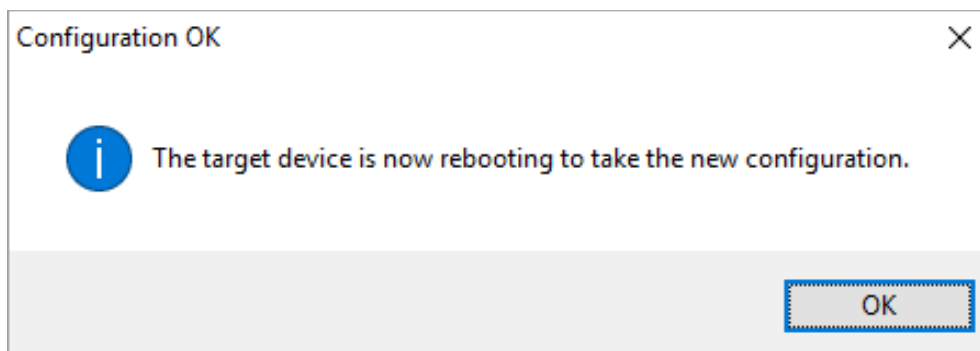
After a few seconds, the list of devices is refreshed and shows the new configuration:
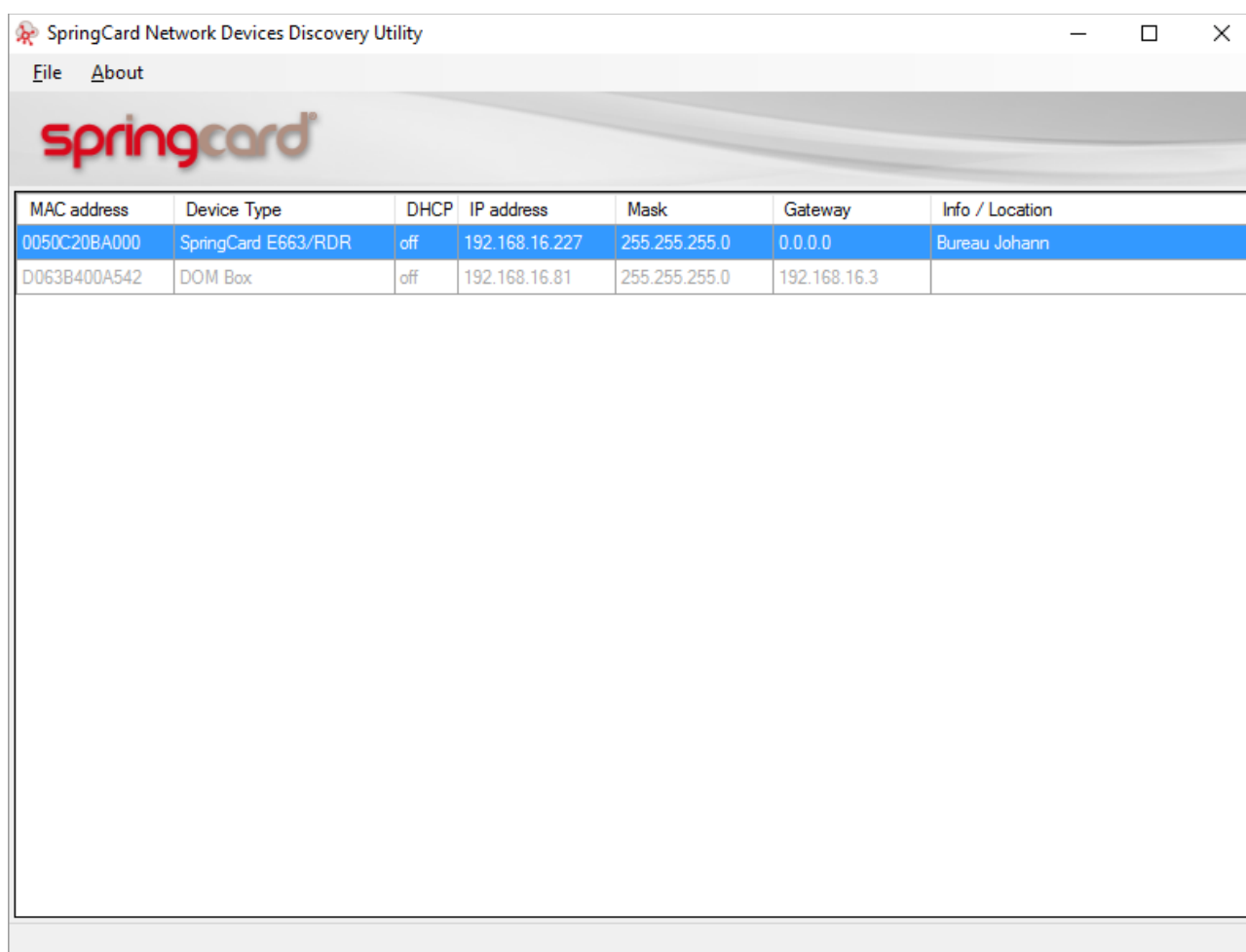
2.2. ASSIGN AN IP ADDRESS TO A READER USING A MASTER CARD

SpringCard Readers in the **FunkyGate** family could be configured by the mean of a contactless Master Card.

The Master Cards are NXP Desfire cards formatted and programmed by **SpringCard Configuration Tool (ScMultiConf.exe, ref # SN14007)** for Windows.

Please refer to this software's documentation for details.

3. TELNET ACCESS TO THE DEVICE

3.1. THE DEVICE'S CONSOLE

The Device features a "human" command processor (shell or console). This feature accessible through the Telnet protocol. It is primarily made for testing and demonstration purpose. Only the few commands depicted in this chapter could safely be used for configuration and diagnostic.

Note that the SEC Configuration Register ($_h6E$, § 8.2) may be used to disable the Console.

3.1.1. Open a Telnet session to the Device

On most operating systems you could find a Telnet client in the default system tools. Open a console and enter

```
telnet xxx.xxx.xxx.xxx
```

where *xxx.xxx.xxx.xxx* is the Device's IP Address as defined in chapter 2.

Windows Vista / 7 / 8 / 10 : the Telnet client may be missing from you OS default install. Go to **Control Panel, Programs and Features** *section, and then enable* **Telnet client** *in the* **Turn Windows features on or off** *tab.*
Alternatively, you may download a free terminal client such as **Putty**, *that is also a Telnet client.*

a. Device's connection prompt

The Device sends its connection prompt. **FunkyGate** Readers say "SpringCard E663/RDR", **HandyDrummer** Modules say "SpringCard E663/MIO".

