

Pepper C2 software manual

Manual version: v1.2¹

13/05/2026

Table of Contents

1. Configuration – C2 client software	5
1.1 RFID	5
2. RFID configuration.....	6
2.1 Polling configuration	6
2.1.1 Supported technologies.....	7
2.1.2 RFID power settings.....	7
2.1.3 Polling loop settings.....	7
2.1.4 Read memory settings.....	7
2.1.5 Polling events	7
2.2 Key storage	10
2.3 Known UID list.....	11
3. Communication interface – binary interface	12
3.1 Overview.....	12
3.2 Frame structure.....	12
3.3 CRC calculation.....	12
3.4 Pepper Cx Client – PC application	14
4. Commands list.....	16
4.1 Generic commands.....	16
4.1.1 Acknowledge frame (0x00)	16
4.1.2 Error response (0xFF).....	16
4.1.3 Dummy command (0x01)	19
4.1.4 Get tag count (0x02)	19
4.1.5 Get tag UID (0x03)	20
4.1.6 Activate TAG (0x04)	21

¹ The newest software manual can be found on our website: https://eccel.co.uk/wp-content/downloads/Pepper_C2/C2-software-manual.pdf

4.1.7	Halt (0x05).....	21
4.1.8	Set polling (0x06).....	21
4.1.9	Set key (0x07).....	22
4.1.10	Save keys (0x08).....	23
4.1.11	Reboot (0x0A).....	23
4.1.12	Get version (0x0B).....	24
4.1.13	Sleep command (0x0D).....	24
4.1.14	GPIO command (0x0E).....	25
4.1.15	Factory reset command (0x11).....	26
4.1.16	Protocol configuration (0x13).....	26
4.1.17	LED command (0x14).....	27
4.1.18	Polling setup (0x16).....	28
4.2	MIFARE Classics commands.....	36
4.2.1	Read block (0x20).....	36
4.2.2	Write block (0x21).....	36
4.2.3	Read value (0x22).....	37
4.2.4	Write value (0x23).....	38
4.2.5	Increment/decrement value (0x24).....	38
4.2.6	Transfer value (0x25).....	39
4.2.7	Restore value (0x26).....	40
4.2.8	Transfer-Restore value (0x27).....	40
4.3	MIFARE Ultralight commands.....	41
4.3.1	Read page (0x40).....	41
4.3.2	Write page (0x41).....	42
4.3.3	Get version (0x42).....	42
4.3.4	Read signature (0x43).....	43
4.3.5	Write signature (0x44).....	43
4.3.6	Lock signature (0x45).....	44
4.3.7	Read counter (0x46).....	44
4.3.8	Increment counter (0x47).....	45
4.3.9	Password auth (0x48).....	45
4.3.10	Check Tearing Event (0x4A).....	46

4.4	ICODE (ISO15693) commands.....	47
4.4.1	Inventory start (0x90)	47
4.4.2	Inventory next (0x91)	47
4.4.3	Read block (0x93)	48
4.4.4	Write block (0x94)	49
4.4.5	Lock block (0x95)	49
4.4.6	Write AFI (0x96)	50
4.4.7	Lock AFI (0x97)	50
4.4.8	Write DSFID (0x98)	50
4.4.9	Lock DSFID (0x99)	51
4.4.10	Get System Information (0x9A).....	51
4.4.11	Get multiple BSS (0x9B)	52
4.4.12	Password protect AFI (0x9C)	52
4.4.13	Read EPC (0x9D)	53
4.4.14	Get NXP System Information (0x9E).....	53
4.4.15	Get random number (0x9F)	54
4.4.16	Set password (0xA0)	54
4.4.17	Write password (0xA1)	55
4.4.18	Lock password (0xA2)	55
4.4.19	Protect page (0xA3)	56
4.4.20	Lock page protection (0xA4)	57
4.4.21	Get multiple block protection status (0xA5)	57
4.4.22	Destroy (0xA6).....	58
4.4.23	Enable privacy (0xA7)	58
4.4.24	Enable 64-bit password (0xA8)	59
4.4.25	Read signature (0xA9).....	59
4.4.26	Read config (0xAA)	59
4.4.27	Write config (0xAB).....	60
4.4.28	Pick random ID (0xAC)	61
4.4.29	Extended read block (0xB3)	61
4.4.30	Extended write block (0xB4)	62
4.4.31	ICODE custom command (15693) (0xBF).....	62

4.5	OTA upgrade	63
4.5.1	OTA begin (0xF0)	63
4.5.2	OTA firmware frame (0xF1)	63
4.5.3	OTA finish (0xF2)	64
5.	Revision history	65

1. Configuration – C2 client software

The C2 Pepper reader can be configured via a binary protocol using the Cx-client Windows software. The user can set up all RFID polling parameters, encryption keys, known tags, and more. This software also allows configuration of the USB interface and device UARTs. Additionally, it enables testing of all commands supported by the binary protocol. By default Cx-client software can be used with the C2 Pepper reader over USB cable.

