

## uFR serial - Communication protocol for uFR series devices

uFR Series devices can establish communication over FTDI's Virtual COM port, so devices are seen as standard COM port hardware.

Communication parameters are :

*Readers with FTDI serial interface:*

**uFR Classic and uFR Advance readers with USB connection:**

Serial communication: 1 Mbps, 8-N-1, Flow control: None;

**uFR BaseHD readers with “uFR support” firmware installed (ex. XR and uFR XRc readers):**

Serial communication (using VCOM FTDI driver): 250 kbps, 8-N-1, Flow control: None;

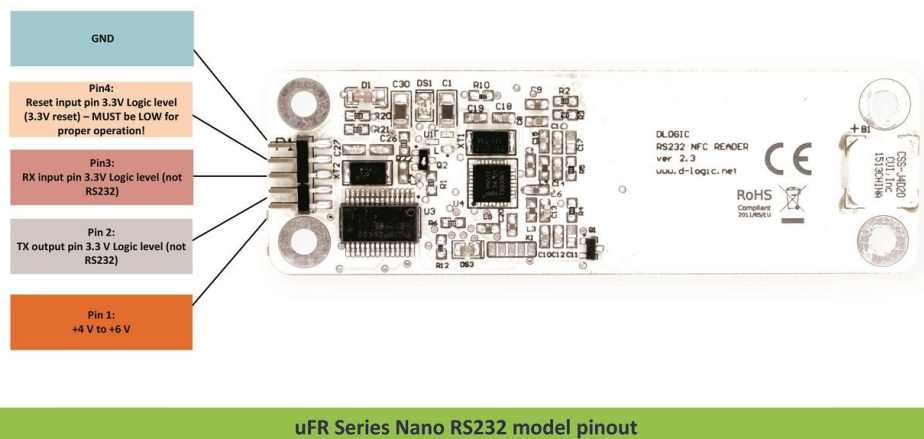
*Readers without FTDI serial interface:*

**RS485 (connection without USB/RS-485 converter):** variable baudrate can be set through software tool. Current baud rate must be known when changing baudrate. Default baudrate is 250 kbps.

**uFR Classic Nano and Card Size:**

UART / TTL: 115200 bps, 8-N-1, Flow control: None

**Pinout for UART / TTL model is presented below:**



For communication purposes between reader devices and host PC, D-Logic's proprietary protocol called "uFR serial" is created.

All communication is initiated by the host (PC or other platform) to which the device is connected.

Maximum data transferred by single command or received by one device response, from firmware version 3.9.44 is 256 bytes, and before is 192 bytes.

Generally, there are two types of packets:

**CMD** – command sent by host to device

**ANS** – answer sent from device to host

CMD can be short or long set. CMD short set is always 7 byte long while CMD long set – called CMD\_EXT can have variable length.

Answer have following types:

**ACK** – Acknowledgment, everything is OK, device is waiting for next CMD or CMD EXT

**ERR** – Error occurred, error byte defines ERR\_TYPE

**RSP** – Response from device on CMD or CMD\_EXT

Communication constants bytes defines type of packet, which can be seen in first three bytes of each packet.

First byte of each packet is HEADER byte. Second byte is always CMD\_CODE. Third byte is TRAILER byte.

<b>CMD_HEADER</b>	<b>0x55</b>	<b>CMD_TRAILER</b>	<b>0xAA</b>
<b>ACK_HEADER</b>	<b>0xAC</b>	<b>ACK_TRAILER</b>	<b>0xCA</b>
<b>RESPONSE_HEADER</b>	<b>0xDE</b>	<b>RESPONSE_TRAILER</b>	<b>0xED</b>
<b>ERR_HEADER</b>	<b>0xEC</b>	<b>ERR_TRAILER</b>	<b>0xCE</b>

## **CHECKSUM**

All checksums in this document are calculated in the same manner: row of bytes is used for checksum calculation, each byte is XOR-ed with next one until the end of row. Final value is incremented with 0x07.

For example, CMD packet has 7 bytes, where 7<sup>th</sup> byte is checksum of previous 6 bytes:

**CHECKSUM = (Byte1 XOR Byte2 XOR Byte3 XOR Byte4 XOR Byte5 XOR Byte6) + 0x07**

**CMD codes**

Each command has its corresponding value - look at [COMMANDS OVERVIEW](#).

**Error codes**

If error occurs, device will answer with ERR packet. Each Error has its corresponding value which can be found in table in [Appendix: ERROR CODES](#).

**CMD packet**

CMD packet can be short – 7 byte long or EXT-ended with variable length. In case of EXT CMD packet, fourth byte of CMD packet is greater than 0, containing integer value – length of CMD\_EXT packet. When issuing CMD\_EXT, always main CMD 7-byte long packet goes first. If everything as expected, device will answer with ACK packet, waiting for CMD\_EXT packet. On error, device will answer with ERR packet. CMD\_EXT consists of various different parameters, depending on command type, so CMD\_EXT does not have fixed length and order of parameters.

CMD packet has following structure:

Mandatory 7 byte CMD packet structure						
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CMD_HEADER	CMD_CODE	CMD_TRAILER	CMD_EXT_Length	CMD_Par0	CMD_Par1	CHECKSUM

**Byte 1:** CMD\_HEADER as defined in Table1.Communication constants, 0x55

**Byte 2:** CMD\_CODE as defined in Table2. CMD\_CODE values

**Byte 3:** CMD\_TRAILER as defined in Table1.Communication constants, 0xAA

**Byte 4:** CMD\_EXT\_Length: If 0 than the “CMD EXT” is not used); ELSE value is length of whole CMD\_EXT packet

**Byte 5:** CMD\_Par0: command parameter0, takes different values depending on command

**Byte 6:** CMD\_Par1: command parameter1, takes different values depending on command

**Byte 7:** CHECKSUM – Checksum of Bytes 1 to 6 as explained above

CMD\_EXT packet has following structure:

CMD_EXT packet structure			
Byte 1	..	Byte N	Byte N+1
Parameter bytes 1 to N			CMD_EXT_CHECKSUM

Parameter bytes 1 to N – different parameters, values depends on type of command

**CMD\_EXT\_CHECKSUM** - Checksum of bytes 1 to N

**CMD\_EXT\_Length** is number of all bytes including CMD\_EXT\_CHECKSUM; e.g. length is N+1

### *ANSWER packet types*

The device can answer with following packet types:

#### **ACK – Acknowledgment packet**

If command and CMD packet are properly configured (structure and checksum) and additional CMD\_EXT packet needs to be sent, device will answer with ACK packet.

#### **ERR – Error packet**

If error occurred, device will answer with ERR packet. Some commands can return ERR\_EXT set. In that case ERR\_EXT packet comes immediately after ERR packet.

#### **RSP – Response packet**

If properly configured CMD or CMD\_EXT packet is sent, device will answer with RSP or RSP\_EXT packet, which depends on command issued. For examples, if CMD needs answer which is short enough for RSP packet, there will be no RSP\_EXT packet. Otherwise, if CMD or CMD\_EXT needs answer with more bytes, RSP\_EXT will come immediately after RSP packet. Common situation is when reading data with LinearRead command, where device will answer with row of card data bytes.

### *ACK – Acknowledgment packet*

ACK packet has following structure:

ACP packet structure						
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
ACK_HEADER	CMD_CODE	CMD_TRAILER	Irrelevant, not used in ACK packet			CHECKSUM

**Byte 1:** ACK\_HEADER as defined in Table1.Communication constants, 0x55

**Byte 2:** CMD\_CODE as defined in Table2. CMD\_CODE values. Device ACK-knowledge that previous

command is properly sent

**Byte 3:** ACK\_HEADER as defined in Table1.Communication constants, 0x55

**Byte 4, Byte 5, Byte 6:** Not used in ACK packet, values are 0x00

**Byte 7:** CHECKSUM – Checksum of Bytes 1 to 6 as explained above

*ERR – error packet*

ERR packet has following structure:

Mandatory 7 byte ERR						
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
ERR_HEADER	ERROR_CODE	ERR_TRAILER	ERR_EXT length	Err_Val0	Err_Val1	CHECKSUM

**Byte 1:** ERR\_HEADER as defined in Table1.Communication constants, 0xEC

**Byte 2:** ERROR\_CODE as defined in Table3. ERROR CODES.

**Byte 3:** ERR\_TRAILER as defined in Table1.Communication constants, 0xCE

**Byte 4:** If ERR\_EXT exists, this byte contains length of ERR\_EXT packet (including ERR\_EXT checksum)

**Byte 5:** Possible additional info on error can be defined in ERR\_Val0

**Byte 6:** Possible additional info on error can be defined in ERR\_Val1

**Byte 7:** CHECKSUM – Checksum of Bytes 1 to 6 as explained above

ERR\_EXT and has following structure:

ERR_EXT packet structure			
Byte 1	..	Byte N	Byte N+1
Error bytes 1 to N			ERR_EXT_CHECKSUM

**Byte 1:** First Byte of ERR\_EXT

...

**Byte N:** N-nth Byte of ERR\_EXT

**Byte N+1:** ERR\_EXT\_CHECKSUM, checksum of Bytes 1 to N, calculated as explained earlier.

*RSP – response packet*

RSP packet has following structure:

Mandatory 7 byte RSP						
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RSP_HEADER	CMD_CODE	RSP_TRAILER	RSP_EXT length	RSP_Val0	RSP_Val1	CHECKSUM

**Byte 1:** RSP\_HEADER as defined in Table1.Communication constants, 0xED

**Byte 2:** CMD\_CODE as defined in Table2. CMD\_CODE values

**Byte 3:** ERR\_TRAILER as defined in Table1.Communication constants, 0xDE

**Byte 4:** If RSP\_EXT exists, this byte contains length of RSP\_EXT packet (including RSP\_EXT checksum)

**Byte 5:** Possible additional info on RESPONSE can be defined in RSP\_Val0

**Byte 6:** Possible additional info on RESPONSE can be defined in RSP\_Val1

**Byte 7:** CHECKSUM – Checksum of Bytes 1 to 6 as explained above

RSP_EXT packet structure			
Byte 1	..	Byte N	Byte N+1
RSP bytes 1 to N			RSP_EXT_CHECKSUM

**Byte 1:** First Byte of RSP\_EXT

...

**Byte N:** N-nth Byte of RSP\_EXT

**Byte N+1:** RSP\_EXT\_CHECKSUM, checksum of Bytes 1 to N, calculated as explained earlier.

## COMMANDS OVERVIEW

Commands are divided into several groups, based on purpose.

### Device related commands

#### General purpose device related commands

GET_READER_TYPE	0x10
GET_READER_SERIAL	0x11
GET_SERIAL_NUMBER	0x40
GET_HARDWARE_VERSION	0x2A
GET_FIRMWARE_VERSION	0x29
GET_BUILD_NUMBER	0x2B
READER_KEY_WRITE	0x12
USER_DATA_READ	0x1B
USER_DATA_WRITE	0x1C
SELF_RESET	0x30

SET_UART_SPEED	0x70
RED_LIGHT_CONTROL	0x71
USER_INTERFACE_SIGNAL	0x26
SET_RF_ANALOG_SETTINGS	0x7D
GET_RF_ANALOG_SETTINGS	0x7E

## Card related commands

### *General purpose card related commands*

GET_CARD_ID	0x13
GET_CARD_ID_EX	0x2C
GET_DLOGIC_CARD_TYPE	0x3C
GET_LAST_CARD_ID_EX	0x7C

### *Trailer block manipulation commands*

SECTOR_TRAILER_WRITE	0x1A
SECTOR_TRAILER_WRITE_UNSAFE	0x2F

### *Block manipulation commands*

BLOCK_READ	0x16
BLOCK_WRITE	0x17
BLOCK_IN_SECTOR_READ	0x18
BLOCK_IN_SECTOR_WRITE	0x19

### *Linear data manipulation commands*

LINEAR_READ	0x14
LINEAR_WRITE	0x15
LINEAR_FORMAT_CARD	0x25
LIN_ROW_READ	0x45

### ***Value block manipulation commands***

#### **Direct block addressing**

VALUE_BLOCK_READ	0x1D
VALUE_BLOCK_WRITE	0x1E
VALUE_BLOCK_INC	0x21
VALUE_BLOCK_DEC	0x22

#### **Indirect block addressing**

VALUE_BLOCK_IN_SECTOR_READ	0x1F
VALUE_BLOCK_IN_SECTOR_WRITE	0x20
VALUE_BLOCK_IN_SECTOR_INC	0x23
VALUE_BLOCK_IN_SECTOR_DEC	0x24

### ***Commands for “asynchronous UID sending” feature***

SET_CARD_ID_SEND_CONF	0x3D
GET_CARD_ID_SEND_CONF	0x3E
SET_BAD_SELECT_NR_MAX	0x3F
GET_BAD_SELECT_NR_MAX	0x44

### ***Power saving commands***

ENTER_SLEEP_MODE	0x46
LEAVE_SLEEP_MODE	0x47
AUTO_SLEEP_SET	0x4D
AUTO_SLEEP_GET	0x4E

### ***Light and display commands***

SET_DISPLAY_DATA	0x72
SET_SPEAKER_FREQUENCY	0x73
SET_DISPLAY_INTENSITY	0x74
GET_DISPLAY_INTENSITY	0x75

### ***uFR BASE Control commands***

UFR_XRC_LOCK_OPEN	0x60
UFR_XRC_SET_RELAY_STATE	0x61
UFR_XRC_GET_IO_STATE	0x62

### ***Shared Ram card emulation commands***

ENTER_SHARE_RAM_COMM_MODE	0x78
EXIT_SHARE_RAM_COMM_MODE	0x79



<b>READ_SHARE_RAM</b>	<b>0x7A</b>
<b>WRITE_SHARE_RAM</b>	<b>0x7B</b>

## **DEVICE RELATED COMMANDS**

### **GENERAL PURPOSE DEVICE RELATED COMMANDS**

#### **GET\_READER\_TYPE (0x10)**

It gives device (reader) type in size of 4 bytes which is hard coded in the firmware.

uFR Classic has value of 0xD1150021.

CMD\_EXT set is not in use.

CMD\_Par0 and CMD\_Par1 are not in use.

If everything operates as expected the RSP packet is sent and after that also the RSP\_EXT packet of 5 bytes which contains 4 byte DeviceType values (little-endian) and CHECKSUM byte.