The second line shows the Info / Location string that has been entered in chapter 2 (if some).

On the third line the Device prompts for a password.

Enter the Reader's password that you have defined in chapter 2.

If you haven't changed the password, the default password is **springcard**.
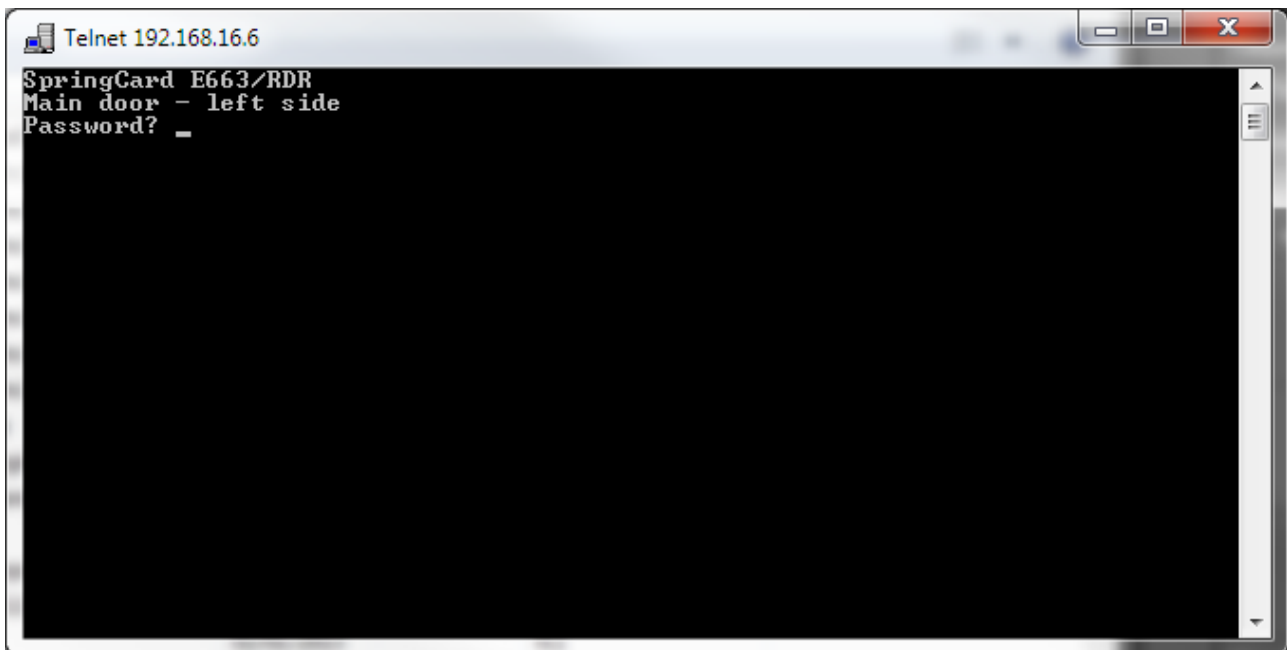
3.1.2.   Sending a command to the Device

Write the command line as documented below, and terminate by hitting the ENTER key.

Note that the Device echoes the entered characters.

3.1.3. List of Console commands

Command	Meaning
version	Show the firmware version
info	Show the firmware information data
show	Show the current configuration
cfg	Dump all Configuration Registers written into persistent memory
cfgXX=YY...YY	Write value $_h$YY...YY to Configuration Register $_h$XX
cfgXX=!!	Erase Configuration Register $_h$XX
cfgXX	Read Configuration Register $_h$XX
exit	Terminate the Telnet session

4. SPRINGCARD NETWORK DEVICE C/S PROTOCOL – PLAIN MODE

4.1. ABSTRACT

SpringCard Network Device C/S Protocol is a light-weight, bandwidth-efficient network protocol. The Device is a TCP Server, and the Host (access control unit or computer in charge) is the Client.

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

This chapter describes the **Plain Transport Layer**.

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. The Host (Client) may also decide to switch to the optional Secure Transport Layer (chapter 5).

> *The **SpringCard Devices** listens on TCP port 3999. This default value could be changed by writing into the IPP configuration register ($_h$81, § 8.3.2).*
> *To enforce security, the Plain Transport Layer could be disabled by setting bit 0 of byte 0 to 1 in the IPS configuration register ($_h$82, § 8.3.3). In this case, the Reader will reject any Host that doesn't switch to the Secure Transport Layer.*

After the initial session-establishment, the Host (Client) and the Device (Server) exchange only I-Blocks (§ 4.2.4.a). **The I-Blocks convey the Application Level Datagrams defined in chapter 6.**

4.2. PRESENTATION LAYER

4.2.1. Block format

Every block transmitted in the channel is formatted as follow:

LENGTH	TYPE	PAYLOAD
1 byte	1 byte	Variable length

4.2.2. Description of the fields

Field	Description
LENGTH	The LENGTH byte is the total length of the block, this byte included.
TYPE	The TYPE byte is used to convey the information required to control the data transmission. There are three fundamental types of blocks: • I-block used to convey information for use by the upper layers • H-block used to exchange control information between the Server and the Client
PAYLOAD	The PAYLOAD field is optional. When present, the PAYLOAD field conveys application data.

4.2.3. Size of the blocks

The size of every block must be less or equal to 66 bytes.

This lead to a PAYLOAD between 0 and 64 bytes.

If the application layer needs to transmit more than 64 bytes, chaining shall be used.

4.2.4. Format of the TYPE byte

a. I-Block

Bit	Description

7 (msb)	Direction
	   • 0 for Host → Device
	   • 1 for Device → Host

6	Shall be set to 0
5	Shall be set to 0

4	Chaining
	   • 0: no chaining – this block is the only one, or the last one in a sequence
	   • 1: chaining enabled – more block(s) to come

3 **2** **1** **0** (lsb)	Shall be set to 0000

b. H-Block

Bit	Description

7 (msb)	Direction
	   • 0 for Host → Device
	   • 1 for Device → Host

6	Shall be set to 1

5
4
	Block type:
	   • $_b$00: HELO (Device's "hello" block)
	   • $_b$01: HELO-OK (Host's "hello" acknowledge)
	   • $_b$10: RFU, do not use
	   • $_b$11: HELO-AUTH (see § 5.3)

3 **2** **1** **0** (lsb)	Protocol Version for HELO block Key Number for the first HELO-AUTH block 0000 for the other blocks.

c. Protocol Version

The Device sets this field to $_b$0000. Any other value shall be interpreted by the Host as an error.