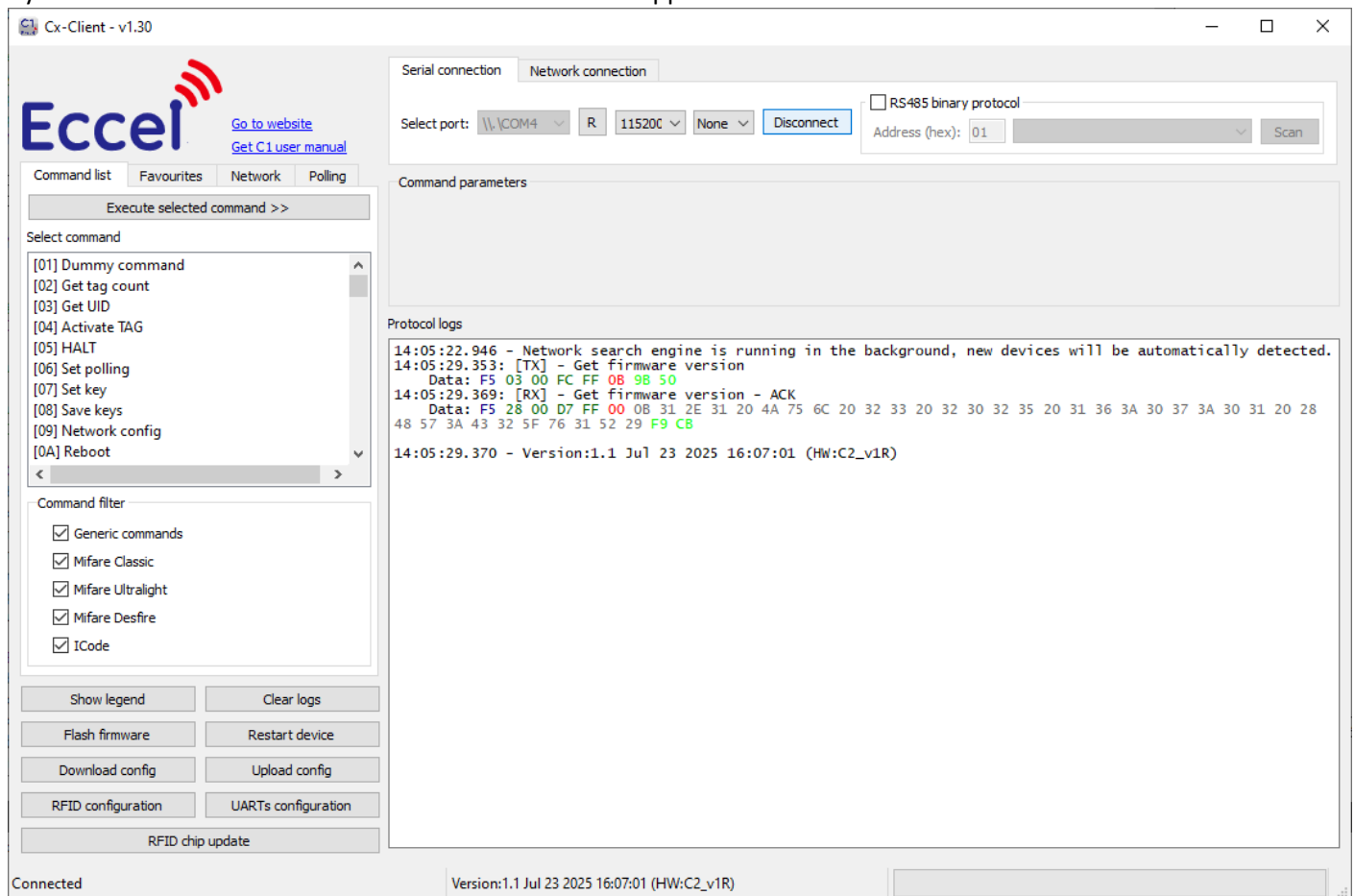


Figure 1. Cx-Client software

1.1 RFID

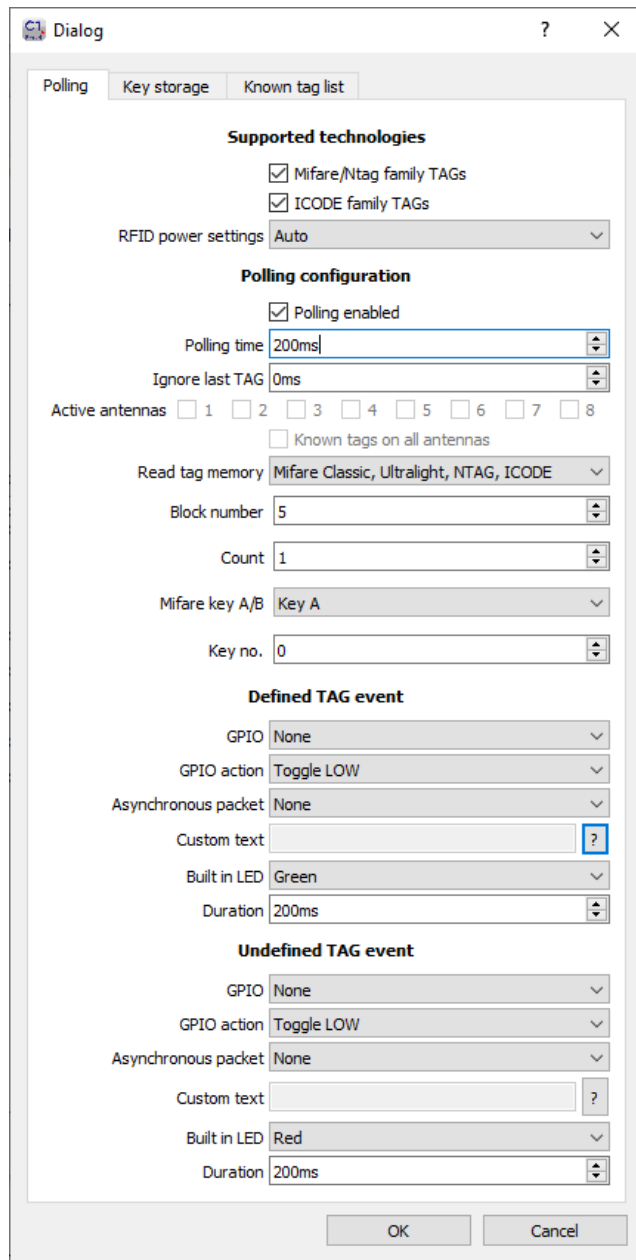
After connecting the Cx-client software to the reader, the software should display the device firmware version on the status bar. To set up RFID parameters, the user needs to click the RFID configuration button to open the RFID polling configuration window. This window contains three tabs required to configure all options related to the RFID functionality of the reader.

- Polling
- Known UIDs
- Key storage

2. RFID configuration

2.1 Polling configuration

In this mode the Pepper C2 device executes the continuous repeated enumerate tags UID command. Depending upon the polling settings, the module can execute some actions as described below. Because the module has built in memory, the user can store known UIDs, and polling mode can trigger different actions depending upon whether the UID is stored in the memory or not. (Whitelist)



The screenshot shows a 'Dialog' window with three tabs: 'Polling', 'Key storage', and 'Known tag list'. The 'Polling' tab is selected. The configuration is organized into several sections:

- Supported technologies:**
 - Mifare/Ntag family TAGs
 - ICODE family TAGs
 - RFID power settings: Auto (dropdown)
- Polling configuration:**
 - Polling enabled
 - Polling time: 200ms (spinner)
 - Ignore last TAG: 0ms (spinner)
 - Active antennas: 1, 2, 3, 4, 5, 6, 7, 8 (checkboxes)
 - Known tags on all antennas
 - Read tag memory: Mifare Classic, Ultralight, NTAG, ICODE (dropdown)
 - Block number: 5 (spinner)
 - Count: 1 (spinner)
 - Mifare key A/B: Key A (dropdown)
 - Key no.: 0 (spinner)
- Defined TAG event:**
 - GPIO: None (dropdown)
 - GPIO action: Toggle LOW (dropdown)
 - Asynchronous packet: None (dropdown)
 - Custom text: [?] (text box)
 - Built in LED: Green (dropdown)
 - Duration: 200ms (spinner)
- Undefined TAG event:**
 - GPIO: None (dropdown)
 - GPIO action: Toggle LOW (dropdown)
 - Asynchronous packet: None (dropdown)
 - Custom text: [?] (text box)
 - Built in LED: Red (dropdown)
 - Duration: 200ms (spinner)