#### **Example:**

Send CMD GET\_READER\_TYPE  
**55 10 AA 00 00 00 F6**

#### **Where**

55 - CMD\_HEADER  
10 - CMD\_CODE  
AA - CMD\_TRAILER  
00 00 00 - CMD\_EX\_Length and CMD\_Par0 and CMD\_Par1 not used  
F6 - CHECKSUM

Reader answer with RESPONSE – RSP packet followed by RSP\_EXT packet

**DE 10 ED 05 00 00 2D 21 00 15 D1 EC**

#### **Where RSP PACKET contains**

DE - RSP\_HEADER  
10 - CMD\_CODE  
ED - RSP\_TRAILER  
05 - RSP\_EXT\_Length  
00 00 - RSP\_Val0 and RSP\_Val1 not used  
2D - CHECKSUM

#### **and RSP\_EXT contains**

21 00 15 D1 - Device type (currently uFR Classic D1 15 00 21, little-endian notation)  
EC - CHECKSUM

### **GET\_READER\_SERIAL (0x11)**

It gives the device (reader) serial number with length of 4 bytes. On the older devices, this serial number has been read from EEPROM MFRC chip.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

If everything operates as expected the RESPONSE set is sent and after that also the RESPONSE EXT set of 5 bytes which contains 4 byte ReaderSerialNumber values (little-endian) and at the end one checksum byte.

#### **Example:**

Send CMD GET\_READER\_SERIAL  
**55 11 AA 00 00 00 F5**

#### **Where**

55 - CMD\_HEADER  
11 - CMD\_CODE  
AA - CMD\_TRAILER  
00 00 00 - CMD\_EX\_Length and CMD\_Par0 and CMD\_Par1 not used  
F5 - CHECKSUM

Reader answer with RESPONSE – RSP packet followed by RSP\_EXT packet

**DE 11 ED 05 00 00 2E 54 7E 1A 5D 74**

#### **Where RSP PACKET contains**

DE - RSP\_HEADER  
11 - CMD\_CODE  
ED - RSP\_TRAILER  
05 - RSP\_EXT\_Length  
00 00 - RSP\_Val0 and RSP\_Val1 not used  
2E - CHECKSUM

#### **and RSP\_EXT contains**

54 7E 1A 5D - Device type (currently serial is 5D 1A 7E 54, little-endian notation)  
74 - CHECKSUM

### **GET\_SERIAL\_NUMBER (0x40)**

Command returns reader serial number in string representation, like “UF123456”.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

#### **Example:**

Send CMD GET\_SERIAL\_NUMBER  
**55 40 AA 00 AA CC E0**

#### **Where**

55 - CMD\_HEADER  
40 - CMD\_CODE  
AA - CMD\_TRAILER

00 AA CC - CMD\_EX\_Length and CMD\_Par0 and CMD\_Par1 not used  
E0 - CHECKSUM

Reader answer with RESPONSE – RSP packet followed by RSP\_EXT packet

**DE 40 ED 09 00 00 81 55 46 31 32 33 34 35 36 1B**

Where RSP PACKET contains

DE - RSP\_HEADER  
40 - CMD\_CODE  
ED - RSP\_TRAILER  
09 - RSP\_EXT\_Length  
00 00 - RSP\_Val0 and RSP\_Val1 not used  
81 - CHECKSUM

and RSP\_EXT contains

55 46 31 32 33 34 35 36 - Device readers number (currently serial is "UF123456")  
1B - CHECKSUM

### **GET\_HARDWARE\_VERSION (0x2A)**

Returns reader hardware version as two byte representation of higher and lower byte.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

High byte of hardware version is RSP\_Val0.

Low byte of hardware version is PSP\_Val1

**Example:**

<b>CMD</b>	<b>55 2A AA 00 00 00 DC</b>
<b>RSP</b>	<b>DE 2A ED 00 01 01 20</b>

### **GET\_FIRMWARE\_VERSION (0x29)**

Returns reader firmware version as two byte representation of higher and lower byte.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

High byte of firmware version is RSP\_Val0.

Low byte of firmware version is PSP\_Val1.

**Example:**

<b>CMD</b>	<b>55 29 AA 00 00 00 DD</b>
<b>RSP</b>	<b>DE 29 ED 00 03 09 17</b>

### **GET\_BUILD\_NUMBER (0x2B)**

Returns reader firmware build version as one byte representation.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

Build number of firmware version is RSP\_Val0.

**Example:**

```
CMD      55 2B AA 00 00 00 DB
RSP      DE 2B ED 00 C8 00 D7
```

**READER\_KEY\_WRITE (0x12)**

Function writes MIFARE key into internal EEPROM of MFRC531, at key index location (0 – 31).

- CMD\_Par0 is key index
- CMD\_Par1 is not in use
- array from 1st to 6th byte of CMD\_EXT set contains 6-byte key
- 7th byte of CMD\_EXT set is CHECKSUM

**Example:**

Write Key FF FF FF FF FF FF into key index 00

```
CMD      55 12 AA 07 00 00 F1
ACK      AC 12 CA 07 00 00 7A

CMD_EXT  FF FF FF FF FF FF 07
RSP      DE 12 ED 00 00 00 28
```

**USER\_DATA\_READ (0x1B)**

Function gives the 16 bytes from internal EEPROM user space.

The CMD\_Par0 and CMD\_Par1 are not in use.

- array from 1st to 16th byte of RSP\_EXT set contains 16 bytes of user data
- 17th byte of RSP\_EXT set is CHECKSUM.

**Example:**

```
CMD      55 1B AA 00 00 00 EB
RSP      DE 1B ED 11 00 00 40
RSP_EXT  6A 6A 00 00 36 00 00 00 30 00 32 00 38 00 41 00 54
```

**USER\_DATA\_WRITE (0x1C)**

Function writes 16 bytes into user space, which is 16 bytes part of internal EEPROM of MFRC531.

The CMD\_Par0 and CMD\_Par1 are not in use.

- array from 1st to 16th byte of CMD\_EXT set contains 16 bytes of user data
- 17th byte of CMD\_EXT set is CHECKSUM.

**Example:**

write into user space values we read in previous example (6A 6A 00 00 36 00 00 00 30 00 32 00 38 00 41 00 54)

**CMD**            55 1C AA 11 00 00 F9  
**ACK**            AC 1C CA 11 00 00 72

**CMD\_EXT**      6A 6A 00 00 36 00 00 00 30 00 32 00 38 00 41 00 54  
**RSP**            DE 1C ED 00 00 00 36

### **SELF\_RESET (0X30)**

Function performs soft restart of device.  
The CMD\_EXT set is not in use.  
The CMD\_Par0 and CMD\_Par1 are not in use

#### **Example:**

**CMD**            55 30 AA 00 00 00 D6  
  
**RSP**            DE 30 ED 00 00 00 0A  
**RSP\_EXT**      03 55 55 BB

### **SET\_UART\_SPEED (0X70)**

Function writes new value of UART's baud rate. For example 115200. Command sending is at current baud rate, ACK is at current baud rate, but response is at new baud rate. In future, the device will communicate at new baud rate.  
The CMD\_Par0 and CMD\_Par1 are not in use.  
• array from 1st to 4th byte of CMD\_EXT set contains 4 byte long baud rate (little-endian)  
• 5th byte of CMD\_EXT set is CHECKSUM.

#### **Example:**

**CMD**            55 70 AA 05 00 00 91  
**ACK**            AC 70 CA 00 00 00 1D  
**CMD\_EXT**      00 01 C2 00  
**RSP**            ED 70 DE ..... .

### **RED\_LIGHT\_CONTROL (0X71)**

This function turns on or off red LED light. If turned on, green LED will stop flashing.  
The CMD\_EXT set is not in use.  
CMD\_Par0 – 0x01 turn red LED on, 0x00 – turn red LED off.  
CMD\_Par1 is not in use.

#### **Example:**

To turn red LED ON, send CMD packet  
**CMD**            55 71 AA 00 01 00 96  
**RSP**            DE 71 ED 00 00 00 49

To turn red LED OFF, send CMD packet

```
CMD      55 71 AA 00 00 00 95
RSP      DE 71 ED 00 00 00 49
```

### USER\_INTERFACE\_SIGNAL (0x26)

This function turns sound and light reader signals. Sound signals are performed by reader's buzzer and light signals are performed by reader's LEDs.

There are predefined signal values for sound and light:

light_signal_mode:		beep_signal_mode:	
0	None	0	None
1	Long Green	1	Short
2	Long Red	2	Long
3	Alternation	3	Double Short
4	Flash	4	Triple Short
		5	Triplet Melody

The CMD\_EXT set is not in use.

CMD\_Par0 is value of light signal mode (0 - 4)

CMD\_Par1 is value of beep signal mode (0 - 5)

#### Example:

light signal mode is Long Green (1), beep signal mode is Long (2)

```
CMD      55 26 AA 00 01 02 E1
RSP      DE 26 ED 00 00 00 1C
```

### SET\_DISPLAY\_DATA (0x72)

This feature working with LED RING 24 display module.

Function enables sending data to the display. A string of data contains information about the intensity of color in each cell of the display. Each cell has three LED (red, green and blue). For each cell of the three bytes is necessary. The first byte indicates the intensity of the green color, the second byte indicates the intensity of the red color, and the third byte indicates the intensity of blue color. For example, if the display has 16 cells, an array contains 48 bytes. Value of intensity is in range from 0 to 255.

CMD\_Par0 number of bytes

CMD\_Par1 not in use

CMD\_EXT contains data for display with checksum

**Example:**

green = 0, red = 0xFF, blue = 0x80

```
CMD      55 72 AA 49 48 00 93
ACK      AC 72 CA 49 48 00 1C
CMD_EXT  00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80
00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80
00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80 00 FF 80
00 FF 80 07
RSP      DE 72 ED 00 00 00 48
```

**SET\_DISPLAY\_INTENSITY (0x74)**

Function sets the intensity of light on the display. Value of intensity is in range 0 to 100.

CMD\_Par0 is display intensity

CMD\_Par1 not in use

CMD\_EXT not in use

**Example:**

display intensity is 50

```
CMD      55 74 AA 00 32 00 C0
RSP      DE 74 ED 00 00 00 4E
```

**GET\_DISPLAY\_INTENSITY (0x75)**

Function gets the intensity of light on the display.

CMD\_Par0 not in use

CMD\_Par1 not in use

CMD\_EXT not in use

RSP\_EXT 1st byte is intensity, 2nd byte is checksum

**Example:**

```
CMD      55 75 AA 00 00 00 91
RSP      DE 75 ED 02 00 00 4B
RSP_EXT  32 39
```

**SET\_SPEAKER\_FREQUENCY (0x73)**

Function sets the frequency of the speaker. The speaker is working on this frequency until a new frequency setting. To stop the operation set frequency to zero.

Period of sound frequency calculated according to the following formula

$$\text{period} = 65535 - 1500000 / (2 * \text{frequency in Hertz})$$

CMD\_Par0 is low byte of sound's period

CMD\_Par1 is high byte of sound's period

**Example:**

set frequency of 1600Hz

<b>CMD</b>	<b>55</b>	<b>73</b>	<b>AA</b>	<b>00</b>	<b>2B</b>	<b>FE</b>	<b>60</b>
<b>RSP</b>	<b>DE</b>	<b>73</b>	<b>ED</b>	<b>00</b>	<b>00</b>	<b>00</b>	<b>47</b>



**SET\_RF\_ANALOG\_SETTINGS (0x7D)**

This function allows you to adjust the value of several registers on PN512. These are registers: RFCfgReg, RxThresholdReg, GsNOnReg, GsNOffReg, CWGsPReg, ModGsPReg. This can be useful if you want to increase the operation distance of card, or when it is necessary to reduce the impact of environmental disturbances.

CMD\_Par0 type of communication with tag

ISO14443 type A	0x01
ISO14443 type B	0x02
ISO14443-4 212 Kbps	0x03
ISO14443-4 424 Kbps	0x04

CMD\_Par1 0 - user settings, 1 - factory default settings

CMD\_EXT

- 1st byte is value of RFCfgReg
- 2nd byte is value of RxThresholdReg
- 3rd byte is value of GsNOnReg
- 4th byte is value of CWGsPReg
- 5th byte is value of GsNOffReg for Type A or ModGsPReg for type B

For ISO14443-4 212 Kbps and ISO14443-4 424 Kbps CMD\_EXT contains just first 2 bytes

**Example:**

RFCfgReg = 0x79, RxThesholdReg = 0x87, GsNonReg = 0x88, CWGsPReg = 0x20, GsNOffReg = 0x88

```

CMD      55 7D AA 06 01 00 8C
ACK      AC 7D CA 06 01 00 23
CMD_EXT  79 87 88 20 88 E5
RSP      DE 7D ED 00 00 00 55
  
```

**GET\_RF\_ANALOG\_SETTINGS (0x7E)**

The function reads the value of the registers RFCfgReg, RxThresholdReg, GsNOnReg, GsNOffReg, CWGsPReg, ModGsPReg.

CMD\_Par0 type of communication with tag

ISO14443 type A	0x01
ISO14443 type B	0x02
ISO14443-4 212 Kbps	0x03
ISO14443-4 424 Kbps	0x04

The CMD\_EXT set is not in use.

#### RSP\_EXT

- 1st byte is value of RFCfgReg
- 2nd byte is value of RxThresholdReg
- 3rd byte is value of GsNOnReg
- 4th byte is value of CWGsPReg
- 5th byte is value of GsNOffReg for Type A or ModGsPReg for type B

For ISO14443-4 212 Kbps and ISO14443-4 424 Kbps RSP\_EXT contains just first 2 bytes

#### **UFR\_BASE\_HD\_LOCK\_OPEN (0x60)**

BASE HD uFR only.

Electric strike switches when the function called. Pulse duration determined by function.

CMD\_Par0 pulse duration in ms low byte

CMD\_Par1 pulse duration in ms high byte

#### **Example:**

Pulse duration is 300ms (0x12C)

```
CMD      55 60 AA 00 2C 01 B9
RSP      DE 60 ED 00 00 00 5A
```

#### **UFR\_BASE\_HD\_SET\_RELAY\_STATE (0x61)**

BASE HD uFR only.

Function switches relay.

CMD\_Par0 1 - relay on, 0 - relay off

#### **Example:**

Relay on.

```
CMD      55 61 AA 00 01 00 A6
RSP      DE 61 ED 00 00 00 59
```

#### **UFR\_BASE\_HD\_GET\_IO\_STATE (0x62)**

BASE HD uFR only.

Function returns states of 3 IO pins.

## RSP\_EXT

1st byte 1- voltage at the intercom terminals detected, 0 - no voltage at the intercom terminals

2nd byte 1 - voltage at DIGIN pin is high, 0 - voltage at DIGIN pin is low.