4.3. GENERAL COMMUNICATION FLOW

4.3.1. Session establishment

The Host tries to connect to one (or many) Device.

When a connection is established on the Device, the Device sends a HELO block. The payload of the HELO block is the Device's MAC address on 6 bytes.

HELO block (Device → Host)

LENGTH	TYPE	PAYLOAD
$_h$08	$_h$C0	Device's MAC address on 6 bytes

The Host may check that the claimed MAC address is coherent with its records.

The Host may check that the Device's Protocol Version is acceptable.

If everything is OK, the Host sends a HELO-OK block. The payload of the HELO-OK block is empty.

HELO-OK block (Host → Device)

LENGTH	TYPE	PAYLOAD
$_h$02	$_h$50	empty

4.3.2. Nominal dialogue

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

The Host sends I-Blocks to transmit its commands or to query the Device. An empty I-Block denotes a Keep Alive request.

The Device sends I-Block to transmit its notifications or its answers. An empty I-Block denotes a Keep Alive response (when no other data is available).

4.3.3. Timings

The Device ensures that it answer to every block coming from the Host by a response block within 2.5s. The Host may use a 3s-timeout to watch-out the Device. This is also applicable to the HELO frame that is sent by the Device immediately when the connection is opened.

The Device expects to receive a block from the Host at least every 60s.

4.3.4. Chaining

If the application data buffer is longer than the max size for the PAYLOAD field, the data shall be divided onto multiple I-Blocks. In this case, the Chaining bit is set to 1 for every I-Block but the last one.

Chaining is not implemented in the current version of the Device's firmware. The Host shall not use this feature (and the Device will not use it).

4.4. ERROR HANDLING AND RECOVERY

4.4.1. For the Device

■ **Bad sequence during session establishment:** is the Device receives a frame before having transmitted its HELO, the Device drops the connection,

■ **Protocol error:** if the Device receives an invalid block from the Host (LENGTH not coherent with actual length, or unallowed value for TYPE), the Device drops the connection,

■ **No more activity error:** if the Host remains silent for 60s, the Device drops the connection.

4.4.2. For the Host

■ **Bad sequence during session establishment:** is the first frame received by Host is not a valid HELO, or the Host receives another frame before having transmitted its HELO-OK, the Host shall drop the connection,

■ **Protocol error:** if the Host receives an invalid block from a Device (LENGTH not coherent with actual length, or unallowed value for TYPE), the Host shall drop the connection,

■ **Timeout error:** if the Device doesn't answer within 3s, the Host shall drop the connection.

4.4.3. Recovery

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

4.5. APPL LEVEL

Chapter 6 contains the Application Level Protocol. The application layer frames are conveyed within I-Blocks.

5. SPRINGCARD DEVICE C/S PROTOCOL – SECURE MODE

5.1. ABSTRACT

SpringCard Device C/S Protocol is a light-weight, bandwidth-efficient network protocol. The Device is a TCP Server, and the Host (access control unit or computer in charge) is the Client.

This chapter describes the **Secure Transport Layer**. In this mode,

- The Device and the Host perform a **3-pass mutual authentication** to prove each other that they share the very same authentication key (one of the 2 Device's secret key). In the same time, the 3-pass authentication establishes a one-time, random session key – that remains also a secret only shared by both peers,

- The blocks conveyed between the two partners are **ciphered and authenticated**, i.e. their content remains undisclosed, and a defrauder could not insert its own packets into the sequence without being noticed.

The Device has 2 secret keys. Both keys are defined in the IPS Configuration Register ($_h$83, § 8.3.3). The Host choose either key when asking for authentication, depending on the actions it wants to perform onto the Device:

- The **Operation Key** gives access to the basic operations of the Device (§ 6.4). This is the key an access control unit would use to operate the Device,

- The **Administration Key** makes it possible to change the Device's configuration (§ 6.5). This key would typically be used by a configuration software, when installing the Device.

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

*The **SpringCard Device** listens on TCP port 3999. This default value could be changed by writing into the IPP configuration register ($_h$81, § 8.3.2).*
To enforce security, remember to disable the Plain Transport Layer by setting bit 0 of byte 0 to 1 in the IPS configuration register ($_h$82, § 8.3.3).

After the initial session-establishment, the Host (Client) and the Device (Server) exchange only I$_S$-Blocks (§ 5.4.4.a). **The I$_S$-Blocks convey the Application Level Datagrams defined in chapter 6**.

5.2. CRYPTOGRAPHIC BACKGROUND

The Device uses the **AES block cipher** (Rijndael) . AES has a fixed 128-bit (16 bytes) block size. The Device supports **128-bit keys** (16 bytes) only.

In the following paragraphs,

- E (K , P) means "AES encrypt operation (cipher) over block P using the key K",
- D (K , C) means "AES decrypt operation (decipher) over block C using the key K".

Note that the size of blocks P and C must exactly 16 bytes.

When more than one block are involved, the encrypt and decrypt operations are performed **in CBC (cipher block chaining) mode**.

In the following paragraphs,

- E_{CBC} (K , V , P) means "AES encrypt operation (cipher) over buffer P using the key K and the Init Vector V",
- D_{CBC} (K , V , C) means "AES decrypt operation (decipher) over buffer C using the key K and the Init Vector V".

Note that the size of buffers P and C must be a multiple of 16 bytes. As a consequence, a padding is generally involved.

5.3. 3-PASS AUTHENTICATION

The 3-pass authentication is initiated by the Host after receiving the HELO frame from the Device (§ 4.3.1)

5.3.1. Device's HELO

HELO block (Device → Host)

LENGTH	TYPE	PAYLOAD
$_h$08	$_h$C0	Device's MAC address on 6 bytes

The HELO block contains the Device's MAC address. This makes it possible for the Host

1. To check this Device is the expected once (table IP address ←→ MAC address)
2. To select this Device's secret key.

5.3.2. Host's HELO-Auth

The Host asks the Device to open a secure session by sending an HELO-Auth block. The payload of the HELO-Auth block is empty. The low-order bit of the TYPE byte selects the key

HELO-Auth block (Host → Device) selecting Operation Key

LENGTH	TYPE	PAYLOAD
$_h$02	$_h$71	empty

HELO-Auth block (Host → Device) selecting Administration Key

LENGTH	TYPE	PAYLOAD
$_h$02	$_h$72	empty

5.3.3.   Authentication, step 1

After receiving the HELO-Auth block from the Host,

- ■ The Device activates the selected **secret key K$_{AUTH}$,**

- ■ The Device generate a random challenge (C$_R$) on 16 bytes,

- ■ The Device sends to the Host a block containing E (K$_{AUTH}$, C$_R$).