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Figure 2. RFID polling configuration

As shown in Figure 2 above, you can configure different actions for a defined tag (stored in device memory) and undefined. Both actions have five parameters to configure:

2.1.1 Supported technologies

The user can select what transponder technology is supported by the reader, MIFARE/NTAG and ICODE technology. Due to this option polling time is shorter and the device can be used with only one of the above two technologies when fastest transponder read performance is needed.

2.1.2 RFID power settings

This setting is related to the RFID output power on the RFID antenna. If AUTO is selected then the device is using dynamic power control function to provide optimum power for the antenna. But if the user need to reduce RFID power to reduce the range or to limit RF emission then lower power settings can be selected from seven predefined levels where 7 is the maximum power on the antenna and 1 is minimum power.

2.1.3 Polling loop settings

These settings are related to the polling period for the RFID loop. By default the reader checks TAGs in range every 200ms. User can also specify “Ignore timeout” parameter. Thanks to this timeout when the same TAG is detected in range of the antenna it will be ignored. If the TAG is presented to the antenna before the selected ignore same tag timeout has expired, then the timeout is restarted.

2.1.4 Read memory settings

The Pepper C2 family supports reading memory content during the polling mode. This is useful if the user wants to read memory content + UID. The content of the memory is reported in the asynchronous packet when it is selected as Plain text or JSON.

Depending upon the transponder technology, the reader can read pages or blocks from MIFARE Classic with authorization, and other tags like Ultralight, NTAG tags and ICODE when the memory is not protected.

2.1.5 Polling events

The user can set up some automatic actions assigned to the reading events. Depending upon whether the TAG is stored on the known list or not, different events can be triggered. For both scenarios, the user can setup these fields:

- **GPIO** - user can select one of the dedicated GPIO to perform an action
- **GPIO action** – there are two options: toggle LOW or HIGH. If the configured action is to toggle HIGH, then the selected GPIO remains LOW until the event occurs and then toggles HIGH for a time defined in the Timeout field. If the selected action is to toggle LOW, then the GPIO remains HIGH until the event occurs and then toggles LOW.
- **Asynchronous packet** – the device can send packets over the communication protocol selected in the RFID tab. Three packet options are available:

- Binary packet format – with these settings, the module sends the frame in the binary protocol format. This is the best method if the user already uses binary protocol as the selected communication method. Here is an example:

Byte no.	0-4	5	6	7	8	9...	Last two bytes
Description	Command header	0xFE CMD ASYNC	0x03 CMD Get UID	Card type: 0x00 - ISO14443A 0x10 - ISO15693	SAK or DSFID	UID (4, 7 or 8 bytes)	CRC16

Example frame in the binary protocol format:

F5 0A 00 F5 FF FE 03 01 08 54 D4 F8 2A 73 64

F5 0A 00 F5 FF – command header

FE – CMD ASYNC (fixed value)

03 – CMD GET UID (fixed value)

01 - ISO/IEC 14443 Type A

08 - MIFARE Classic 1k (SAK value – 0x08)

54 D4 F8 2A – UID (4 bytes long)

73 64 – CRC16

- Plain text – the device sends text strings with basic information about the TAG eg:

UID:54D4F82A; TYPE:1; KNOWN:0<\r><\n>

- JSON frame – the module sends a JSON string using the configured communication method. This is the best option if you want to connect this device to IOT systems. Example below

```
{
  "type": "uid",
  "uid": "E0040100206D3A86",
  "string": "ICODE SLI",
  "tag_index": 4,
  "uptime_ms": 49242,
  "known_tag": false
}
```

Figure 3 JSON frame example

- Custom text format – the device is capable to send a text format frame for RFID event defined by the user. To specify the custom frame format the user can use special macros in the format defined below:

- %u - tag UID
- %a - antenna index

- %m - memory content (only if reading memory is setup correctly)
- %i - idx msg/tag counter since restart
- %t - timestamp in milliseconds
- %T - tag type 1-MIFARE family, 16 - ICODE
- %p - tag parameter. SAK for MIFARE family and DSFID for ICODE
- <CR> - Carriage return
- <LF> - Line feed

Example 1:

Format:

Uptime:%t, idx:%i, ant:%a, sub:%s, uid:%u<CR><LF>

Output:

Uptime:2713, idx:2, ant:0, sub:ICODE SLI, uid:E004010042286400
Uptime:13114, idx:3, ant:0, sub:MIFARE Classic 1k/Plus 2k, uid:438076F7

Example 2:

Format:

[%t]UID:%u<CR><LF>

Output:

[2321]UID:E004010042286400
[4094]UID:438076F7

- **Built in LED** – the user can configure the device to toggle the LED in selected colours (Red, Green, Blue, White)
- **Timeout** – time used for toggling the GPIO action and LED

2.2 Key storage

To perform some operations on TAGs authority keys maybe required. The user can set these keys using the SET_KEY command anytime this is required. However it is also possible store up to 5 keys in non-volatile memory and the module will then load these keys after bootup.

Storing keys in memory can be done in two ways: In the Cx-client on the RFID tab and by using commands.

In the latter scenario, the command SET_KEY needs to be executed to save a KEY in volatile memory temporarily and then execute the SAVE_KEYS command to save these keys to non-volatile memory. Please refer to these commands for full details.