3rd byte 1 - relay is turn on, 0 - relay is turn off

### Example:

```
CMD      55 62 AA 00 00 00 A4
RSP      DE 62 ED 04 00 00 5C
RSP_EXT  01 00 01 07
```

## CARD RELATED COMMANDS

For all the functions for operations with cards the following applies:

- They operates only with one card in the device field
- If there is no card in the field device return error NO\_CARD (0x08).
- If there is more than one card in the field the behavior of the device is unpredictable but some of the next cases are possible:
  - Gives NO\_CARD error or
  - Just one card is detected and the device gives its type (this is due to the lack of a cascade of selection and the collision process as described in the ISO14443 standard).

## GENERAL PURPOSE CARD RELATED COMMANDS

### GET\_CARD\_ID (0x13)

This function return the serial number of the card which is currently in the readers field and the one byte value that represents its type. For Mifare Classic 1K the type is 0x08, Mifare Classic 4k type is 0x18 and Mifare Classic Mini cards type is 0x09.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

If everything operates as expected the RESPONSE set is sent and after that also the RESPONSE EXT set of 5 bytes which contains 4 byte Card UID values (little-endian) and CHECKSUM byte.

RSP\_Val0 contains value of the card type.

This function applies only for card with 4-byte UID. For longer UID's, use GET\_CARD\_ID\_EX (0x2C)

**Example:**

```
CMD      55 13 AA 00 00 00 F3
RSP      DE 13 ED 05 08 00 34
RSP_EXT  13 E2 0A 87 83
```

Where in RSP packet byte 05 represents RSP\_EXT\_length and byte 08 represents CardType – 0x08 – Mifare Classic.

RSP\_EXT returns Card UID (little-endian) and CHECKSUM of UID bytes.

If error occurs, like NO\_CARD, device will answer with ERR packet

```
CMD      55 13 AA 00 00 00 F3
ERR      EC 08 CE 00 00 00 31
```

Where byte 08 represents ERR\_CODE for NO\_CARD error.

**GET\_CARD\_ID\_EX (0x2C)**

Use this function for cards with UID longer than 4 byte.

This function return the serial number of the card which is currently in the readers field, length of serial number (4 (UID size: single), 7 (UID size: double) or 10 (UID size: triple)), and the one byte value that represents its type. For Mifare Classic 1K the type is 0x08, Mifare Classic 4k type is 0x18 and Mifare Classic Mini cards type is 0x09.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

If everything operates as expected the RSP packet is sent and after that also the RSP\_EXT packet of 11 bytes which contains card serial number and at the end one checksum byte.

RSP\_Val0 contains value of the card type.

RSP\_Val1 contains length of card serial number.

**Example:**

```
CMD      55 2C AA 00 00 00 DA
RSP      DE 2C ED 0B 08 04 1F
RSP_EXT  13 E2 0A 87 00 00 00 00 00 00 83
```

Where in RSP packet byte 0B represents RSP\_EXT\_Length, byte 08 means Card Type – Mifare Classic 1K, and byte 04 is length of card UID in RSP\_EXT packet.

RSP\_EXT packet contains card UID bytes and CHECKSUM.

If error occurs, like NO\_CARD, device will answer with ERR packet

```
CMD      55 2C AA 00 00 00 DA
ERR      EC 08 CE 00 00 00 31
```

Where byte 08 represents ERR\_CODE for NO\_CARD error.

### **GET\_LAST\_CARD\_ID\_EX (0x7C)**

This function returns UID of last card which was present in RF field of reader. It can handle all three known types: 4, 7 and 10 byte long UIDs. Difference with GetCardIdEx is that card does not be in RF field mandatory, UID value is stored in temporary memory area.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

If everything operates as expected the RSP packet is sent and after that also the RSP\_EXT packet of 11 bytes which contains card serial number and at the end one checksum byte.

RSP\_Val0 contains value of the card type.

RSP\_Val1 contains length of card serial number.

#### **Example:**

```

CMD          55 7C AA 00 AA CC EC

RSP          DE 7C ED 0B 08 04 4F
RSP_EXT     52 DA D9 95 00 00 00 00 00 00 CB
  
```

Where in RSP packet byte 0B represents RSP\_EXT\_Length, byte 08 means Card Type – Mifare Classic 1K, and byte 04 is length of card UID in RSP\_EXT packet.

RSP\_EXT packet contains card UID bytes and CHECKSUM.

If error occurs, like NO\_CARD, device will answer with ERR packet

```

CMD          55 7C AA 00 AA CC EC
ERR          EC 08 CE 00 AA CC 53
  
```

Where byte 08 represents ERR\_CODE for NO\_CARD error.

### **GET\_DLOGIC\_CARD\_TYPE (0x3C)**

This function returns card type according to following enumeration list:

DL_MIFARE_ULTRALIGHT	0x01
DL_MIFARE_ULTRALIGHT_EV1_11	0x02
DL_MIFARE_ULTRALIGHT_EV1_21	0x03
DL_MIFARE_ULTRALIGHT_C	0x04
DL_NTAG_203	0x05
DL_NTAG_210	0x06
DL_NTAG_212	0x07
DL_NTAG_213	0x08

DL_NTAG_215	0x09
DL_NTAG_216	0x0A
MIKRON_MIK640D	0x0B
DL_MIFARE_MINI	0x20
DL_MIFARE_CLASSIC_1K	0x21
DL_MIFARE_CLASSIC_4K	0x22
DL_MIFARE_PLUS_S_2K	0x23
DL_MIFARE_PLUS_S_4K	0x24
DL_MIFARE_PLUS_X_2K	0x25
DL_MIFARE_PLUS_X_4K	0x26
DL_MIFARE_DESFIRE	0x27
DL_MIFARE_DESFIRE_EV1_2K	0x28
DL_MIFARE_DESFIRE_EV1_4K	0x29
DL_MIFARE_DESFIRE_EV1_8K	0x2A

**Example:**

```

CMD      55 3C AA 00 00 00 CA
RSP      DE 3C ED 00 21 00 35

```

Where byte 21 in RSP packet represents card type – 0x21 – Mifare Classic 1K.

If error occurs, like NO\_CARD, device will answer with ERR packet

```

CMD      55 3C AA 00 00 00 CA
ERR      EC 08 CE 00 00 00 31

```

Where byte 08 represents ERR\_CODE for NO\_CARD error.

## FUNCTIONS FOR READING AND WRITING THE DATA INTO THE CARD

### *Authentication mode considerations for Mifare Classic tags*

The parameter AUTH\_MODE affects all the functions and determines authorization before reading or entering data in the card sector. This parameter can have the following values:

- **RKA\_AUTH1A**    0x00
- **RKA\_AUTH1B**    0x01
- **AKM1\_AUTH1A**   0x20
- **AKM1\_AUTH1B**   0x21
- **AKM2\_AUTH1A**   0x40
- **AKM2\_AUTH1B**   0x41
- **PK\_AUTH1A**      0x60
- **PK\_AUTH1B**      0x61

From the names of each of these constants can be concluded that the suffixes 1A and 1B indicate that you want to perform authentication key A or key B.

**Prefixes in the names of constants represents modes of authentication, as following:**

RKA – abbreviation of Reader Key Authentication. This means that authentication will be done with one of the 32 keys that are stored in reader device. It is assumed that as one of the command parameter that is sent to the reader is the index of the desired key. Indexes are in range 0..31.

AKM1 and AKM2 – abbreviation of Automatic Key Modes. This means that the authentication will be done automatically with the keys stored in reader device and they are indexed on the basis of the block or sector address where the writing or reading is currently done.

This applies to any function for card writing and reading, even for linear modes. I

When using AKM1 mode, keys in range 0 to 15 are used as Key A for corresponding sectors, while keys indexed from 16 to 31 are used as Key B for corresponding sectors.

**Example for AKM1 keys indexes:**

Key[00] = Key A Sector 0; Key [01] = Key A Sector [1]; ... Key [15] = Key A Sector 15;  
Key[16] = Key B Sector 0; Key [17] = Key B Sector [1]; ... Key [31] = Key B Sector 15;

When using AKM2, keys are indexed by odd and even order, so even keys indexes are used as Key A and odd keys indexes are used as Key B.

**Example for AKM2 keys indexes:**

Key[00] = Key A Sector 0; Key [02] = Key A Sector [1]; ... Key [30] = Key A Sector 15;  
Key[01] = Key B Sector 0; Key [03] = Key B Sector [1]; ... Key [31] = Key B Sector 15;

For 4k cards, which have 24 sectors more than 1k cards (total 40) for sectors 16 to 31 is used the same method as for indexing sectors 0 to 15 and for sectors 32 to 39 used the same method of indexing and for sectors 0 to 8.

PK – abbreviation for Provided Key refers to the authentication which is performed with key that is sent as a command parameter. Generally, this mode of authentication should be avoided due to the low level of security it provides, since key is passed as command parameter.

***Authentication mode considerations for NTAG 21x and other T2T tags (supported from firmware version 3.9.10)***

NTAG 21x and some other T2T tags (such as Ultralight EV2) support different authentication method from the Mifare Classic tags. NTAG 21x tags authentication is done using ISO 14443A-3 PWD\_AUTH command, requiring from the reader to transmit secret code (PWD) of 4 bytes the

tag, which responds with a PACK (PWD ACKNOWLEDGE). If the transmitted code is equal to that programmed in the tag, he responds with the correct PACK (length 2 bytes). PWD and PACK is typically written into the tag during the personalization process. The configuration pages are used to configure the memory access restriction of the tag. In order to familiarize with the methods of authentication of the NTAG 21x we recommend that you read "NTAG210 / 212, NFC Forum Type 2 Tag IC compliant with 48/128 bytes user memory Product data sheet" or "NTAG213 / 215/216, NFC Forum Type 2 Tag IC compliant with 144/504/888 bytes user memory data sheet Product" or "MF0ULx1, MIFARE Ultralight EV1 - Contactless IC ticket Product data sheet" that can be found on the manufacturer website. All these documents are marked "PUBLIC COMPANY".

NTAG 21x, Ultralight EV2 and other T2T tags supporting PWD\_AUTH, practically use 6 bytes (4 bytes that make up the PWD and 2 bytes of the PACK response) in our uFR readers we use the same mechanism as for Mifare Classic tags. The only difference is that a combined PWD (first 4 bytes of the key) and PACK (the last 2 bytes of the key) now forming a key (6 bytes in length). The resultant key can be prepared in advance and written in the card reader internal EEPROM (NV Memory) for using with Reader Key Authentication (RKA) method, or sent as a parameter of the uFR\_COM protocol command using Provided Key (PK) methods.

Note: Reader Key Authentication (RKA) methods with NTAG 21x, Ultralight EV2 and other T2T tags can not be used with uFR Classic and uFR Advanced commercial readers. These methods are possible only with newer reader series like uFR nano, uFR card size readers and HD Base with uFR support installed. On older models for this purpose can be used only Provided Key (PK) methods.

The following constants are declared for the parameter that determines the method for PWD\_AUTH for NTAG 21x, Ultralight EV2 and other T2T tags:

```
T2T_NO_PWD_AUTH 0x00
T2T_RKA_PWD_AUTH 0x01
T2T_PK_PWD_AUTH 0x61
```

These constants are used with the following uFR\_COM protocol commands:

```
BLOCK_READ
BLOCK_WRITE
LINEAR_READ
LINEAR_WRITE
LIN_ROW_READ
```

and passed as a parameter value controls AUTH\_MODE. If you use any other undeclared value as AUTH\_MODE, the effect will be the same as if you sent T2T\_NO\_PWD\_AUTH.



When for the AUTH\_MODE command parameter you send T2T\_RKA\_PWD\_AUTH or T2T\_PK\_PWD\_AUTH reader will always try to perform PWD\_AUTH regardless of the settings in the configuration pages of the tag. For the implementation of the adequate authentication scheme developer is responsible to use T2T\_NO\_PWD\_AUTH for access of the public data that are not protected by a pair of PWD, PACK.

## TRAILER BLOCK MANIPULATION COMMANDS

Special blocks called “trailer blocks” defines access bits and rights for Keys A and B for each sector. To read more, refer to NXP documentation about Mifare cards, see [http://www.nxp.com/documents/data\\_sheet/M001053\\_MF1ICS50\\_rev5\\_3.pdf](http://www.nxp.com/documents/data_sheet/M001053_MF1ICS50_rev5_3.pdf) and [http://www.nxp.com/documents/data\\_sheet/MF1S50YYX.pdf](http://www.nxp.com/documents/data_sheet/MF1S50YYX.pdf)

### SECTOR\_TRAILER\_WRITE (0x1A)

Function is used to write keys and access bits into the trailers of the sector. It could be used or sector address mode (without need for block\_in\_sector\_address to be sent because the given sector is always known) either the block address mode that determines the addressing\_mode u CMD\_EXT set parameter which can have the following values:

BLOCK\_ADDRESS\_MODE = 0

SECTOR\_ADDRESS\_MODE = 1

Access bits are sent separately as 4 bytes that has possible values 0 up to 7.

The device Firmware is formatting the access bits according to the cards specification irreversible blocking of that sector.