Authentication, step 1: block Device → Host

LENGTH	TYPE	PAYLOAD
$_h$12	$_h$F0	E (K$_{AUTH}$, C$_R$) on 16 bytes

The Init Vector of the AES cipher is cleared to (00..00) before computing E (K$_{AUTH}$, C$_R$).
1 block is crypted, no padding is applied.

5.3.4.   Authentication, step 2

- ■ The Host activates the secret key K$_{AUTH}$,

- ■ The Host decrypts the payload received from the Device, and retrieves C$_R$,

- ■ The Host computes C$_R$' = C$_R$ << 1 || C$_R$ >> 127 (shift left with carry),

- ■ The Host generate a random challenge (C$_H$) on 16 bytes,

- ■ The Host sends to the Device a block containing E (K$_{Auth}$, C$_H$ || C$_R$'),

Authentication, step 2: block Host → Device

LENGTH	TYPE	PAYLOAD
$_h$22	$_h$70	E (K$_{AUTH}$, C$_H$ || C$_R$') on 32 bytes

The Init Vector of the AES cipher is cleared to (00..00) before computing E (K$_{AUTH}$, C$_H$ || C$_R$').
2 block are crypted in CBC mode, no padding is applied.

5.3.5. Authentication, step 3

- The Device decrypts the payload received from the Host, and retrieves C_H and C_R',

- The Device checks that C_R' is valid. This is the proof that the Host knows the secret key,

- The Device computes $C_H' = C_H << 1 \,||\, C_H >> 127$ (shift left with carry),

- The Device sends to the Host a block containing $E(K_S, C_H')$,

Authentication, step 3: block Device → Host

LENGTH	TYPE	PAYLOAD
$_h12$	$_hF0$	$E(K_{AUTH}, C_H')$ on 16 bytes

The Init Vector of the AES cipher is cleared to (00..00) before computing $E(K_{AUTH}, C_H')$.
1 block is crypted, no padding is applied.

5.3.6. Conclusion of the Authentication sequence – Host's HELO-OK

- The Host deciphers the payload received from the Device, and retrieves C_H',

- The Host checks that C_H' is valid. This is the proof that the Device knows the secret key,

- The Host generates a random nounce (N_H) on 16 bytes

- The Host sends to the Device a HELO-OK block containing $E(K_{SESS}, N_H)$

HELO-OK block (Host → Device)

LENGTH	TYPE	PAYLOAD
$_h22$	$_h50$	$E(K_{SESS}, N_H \,||\, CMAC \,||\, PADD)$ on 32 bytes

The CMAC is computed as specified in 5.6.1 . The sequence number of the CMAC is cleared to 0 before computing the CMAC. The initial size of the payload (before CMAC and Padding) is $_h10$ (size of N_H).
After CMAC, the size of the payload is $_h18$ (size of CMAC is 8 bytes).
The Padding is applied as specified in 5.6.2.b to reach 32 bytes of payload. The final Length of the packet is therefore $_h22$ (2-byte header + 32-byte payload).
The Init Vector of the AES cipher is cleared to (00..00) before computing $E(K_{SESS}, ...)$. 2 blocks are crypted in CBC mode as specified in 5.6.2.c.

5.4. PRESENTATION LAYER AFTER AUTHENTICATION

5.4.1. Block format

Every block transmitted in the channel is formatted as follow:

LENGTH	TYPE	CIPHERED PAYLOAD		
		PAYLOAD	CMAC	PADDING
1 byte	1 byte	Variable length	8 bytes	Variable length

5.4.2. Description of the fields

Field	Description
LENGTH	The LENGTH byte is the total length of the block, this byte included.
TYPE	The TYPE byte is used to convey the information required to control the data transmission. After authentication, only I_S-Blocks could be transmitted
PAYLOAD	The PAYLOAD field is optional. When present, the PAYLOAD field conveys application data.
CMAC	The CMAC field is computed over the initial PAYLOAD field and the TYPE and SEQUENCE fields, as specified in 5.6.1 .
PADDING	The cipher algorithm uses fixed-size blocks. Therefore a PADDING shall be applied to ensure that the size of content to be ciphered is a multiple of the cipher's block size. The PADDING is specified in 5.6.2 .
CIPHERED PAYLOAD	After addition of the CMAC and PADDING field, the whole PAYLOAD is ciphered (encrypted) as specified in 5.6.2 .

5.4.3. Size of the blocks

If the application layer needs to transmit more than 64 bytes, chaining shall be used.

With a PAYLOAD between 0 and 64 bytes, the total size of every block is between 18 and 82.

5.4.4. Format of the TYPE byte

a. I_S-Block

Bit	Description
7 (msb)	Direction • 0 for Host → Device • 1 for Device → Host
6	Shall be set to 0
5	Shall be set to 1
4	Chaining • 0: no chaining – this block is the only one, or the last one in a sequence • 1: chaining enabled – more block(s) to come
3 **2** **1** **0** (lsb)	Shall be set to 0000

5.5. SESSION KEYS

In order to secure the communication, two session keys are derived from the two random challenges exchanged during the authentication:

- **K_{SESS} is the session Encryption Key**, used to ensure confidentiality over the TCP channel,

- **K_{CMAC} is the session CMAC Key**, used to ensure confidence over the TCP channel.

5.5.1. Session Encryption Key

Let C_R be the Device's random challenge (§ 5.3.3). C_R is a 16-byte value ($C_R[0]$ ... $C_R[15]$).

Let C_H be the Host's random challenge (§ 5.3.4). C_H is a 16-byte value ($C_H[0]$ ... $C_H[15]$).

Let K_{AUTH} be the key used for authentication.

Construct T, a 16-byte buffer, as follow:

- $T[0] = C_H[11]$
- $T[1] = C_H[12]$
- $T[2] = C_H[13]$
- $T[3] = C_H[14]$
- $T[4] = C_H[15]$
- $T[5] = C_R[11]$
- $T[6] = C_R[12]$
- $T[7] = C_R[13]$

- $T[8] = C_R[14]$
- $T[9] = C_R[15]$
- $T[10] = C_H[4] \oplus C_R[4]$
- $T[11] = C_H[5] \oplus C_R[5]$
- $T[12] = C_H[6] \oplus C_R[6]$
- $T[13] = C_H[7] \oplus C_R[7]$
- $T[14] = C_H[8] \oplus C_R[8]$
- $T[15] = {}_h11$

Compute $K_{SESS} = E (K_{AUTH}, T)$.