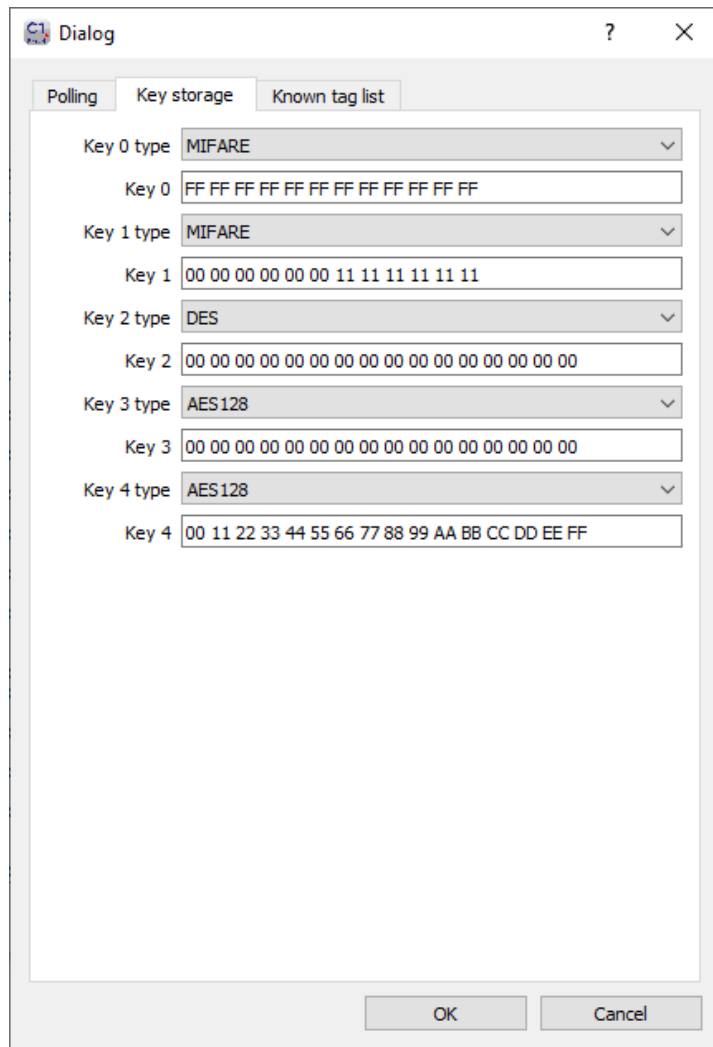


Figure 3 Cx-client – Key storage TAB

2.3 Known UID list

This tab in the Cx-client is used to manage known UIDs stored in the device memory. Thanks to this, in standalone mode, the Pepper C2 can perform different actions for known and unknown UIDs.

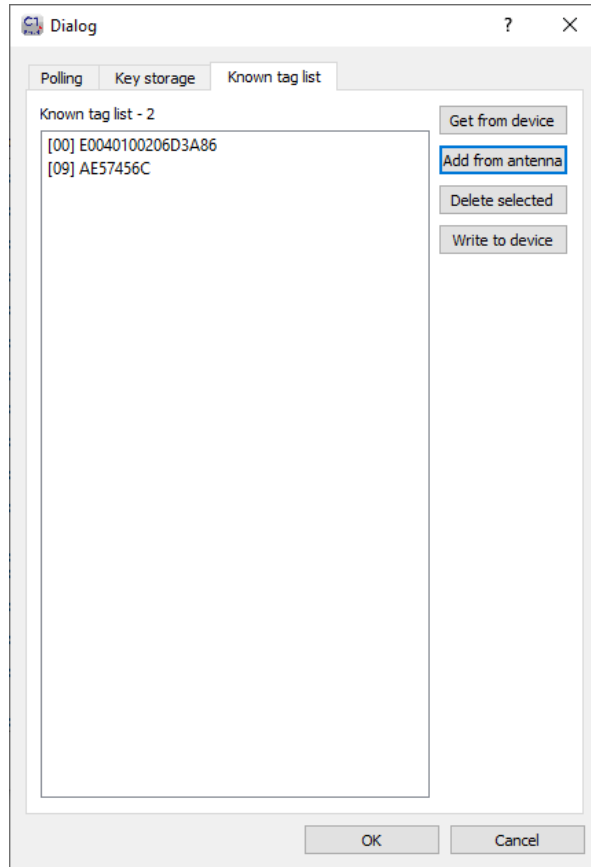


Figure 4 Cx-client – know UID list

3. Communication interface – binary interface

3.1 Overview

The Pepper C2 can be controlled using a simple binary protocol available over USB, UART0 or UART2 (using the built in USB-TTL converter). This binary protocol was designed to be as simple as possible to implement on the host side whilst still providing robust communication.

The default configuration provides communication over USB with the following parameters:

- Baud rate: 115200bps
- Data: 8 bit
- Parity: None
- Stop bits: 1 bit
- Flow Control: none

The baud rate can be changed in the Cx-Client Interface from 9600 up to 921600. The same settings can be applied when communication is switched to UART2.

3.2 Frame structure

Communication with the module is symmetric so frames sent to, and received from the module are coded in the same way. All frames contain fields as described in the table below.

Frame STX	Command body length + 2bytes CRC	Command length XOR	Command body		CRC16
1 byte	2-bytes	2-bytes	1-byte	n-bytes	2-bytes
0xF5	Command body length, LSB, maximum value 1024	XOR with 0xffff of command length bytes	Command	Command parameters	Command body CRC, LSB

3.3 CRC calculation

CRC is a 16-bit CRC-CCITT with a polynomial equal to 0x1021. The initial value is set to 0xFFFF, the input data and the output CRC is not negated. In addition, no XOR is performed on the output value. Example C code is shown below.

```
static const uint16_t CCITTCRCTable [256] = {
0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5,
```