The CMD\_EXT set is used and its length depends on the authentication mode that is in use. CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

### RKA\_AUTH1x:

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of the set contains sector\_(block\_)address
- 2<sup>nd</sup> byte of the set contains dummy value
- 3<sup>rd</sup> byte of the set contains addressing mode
- 4<sup>th</sup> byte contains 9-byte sector trailer value (anything could be written)
- in 5<sup>th</sup> to 10<sup>th</sup> byte of the set is an unencrypted key A for writing
- in 11<sup>th</sup> to 14<sup>th</sup> byte are the access bits values for 0 to 3 blocks inside the sector respectively (for Classic 4k cards also the second half of their address space – the rest 2K of space, 11<sup>th</sup> byte of CMD\_EXT set determines the access bits values for the blocks 0 to 4, the 12<sup>th</sup> byte for blocks 5 to 9 and the 13<sup>th</sup> byte for blocks 10 to 14 and at the end 14<sup>th</sup> byte for sector trailer)
- the 15<sup>th</sup> to 20<sup>th</sup> byte of the set contains an unencrypted key B for writing

- 21<sup>st</sup> byte contains checksum

### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the set contains sector\_(block\_)address
- 2<sup>nd</sup> byte of the set contains dummy value
- 3<sup>rd</sup> byte of the set contains addressing mode
- 4<sup>th</sup> byte contains 9-byte sector trailer value (anything could be written)
- in 5<sup>th</sup> to 10<sup>th</sup> byte of the set is an unencrypted key A for writing
- in 11<sup>th</sup> to 14<sup>th</sup> byte are the access bits values for 0 to 3 blocks inside the sector respectively (for Classic 4k cards also the second half of their address space – the rest 2K of space, 11<sup>th</sup> byte of CMD\_EXT set determines the access bits values for the blocks 0 to 4, the 12<sup>th</sup> byte for blocks 5 to 9 and the 13<sup>th</sup> byte for blocks 10 to 14 and at the end 14<sup>th</sup> byte for sector trailer)
- the 15<sup>th</sup> to 20<sup>th</sup> byte of the set contains an unencrypted key B for writing
- 21<sup>st</sup> byte contains checksum

### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the set contains sector\_(block\_)address
- 2<sup>nd</sup> byte of the set contains dummy value
- 3<sup>rd</sup> byte of the set contains addressing\_mode
- 4<sup>th</sup> byte contains 9-byte sector trailer value (anything could be written)
- array from 5<sup>th</sup> up to 10<sup>th</sup> byte contains 6-byte key.
- in 11<sup>th</sup> to 16<sup>th</sup> byte of the set is an unencrypted key A for writing
- in 17<sup>th</sup> to 20<sup>th</sup> byte are the access bits values for 0 to 3 blocks inside the sector respectively (for Classic 4k cards also the second half of their address space – the rest 2K of space, 11<sup>th</sup> byte of CMD\_EXT set determines the access bits values for the blocks 0 to 4, the 12<sup>th</sup> byte for blocks 5 to 9 and the 13<sup>th</sup> byte for blocks 10 to 14 and at the end 14<sup>th</sup> byte for sector trailer)
- the 21<sup>st</sup> do 26<sup>th</sup> byte of the set contains an unencrypted key B for writing
- 27<sup>th</sup> byte contains checksum

If everything is done as it should it returns the RESPONSE set.

RESPONSE\_EXT is not used.

### **Example:**

authentication RKA key A, key number 0, sector address 0, addressing mode 1, key A = 0xFFFFFFFFFFFF, key B = 0xFFFFFFFFFFFF, access bits values 0, 0, 0, 1

<b>CMD</b>	<b>55 1A AA 15 00 00 F7</b>
<b>ACK</b>	<b>AC 1A CA 15 00 00 70</b>

```
CMD_EXT    00 00 01 69 FF FF FF FF FF FF 00 00 00 01 FF FF FF FF FF FF 70
RESP       DE 1A ED 00 00 00 30
```

### **SECTOR\_TRAILER\_WRITE\_UNSAFE (0x2F)**

It operates as SECTOR\_TRAILER\_WRITE except it send already formatted sector trailer block to be written without the access bits value check. The command is unsafe because it could lead to irreversible blocking of the entire sector of the card due to improperly formatted value of access bits. Made only for advanced users.

The CMD\_EXT set is used and its length depends on the authentication mode that is in use. CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

#### **RKA\_AUTH1x:**

- CMD\_Par1 u CMD set contains readers index key
- 1<sup>st</sup> byte of the set contains sector\_(block\_)address
- 2<sup>nd</sup> byte of the set contains dummy value
- 3<sup>rd</sup> byte of the set contains addressing\_mode
- 4<sup>th</sup> byte of the set contains dummy value
- in 5<sup>th</sup> to 20<sup>th</sup> byte of the set is the content of the sector trailer for writing
- 21<sup>st</sup> byte contains checksum

#### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the set contains sector\_(block\_)address
- 2<sup>nd</sup> byte of the set contains dummy value
- 3<sup>rd</sup> byte of the set contains addressing\_mode
- 4<sup>th</sup> byte of the set contains dummy value
- in 5<sup>th</sup> to 20<sup>th</sup> byte of the set is the content of the sector trailer for writing
- 21<sup>st</sup> byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the set contains sector\_(block\_)address
- 2<sup>nd</sup> byte of the set contains dummy value
- 3<sup>rd</sup> byte of the set contains addressing\_mode
- 4<sup>th</sup> byte of the set contains dummy value
- array from 5<sup>th</sup> up to 10<sup>th</sup> bytes contains 6-byte key.
- in 11<sup>th</sup> to 26<sup>th</sup> byte of the set is the content of the sector trailer for writing
- 27<sup>th</sup> byte contains checksum

If everything is done as it should it returns the RESPONSE set.

RESPONSE\_EXT is not used.

**Example:**

authentication RKA key A, key number 0, sector address 0, addressing mode 1, key A = 0xFFFFFFFFFFFF, key B = 0xFFFFFFFFFFFF, access bits values 0xFF078069 (default configuration)

```
CMD      55 2F AA 15 00 00 CC
ACK      AC 2F CA 15 00 00 63
```

```
CMD_EXT  00 00 01 00 FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF 17
RESP     DE 2F ED 00 00 00 23
```

**BLOCK MANIPULATION COMMANDS**

Following commands used direct block addressing, meaning that blocks are indexed in range 0 to 63 for Mifare 1K cards.

**BLOCK\_READ (0x16)**

Reads the whole data block from the card which is in the reader field.

The CMD\_EXT set is used and its length depends on authentication mode that is used.

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

**RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains key index in the reader
- 1<sup>st</sup> byte of CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains dummy data
- 5<sup>th</sup> byte contains checksum

**Example:**

read block 01 with RKA\_AUTH1A

```
CMD      55 16 AA 05 00 00 F3
ACK      AC 16 CA 05 00 00 7C
```

```
CMD_EXT  01 00 00 00 08
```

```
RSP      DE 16 ED 11 00 00 3B
```

```
RSP_EXT  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 07
```

**AKMy\_AUTH1x:**

- CMD\_Par1 is not used.

- 1<sup>st</sup> byte of CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains dummy data
- 5<sup>th</sup> byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains dummy data
- array from 5<sup>th</sup> to 10<sup>th</sup> byte contains 6-byte key.
- 11<sup>th</sup> byte contains checksum

If all operates as it should it turns the RESPONSE set and the RESPONSE\_EXT is following with 16 read bytes and checksum at the end.

#### **BLOCK\_WRITE (0x17)**

Writes the whole data block into the card that is currently in the readers field. Address mode is used for so called block addressing where for example the first block on Mifare Classic 1k has an address 0 and the last one has the address 63. This command doesn't allow the direct writing into the sector trailer and in the case of its addressing it gives back the FORBIDDEN\_DIRECT\_WRITE\_IN\_SECTOR\_TRAILER.

The CMD\_EXT set is used and its length depends on the authentication mode that is in use.

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

#### **RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains dummy data
- in 5<sup>th</sup> to 20<sup>th</sup> byte of set are placed data for writing into the data block
- 21<sup>st</sup> byte contains checksum

#### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains dummy data
- in 5<sup>th</sup> to 20<sup>th</sup> byte of the set are placed the data for writing into the data block
- 21<sup>st</sup> byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of CMD\_EXT set contains block\_address

- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte CMD\_EXT set contains dummy data
- array from 5<sup>th</sup> to 10<sup>th</sup> byte contains 6-byte key.
- in 11<sup>th</sup> to 26<sup>th</sup> byte are placed the data for writing into the data block
- 27<sup>th</sup> byte contains checksum.

If everything is done as it should device answer with RSP packet.

**Example:**

write "01 02 03 04 05 06 07 08" into block 1 using key "FF FF FF FF FF FF"

**CMD**            55 17 AA 1B 60 00 9A

**ACK**            AC 17 CA 1B 60 00 11

**CMD\_EXT**      01 00 00 00 FF FF FF FF FF FF 01 02 03 04 05 06 07 08 00 00 00  
00 00

00 00 00 10

**RSP**            DE 17 ED 00 00 00 2B

***BLOCK\_IN\_SECTOR\_READ (0x18)***

It has the same function as the BLOCK\_READ but uses the different address mode for so called sector addressing where is always given the address of the sector and the sector block (as specified in the NXP documentation for Mifare Classic cards). The first sector of the Mifare Classic 1k card for example has the address 0 and the last one has 15. The block addresses of the sector are defined in the interval from 0 to 3 (3<sup>rd</sup> block of each sector is sector trailer) excluding Mifare Classic 4k cards for which in its second line of address space (the second 2k that is 32<sup>nd</sup> up to 39<sup>th</sup> sector) have the block addresses in sector 0 to 15 and the 15<sup>th</sup> is sector trailer.

Communication command protocol is the same as with BLOCK\_READ with following exception:

- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_in\_sector\_address
- 2<sup>nd</sup> byte of the CMD\_EXT set contains sector\_address
- 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data

**Example:**

read block 0 in sector 0 with RKA\_AUTH1A, key number 0

**CMD**            55 18 AA 05 00 00 E9

**ACK**            AC 18 CA 05 00 00 82

**CMD\_EXT**      00 00 00 00 07

```
RSP          DE 18 ED 11 00 00 41
RSP_EXT     47 8F 90 61 39 08 04 00 01 F1 0A F0 1A A2 EB 1D 4F
```

### **BLOCK\_IN\_SECTOR\_WRITE (0x19)**

Has the same function as the BLOCK\_WRITE but uses the different address mode, so called sector addressing where the sector address and the address of the block in the sector is always given (as mentioned in NXP documentation for Mifare Classic cards). For example the first sector on Mifare Classic 1k card has the address 0 and the last one has the address 15. The block addresses in sector are in the interval from 0 to 3 (3<sup>rd</sup> block of each sector is sector trailer) excluding Mifare Classic 4k cards for which in its second line of address space (the second 2k that is 32<sup>nd</sup> up to 39<sup>th</sup> sector) have the block addresses in sector 0 to 15 and the 15<sup>th</sup> is sector trailer. Communication command protocol is the same as with BLOCK\_WRITE with following exception:

- 1<sup>st</sup> byte of CMD\_EXT set contains block\_in\_sector\_address
- 2<sup>nd</sup> byte of CMD\_EXT set contains sector\_address
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains dummy data

#### **Example:**

write block 1 in sector 0 with RKA\_AUTH1A, key number 0

```
CMD          55 19 AA 15 00 00 FA
ACK          AC 19 CA 15 00 00 71
CMD_EXT     01 00 00 00 00 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF 17
RSP         DE 19 ED 00 00 00 31
```

## **LINEAR DATA MANIPULATION COMMANDS**

### **LINEAR\_READ (0x14)**

Linear read data from the card. This command concatenates data for successive blocks and sectors into one array of data. It performs something like “continuous reading” of data. It is very convenient for reading data from more blocks or sectors which are in successive order.

The CMD\_EXT set is used whose length depends on the mode of authentication that is used.

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT sets contains:

#### **RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains key index in the
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- 5<sup>th</sup> byte contains checksum

**Example:**

Read linear data from 0 to 63, length is 64 bytes, using RK AUTH1A

**CMD**            55 14 AA 05 00 00 F5  
**ACK**            AC 14 CA 05 00 00 7E

**CMD\_EXT**       00 00 40 00 47  
**RSP**            DE 14 ED 41 00 00 6D

and DATA we asked for in RSP\_EXT

31 32 33 34 35 36 37 38 39 30 00 00 00 00 00 31  
32 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

With checksum

38

**AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- 5<sup>th</sup> byte contains checksum

**Example:** Read linear data from 0 to 31, length is 32 bytes, using AKM1 AUTH1A

**CMD**            55 14 AA 05 20 00 D5  
**ACK**            AC 14 CA 05 20 00 5E

**CMD\_EXT**       00 00 20 00 27  
**RSP**            DE 14 ED 21 00 00 0D

and DATA we asked for in RSP\_EXT

31 32 33 34 35 36 37 38 39 30 00 00 00 00 00 31  
32 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00

With checksum

38

**Example:** Read linear data from 0 to 31, length is 32 bytes, using AKM1 AUTH1B



**CMD**            55 14 AA 05 21 00 D6  
**ACK**            AC 14 CA 05 21 00 5D

**CMD\_EXT**       00 00 20 00 27  
**RSP**            DE 14 ED 21 00 00 0D

and DATA we asked for in RSP\_EXT

31 32 33 34 35 36 37 38 39 30 00 00 00 00 00 31  
32 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00

With checksum

38

Same applies to AKM2 AUTHA and AUTHB commands.

### PK\_AUTH1x:

- CMD\_Par1 is not used.
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- array from 5<sup>th</sup> do 10<sup>th</sup> byte contains 6-byte key.
- 11<sup>th</sup> byte contains checksum.

**Example:** Read linear data from 16 to 31, length is 16 bytes, using PK AUTH1B and provided key 6 x FF

**CMD**            55 14 AA 0B 61 00 88  
**ACK**            AC 14 CA 0B 61 00 1F

**CMD\_EXT**       10 00 10 00 FF FF FF FF FF FF 07  
**RSP**            DE 14 ED 11 00 00 3D

and DATA we asked for in RSP\_EXT

32 33 00 00 00 00 00 00  
00 00 00 00 00 00 00 00

with checksum

08

If everything operates as expected the RSP packet is sent and after that also the RSP\_EXT with number of bytes according to the data\_length command with checksum at the end.

In case the card is removed from the field or in case of wrong authentication including that some block is read anyway, it turns ERR set with NO\_CARD error code or AUTH\_ERROR and then the ERR\_EXT set which contains the array of the read bytes and CHECKSUM at the end.

LINEAR\_READ command utilise FAST\_READ ISO 14443-3 command with NTAG21x and Mifare Ultralight EV1 tags.

### **LINEAR\_WRITE (0x15)**

Linear data writing into the card which is currently in the field of the reader. The verification of each written block is done during the writing.

The CMD\_EXT set is used and its length depends on the authentication mode that is used

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT sets contains:

#### **RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains key index in the reader
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- from 5<sup>th</sup> byte up (data\_length + 4) contains data array for writing
- (data\_length + 5) byte contains checksum

**Example:** Write 8 bytes into card string at linear address 08, using RK\_AUTH1A, bytes are 10 11...17

**CMD**            55 15 AA 0D 00 00 EE

**ACK**            AC 15 CA 0D 00 00 85

**CMD\_EXT**       08 00 08 00 10 11 12 13 14 15 16 17 07

**RSP**            DE 15 ED 00 00 00 2D

We can check now if bytes are written using previous examples of LinearRead command.

#### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- from 5<sup>th</sup> byte up (data\_length + 4) contains data array for writing
- (data\_length + 5) byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- array from 5<sup>th</sup> do 10<sup>th</sup> byte contains 6- byte key

- 11<sup>th</sup> byte and up to (data\_length + 10) contains data array for writing
- (data\_length + 11) byte contains checksum.