5.5.2. Session CMAC Key

Let C_R be the Device's random challenge (§ 5.3.3). C_R is a 16-byte value ($C_R[0]$ ... $C_R[15]$).

Let C_H be the Host's random challenge (§ 5.3.4). C_H is a 16-byte value ($C_H[0]$ ... $C_H[15]$).

Let K_{AUTH} be the key used for authentication.

Construct T, a 16-byte buffer, as follow:

- $T[0] = C_H[7]$
- $T[1] = C_H[8]$
- $T[2] = C_H[9]$
- $T[3] = C_H[10]$
- $T[4] = C_H[11]$
- $T[5] = C_R[7]$
- $T[6] = C_R[8]$
- $T[7] = C_R[9]$

- $T[8] = C_R[10]$
- $T[9] = C_R[11]$
- $T[10] = C_H[0] \oplus C_R[0]$
- $T[11] = C_H[1] \oplus C_R[1]$
- $T[12] = C_H[2] \oplus C_R[2]$
- $T[13] = C_H[3] \oplus C_R[3]$
- $T[14] = C_H[4] \oplus C_R[4]$
- $T[15] = {}_h22$

Compute $K_{CMAC} = E (K_{AUTH}, T)$.

5.6. SECURE COMMUNICATION CHANNEL

After the authentication, the communication is secured by the combination of:

- A **8-byte CMAC** computed over the plain-text using K_{CMAC},

- The **AES-CBC encryption** of the plain-text plus the CMAC using K_{SESS},

- The synchronisation of a **Sequence number** and the synchronisation of the **Init Vectors (IV)** between sender and receiver, to prevent any kind of injection.

5.6.1. CMAC

a. Sequence numbers

The calculation of the CMAC includes a Sequence number, to protect against the injection or the removal of frames. Both the Device and the Host shall maintain 2 Sequence numbers for the CMAC:

- **SEQ_H is used by the Host to compute its outgoing CMAC**, and by the Device to verify the its incoming CMAC. SEQ_H is incremented every time the Hosts sends a frame.

- **SEQ_R is used by the Device to compute its outgoing CMAC**, and by the Host to verify the its incoming CMAC. SEQ_R is incremented every time the Hosts sends a frame.

Both SEQ_H and SEQ_R are cleared at the end of the authentication (§ 5.3.6) and evolve independently afterwards.

b. Computing the CMAC

Let P be the (plain) payload of the packet. P is an arbitrary-length buffer.

Construct H, an 8-byte buffer, as follow:

- H[0] to H[3] = SEQ_H or SEQ_R (sequence number of the sender), expressed in MSB-first format

- H[4] = TYPE of the packet (see § 5.4.2)

- H[5] = size of P (see § 5.4.2)

- H[6] = $_h$FF $\oplus$ TYPE

- H[7] = $_h$FF $\oplus$ LENGTH

Construct T = H || P

If size of T is not a multiple of 16 bytes, padd T as follow:

- T = T || $_h$80

- While size of T is not a multiple of 16 bytes, T = T || $_h$00

Compute C = E_{CBC} (K_{CMAC} , $_h$00 .. $_h$00 , T)

(T encrypted in CBC mode, using K_{CMAC} and an all-zero Init Vector)

Keep C_{LAST} , the last 16 bytes of C.

Extract CMAC, a 8-byte buffer, as follow:

- CMAC[0] = C_{LAST}[0]
- CMAC[1] = C_{LAST}[2]
- CMAC[2] = C_{LAST}[4]
- CMAC[3] = C_{LAST}[6]

- CMAC[4] = C_{LAST}[8]
- CMAC[5] = C_{LAST}[10]
- CMAC[6] = C_{LAST}[12]
- CMAC[7] = C_{LAST}[14]

c. New payload

The AES-CBC encryption will now be applied over P' = P || CMAC

5.6.2. AES-CBC encryption

a. Init Vectors

All operation are performed in CBC mode. The Init Vector is preserved among all operations on both sides. Both the Device and the Host shall maintain 2 Init Vectors for the encryption/decryption:

- **IV_H is used by the Host to send** (encrypt), and by the Device to receive (decrypt),

- **IV_R is used by the Device to send** (encrypt), and by the Device to receive (decrypt).

When receiving the HELO-OK block (§ 5.3.6), the Device decrypts the received cryptogram after starting from a clear IV (00..00). As a consequence, the Device's IV_H becomes synchronised with the Host's IV_H . At this step, both **the Device and the Host copy IV_H into IV_R.**

Afterwards, IV_H into IV_R will evolve independently.

b. Padding

The AES-CBC encryption could be performed only if the size of the text is a multiple of 16 bytes. A padding is always applied.

Let P' be the packet's payload (P' = P || CMAC as per § 5.6.1.c).

Compute p = 16 - (sizeof (P') [16])

(p is 16 minus the reminder of the size of P' in the division by 16)

If p = 0, then set p = 16.

Construct T, a p-byte buffer, whose every byte value is b:

- T[0] = p
- T[1] = p
- …
- T[p-1] = p

Set P'' = P' || T

Size of P'' is now a multiple of 16 bytes.

c. Encryption

Compute C = E_{CBC} (K_{SESS} , IV_H or IV_R , P'')

(P'' encrypted in CBC mode, using K_{SESS} and using either the Init Vector of the sender)

The final IS-Block packet (§ 5.4.4.a) is then

- LENGTH = 2 + sizeof (C)
- TYPE = I_S-Block
- Actual payload = C

5.6.3. Receiving

When receiving a I_S-Block packet, the receiver must follow the reverse path:

1. Check that LENGTH and TYPE are valid

2. Check that the size of the packet's content (C) is multiple of 16

3. Retrieve P'' = D_{CBC} (K_{SESS} , IV , C)

   (remember to use the Init Vector of the sender)

4. Check the padding, suppress the padding to recover P' from P''

5. Extract P and CMAC from P', verify the CMAC using K_{CMAC} and the sequence number of the sender

5.7. NEW AUTHENTICATION – GENERATION OF A NEW SESSION KEY

The Host may require a new authentication at any time, by sending a new HELO-Auth block as specified in § 5.3.2 .

5.8. GENERAL COMMUNICATION FLOW

5.8.1. Nominal dialog

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

The Host sends I_S-Blocks to transmit its commands or to query the Device. An empty I_S-Block denotes a Keep Alive request.