0x60c6, 0x70e7, 0x8108, 0x9129, 0xa14a, 0xb16b,
0xc18c, 0xd1ad, 0xe1ce, 0xf1ef, 0x1231, 0x0210,
0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c,
0xf3ff, 0xe3de, 0x2462, 0x3443, 0x0420, 0x1401,
0x64e6, 0x74c7, 0x44a4, 0x5485, 0xa56a, 0xb54b,
0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6,
0x5695, 0x46b4, 0xb75b, 0xa77a, 0x9719, 0x8738,
0xf7df, 0xe7fe, 0xd79d, 0xc7bc, 0x48c4, 0x58e5,
0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969,
0xa90a, 0xb92b, 0x5af5, 0x4ad4, 0x7ab7, 0x6a96,
0x1a71, 0x0a50, 0x3a33, 0x2a12, 0xdbfd, 0xcdbc,
0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03,
0x0c60, 0x1c41, 0xedae, 0xfd8f, 0xcdec, 0xddcd,
0xad2a, 0xbd0b, 0x8d68, 0x9d49, 0x7e97, 0x6eb6,
0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
0xff9f, 0xfbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a,
0x9f59, 0x8f78, 0x9188, 0x81a9, 0xb1ca, 0xa1eb,
0xd10c, 0xc12d, 0xf14e, 0xe16f, 0x1080, 0x00a1,
0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c,
0xe37f, 0xf35e, 0x02b1, 0x1290, 0x22f3, 0x32d2,
0x4235, 0x5214, 0x6277, 0x7256, 0xb5ea, 0xa5cb,
0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447,
0x5424, 0x4405, 0xa7db, 0xb7fa, 0x8799, 0x97b8,
0xe75f, 0xf77e, 0xc71d, 0xd73c, 0x26d3, 0x36f2,
0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9,
0xb98a, 0xa9ab, 0x5844, 0x4865, 0x7806, 0x6827,

```
0x18c0, 0x08e1, 0x3882, 0x28a3, 0xcb7d, 0xdb5c,  
0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,  
0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0,  
0x2ab3, 0x3a92, 0xfd2e, 0xed0f, 0xdd6c, 0xcd4d,  
0xbdaa, 0xad8b, 0x9de8, 0x8dc9, 0x7c26, 0x6c07,  
0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,  
0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba,  
0x8fd9, 0x9ff8, 0x6e17, 0x7e36, 0x4e55, 0x5e74,  
0x2e93, 0x3eb2, 0x0ed1, 0x1ef0 };
```

```
static uint16_t GetCCITTCRC(const uint8_t* Data, uint32_t Size) {  
    uint16_t CRC;  
    uint16_t Temp;  
    uint32_t Index;  
    if (Size == 0) {  
        return 0;  
    }  
    CRC = 0xFFFF;  
    for (Index = 0; Index < Size; Index++){  
        Temp = (uint16_t)( (CRC >> 8) ^ Data[Index] ) & 0x00FF;  
        CRC = CCITTCRCTable[Temp] ^ (CRC << 8);  
    }  
    return CRC;  
}
```

3.4 Pepper Cx Client – PC application

Eccel provides the Pepper Cx-Client – the PC application written in QT (source code available) to easily test all commands with the Pepper C2 reader over the binary protocol.

The Cx-Client can be downloaded here:

https://eccel.co.uk/wp-content/downloads/Pepper_Cx/Cx-client.zip

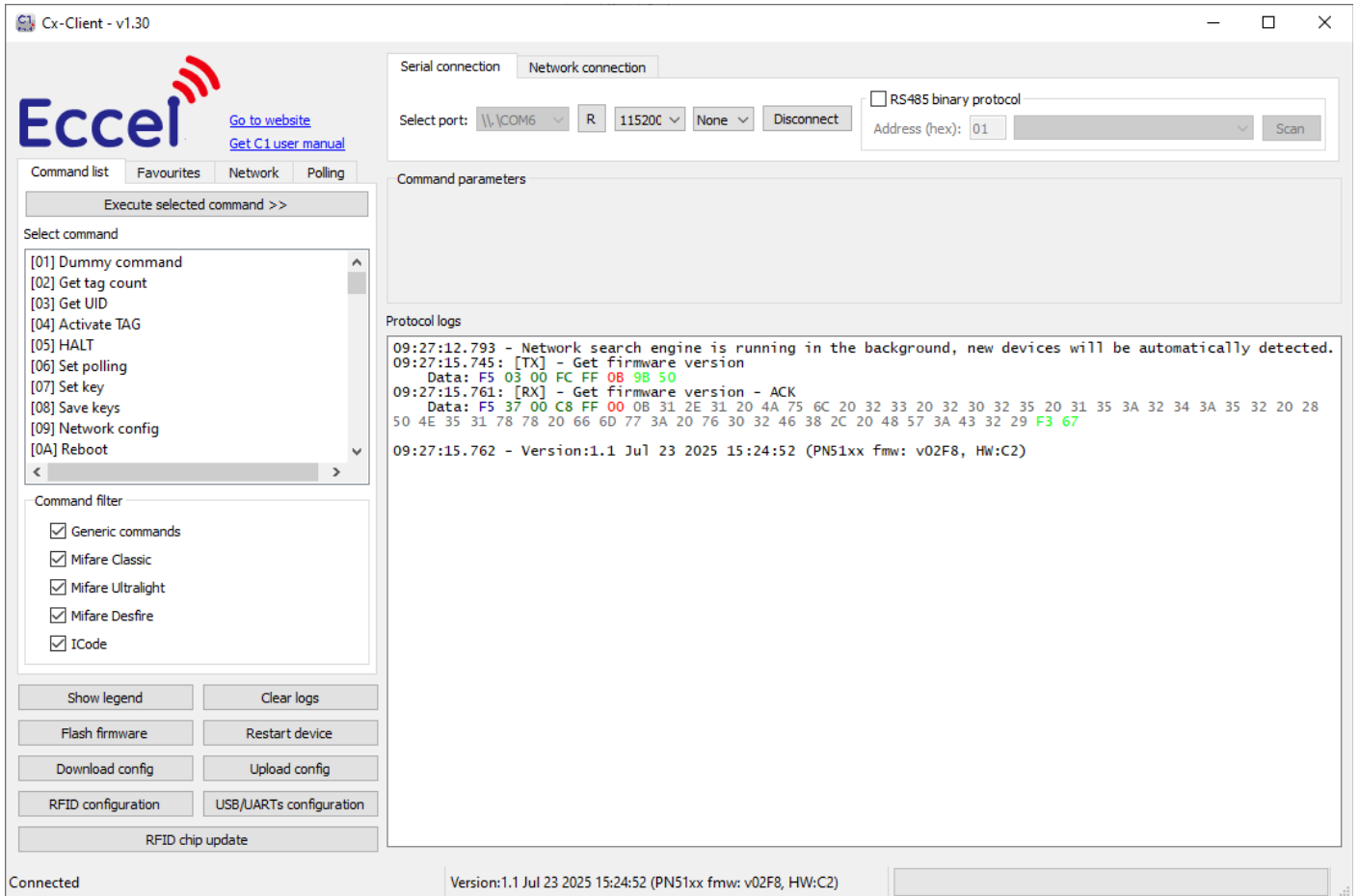


Figure 4. Cx Client

4. Commands list

Commands are exchanged with the module using the protocol described above. All frames contain a command byte and command arguments. Depending upon the command, arguments can be optional, so a command length can be in the range from 1-512 bytes.

4.1 Generic commands

4.1.1 Acknowledge frame (0x00)

This is the response message from the module to the host. This frame always contains 1-byte with command ID and optional arguments.