If everything went as expected device answer with RSP packet.

In error case it turns the ERR packet where the RSP\_Val0 contains the number of eventual written bytes.

### **LINEAR\_FORMAT\_CARD (0x25)**

The CMD\_EXT set is used and its length depends on the authentication mode that is used.

Since this command can erase data or block card reading if wrong access bits are provided, we strongly suggest to test it first through SDK API examples to figure out what this command does.

For pure erasing data or filling card with 0x00 without changing the keys, it is much easier to use Linear\_Write command.

#### **Usage:**

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

#### **RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of the set contains access bits value for blocks in sector
- 2<sup>nd</sup> byte of the set contains access bits value for sector trailers
- 3<sup>rd</sup> byte of the set contains dummy value
- 4<sup>th</sup> byte of the set has 9-byte sector trailer value (anything could be written)
- in 5<sup>th</sup> to 10<sup>th</sup> byte of the set is new key A
- in 11<sup>th</sup> to 16<sup>th</sup> byte of the set is new key B
- 17<sup>th</sup> byte contains checksum

#### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the set contains access bits value for blocks in sector
- 2<sup>nd</sup> byte of the set contains access bits value for sector trailers
- 3<sup>rd</sup> byte of the set contains dummy value
- 4<sup>th</sup> byte of the set has 9-byte sector trailer value (anything could be written)
- in 5<sup>th</sup> to 10<sup>th</sup> byte of the set is new key A
- in 11<sup>th</sup> to 16<sup>th</sup> byte of the set is new key B
- 17<sup>th</sup> byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the set contains access bits value for blocks in sector
- 2<sup>nd</sup> byte of the set contains access bits value for sector trailers
- 3<sup>rd</sup> byte of the set contains dummy value

- 4<sup>th</sup> byte of the set has 9-byte sector trailer value (anything could be written)
- array from 5<sup>th</sup> up to 10<sup>th</sup> byte contains 6-byte key for authentication (previous)
- in 11<sup>th</sup> to 16<sup>th</sup> byte of the set is new key A
- in 17<sup>th</sup> to 22<sup>nd</sup> byte of the set is new key B
- 23<sup>rd</sup> byte contains checksum

If everything is done as it should device answer with RSP packet.  
RSP\_EXT is not used.

**Example:**

Key A is 0xFFFFFFFFFFFF, Key B is 0xFFFFFFFFFFFF, access bits value for blocks is 0, access bits value for sector trailers is 1, authentication mode is RKA\_AUTH1A, key number is 0

```

CMD          55 25 AA 11 00 00 D2
ACK          AC 25 CA 11 00 00 59
CMD_EXT     00 01 00 69 FF FF FF FF FF FF FF FF FF FF FF 6F
RSP         DE 25 ED 00 10 00 0D
    
```

**LIN\_ROW\_READ(0x45)**

Functions allow you to quickly read data from the card including the sector trailer blocks. These functions are very similar to the functions for linear reading of users data space. Using this command is the same as using the command LINEAR\_READ(0x14)

The CMD\_EXT set is used whose length depends on the mode of authentication that is used. CMD\_Par0 contains AUTH\_MODE. Depending on AUTH\_MODE, CMD and CMD\_EXT sets contains:

**RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains key index in the
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- 5<sup>th</sup> byte contains checksum

**AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)
- 5<sup>th</sup> byte contains checksum

**PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> and 2<sup>nd</sup> byte of CMD\_EXT set contains linear\_address (little endian)
- 3<sup>rd</sup> and 4<sup>th</sup> byte of CMD\_EXT set contains data\_length (little endian)

- array from 5<sup>th</sup> do 10<sup>th</sup> byte contains 6-byte key.
- 11<sup>th</sup> byte contains checksum.

**Example:**

Read data from 0 to 47, length is 48 bytes, using RK AUTH1A key number 0

```
CMD      55 45 AA 05 00 00 C6
ACK      AC 45 CA 05 00 00 2D
CMD_EXT  00 00 30 00 37
RSP      DE 45 ED 31 00 00 4E
RSP_EXT  47 8F 90 61 39 08 04 00 01 F1 0A F0 1A A2 EB 1D 00 00 00 00 00
00 FF
          07 80 69 FF FF FF FF FF FF 00 00 00 00 00 00 FF 07 80 69 FF FF
FF FF
          FF FF 4F
```

## VALUE BLOCK MANIPULATION COMMANDS

### DIRECT BLOCK ADDRESSING

#### VALUE\_BLOCK\_READ (0x1D)

Reads the 4-byte value of the “value block” of the card which is currently in the reading field. Address mode that is used is so called block addressing where for example the first block of Mifare Classic 1k card has the address 0 and the last one has the address 63.

The CMD\_EXT set is used and its length depends on the authentication mode that is used. CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

#### RKA\_AUTH1x:

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- 5<sup>th</sup> byte contains checksum

#### AKMy\_AUTH1x:

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- 5<sup>th</sup> byte contains checksum

### PK\_AUTH1x:

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- array from 5<sup>th</sup> up to 10<sup>th</sup> byte contains 6-byte key.
- 11<sup>th</sup> byte contains checksum

If everything is OK, device answer with RSP packet followed by RSP\_EXT containing 4-byte value and checksum.

RSP\_Val0 contains block address (read from block value for powerful backup as mentioned in the Mifare card documentation).

In the case of error the VALUE\_BLOCK\_ADDR\_INVALID (read value of the value block is formatted properly but the address bytes aren't) it returns ERR\_EXT set which contains the value of the value block.

Notice that value is in little-endian notation, where negative values are stored as "Two complement's".

### Example:

Read Value Block 05 with PK\_AUTH1A:

```
CMD          55 1D AA 0B 60 00 90
ACK          AC 1D CA 0B 60 00 17

CMD_EXT      05 00 00 00 FF FF FF FF FF FF 0C
RSP          DE 1D ED 05 00 00 32
RSP_EXT      00 00 00 00 07
```

### VALUE\_BLOCK\_WRITE (0x1E)

Store 4-byte value into "value block".

This command disallow the writing into the trailers of the sector and in case of their addressing it returns the FORBIDEN\_DIRECT\_WRITE\_IN\_SECTOR\_TRAILER.

The CMD\_EXT set is used and its length depends on the authentication mode that is used.

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

### RKA\_AUTH1x:

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup> and 3<sup>rd</sup> byte of the CMD\_EXT set contains dummy data
- 4<sup>th</sup> byte contains value address
- in 5<sup>th</sup> to 8<sup>th</sup> byte of the set is placed the data for writing into the value block
- 9<sup>th</sup> byte contains checksum

#### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup> and 3<sup>rd</sup> byte of the CMD\_EXT set contains dummy data
- 4<sup>th</sup> byte contains value address
- in 5<sup>th</sup> to 8<sup>th</sup> byte of the set is placed the data for writing into the value block
- 9<sup>th</sup> byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup> and 3<sup>rd</sup> byte of the CMD\_EXT set contains dummy data
- 4<sup>th</sup> byte contains value address
- array from 5<sup>th</sup> up to 10<sup>th</sup> byte contains 6-byte key.
- in 11<sup>th</sup> to 14<sup>th</sup> byte of the set is placed the data for writing into the value block
- 15<sup>th</sup> byte contains checksum

Example: Store value 01 01 01 01 into block 5 using PK\_AUTH1A key FF FF FF FF FF FF

```
CMD      55 1E AA 0F 60 00 95
ACK      AC 1E CA 0F 60 00 1E
```

```
CMD_EXT  05 00 00 05 FF FF FF FF FF FF 01 01 01 01 07
RSP      DE 1E ED 00 00 00 34 DE
```

If everything is OK, device answer with RSP packet. RSP\_EXT is not used.

Notice that value is in little-endian notation, where negative values are stored as “Two complement's”. For example, decimal value 65535 should be stored as FF FF 00 00.

#### **VALUE\_BLOCK\_INC (0x21)**

It increases the value of the addressed value block for the 4-byte value increment\_val that is send as a command parameter and is been used for so-called block address mode.

The CMD\_EXT set is used and its length depends on the authentication mode that is used.

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

**RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- in 5<sup>th</sup> to 8<sup>th</sup> byte set is increment\_val
- 9<sup>th</sup> byte contains checksum

**AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- in 5<sup>th</sup> to 8<sup>th</sup> byte set is increment\_val
- 9<sup>th</sup> byte contains checksum

**PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- array from 5<sup>th</sup> up to 10<sup>th</sup> byte contains 6-byte key
- in 11<sup>th</sup> to 14<sup>th</sup> bytes of the set is increment\_val
- 15<sup>th</sup> byte contains checksum.

If everything is OK, device answer with RSP packet. RSP\_EXT packet is not used.

**Example:**

Increase Value Block 5 with "F0 F0 F0 F0" using PK\_AUTH1A with key FF FF FF FF FF FF

**CMD**            55 21 AA 0F 60 00 B8

**ACK**            AC 21 CA 0F 60 00 2F

**CMD\_EXT**      05 00 00 00 FF FF FF FF FF FF F0 F0 F0 F0 0C

**RSP**            DE 21 ED 00 00 00 19 DE

Notice that when we read now Value Block 5 we will get

RSP and RSP\_EXT      DE 1D ED 05 05 00 35 F1 F1 F1 71 87,

with value F1 F1 F1 71, stored in little-endian notation, where byte 71 is represented in Two Complement's manner (change of sign +/-).



### **VALUE\_BLOCK\_DEC (0x22)**

Decrement the value of the addressed value block for 4-byte value decrement\_val which is sent as the command parameter. The so-called block address mode is used.

The CMD\_EXT set is used and the length of the authentication mode is used.

CMD\_Par0 contains AUTH\_MODE.

Depending on AUTH\_MODE, CMD and CMD\_EXT set contains:

#### **RKA\_AUTH1x:**

- CMD\_Par1 in CMD set contains readers index key
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte CMD\_EXT set contains dummy data
- in 5<sup>th</sup> to 8<sup>th</sup> byte of the set is decrement\_val
- 9<sup>th</sup> byte contains checksum

#### **AKMy\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte CMD\_EXT set contains dummy data
- in 5<sup>th</sup> to 8<sup>th</sup> byte of the set is decrement\_val
- 9<sup>th</sup> byte contains checksum

#### **PK\_AUTH1x:**

- CMD\_Par1 is not used.
- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_address
- 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data
- array from 5<sup>th</sup> up to 10<sup>th</sup> byte contains 6-byte key.
- in 11<sup>th</sup> to 14<sup>th</sup> byte of the set is decrement\_val
- 15<sup>th</sup> byte contains checksum.

If everything is OK, device answer with RSP packet. RSP\_EXT packet is not used

#### **Example:**

Decrement Value Block 5 with 00 00 00 F0 using PK\_AUTH1A with key FF FF FF FF FF FF

```
CMD      55 22 AA 0F 60 00 B9
ACK      AC 22 CA 0F 60 00 32

CMD_EXT  05 00 00 00 FF FF FF FF FF FF 00 00 00 F0 FC
RSP      DE 22 ED 00 00 00 18
```

Notice that when we read now Value Block 5 we will get

```
RSP and RSP_EXT  DE 1D ED 05 05 00 35  F1 F1 F1 01 F7
```

with value F1 F1 F1 01, stored in little-endian notation, where byte 01 is represented in Two Complement's manner (change of sign +/-).

## INDIRECT BLOCK ADDRESSING

### VALUE\_BLOCK\_IN\_SECTOR\_READ (0x1F)

It operates as VALUE\_BLOCK\_READ but uses the different address mode, so-called sector addressing where are always given the sector address and the block address in the sector (as mentioned in NXP documentation for Mifare Classic cards).

For example the first sector of the Mifare Classic 1k card has the 0 and the last one has the address 15. Block addresses in the sector are in the interval from 0 to 3 (3<sup>rd</sup> block of each sector is sector trailer) excluding Mifare Classic 4k cards for which in its second half of address space (second 2k with 32 to 39 sector) the addresses of the blocks in sector 0 to 15 and the block 15 is sector trailer.

Communication command protocol is the same as with VALUE\_BLOCK\_READ with following exception:

- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_in\_sector\_address
- 2<sup>nd</sup> byte of the CMD\_EXT set contains sector\_address
- 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data.

Device will answer with RSP and RSP\_EXT. RSP\_Val0 contains direct block address.

### Example:

Read Value Block 01 in Sector 01 (is equal to Value Block 5 using direct addressing) using PK\_AUTH1A mode with key FF FF FF FF FF FF

```
CMD      55 1F AA 0B 60 00 92
ACK      AC 1F CA 0B 60 00 19

CMD_EXT  01 01 00 00 FF FF FF FF FF FF 07
RSP      DE 1F ED 05 05 00 33
RSP_EXT  F1 F1 F1 01 F7
```

### VALUE\_BLOCK\_IN\_SECTOR\_WRITE (0x20)

It operates as VALUE\_BLOCK\_WRITE but uses different address mode, so-called sector addressing where are always given the sector address and the block address in the sector (as mentioned in NXP documentation for Mifare Classic cards). For example the first sector of the Mifare Classic 1k card has the 0 and the last one has the address 15. Block addresses in the sector are in the interval from 0 to 3 (3<sup>rd</sup> block of each sector is sector trailer) excluding Mifare Classic 4k cards for which in its second half of address space (second 2k with 32 to 39 sector) the addresses of the blocks in sector 0 to 15 and the block 15 is sector trailer.

Communication command protocol is the same as with VALUE\_BLOCK\_IN\_SECTOR\_READ with following exception:

- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_in\_sector\_address
- 2<sup>nd</sup> byte of the CMD\_EXT set contains sector\_address
- 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data

#### Example:

Write Value Block 00 in Sector 01 (is equal to Value Block 5 using direct addressing) value "80 80 80 80" using PK\_AUTH1A mode with key FF FF FF FF FF FF

**CMD**            55 20 AA 0F 60 00 B7

**ACK**            AC 20 CA 0F 60 00 30

**CMD\_EXT**      01 01 00 00 FF FF FF FF FF FF 80 80 80 80 07

**RSP**            DE 20 ED 00 00 00 1A

### VALUE\_BLOCK\_IN\_SECTOR\_INC (0x23)

It operates as VALUE\_BLOCK\_IN\_SECTOR\_INC but uses the different address mode, so-called sector addressing where are always given the sector address and the block address in the sector (as mentioned in NXP documentation for Mifare Classic cards). For example the first sector of the Mifare Classic 1k card has the 0 and the last one has the address 15. Block addresses in the sector are in the interval from 0 to 3 (3<sup>rd</sup> block of each sector is sector trailer) excluding Mifare Classic 4k cards for which in its second half of address space (second 2k with 32 to 39 sector) the addresses of the blocks in sector 0 to 15 and the block 15 is sector trailer.