The Device sends I-Block to transmit its notifications or its answers. An empty I_S-Block denotes a Keep Alive response (when no other data is available).

5.8.2. Timings

The Device ensures that it answer to every block coming from the Host by a response block within 2.5s. The Host may use a 3s-timeout to watch-out the Device. This is also applicable to the HELO frame that is sent by the Device immediately when the connection is opened.

The Device expects to receive a block from the Host at least every 60s.

5.8.3. Chaining

If the application data buffer is longer than the max size for the PAYLOAD field, the data shall be divided onto multiple I_S-Blocks.

In this case,

- The Chaining bit is set to 1 for every I_S-Block but the last one,
- Only the first I_S-Block contains the SEQUENCE field,
- Only the last I_S-Block contains the CRC32 and PADDING fields.

5.9. ERROR HANDLING AND RECOVERY

5.9.1. For the Device

- **Bad sequence during session establishment:** is the Device receives a frame before having transmitted its HELO, the Device drops the connection,

- **Protocol error:** if the Device receives an invalid block from the Host (LENGTH not coherent with actual length, or unallowed value for TYPE), the Device drops the connection,

- **No more activity error:** if the Host remains silent for 60s, the Device drops the connection.

5.9.2. For the Host

- **Bad sequence during session establishment:** is the first frame received by Host is not a valid HELO, or the Host receives another frame before having transmitted its HELO-OK, the Host shall drop the connection,

- **Protocol error:** if the Host receives an invalid block from a Device (LENGTH not coherent with actual length, or unallowed value for TYPE), the Host shall drop the connection,

- **Timeout error:** if the Device doesn't answer within 3s, the Host shall drop the connection.

5.9.3. Recovery

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

5.10. APPLICATION LAYER

Chapter 6 contains the application layer protocol. The application layer frames are conveyed within I_S-Blocks.

6. SPRINGCARD NETWORK DEVICE C/S PROTOCOL – APPLICATION LAYER

6.1. PRINCIPLES

SpringCard Device C/S Protocol is a light-weight, bandwidth-efficient network protocol, that makes it possible for a low-end access control unit to be the client of numerous Devices, each Device being a TCP server.

This chapter describes the **Application Layer**, whose Application-Level Datagrams are transmitted in either **Plain** Transport Datagrams (§ 4) or **Secure** Transport Datagrams (§ 5).

*The **SpringCard Device C/S Protocol** is not a Request/Response Protocol; since a TCP channel is truly full-duplex, both the Host and the Device may talk at any time. Therefore, the Host must be ready to process (or at least to queue) an Application-Level Datagram coming from the Device at any time.*

6.2. FORMAT OF THE APPLICATION LEVEL DATAGRAM UNITS

The **Application Level Datagrams** are obeys to a T,L,V scheme:

- **T (Tag):** this is the operation-code of a command, or the identifier of a data field. The Tag is on either 1 or 2 bytes,

- **L (Length)**: this is the length of the following Value, on 1 byte. Allowed values are $_h00$ to $_h7F$,

- **V (Value):** the parameters to the command, or the data field itself. The length is specified by L, from 0 to 127 bytes.

6.3. LIST OF OPERATION-CODES AND DATA-FIELD IDENTIFIERS

6.3.1. Operation-codes (Host → Device)

T (Tag)	Operation	See §
$_h$00	Get Global Status	6.4.1
$_h$01	Get Device Name	6.4.2
$_h$02	Get Device Capabilities	6.4.3
$_h$03	Get Device Serial Number	6.4.4
$_h$04	Read Inputs	6.4.5
$_h$0A	Start / Stop Reader	6.4.6
$_h$90xx	Set Output	6.4.7
$_h$A0xx	Clear Output	6.4.8
$_h$D000	Clear LEDs Set LEDs Start LEDs	6.4.9 6.4.10 6.4.11
$_h$D100	Buzzer	6.4.12
Restricted operations (available only after authentication using Administration Key)		
$_h$0C	Write Configuration Erase Configuration Reset the Device (to apply the Configuration)	6.5.1 6.5.2 6.5.3

6.3.2. Data-field identifiers (Device → Host)

T (Tag)	Operation	See §
$_h$01	Device Name	6.6.1
$_h$02	Device Capabilities	6.6.2
$_h$03	Device Serial Number	6.6.3
$_h$8100	Reader Name	6.6.4
$_h$2F	Tamper Status	6.6.5
$_h$B000	Card Read	6.6.6
$_h$B100	Card Inserted Card Removed	6.6.7 6.6.8
$_h$C0xx	Input Changed	6.6.9

6.4. HOST → DEVICE, BASIC OPERATIONS

The operations listed in this chapter are available **whatever the mode**:

- Plain (no authentication),

- Secure, after authentication using the Operation Key,

- Secure, after authentication using the Administration Key.

6.4.1. Get Global Status

T	L
$_h00$	$_h00$

The Device answers by a sequence of messages:

1. **Reading Head Identifier** (§ 6.6.4) if the Device is a Reader (all RDR),

2. **Tamper Status** (§ 6.6.5) if the Device has Tampers (FunkyGate only).

6.4.2. Get Device Name

T	L
$_h01$	$_h00$

The Device answers by sending the **Device Name** message (§ 6.6.1).

6.4.3. Get Device Capabilities

T	L
$_h02$	$_h00$

The Device answers by sending the **Device Capabilities** message (§ 6.6.2).

6.4.4. Get Device Serial Number

T	L
$_h$03	$_h$00

The Device answers by sending the **Device Serial Number** message (§ 6.6.3).

6.4.5. Read Inputs *(MIO only)*

T	L
$_h$0C	$_h$00

The MIO answers by sending one **Input Changed** message (§ 6.6.9) for every input line it has.

6.4.6. Start/Stop Reader *(RDR only)*

T	L	V
$_h$0A	$_h$01	**mode**

- **mode:** start/stop command
    - $_h$00 Reader goes OFF (RF field OFF, no activity on RF)
    - $_h$01 Reader goes ON

6.4.7.   Set Output command *(MIO only)*

a.   Permanent

The MIO asserts the Output until the **Clear Output** (§ 6.4.8) command is received.

T	L
h90xx	h00

The 'xx' part in the Tag is the number of the Output.

b.   Temporary

The MIO asserts the Output for the specified time (in seconds). If the time is 0s, the Output is asserted for about 100ms.

T	L	V
h90xx	h02	Time-out (s)

The 'xx' part in the Tag is the number of the Output.

6.4.8.   Clear Output command *(MIO only)*

The MIO de-assert the specified Output.