Command description:

Argument	Size	Value	Description
Command ID	1	0x00	
Related command ID	1	X	Related command code
Other parameters	n	X	Depending on the requested command this parameter is n-bytes long and contains parameters

Example:

```

HOST=>C2: 0x02 - GET_TAG_COUNT command
C2=>HOST: 0x00 - ACK byte
          0x02 - related command code GET_TAG_COUNT
          0x01 - argument for GET_TAG_COUNT - 0x01 - one tag detected
  
```

4.1.2 Error response (0xFF)

In case of any problems with executing the command, the device can send back ERROR response with error number returned by the RFID chip. The most common errors are described below.

Command description			
Argument	Size	Value	Description
ERROR	1	0xFF	
Command ID	1	0x01	DUMMY_COMMAND

Example:

```

C2=>HOST: 0xFF - Error byte
          0x01 - related command code DUMMY_COMMAND
          0x02 - layer byte
          0x01 - Error number
  
```

Here is a list with the most common errors:

ICODE specific errors – layer byte 0x15

Error byte:

- 0x01 - The command is not supported, i.e. the request code is not recognized
- 0x02 - The command is not recognized, for example: a format error occurred
- 0x03 - The command option is not supported
- 0x0F - Error with no information given or a specific error code is not supported
- 0x10 - The specified block is not available (doesn't exist)
- 0x11 - The specified block is already locked and thus cannot be locked again
- 0x12 - The specified block is locked and its content cannot be changed
- 0x13 - The specified block was not successfully programmed
- 0x14 - The specified block was not successfully locked
- 0x15 - The specified block is protected
- 0x40 - Generic cryptographic error
- 0x81 - The command is not supported, i.e. the request code is not recognized
- 0x82 - The command is not recognized, for example: a format error occurred
- 0x83 - The command option is not supported
- 0x84 - Error with no information given or a specific error code is not supported
- 0x85 - The specified block is not available (doesn't exist)
- 0x86 - The specified block is already locked and thus cannot be locked again
- 0x87 - The specified block is locked and its content cannot be changed
- 0x88 - The specified block was not successfully programmed
- 0x89 - The specified block was not successfully locked
- 0x8A - The specified block is protected
- 0x8B - Generic cryptographic error

Other layers errors:

- 0x01 - No reply received, e.g. PICC removal
- 0x02 - Wrong CRC or parity detected
- 0x03 - A collision occurred
- 0x04 - Attempt to write beyond buffer size
- 0x05 - Invalid frame format
- 0x06 - Received response violates protocol
- 0x07 - Authentication error
- 0x08 - A Read or Write error occurred in RAM/ROM or Flash
- 0x09 - The RC sensors signal over heating
- 0x0A - Error due to RF.
- 0x0B - An error occurred in RC communication
- 0x0C - A length error occurred
- 0x0D - An resource error
- 0x0E - TX Rejected sanely by the counterpart
- 0x0F - RX request Rejected sanely by the counterpart
- 0x10 - Error due to External RF

- 0x11 - EMVCo EMD Noise Error
- 0x12 - Used when HAL ShutDown is called
- 0x20 - Invalid data parameters supplied (layer id check failed)
- 0x21 - Invalid parameter supplied
- 0x22 - Reading/Writing a parameter would produce an overflow.
- 0x23 - Parameter not supported
- 0x24 - Command not supported
- 0x25 - Condition of use not satisfied
- 0x26 - A key error occurred
- 0x7F - An internal error occurred
- 0xF0 – Protocol authorization error. This command is not allowed without protocol authorization (Command 0x12)

4.1.3 Dummy command (0x01)

This command takes no arguments. It is used to check that the module is alive. The module replies to this command with an ACK response and no optional parameters.

Command description			
Argument	Size	Value	Description
Command ID	1	0x01	DUMMY_COMMAND
Response description			
ACK	1	0x00	
Command ID	1	0x01	DUMMY_COMMAND

Example:

```
HOST=>C2: 0x01 -DUMMY_COMMAND
C2=>HOST: 0x00 - ACK byte
          0x01 - related command code DUMMY_COMMAND
```

4.1.4 Get tag count (0x02)

The command is sent to the module to read how many TAGS are in range of the antenna no matter which technology of tag, so it returns the total amount present of all supported tag types. The maximum number for this standard discovery loop is 5. If you want to perform a full inventory command for ICODE tag types please refer to ICODE_INVENTORY_xxx commands.

After this command, the module holds all UID's and basic information about TAGs present in volatile memory and the user can read it using the GET_TAG_UID command.

Command description			
Argument	Size	Value	Description
Command ID	1	0x02	GET_TAG_COUNT
Response description			
ACK	1	0x00	
Command ID	1	0x02	GET_TAG_COUNT
TAG count	1	X	Maximum discovered tags is 5

Example:

```
HOST=>C2: 0x02 - GET_TAG_COUNT
C2=>HOST: 0x00 - ACK byte
          0x02 - related command code GET_TAG_COUNT
          0x01 - number of tags in range
```

4.1.5 Get tag UID (0x03)

This command should be executed after GET_TAG_COUNT frame to read information about the tag.

Command description			
Argument	Size	Value	Description
Command ID	1	0x03	GET_TAG_UID
TAG idx	1	X	TAG index in module memory, must be less than number of tags reported by GET_TAG_COUNT command
Response description			
ACK	1	0x00	
Command ID	1	0x03	GET_TAG_UID
TAG type	1	X	0x01 - MIFARE Ultralight 0x02 - MIFARE Ultralight-C 0x03 - MIFARE Classic 0x04 - MIFARE Classic 1k 0x05 - MIFARE Classic 4k 0x06 - MIFARE Plus 0x07 - MIFARE Plus 2k 0x08 - MIFARE Plus 4k 0x09 - MIFARE Plus 2k sl2 0x0A - MIFARE Plus 4k sl2 0x0B - MIFARE Plus 2k sl3 0x0C - MIFARE Plus 4k sl3 0x0D - MIFARE DESFire 0x0F - JCOP 0x10 - MIFARE Mini 0x21 - ICODE SLI 0x22 - ICODE SLI-S 0x23 - ICODE SLI-L 0x24 - ICODE SLIX 0x25 - ICODE SLIX-S 0x26 - ICODE SLIX-X 0x27 - ICODE SLIX2 0x28 - ICODE DNA
TAG parameter	1	X	SAK - byte for MIFARE family tags DSFID - byte for ICODE family tags
UID	N	X	UID bytes. Max length is 8.