Communication command protocol is the same as with VALUE\_BLOCK\_INC with following exception:

- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_in\_sector\_address
- 2<sup>nd</sup> byte of the CMD\_EXT set contains sector\_address
- 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data.

**Example:**

```
CMD      55 23 AA 0F 60 00 BA
ACK      AC 23 CA 0F 60 00 31
```

```
CMD_EXT  01 01 00 00 FF FF FF FF FF FF 60 60 60 60 07
RSP      DE 23 ED 00 00 00 17
```

**VALUE\_BLOCK\_IN\_SECTOR\_DEC (0x24)**

It operates as VALUE\_BLOCK\_IN\_SECTOR\_DEC but uses different address mode, so-called sector addressing where are always given the sector address and the block address in the sector (as mentioned in NXP documentation for Mifare Classic cards). For example the first sector of the Mifare Classic 1k card has the 0 and the last one has the address 15. Block addresses in the sector are in the interval from 0 to 3 (3<sup>rd</sup> block of each sector is sector trailer) excluding Mifare Classic 4k cards for which in its second half of address space (second 2k with 32 to 39 sector) the addresses of the blocks in sector 0 to 15 and the block 15 is sector trailer.

Communication command protocol is the same as with VALUE\_BLOCK\_DEC with following exception:

- 1<sup>st</sup> byte of the CMD\_EXT set contains block\_in\_sector\_address
- 2<sup>nd</sup> byte of the CMD\_EXT set contains sector\_address
- 3<sup>rd</sup> and 4<sup>th</sup> byte of the CMD\_EXT set contains dummy data

**Example:**

```
CMD      55 24 AA 0F 60 00 BB
ACK      AC 24 CA 0F 60 00 34
```

```
CMD_EXT  01 01 00 00 FF FF FF FF FF FF 60 60 60 60 07
RSP      DE 24 ED 00 00 00 1E
```

**COMMANDS FOR “ASYNCHRONOUS UID SENDING” FEATURE**

This feature “Async UID sending” is capability of reader device to send Card UID immediately when card enters into device RF field, without any action initiated by host. This is also exception from rule that communication is always initiated by host to device. Feature can be turned on and off. Baudrate for this feature is different than baudrate of device, e.g. it can be different. Prefix and suffix are bytes that are used to diversify UID's, like header and trailer bytes of UID.

Device can send UID encapsulated in [Prefix] and [Suffix] when card enters into RF field.

Device can also send “empty UID” when card leaves RF field, meaning only [Prefix][Suffix] will be sent.

Best practice is to set Baud rate different than device communication speed, anything bigger than 9600 Bps to avoid colision with standard communication between device and host.

### SET\_CARD\_ID\_SEND\_CONF (0x3D)

Set the asynchronously card ID sending parameters.

CMD\_Par0 contains send enable flag (bit 0), prefix enable flag (bit 1) and send removed enable flag (bit2).

When using option Send removed flag, Prefix byte is mandatory

1<sup>st</sup> byte of the CMD\_EXT contains prefix character

2<sup>nd</sup> byte of the CMD\_EXT contains suffix character

array from 3<sup>rd</sup> byte up to 6<sup>th</sup> byte of the CMD\_EXT contains baud rate value

7<sup>th</sup> byte of the CMD\_EXT contains internal CRC (xor of bytes CMD\_Par0 to 6<sup>th</sup> byte + 7)

8<sup>th</sup> byte of the CMD\_EXT contains checksum

If everything is OK, device answer with RSP packet. RSP\_EXT is not used.

#### Example:

**CMD**            55 3D AA 08 07 00 D4 (send command 3D, bits 0,1,2 high), D4  
**checksum**

**ACK**            AC 3D CA 08 07 00 5B (ACK OK)

**CMD\_EXT**    CC EE 80 25 00 00 87 07 (prefix CC, suffix EE, speed 9600  
(0x2580) ,

(87 checksum -

07,00,CC,EE,80,25,00,00) ,

(07 - checksum of CMD\_EXT)

**RSP**            DE 3D ED 00 00 00 15            (RESPONSE OK) speed 9600 (0x2580) ,

When card enter the field, event will occur:

<b>HEX</b>	CC	30	34	32	32	43	33	36	32	34	42	32	44	38	31	EE
<b>ASCII</b>	?	0	4	2	2	C	3	6	2	4	B	2	D	8	1	?

meaning card UID is 04 22 C3 62 4B 2D 81

On card removal, event will occur:

**CC EE**

To disable feature, send bits 0,1,2 low:

```
CMD  55 3D AA 00 00 00 C9
RSP  DE 3D ED 00 00 00 15
```

### GET\_CARD\_ID\_SEND\_CONF (0x3E)

Get the asynchronously card ID sending parameters.

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

If everything is OK, device answer with RSP packet and after that also the RSP\_EXT packet of 9 bytes.

RSP\_Val0 and RSP\_Val1 are not in use.

1<sup>st</sup> byte of the RESPONSE\_EXT contains send enable flag (bit 0), prefix enable flag (bit 1) and send removed enable flag (bit2).

2<sup>nd</sup> byte of the RESPONSE\_EXT contains prefix character

3<sup>rd</sup> byte of the RESPONSE\_EXT contains suffix character

array from 4<sup>th</sup> byte up to 7<sup>th</sup> byte of the RESPONSE\_EXT contains baud rate value

8<sup>th</sup> byte of the RESPONSE\_EXT contains internal CRC

9<sup>th</sup> byte of the RESPONSE\_EXT contains checksum

#### Example:

```
CMD      55 3E AA 00 00 00 C8 (send CMD 3E, C8 checksum)
RSP      DE 3E ED 09 00 00 0B (RSP command 3E, 9 byte follows, 0B
checksum)
RSP_EXT  07 CC EE 80 25 00 00 87 0E (07 -bits 0,1,2 high, CC Prefix, EE
suffix,
                                speed 9600 (0x2580),
                                87 - checksum (
07,CC,EE,80,25,00,00) ,
                                0E - checksum of RSP_EXT)
```

## COMMANDS FOR WORKS WITH DESFIRE CARDS

### ERROR CODES FOR DESFIRE CARD OPERATIONS

```
#define DATA_OVERFLOW          2990
#define READER_ERROR            2999
#define NO_CARD_DETECTED        3000
#define CARD_OPERATION_OK        3001
#define WRONG_KEY_TYPE          3002
#define KEY_AUTH_ERROR          3003
#define CARD_CRYPTO_ERROR       3004
```

```
#define READER_CARD_COMM_ERROR      3005
#define PC_READER_COMM_ERROR       3006
```

### **DESFIRE\_WRITE\_AES\_KEY (0x8E)**

Command writes AES key into reader.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT contains ordinal number of AES key into reader

array from 2<sup>nd</sup> byte up to 17<sup>th</sup> byte of the CMD\_EXT contains AES key

18<sup>th</sup> byte of the CMD\_EXT contains checksum

Device answer with RSP packet. RSP\_EXT is not used.

#### **Example:**

AES key is 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF, and ordinal number is 3

```
CMD      55 8E AA 12 00 00 6A (send command 8E) , 6A checksum
ACK      AC 8E CA 12 00 00 01 (ACK OK)
```

```
CMD_EXT  03 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 0A (0A
checksum)
RSP      DE 8E ED 00 00 00 C4          (RESPONSE OK)
```

### **GET\_DESFIRE\_UID (0x80)**

Command returns Unique ID of card, if the Random ID is used.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte contains ordinal key number into application

23<sup>rd</sup> byte contains checksum

If no error, i.e. error code is CARD\_OPERATION\_OK, device answer with RSP packet and after that also the RSP\_EXT packet of 12 bytes.

RSP\_Val0 and RSP\_Val1 are not in use.

array from 1<sup>st</sup> to 7<sup>th</sup> byte of RSP\_EXT contains 7 bytes length card UID

8<sup>th</sup> and 9<sup>th</sup> bytes represents error code of operation (b9 \* 256 + b8)

10<sup>th</sup> and 11<sup>th</sup> bytes represents execution time of command

12<sup>th</sup> byte is checksum.

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

**Example:**

Authentication using the internal key ordinal number 3, AID = 0xF00001, ordinal key number into application is 1.

<b>CMD</b>	55 80 AA 17 00 00 6F	(send command 80), 6F
<b>checksum</b>		
<b>ACK</b>	AC 80 CA 17 00 00 F8	(ACK OK)
<b>CMD_EXT</b>	01 03 00 00 00 00 00 00 00 00 00 00	(internal key uses so AES
<b>key</b>	00 00 00 00 00 00 01 00 F0 F9	bytes may have any value
<b>(all</b>		00), F9 checksum)
<b>RSP</b>	DE 80 ED 0B 00 00 AC	(RSP command 80, 12 bytes
		follows, 0B checksum)
<b>RSP_EXT</b>	04 01 02 03 05 06 07 B9 0B 0A 00 BF	(UID is 04010203050607,
<b>error</b>		code is 0BB9, execution
<b>time</b>		is 000A , checksum is BF

**DESFIRE\_FREE\_MEM (0x8D)**

Command returns the available bytes on the card

The CMD\_EXT set is not in use.

The CMD\_Par0 and CMD\_Par1 are not in use.

If no error, i.e. error code is CARD\_OPERATION\_OK, device answer with RSP packet and after that also the RSP\_EXT packet of 9 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

array from 5<sup>th</sup> to 8<sup>th</sup> of RSP\_EXT contains quantity of available bytes on card

9<sup>th</sup> byte is checksum



**Example:**

```

CMD          55 8D AA 00 00 00 79          (send CMD 6D, 79 checksum)
RSP          DE 8D ED 09 00 00 BE          (RSP command 8D, 9 byte follows,
BE                                                  checksum)
RSP_EXT      B9 0B 0A 00 E8 03 00 00 5A    (error code 0BB9, execution time
000A,                                                  free mem 000003E8 i.e. 1000)

```

**DESFIRE\_FORMAT\_CARD(0x8C)**

Function releases all allocated user memory on the card. All applications will be deleted, also all files within those applications will be deleted. Only the card master key, and card master key settings will not be deleted. This operation requires authentication with the card master key.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

19<sup>th</sup> byte is checksum

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

**Example:**

Authentication using the internal key ordinal number 1

```
CMD      55 8C AA 13 00 00 67      (send command 8C), 67
checksum
ACK      AC 8C CA 13 00 00 00      (ACK OK)

CMD_EXT  01 01 00 00 00 00 00 00 00 00 00 00 00 00  (internal key uses so AES
key                                             bytes may have any value
(all
                                                00), 07 checksum)
RSP      DE 8C ED 05 00 00 C1      (RSP command 8C, 5 byte follows,
BD checksum)
RSP_EXT  B9 0B AC 0D 1A (error code 0BB9, execution time 0DAC)
```

### **DESFIRE\_SET\_CONFIGURATION(0x8B)**

Function allows you to activate the Random ID option, and/or Format disable option.

If these options are activated, then they can not be returned to the factory setting (Random ID disabled, Format card enabled).

This operation requires authentication with the card master key.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

19<sup>th</sup> byte is 1 if Random ID enabled or 0 if Random ID disabled

20<sup>th</sup> byte is 1 if format card disabled or 0 if format card enabled

21<sup>st</sup> byte is checksum

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b_2 * 256 + b_1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### **Example:**

Authentication using the internal key ordinal number 1, Random ID enabled, format card disabled

<b>CMD</b>	<b>55 8B AA 15 00 00 68</b>	<b>(send command 8B), 68</b>
<b>checksum</b>		
<b>ACK</b>	<b>AC 8B CA 15 00 00 FF</b>	<b>(ACK OK)</b>
<b>CMD_EXT</b>	<b>01 01 00 00 00 00 00 00 00 00 00 00 00</b>	<b>(internal key uses so AES</b>
<b>key</b>		
	<b>00 00 00 00 00 00 01 00 08</b>	<b>bytes may have any value</b>
<b>(all</b>		
<b>format card</b>		<b>00), Random ID 01,</b>
		<b>00, 08 checksum)</b>
<b>RSP</b>	<b>DE 8B ED 05 00 00 C4</b>	<b>(RSP command 8B, 5 byte</b>
<b>follows,</b>		
		<b>BD checksum)</b>
<b>RSP_EXT</b>	<b>B9 0B 1A 00 AF</b>	<b>(error code 0BB9, execution time</b>
<b>001A)</b>		

### **DESFIRE\_GET\_KEY\_CONFIG(0x87)**

Function allows to get card master key and application master key configuration settings. In addition it returns the maximum number of keys which can be stored within selected application.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte contains checksum.

If no error, i.e. error code is CARD\_OPERATION\_OK, device answer with RSP packet and after that also the RSP\_EXT packet of 7 bytes.

RSP\_Val0 and RSP\_Val1 are not in use.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation (b2 \* 256 + b1)

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is key settings

6<sup>th</sup> byte is maximum number of keys within selected application.

7<sup>th</sup> byte is checksum

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 2, AID = 0xF00001

**CMD**            55 87 AA 16 00 00 75 (send command 87), 75 checksum

**ACK**            AC 87 CA 16 00 00 FE (ACK OK)

**CMD\_EXT**    01 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 F0  
CA(internal key uses so AES key bytes may have any value (all 00), CA  
checksum)

**RSP**            DE 87 ED 07 00 00 BA                    (RSP command 87, 7 bytes  
follows, BA checksum)

**RSP\_EXT**    B9 0B 1A 00 09 03 A9            (error code 0BB9, execution time  
001A, key settings 9, maximum number of key 3)

### **DESFIRE\_CHANGE\_KEY\_CONFIG(0x88)**

Function allows to set card master key, and application master key configuration settings.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is key settings

23<sup>rd</sup> byte contains checksum.

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 2, AID = 0xF00001, key settings is 9

**CMD**            55 88 AA 17 00 00 67 (send command 88), 67 checksum  
**ACK**            AC 88 CA 17 00 00 00 (ACK OK)

**CMD\_EXT**      01 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 F0  
 09 02 (internal key uses so AES key bytes may have any value (all 00), 02  
 checksum)

**RSP**            DE 88 ED 05 00 00 C6                    (RSP command 88, 5 bytes  
 follows, C5 checksum)

**RSP\_EXT**      B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

### **DESFIRE\_CHANGE\_AES\_KEY(0x86)**

Function allow to change any AES key on the card. Changing the card master key require current card master key authentication. Authentication for the application keys changing depend on the application master key settings (which key uses for authentication).