T	L
hA0xx	h00

The 'xx' part in the Tag is the number of the Output.

6.4.9. Clear LEDs command *(RDR only)*

Both LEDs go OFF.

T	L
$_h$D000	$_h$00

6.4.10. Set LEDs command *(RDR only)*

Both LEDs are driven – until a Clear LEDs command is received.

T	L	V	
$_h$D000	$_h$02	red	green

- **red:** command for red LED
    - $_h$00 OFF
    - $_h$01 ON
    - $_h$02 blinks slowly
    - $_h$03 blinks quickly
- **green:** command for green LED
    - $_h$00 OFF
    - $_h$01 ON
    - $_h$02 blinks slowly
    - $_h$03 blinks quickly

6.4.11. Start LED sequence command *(RDR only)*

Both LEDs are driven – until a Clear LEDs command is received or a timeout occurs.

T	L	V		
$_h$D000	$_h$04	red	green	time (sec)

- **red:** same as above,

- **green:** same as above,

- **time:** time (in seconds, MSB-first) before returning to all-LED-OFF state.

6.4.12. Buzzer command *(RDR only)*

T	L	V
$_h$D100	$_h$01	seq.

- **seq:**

    - $_h$00 buzzer OFF,

    - $_h$01 buzzer ON,

    - $_h$02 buzzer short sequence,

    - $_h$03 buzzer long sequence.

6.5. HOST → DEVICE, RESTRICTED OPERATIONS

The operations listed in this chapter are available only in **Secure mode, after authentication using the Administration Key**.

6.5.1. Write Configuration Register

The Device's behaviour is defined by Configuration Registers. The Write Configuration Register command allows to write into any Configuration Register given its address.

<addr> is the Register number on one byte (valid values are $_h00$ to $_hFE$).

T	L	V	
$_h0C$	<var.>	<addr>	<value>

6.5.2. Erase Configuration Register

The Device's behaviour is defined by Configuration Registers. The Erase Configuration Register command allows to delete any Configuration Register given its address. Once a Register is deleted, the default value for this Register is used.

<addr> is the Register number on one byte (valid values are $_h00$ to $_hFE$).

T	L	V
$_h0C$	$_h01$	<addr>

6.5.3. Reset the Device

The Device must be re-setted in order for the new configuration to take effect. When receiving this command, the Device drops the connection and resets.

T	L
$_h0C$	$_h00$

6.6.   DEVICE → HOST

6.6.1.   Device Name

This T,L,V is transmitted in response to the **Get Device Name** command (§ 6.4.2).

a.   For a RDR

T	L	V
$_h$01	$_h$1C	`SpringCard E663/RDR x.xx`

b.   For a MIO

T	L	V
$_h$01	$_h$1C	`SpringCard E663/MIO x.xx`

6.6.2.   Device Capabilities

This T,L,V is transmitted in response to the **Get Device Capabilities** command (§ 6.4.3).

T	L	V		
$_h$02	$_h$03	Number of Reading Heads	Number of Inputs	Number of Outputs

A RDR would return V = $_h$01, $_h$00, $_h$00.

A MIO with 8 input lines and 8 output lines would return V = $_h$00, $_h$08, $_h$08.

6.6.3.   Device Serial Number

This T,L,V is transmitted in response to the **Get Device Serial Number** command (§ 6.4.4).

T	L	V
$_h$03	$_h$06	Serial number (MAC address)

6.6.4.   Reading Head Identifier

This T,L,V is transmitted in response to the **Get Global Status** command, for every Reading Head that is available on the Device.

(Basically, there's only one Reading Head on a RDR device, and none on a MIO device).

a.   For a RFID/NFC Reading Head

T	L	V
$_h$8100	$_h$1C	`SpringCard E663/RDR x.xx`

b.   For a RFID/NFC Reading Head with a pinpad

T	L	V
$_h$8100	$_h$1C	`SpringCard E663/RDR+PIN x.xx`

6.6.5.   Tamper Status

This T,L,V is transmitted in response to the **Get Global Status** command or when one of the tampers is broken/restored.

T	L	V
$_h$2F	$_h$01	Bit field, the broken tampers are denoted by the corresponding bit set to 1. V = $_h$00 when all tampers are OK.

6.6.6.   Card Read *(RDR only)*

This T,L,V is transmitted when the Reader has read a card, if the Insert/Remove mode is disabled.

T	L	V
$_h$B000	<var.>	Card Identifier

6.6.7.   Card Inserted *(RDR only)*

This T,L,V is transmitted when the Reader has read a card, if the Insert/Remove mode is enabled.

T	L	V
$_h$B100	<var.>	Card Identifier

6.6.8.   Card Removed *(RDR only)*

This T,L,V is transmitted when the card is removed, if the Insert/Remove mode is enabled.

T	L
$_h$B100	$_h$00

SPRINGCARD, the SPRINGCARD logo, PRO ACTIVE and the PRO ACTIVE logo are registered trademarks of PRO ACTIVE SAS.
All other brand names, product names, or trademarks belong to their respective holders.
Information in this document is subject to change without notice. Reproduction without written permission of PRO ACTIVE is forbidden.

6.6.9. Input Changed *(MIO only)*

This T,L,V is transmitted when an Input changes.

T	L	V
$_h$C0xx	$_h$01	$_h$00 : Input not asserted $_h$01 : Input asserted

The 'xx' part in the Tag is the number of the Input.

When the Host sends the **Read Inputs** command, the MIO sends one **Input Changed** messages for every Input it has.

7. EDITING DEVICE'S CONFIGURATION

The Device's configuration is stored in a set of non-volatile Configuration Registers. There are two groups of Registers:

■ The Registers that control the IP configuration and the operation on the network are fully documented in this document,

■ The Registers that are specific to the Device family (either **SpringCard FunkyGate** Reader or **SpringCard HandyDrummer** I/O Module) are documented only in the Integration and Configuration Guide of the family**.**

There are four ways to edit the Device's Configuration Registers:

1. Through the Telnet link

2. Using Master Cards (only available if the Device is a Reader)

3. Using the SpringCard Network Device C/S Protocol, after authentication with the Administration Key.

Note that the SEC Configuration Register ($_h$6E, § 8.2) may be used to disable either way to access the Configuration Registers.
Administration Key is defined in the IPS Configuration Register ($_h$83, § 8.3.3)

7.1. THROUGH THE TELNET LINK

Open a Telnet session to the Device as instructed in § 3.1.