Example:

```

HOST=>C2: 0x03 - GET_TAG_UID
          0x00 - TAG idx

C2=>HOST: 0x00 - ACK byte
          0x03 - related command code GET_TAG_UID
          0x01 - MIFARE tag type
  
```

0x20 - tag parameter:
 SAK byte for MIFARE family tags
 DSFID byte for ICODE family tags
 0x74 0x54 0x12 0x65 - tag UID bytes

4.1.6 Activate TAG (0x04)

The command executed to activate a TAG after the discovery loop if more than one TAG is detected.

Command description			
Argument	Size	Value	Description
Command ID	1	0x04	ACTIVATE_TAG
TAG idx	1	X	TAG index in module memory, must be less than number of tags reported by GET_TAG_COUNT command
Response description			
ACK	1	0x00	
Command ID	1	0x04	ACTIVATE_TAG

Example:

```
HOST=>C2: 0x04 - ACTIVATE_TAG
          0x00 - TAG idx

C2=>HOST: 0x00 - ACK byte
          0x04 - related command code ACTIVATE_TAG
```

4.1.7 Halt (0x05)

The Halt command takes no arguments. It halts the tag and turns off the RF field. It must be executed at the end of each operation on a tag to disable the antenna and reduce the power consumption.

Command description			
Argument	Size	Value	Description
Command ID	1	0x05	HALT
Response description			
ACK	1	0x00	
Command ID	1	0x05	HALT

Example:

```
HOST=>C2: 0x05 - HALT

C2=>HOST: 0x00 - ACK byte
          0x05 - related command code HALT
```

4.1.8 Set polling (0x06)

The module can't perform polling mode and RFID requests over the communication channels simultaneously. When polling is enabled and the host wants to request an RFID command, this command should be executed first with a STOP

parameter, and then START again if needed afterwards. This command does not change polling configuration permanently, so after a reset, the module performs polling as configured in the Cx-Client.

Command description			
Argument	Size	Value	Description
Command ID	1	0x06	SET_POLLING
Start/Stop	1	X	0x00 – Stop polling 0x01 – Start polling
Response description			
ACK	1	0x00	
Command ID	1	0x06	SET_POLLING

Example:

```

HOST=>C2: 0x06 - SET_POLLING
          0x00 - Stop polling temporary

C2=>HOST: 0x00 - ACK byte
          0x06 - related command code SET_POLLING

```

4.1.9 Set key (0x07)

This command sets a KEY in Key Storage Memory on a selected slot. Set key can be used for all RFID functions needing authorization like e.g. READ/WRITE memory on the TAG etc. This command changes a key in volatile memory, so if the user wants to save it permanently and load automatically after boot-up, then the user should use the CMD_SAVE_KEYS command.

Command description			
Argument	Size	Value	Description
Command ID	1	0x07	SET_KEY
Key number	1	0-4	Key number in Key Storage Memory.
Key type	1	0 - 6	0x00 - AES 128 Key. (length = 16 bytes) 0x01 - AES 192 Key. (length = 24 bytes) 0x02 - AES 256 Key. (length = 32 bytes) 0x03 - DES Single Key. (length = 16 bytes) 0x04 - 2 Key Triple Des. (length = 16 bytes) 0x05 - 3 Key Triple Des. (length = 24 bytes) 0x06 - MIFARE (R) Key. (length = 12 bytes, key A+B)
Key	12-32	X	Key bytes. Length must match to the type.
Response description			
ACK	1	0x00	
Command ID	1	0x07	SET_KEY

Example:

```

HOST=>C2: 0x07 - SET_KEY
          0x00 - Key number
          0x06 - MIFARE key type

```

0x00 0x00 0x00 0x00 0x00 0x00
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF – Key bytes

C2=>HOST: 0x00 – ACK byte
0x07 – related command code SET_KEY

4.1.10 Save keys (0x08)

This command should be called if the user wants to save keys changed using the SET_KEY command in the module non-volatile memory. Saved keys will be automatically loaded after power up or reboot.

Command description			
Argument	Size	Value	Description
Command ID	1	0x08	SAVE_KEYS
Response description			
ACK	1	0x00	
Command ID	1	0x08	SAVE_KEYS

Example:

HOST=>C2: 0x08 – SAVE_KEYS
C2=>HOST: 0x00 – ACK byte
0x08 – related command code SAVE_KEYS

4.1.11 Reboot (0x0A)

This command requests a software reboot for the Pepper C2 module. After this command the device will not accept any protocol commands for 1 second.

Command description			
Argument	Size	Value	Description
Command ID	1	0x0A	REBOOT
Response description			
ACK	1	0x00	
Command ID	1	0x0A	REBOOT

Example:

HOST=>C2: 0x0A – REBOOT
C2=>HOST: 0x00 – ACK byte
0x0A – related command code REBOOT

4.1.12 Get version (0x0B)

This command requests a version string from the device.

Command description			
Argument	Size	Value	Description
Command ID	1	0x0B	GET_VERSION
Response description			
ACK	1	0x00	
Command ID	1	0x0B	GET_VERSION
Version string	X	X	Version string, contains major and minor version and build data and time e.g.: 1.1 Jan 18 2019 15:35:03

Example:

```

HOST=>C2: 0x0B - GET_VERSION
C2=>HOST: 0x00 - ACK byte
          0x0B - related command code GET_VERSION
          0x31 0x2e 0x31 0x20 0x4a 0x61 0x6e 0x20
          0x31 0x38 0x20 0x32 0x30 0x31 0x39 0x20
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 - version string bytes

```

4.1.13 Sleep command (0x0D)

This command instructs the device to enter sleep mode. In this state, the device consumes approximately 2mA for module version and 4mA for full board at 5V . To wake the device, the user can send any byte on UART0 or UART2, or simply toggle the BUTTON input.

Command description			
Argument	Size	Value	Description
Command ID	1	0x0D	SLEEP
Response description			
ACK	1	0x00	
Command ID	1	0x0D	SLEEP

Example:

```

HOST=>C1: 0x0D - SLEEP
C1=>HOST: 0x00 - ACK byte
          0x0D - related command code SLEEP

```

4.1.14 GPIO command (0x0E)

This command should be used to setup GPIO pins on the reader. All of these pins can be used as inputs (with pull up/pull down option) or as output pins. For the GPIO output command, the user doesn't need to setup a pin as an output, this is done automatically when the first command setting level or toggling level on the pin is requested. For the input command, the host application should first setup the pin as input with option like pull up/down if needed.