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT bit 0 set if uses internal AES key for authentication, bit 1 set if internal AES key uses as new key, bit 3 set if internal AES key uses as old key, high nibble is ordinal number of internal AES key which uses as old key, if they uses.

2<sup>nd</sup> byte of the CMD\_EXT low nibble is ordinal number of internal AES key which uses for authentication or 0 if uses external AES key, high nibble is ordinal number of internal AES key which uses as new key of 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key for authentication

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is key number into application which uses for authentication

array from 23<sup>rd</sup> to 38<sup>th</sup> byte of CMD\_EXT contains new AES key

38<sup>th</sup> byte is key number into application that will be changed

array from 39<sup>th</sup> to 54<sup>th</sup> byte of CMD\_EXT contains new AES key

55<sup>th</sup> byte contains checksum.

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### Example:

Change the key number 2, into AID 0xF00001. Authentication with master application key key number 0. Key for authentication is internal key number 1, new key is internal key number 2, and old key is internal key number 3.

**CMD**            55 86 AA 37 00 00 55 (send command 88, 0x37 bytes follows 55  
checksum)

**ACK**            AC 86 CA 37 00 00 DE (ACK OK)

**CMD\_EXT**      33 21 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 F0  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 E8 (internal key uses so AES key bytes may  
have any value (all 00), E8 checksum)

**RSP**            DE 86 ED 05 00 00 B7                    (RSP command 86, 5 bytes  
follows, C5 checksum)

**RSP\_EXT**      B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

### **DESFIRE\_CREATE\_APPLICATION(0x84)**

Function allows to create new application on the card. Is the card master key authentication is required, depend on the card master key settings. Maximal number of applications on the card is 28. Each application is linked to set of up 14 different user definable access keys.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is 1 if authentication required, or 0 if no need the authentication

23<sup>rd</sup> byte is application key settings

24<sup>th</sup> byte is maximal number of keys into application

25<sup>th</sup> contains checksum.

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is **READER\_ERROR** or **NO\_CARD\_DETECTED**, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command  
5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 1, AID = 0xF00002, key settings is 9, maximal number of application keys is 3, authentication required

**CMD**            55 84 AA 19 00 00 69 (send command 84), 69 checksum  
**ACK**            AC 84 CA 19 00 00 02 (ACK OK)

**CMD\_EXT**      01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 F0  
01 09 03 00 (internal key uses so AES key bytes may have any value (all  
00), 00 checksum)

**RSP**            DE 84 ED 05 00 00 B9                            (RSP command 84, 5 bytes  
follows, B9 checksum)

**RSP\_EXT**      B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

### **DESFIRE\_DELETE\_APPLICATION(0x89)**

Function allows to deactivate application on the card. AID allocation is removed, but deleted memory blocks can only recovered by using Format card function.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte contains checksum

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b2 * 256 + b1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 1, AID = 0xF00002

**CMD**            55 89 AA 16 00 00 67 (send command 89), 67 checksum  
**ACK**            AC 89 CA 16 00 00 00 (ACK OK)

**CMD\_EXT**      01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 F0  
**F9** (internal key uses so AES key bytes may have any value (all 00), F9  
checksum)

**RSP**            DE 89 ED 05 00 00 C6                    (RSP command 89, 5 bytes  
follows, C6 checksum)

**RSP\_EXT**      B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

### **DESFIRE\_CREATE\_STD\_FILE(0x85)**

Function allows to create file for the storage unformatted user data within existing application on the card. Maximal number of files into application is 32. The file will be created in the currently selected application. Is the application master key authentication is required, depend on the application master key settings.

Communication settings define communication mode between reader and card. The communication modes are:

- plain communication communication settings value is 0x00
- plain communication secured by MACing communication settings value is 0x01
- fully enciphered communication communication settings value is 0x11

Access rights for read, write, read&write and changing, references certain key within application's keys (0 – 13). If value is 14, this means free access, independent of previous authentication. If value is 15, this means deny access (for example if write access is 15 then the file type is read only).

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is ID of file that will be created (0 – 31)

23<sup>rd</sup> and 24<sup>th</sup> bytes represented access rights for read, write, read&write and changing

array from 25<sup>th</sup> to 28<sup>th</sup> of CMD\_EXT contains file size in bytes

29<sup>th</sup> byte is 1 if authentication required, or 0 if no need the authentication

30<sup>th</sup> byte is communication settings

31<sup>st</sup> byte is checksum

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is **READER\_ERROR** or **NO\_CARD\_DETECTED**, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation (b2 \* 256 + b1)



3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command  
5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 1, AID = 0xF00002, authentication required, file ID is 1, communication settings is 0x11, access rights is 0x2110 (read with key 2, write with key 1, read&write with key 1, changing with key 0), file size is 1000 (0x000003E8)

**CMD**            55 85 AA 1F 00 00 67 (send command 89) , 67 checksum  
**ACK**            AC 85 CA 1F 00 00 00 (ACK OK)

**CMD\_EXT**    01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 F0  
01 10 21 E8 03 00 00 01 11 40 (internal key uses so AES key bytes may  
have any value (all 00) , 40 checksum)

**RSP**            DE 85 ED 05 00 00 BA                    (RSP command 85, 5 bytes  
follows, BA checksum)

**RSP\_EXT**    B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

### DESFIRE\_DELETE\_FILE(0x8A)

Function deactivates a file within currently selected application. Allocated memory blocks associated with deleted file not set free. Only format card function can delete the memory blocks. Is the application master key authentication is required, depend on the application master key settings.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is ID of file that will be deleted (0 – 31)

23<sup>rd</sup> byte is 1 if authentication required, or 0 if no need the authentication

24<sup>th</sup> byte is checksum

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is READER\_ERROR or NO\_CARD\_DETECTED, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation (b2 \* 256 + b1)

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

## Example:

Authentication using the internal key ordinal number 1, AID = 0xF00002, authentication required, file ID is 1

**CMD**            55 8A AA 18 00 00 74 (send command 8A) , 74 checksum

**ACK**            AC 8A CA 18 00 00 FB (ACK OK)

**CMD\_EXT**    01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 F0  
01 01 F9 (internal key uses so AES key bytes may have any value (all  
00) , F9 checksum)

**RSP**            DE 8A ED 05 00 00 C3                    (RSP command 8A, 5 bytes  
follows, C3 checksum)

**RSP\_EXT**    B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

## **DESIRE\_READ\_FROM\_STD\_FILE(0x83)**

Function allow to read data from Standard Data File. Read command requires a preceding authentication either with the key specified for Read or Read&Write access.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is application key number for reading

23<sup>rd</sup> byte is ID of file (0 – 31)

23<sup>rd</sup> byte is 1 if authentication required, or 0 if no need the authentication

24<sup>th</sup> and 25<sup>th</sup> bytes represents start position for read operation within file

26<sup>th</sup> and 27<sup>th</sup> bytes represents number of data to be read

28<sup>th</sup> byte is communication settings

29<sup>th</sup> byte is checksum

Reading the data is specific and is done in a loop. Reads one data, and if it is 0, then reads another that indicates how much data follows in the package. This is repeated until the required amount of data read. If the first data is different from 0, then reader will be sent standard response.

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is **READER\_ERROR** or **NO\_CARD\_DETECTED**, device answer with RSP\_EXT packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with RSP\_EXT packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b_2 * 256 + b_1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 3, AID = 0xF00002, authentication required, file ID is 1, reading key number is 2, bytes for read 50 from start address 10, communication settings 0x11

**CMD**            55 83 AA 1D 00 00 68 (send command 83), 68 checksum

**ACK**            AC 83 CA 1D 00 00 FB (ACK OK)

**CMD\_EXT**      01 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 F0  
02 01 01 0A 00 32 00 11 E2 (internal key uses so AES key bytes may have  
any value (all 00), E2 checksum)

**DATA**           00 32 01 02 03 04 05 06 07 08 09 0A 01 02 03 04 05 06 07 08 09  
0A 01 02 03 04 05 06 07 08 09 0A 01 02 03 04 05 06 07 08 09 0A 01 02 03  
04 05 06 07 08 09 0A

**RSP**            DE 8A ED 05 00 00 C3                    (RSP command 8A, 5 bytes  
follows, C3 checksum)

**RSP\_EXT**      B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

### **DESFIRE\_WRITE\_TO\_STD\_FILE(0x82)**

Function allow to write data to Standard Data File, or to Backup Data File. Write command requires a preceding authentication either with the key specified for Write or Read&Write access.

CMD\_Par0 and CMD\_Par1 are 0

1<sup>st</sup> byte of the CMD\_EXT is 1 if uses internal AES key, or 0 if uses external AES key

2<sup>nd</sup> byte of the CMD\_EXT contains ordinal number of internal AES key, or 0 if uses external AES key

array from 3<sup>rd</sup> to 18<sup>th</sup> byte of CMD\_EXT contains AES key

array from 19<sup>th</sup> to 21<sup>st</sup> byte of CMD\_EXT contains AID (Application ID 3 bytes)

22<sup>nd</sup> byte is application key number for writing

23<sup>rd</sup> byte is ID of file (0 – 31)

24<sup>th</sup> byte is 1 if authentication required, or 0 if no need the authentication

25<sup>th</sup> and 26<sup>th</sup> bytes represents start position for read operation within file

27<sup>th</sup> and 28<sup>th</sup> bytes represents number of data to be write

29<sup>th</sup> byte is communication settings

array from 30<sup>th</sup> to 30 + block size number of data for writing contains maximal 160 data for writing

31 + block size byte is checksum

If you want to enter more than 160 bytes, then it is done in blocks of up to 160 bytes. After the first block of data reader sent 0xAD if necessary to receive more data, or 0xDD if no need more data, or at any error. When you receive 0xAD then sends a packet in which the first byte indicates how many bytes follow. When you receive 0xDD then follow standard response.

RSP\_Val0 and RSP\_Val1 are not in use.

If error code is `READER_ERROR` or `NO_CARD_DETECTED`, device answer with `RSP_EXT` packet of 3 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents execution time of command

3<sup>rd</sup> byte is checksum.

In other cases, device answer with `RSP_EXT` packet of 5 bytes.

1<sup>st</sup> and 2<sup>nd</sup> bytes represents error code of operation ( $b_2 * 256 + b_1$ )

3<sup>rd</sup> and 4<sup>th</sup> bytes represents execution time of command

5<sup>th</sup> byte is checksum.

### Example:

Authentication using the internal key ordinal number 3, AID = 0xF00002, authentication required, file ID is 1, writing key number is 1, bytes for write 50 from start address 10, communication settings 0x11

**CMD**            55 82 AA 51 00 00 33 (send command 82), 33 checksum

**ACK**            AC 82 CA 51 00 00 BC (ACK OK)

**CMD\_EXT**      01 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 F0  
 01 01 01 0A 00 32 00 11 01 02 03 04 05 06 07 08 09 0A 01 02 03 04 05 06  
 07 08 09 0A 01 02 03 04 05 06 07 08 09 0A 01 02 03 04 05 06 07 08 09 0A  
 01 02 03 04 05 06 07 08 09 0A CRC (internal key uses so AES key bytes  
 may have any value (all 00), CRC checksum)

**DATA**            DD (no need more data)

**RSP**            DE 82 ED 05 00 00 BB (RSP command 82, 5 bytes  
 follows, BB checksum)

**RSP\_EXT**      B9 0B 1A 00 AF (error code 0BB9, execution time 001A)

## **COMMANDS FOR READER SETTINGS**

### **SET\_BAD\_SELECT\_NR\_MAX(0x3F)**

The function allows you to set the number of unsuccessful card selections before it can be considered that the card is not placed on the reader. Period between two card selections is approximately 10ms. Default value of this parameter is 20 i.e. 200ms. This parameter can be set in the range of 0 to 254.

The CMD\_EXT set is not in use.

CMD\_Par0 is bad select card number maximal

CMD\_Par1 = (CMD\_Par0 xor A3) + 7

The RSP\_EXT is not in use

#### **Example:**

Bad select card maximal is 10

CMD\_Par0 = 0x0A, CMD\_Par1 = (0A xor A3) + 7 = B0

```
CMD      55 3F AA 00 0A B0 81 (send command 3F) , 81 checksum
RSP      DE 3F ED 00 00 00 13
```

### **GET\_BAD\_SELECT\_NR\_MAX(0x44)**

The function returns value of maximal unsuccessful card selections, which is set in reader.

The CMD\_EXT set is not in use.

CMD\_Par0 and CMD\_Par1 are 0

RSP\_EXT - 1<sup>st</sup> byte is maximal value of bad select card number

#### **Example:**

```
CMD      55 44 AA 00 00 00 C2 (send command 44) , C2 checksum
RSP      DE 44 ED 02 00 00 7C
RSP_EXT  0A 11 (number is 0x0A)
```

## **FUNCTIONS FOR THE READER LOW POWER MODE CONTROL**

### **ENTER\_SLEEP\_MODE (0x46)**

Function allows the low power reader mode. Reader is in sleep mode. RF field is turned off. The reader is waiting for the command to return to normal working mode.

The CMD\_EXT set is not in use.

CMD\_Par0 and CMD\_Par1 are 0

The RSP\_EXT is not in use.

#### **Example:**

```
CMD      55 46 AA 00 00 00 C0 (send command 46) , C0 checksum
RSP      DE 46 ED 00 00 00 7C
```

### **LEAVE\_SLEEP\_MODE (0x47)**

Function allows return from low power reader mode to normal working mode.

The CMD\_EXT set is not in use.

CMD\_Par0 and CMD\_Par1 are 0

The RSP\_EXT is not in use.

#### **Example:**

```
WAKE UP BYTE  00 (send just before command)
CMD      55 47 AA 00 00 00 BF (send command 47) , BF checksum
RSP      DE 47 ED 00 00 00 7B
```

### **AUTO\_SLEEP\_SET (0x4D)**

supported from firmware version 3.8.18

#### **Command description:**

This function permanently set auto-sleep functionality of the device. Valid value for the CMD\_Par0 range is from 1 to 254 seconds. To permanently disable auto-sleep functionality use 0 or 0xFF for the CMD\_Par0 value.

The CMD\_EXT is not in use.

CMD\_Par1 are 0 (not in use).

The RSP\_EXT is not in use.