7.1.1. Reading Configuration Registers

Enter "cfg" to list all Configuration registers currently defined (registers that are not explicitly defined keep their default value).

Enter "cfgXX" to read the value of the Configuration register $_h$XX.

Note that Configuration registers $_h$55, $_h$56, $_h$6E and $_h$6F that hold sensitive data (the keys used by Master Cards and the Device's secret keys and password) are masked.

7.1.2. Writing Configuration Registers

Enter "cfgXX=YYYY" to update Configuration Register $_h$XX with value $_h$YYYY. YYYY can me any length between 1 and 32 bytes.

Enter "cfgXX=!!" to erase Configuration Register $_h$XX.

7.2. USING MASTER CARDS (ONLY AVAILABLE ON A READER)

The Master Cards are NXP Desfire cards formatted and programmed by **SpringCard Configuration Tool (ScMultiConf.exe, ref # SN14007)** for Windows.

Please refer to this software's documentation for details.

7.3. THROUGH THE SPRINGCARD NETWORK DEVICE C/S PROTOCOL

Please refer to § 6.5.

8. Common configuration registers for SpringCard Network Devices

8.1. General options

The General Options register is completely defined in every product's detailed documentation. Only the common part is described here.

Name	Tag	Description	Size
OPT	$_h60$	General option, see table below	1 or 2

General options bits

Bits	Value	Meaning
Byte 0		
7 - 2		See the detailed documentation of the actual product you are using
1 - 0	00	*Communication mode* The device uses SpringCard Network Device C/S Protocol (chapters 4 and 5)
	01	The device runs in HTTP server mode
	10	The device runs in HTTP client mode
	11	*RFU* HTTP server and HTTP client modes are detailed in the product's documentation
Byte 1 (optional)		
7 - 0		See the detailed documentation of the actual product you are using

Default value: $_b$XXXXXX00 XXXXXXXX

8.2. SECURITY OPTIONS

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

Security option bits

Bits	Value	Meaning	
		Standard network servers	
7	0	Telnet server is disabled	
	1	Telnet server is enabled	
6	0	*RFU (set to 0)*	
5	0	*RFU (set to 0)*	
4	0	SpringField Colorado notifier is disabled	RDR only[1]
	1	SpringField Colorado notifier is enabled	
3	0	*RFU (set to 0)*	
		Tampers	
2	0	Do not signal tamper alarms on buzzer	
	1	Signal tamper alarms on buzzer	
1	0	Reader keeps on reading even if a tamper is broken	
	1	Reader stops reading when a tamper is broken	
0	0	Do not raise alarm if a tamper is broken at power up	
	1	Raise alarm on tamper broken even at power up	

Default value: $_b$10010100

---

[1] SpringField Colorado is a NFC-enabled application running on Android, or embedded in a specific NFC Tag, that retrieves and shows the Reader's data: firmware name and version, serial number, IP address etc.

8.3. TCP CONFIGURATION

8.3.1. IPv4 address, mask, and gateway

Name	Tag	Description	Size
IPA	$_h$80	IPv4 configuration bytes, see table below	4 to 20

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	
12	DNS server 1, 1st byte	Address of 1st DNS server.
13	DNS server 1, 2nd byte	
14	DNS server 1, 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 DNS server.
15	DNS server 1, 4th byte	
16	DNS server 2, 1st byte	Address of 2nd DNS server.
17	DNS server 2, 2nd byte	
18	DNS server 2, 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 DNS server.
19	DNS server 2, 4th byte	

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

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

8.3.2.   SpringCard Network Device C/S Protocol – Server port

Name	Tag	Description	Size
IPP	h81	Listen TCP port for the server (2 bytes, MSB-first)	2

Default value: h0F 9F (server TCP port = 3999)

8.3.3.   SpringCard Network Device C/S Protocol – Security settings and authentication keys

Name	Tag	Description	Size
IPS	h84	Server security settings bits, see table below	1

Security settings bits

Bits	Value	Meaning
7	0	*RFU (set to 0)*
6	0	*RFU (set to 0)*
5	0	*RFU (set to 0)*
4	0	*RFU (set to 0)*
3	0	*RFU (set to 0)*
2	0	The Administration Key is enabled
	1	The Administration Key is disabled
1	0	The Operation Key is enabled
	1	The Operation Key is disabled
0	0	Plain communication is allowed
	1	Secure communication is mandatory

Default value: b00000100

(only Operation Key is enabled, plain communication is allowed)

8.3.4.   SpringCard Network Device C/S Protocol – Operation Key

Name	Tag	Description	Size
IPK.OPE	h85	C/S Protocol Operation Key	16

Default value: h00 … h00

8.3.5.   SpringCard Network Device C/S Protocol – Administration Key

Name	Tag	Description	Size
IPK.ADM	h86	C/S Protocol Administation Key	16

Default value: h00 … h00

8.3.6. Ethernet configuration

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

Ethernet configuration bits

Bits	Value	Meaning
7	0	*RFU (set to 0)*
6	0	*RFU (set to 0)*
5	0	*RFU (set to 0)*
4	0	*RFU (set to 0)*
3	0	*RFU (set to 0)*
2	0	*RFU (set to 0)*
1	0	*RFU (set to 0)*
0	0	Use auto-configuration (10/100Mbps, half or full-duplex)
	1	Force bitrate = 10Mbps, half-duplex

Default value: $_b$00000000

8.3.7. Info / Location

Name	Tag	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 (§ 2.1.3).

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

8.3.8. Password for Telnet access

Name	Tag	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 Device using Telnet protocol.

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

9. 3RD-PARTY LICENSES

SpringCard has been developed using open-source software components.

9.1. FREERTOS

FreeRTOS is a market leading real time operating system (or RTOS) from Real Time Engineers Ltd. **SpringCard** runs on FreeRTOS v7.5.2.

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

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

<div align="center">

www.freertos.org

</div>

9.2. μIP

μIP is an open-source TCP/IP stack initially developed by Adam Dunkels and licensed under a BSD style license.

SpringCard uses FreeTCPIP, a modified version of μIP that comes with FreeRTOS. To comply with the original license of μIP, we have to copy the full text here:

> Copyright (c) 2001-2003, Adam Dunkels.
>
> All rights reserved.
>
> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
>
> 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
>
> 3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SPRINGCARD, the SPRINGCARD logo, PRO ACTIVE and the PRO ACTIVE logo are registered trademarks of PRO ACTIVE SAS.
All other brand names, product names, or trademarks belong to their respective holders.
Information in this document is subject to change without notice. Reproduction without written permission of PRO ACTIVE is forbidden.