## **AUTO\_SLEEP\_GET (0x4E)**

supported from firmware version 3.8.18

### **Command description:**

This command returns permanently configured auto-sleep wait seconds.

The CMD\_EXT is not in use.

CMD\_Par0 and CMD\_Par1 are 0 (not in use).

The RSP\_EXT is not in use.

RSP\_Val0 containing configured auto-sleep wait seconds.

RSP\_Val1 is 0 (not in use).

## **Commands for Reader NTAG Emulation Mode**

### **WRITE\_EMULATION\_NDEF (0x4A)**

supported from firmware version 3.8.0

### **Command description:**

Command store a message record for NTAG emulation mode in to the reader. The CMD\_EXT is used and contains NDEF message for tag emulation mode.

1<sup>st</sup> and 2<sup>nd</sup> byte of the CMD\_EXT set contains length of the following NDEF message (parameter called ndef\_len).

next ndef\_len bytes contains NDEF message.

last byte of the CMD\_EXT set contains checksum

### **Example:**

(NDEF message is URI type with "[www.d-logic.net](http://www.d-logic.net)" payload):

<b>CMD</b>	<b>55 4A AA 16 00 00 AA</b>
<b>ACK</b>	<b>AC 4A CA 16 00 00 41</b>
<b>CMD_EXT</b>	<b>14 00 03 10 D1 01 0C 55 01 64 2D 6C 6F 67 69 63 2E 6E 65 74 FE 0E</b>
<b>RSP</b>	<b>DE 4A ED 00 00 00 80</b>

### **Possible error codes:**

`WRITE_VERIFICATION_ERROR = 0x70`

`MAX_SIZE_EXCEEDED = 0x10`

### **TAG\_EMULATION\_START (0x48)**

supported from firmware version 3.8.0

Put the reader permanently in a NDEF tag emulation mode. Only way for a reader to exit from this mode is to receive the TAG\_EMULATION\_STOP command.

In this mode, the reader can only answer to the following commands:

`WRITE_EMULATION_NDEF (0x4A)`

`TAG_EMULATION_STOP (0x49)`

`TAG_EMULATION_START (0x48)`

`GET_READER_TYPE (0x10)`

`GET_READER_SERIAL (0x11)`

`GET_FIRMWARE_VERSION (0x29)`

`GET_HARDWARE_VERSION (0x2A)`

`GET_BUILD_NUMBER (0x2B)`

`GET_SERIAL_NUMBER (0x40)`

Issuing another commands in this mode, results with the following error code:

`FORBIDDEN_IN_TAG_EMULATION_MODE = 0x90`

#### **Possible error codes:**

`WRITE_VERIFICATION_ERROR = 0x70`

*(command resulting in a direct write to a device non-volatile memory)*

#### **Example:**

CMD	55 48 AA 00 00 00 BE
RSP	DE 48 ED 00 00 00 82

### **TAG\_EMULATION\_STOP (0x49)**

supported from firmware version 3.8.0

Allows the reader permanent exit from a NDEF tag emulation mode.

#### **Possible error codes:**

`WRITE_VERIFICATION_ERROR = 0x70`

*(command resulting in a direct write to a device non-volatile memory)*



**Example:**

```
CMD      55 49 AA 00 00 00 BD
RSP      DE 49 ED 00 00 00 81
```

**Ad-Hoc emulation mode:**

This mode enables user controlled emulation from the user application. There is “nfc-rfid-reader-sdk/ufr-examples-ad\_hoc\_emulation-c” console example written in C, using our uFCoder library (see uFR API). This example demonstrate usage of the uFCoder library functions that implement sending of the following commands:

***AD\_HOC\_EMULATION\_START (0x76)***

supported from firmware version 3.9.34

Put uFR in emulation mode with ad-hoc emulation parameters (see. SET\_AD\_HOC\_EMULATION\_PARAMS and GET\_AD\_HOC\_EMULATION\_PARAMS). uFR stays in emulation mode until AD\_HOC\_EMULATION\_STOP command is sent or reader reset.

- The CMD\_EXT set is not in use.
- CMD\_Par0 and CMD\_Par1 are not in use.
- The RSP\_EXT is not in use

**Example:**

```
CMD      55 76 AA 00 AA CC F6
RSP      DE 76 ED 00 00 00 4C
```

***AD\_HOC\_EMULATION\_STOP (0x77)***

supported from firmware version 3.9.34

Terminate uFR ad-hoc emulation mode.

- The CMD\_EXT set is not in use.
- CMD\_Par0 and CMD\_Par1 are not in use.
- The RSP\_EXT is not in use

**Example:**

```
CMD      55 77 AA 00 AA CC F5
RSP      DE 77 ED 00 00 00 4B
```

**GET\_EXTERNAL\_FIELD\_STATE (0x9F)**

supported from firmware version 3.9.34

This command returns external field state when uFR is in ad-hoc emulation mode.

The CMD\_EXT set is not in use.

CMD\_Par0 and CMD\_Par1 are not in use.

RSP\_Val0 is 0 if external field isn't present or 1 if field is present.

RSP\_Val1 is not in use.

The RSP\_EXT is not in use

**Example:**

```
CMD      55 9F AA 00 AA CC 0D
RSP      DE 9F ED 00 01 00 B4
```

**GET\_AD\_HOC\_EMULATION\_PARAMS (0x9D)**

supported from firmware version 3.9.35

This command returns current ad-hoc emulation parameters. On uFR power on or reset ad-hoc emulation parameters are set back to their default values.

The CMD\_EXT set is not in use.

CMD\_Par0 and CMD\_Par1 are not in use.

RSP\_Val0 contains current ad-hoc threshold parameters. Default value is 0xF7.

RSP\_Val1 contains current ad-hoc receiver gain and RF level values of the RFCfgReg register (most significant bit of this value should be 0 all the time). Default value is 0x79.

The RSP\_EXT is not in use

**Example:**

```
CMD      55 9D AA 00 AA CC 0B
RSP      DE 9D ED 00 F7 79 27
```

**SET\_AD\_HOC\_EMULATION\_PARAMS (0x9E)**

supported from firmware version 3.9.35

This command set ad-hoc emulation parameters. On uFR power on or reset ad-hoc emulation parameters are set back to their default values.

The CMD\_EXT set is not in use.

CMD\_Par0 contains current ad-hoc threshold parameters. Default value is 0xF7.

CMD\_Par1 contains current ad-hoc receiver gain and RF level values of the RFCfgReg register (most significant bit of this value should be 0 all the time). Default value is 0x79.

**Example:**

```
CMD      55 9E AA 00 F7 79 F6
RSP      DE 9E ED 00 00 00 B4
```

**SET\_SPEED\_PERMANENTLY (0x4B)**

supported from firmware version 3.8.4

Permanently set the requested transceive data rates between reader and ISO14443 – 4A card / tag.

CMD\_EXT set not in use.

CMD\_Par0 containing requested transmit speed constant

CMD\_Par1 containing requested receive speed constant

The RSP\_EXT not in use.

Valid speed constants are:

<i>Const</i>	<i>Requested speed</i>
0	106 kbps (default)
1	212 kbps
2	424 kbps

**Possible error codes:**

WRITE\_VERIFICATION\_ERROR = 0x70

(command resulting in a direct write to a device non-volatile memory)

**Example:**

```
CMD      55 4B AA 00 02 02 BB
RSP      DE 4B ED 00 00 00 7F
```

## GET\_SPEED\_PARAMETERS (0x4C)

supported from firmware version 3.8.4

This command returns permanently configured transceive data rates between reader and ISO14443 – 4A card / tag.

CMD\_EXT set not in use.

The RSP\_EXT not in use.

RSP\_Val0 containing configured transmit speed constants

RSP\_Val1 containing configured receive speed constants

Valid speed constants are:

<i>Const</i>	<i>Configured speed</i>
0	106 kbps (default)
1	212 kbps
2	424 kbps

### Example:

```
CMD      55 4C AA 00 00 00 BA
RSP      DE 4C ED 00 02 02 86
```

## Support for ISO 14443-4A protocol commands

### Basic commands

#### SET\_ISO14433\_4\_MODE (0x93)

supported from firmware version 3.9.36

After issuing this command, ISO 14443-4A tag in a field will be selected and RF field polling will be stopped. Furthermore all the others ISO 14443-4A protocol commands can be issued in a sequence (including APDU\_TRANSCEIVE). Last command in those sequences should be S\_BLOCK\_DESELECT.

**Example:**

```
CMD      55 93 AA 00 AA CC 11
RSP      DE 93 ED 00 00 00 A7
```

***I\_BLOCK\_TRANSCEIVE (0x90)***

supported from firmware version 3.9.36

Used to convey information for use by the application layer.

CMD\_Par0 contains command specific flags (0x0C additional chained i block , 0x04 single i block)

CMD\_Par1 containing timeout value in [ms]

CMD\_EXT contains i-block body.

RSP\_EXT contains i-block response.

### **R\_BLOCK\_TRANSCEIVE (0x91)**

supported from firmware version 3.9.36

Used to convey positive or negative acknowledgements. An R-block never contains an INF field. The acknowledgement relates to the last received block.

CMD\_Par0 contains acknowledge flag (1 = ACK, 0 = NOT ACK)

CMD\_Par1 containing timeout value in [ms]

CMD\_EXT not in use.

RSP\_EXT contains i-block response.

### **S\_BLOCK\_DESELECT (0x92)**

supported from firmware version 3.9.36

Issue this command to deselect tag and restore RF field polling. This command is mandatory at the end of any

ISO 14443-4A protocol command sequence.

#### **Example:**

<b>CMD</b>	<b>55 92 AA 00 64 00 10</b>
<b>RSP</b>	<b>DE 92 ED 00 00 00 A8</b>

## Support for APDU commands in ISO 14443-4A tags

### APDU\_TRANSCEIVE (0x94)

supported from firmware version 3.9.39

Some ISO 14443-4A tags supports the APDU message structure according to ISO/IEC 7816-4. For more details you have to check the manual for the tags that you planning to use.

Issuing APDU\_TRANSCEIVE command you will send C-APDU to ISO 14443-4A tag selected using SET\_ISO14433\_4\_MODE. After successfully executed APDU\_TRANSCEIVE command uFR returns byte array which contains R-APDU including data field (body) following by the trailer (SW1 and SW2 APDU status bytes).

CMD\_Par0 not in use

CMD\_Par1 containing timeout value in [ms]

CMD\_EXT contains C-APDU (i.e. {CLA, INS, P0, P1, Lc, ... Nc bytes ... , Le} )

RSP\_EXT contains R-APDU including data field (body) following by the trailer (SW1 and SW2 APDU status bytes).

### Example:

Issuing NDEF Tag Application Select command: 00 A4 04 00 07 D2 76 00 00 85 01 01 00

```
CMD      55 94 AA 0E 00 CC B0
ACK      AC 94 CA 0E 00 CC 37
CMD_EXT  00 A4 04 00 07 D2 76 00 00 85 01 01 00 8D
RSP      DE 94 ED 03 00 00 AB
RSP_EXT  90 00 97
```

## PKI infrastructure and digital signature support

### Fully supported from firmware version 3.9.55

In our product range, we have special cards called “D-Logic JCAp” (working title), which contains support for PKI infrastructure and digital signing. To use these features you have to implement specific APDU command sequences using APDU\_TRANSCEIVE command described before. We have PKI infrastructure and digital signature support implemented in our API (for reference read “uFR Series NFC reader’s API”).

**Appendix: ERROR CODES**

<b>ERROR</b>	<b>VALUE</b>
OK	0x00
COMMUNICATION_ERROR	0x01
CHKSUM_ERROR	0x02
READING_ERROR	0x03
WRITING_ERROR	0x04
BUFFER_OVERFLOW	0x05
MAX_ADDRESS_EXCEEDED	0x06
MAX_KEY_INDEX_EXCEEDED	0x07
NO_CARD	0x08
COMMAND_NOT_SUPPORTED	0x09
FORBIDDEN_DIRECT_WRITE_IN_SECTOR_TRAILER	0x0A
ADDRESSED_BLOCK_IS_NOT_SECTOR_TRAILER	0x0B
WRONG_ADDRESS_MODE	0x0C
WRONG_ACCESS_BITS_VALUES	0x0D
AUTH_ERROR	0x0E
PARAMETERS_ERROR	0x0F
MAX_SIZE_EXCEEDED	0x10
UNSUPPORTED_CARD_TYPE	0x11
COUNTER_ERROR	0x12
WRITE_VERIFICATION_ERROR	0x70
BUFFER_SIZE_EXCEEDED	0x71
VALUE_BLOCK_INVALID	0x72
VALUE_BLOCK_ADDR_INVALID	0x73
VALUE_BLOCK_MANIPULATION_ERROR	0x74
WRONG_UI_MODE	0x75
KEYS_LOCKED	0x76
KEYS_UNLOCKED	0x77
WRONG_PASSWORD	0x78



CAN_NOT_LOCK_DEVICE	0x79
CAN_NOT_UNLOCK_DEVICE	0x7A
DEVICE_EEPROM_BUSY	0x7B
RTC_SET_ERROR	0x7C
EEPROM_ERROR	0x7D
NO_CARDS_ENUMERATED	0x7E
CARD_ALREADY_SELECTED	0x7F
WRONG_CARD_TYPE	0x80
AIS_FOREIGN_CARD	0x81
AIS_WRONG_FORMAT_CARD	0x82
FORBIDDEN_IN_TAG_EMULATION_MODE	0x90

### Change log:

Date	Description	Document revision	refers to the firmware ver.
2018-05-29	PKI infrastructure and digital signature support	1.1	3.9.55
2018-05-29	Changed date format in a Change log. Now we use more universal 'yyyy-mm-dd' date format.	1.1	-
2017-06-29	Support for APDU commands in ISO 14443-4A tags	1.0	3.9.39
2017-05-23	Support for ISO 14443-4A protocol commands	1.0	3.9.36
2017-05-03	Commands for a Ad-Hoc emulation mode parameters manipulation. (GET_AD_HOC_EMULATION_PARAMS and SET_AD_HOC_EMULATION_PARAMS).	1.0	3.9.35
2017-05-03	Ad-Hoc emulation mode commands.	1.0	3.9.34
2016-08-06	FAST_READ ISO14443-3 command with LINEAR_READ utilisation.	1.0	3.9.14
2016-06-06	Title "Authentication mode considerations" changed to "Authentication mode considerations for Mifare Classic tags"	1.0	
2016-06-06	New Title "Authentication mode considerations for NTAG 21x and other T2T tags"	1.0	3.9.10