Command description			
Argument	Size	Value	Description
Command ID	1	0x0E	GPIO command
Subcommand ID	1	X	0x00 – setup pin as GPIO_INPUT 0x01 – setup pin as GPIO_INPUT with PULL_UP enabled 0x02 – setup pin as GPIO_INPUT with PULL_DOWN enabled 0x03 – setup pin as GPIO_OUTPUT with level HIGH 0x04 – setup pin as GPIO_OUTPUT with level LOW 0x05 – toggle GPIO low for specified time 0x06 – toggle GPIO high for specified time 0x07 – read GPIO pin status
GPIO number	1	X	GPIO number in hex format. Values allowed are 0-2.
Toggle timeout	2	X	Optional bytes for TOGGLE_LOW/TOGGLE_HIGH subcommands. Number of milliseconds defined as unsigned 16bit value with LSB order.
Response description			
ACK	1	0x00	
Command ID	1	0x0E	GPIO command
GPIO level	1	X	Optional byte received when READ command is requested 0x00 – GPIO is in LOW state 0x01 – GPIO is in HIGH state

Example1 – setup GPIO1 as input port with pull up enabled:

```

HOST=>C2: 0x0E – GPIO command
          0x01 – input port with PULL UP enabled
          0x01 – GPIO1

C2=>HOST: 0x00 – ACK byte
          0x0E – related command code GPIO
  
```

Example2 – read state of GPIO32:

```

HOST=>C2: 0x0E – GPIO command
          0x07 – read pin status
          0x01 – GPIO1

C2=>HOST: 0x00 – ACK byte
          0x0E – related command code GPIO
          0x01 – HIGH value on the GPIO1
  
```

4.1.15 Factory reset command (0x11)

This command should be user to perform a factory reset. To prevent resetting to factory default by accident, this commands requires four extra bytes as extra parameters described in the table below.

Command description			
Argument	Size	Value	Description
Command ID	1	0x11	FACTORY_RESET
Extra bytes	4	0x01 0x02 0x03 0x04	Four digits pin number (optional)
Response description			
ACK	1	0x00	
Command ID	1	0x11	FACTORY_RESET_PIN

Example – setup new PIN:

```
HOST=>C2: 0x11 - FACTORY_RESET
          0x01 0x02 0x03 0x04 - Extra parameters

C2=>HOST: 0x00 - ACK byte
          0x11 - related command code FACTORY_RESET
```

4.1.16 Protocol configuration (0x13)

This set of frames can be used to setup all parameters for different communication methods. The first byte is the subtype of the frame. To get current settings, the host has to send this frame with a subcommand ID only.

4.1.16.1 UART settings

With this command the host can setup UART parameters.

Command description			
Argument	Size	Value	Description
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x01	UART subcommand
UART0 protocol	1	X	0x00 – Binary protocol 0x01 – Console logs
UART0 baud	1	X	0x00 – 9600 bps 0x01 – 19200 bps 0x02 – 38400 bps 0x03 – 57600 bps 0x04 – 115200 bps 0x05 – 230400 bps 0x06 – 460800 bps 0x07 – 921600 bps
UART2 protocol	1	X	0x00 – Binary protocol 0x01 – Console logs

UART2 baud	1	X	0x00 – 9600 bps 0x01 – 19200 bps 0x02 – 38400 bps 0x03 – 57600 bps 0x04 – 115200 bps 0x05 – 230400 bps 0x06 – 460800 bps 0x07 – 921600 bps
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x01	

Example:

HOST=>C2: 0x13 – PROTOCOL_CONFIG
0x01 – UART subcommand
0x01 – Console logs on UART0
0x04 – 115200 baud
0x00 – Binary protocol on UART2
0x04 – 115200 baud

C2=>HOST: 0x00 – ACK byte
0x13 – related command code PROTOCOL_CONFIG
0x01 – UART subcommand ID

4.1.17 LED command (0x14)

This command should be used to control the built-in LED. The first three bytes are the RGB value of the colour and the optional two bytes are the timeout in milliseconds.

Command description			
Argument	Size	Value	Description
Command ID	1	0x14	LED command
GPIO number	3	RRGGBB	RGB colour value
Timeout	2	X	Number of milliseconds defined as unsigned 16bit value LSB order.
Response description			
ACK	1	0x00	
Command ID	1	0x14	LED command

Example:

HOST=>C2: 0x14 – LED command
0xFF 0x00 0x00 – set red colour
0x64 0x00 – timeout 100ms

C2=>HOST: 0x00 – ACK byte
0x14 – related command code LED

4.1.18 Polling setup (0x16)

This set of frames can be used to setup most of parameters for RFID polling. The first byte is the subtype of the frame. To get current settings, the host has to send this frame with a subcommand ID only.

4.1.18.1 Supported technologies (0x00)

With this command the host can setup general settings for the device like MDNS service and UDP discovery service. As an optional argument, the user can send a new device name.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x00	Supported technologies subcommand ID
Technologies	1	X	0x01 – MIFARE, 0x10 – ICODE, 0x11 – MIFARE + ICODE
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x00	Supported technologies subcommand ID

Example set command:

```
HOST=>C2: 0x16 - POLLING_SETUP
           0x00 - Supported technologies subcommand ID
           0x11 - Enable both MIFARE and ICODE

C2=>HOST: 0x00 - ACK byte
           0x16 - related command code POLLING_SETUP
           0x00 - Supported technologies subcommand ID
```

Example get command:

```
HOST=>C2: 0x16 - POLLING_SETUP
           0x00 - query supported technologies subcommand ID

C2=>HOST: 0x00 - ACK byte
           0x16 - POLLING_SETUP
           0x00 - Supported technologies subcommand ID
           0x01 - MIFARE technology enabled
```

4.1.18.2 Internal polling control (0x02)

With this command the host can enable/disable internal polling. Comparing to command set polling this one is permanent and it is saved in the device memory and restored after restart.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x02	Polling enabled subcommand ID
Enable flag	1	X	0x00 – Disabled, 0x01 – Enabled

Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x02	Polling enabled subcommand ID

Example set command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x02 - Polling enabled subcommand ID
 0x01 - Set polling enabled

C2=>HOST: 0x00 - ACK byte
 0x16 - related command code POLLING_SETUP
 0x02 - RFID power subcommand ID

Example get command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x02 - Query polling enabled flag

C2=>HOST: 0x00 - ACK byte
 0x16 - POLLING_SETUP
 0x02 - Polling enabled subcommand ID
 0x01 - enabled flag

4.1.18.3 Polling timeout (0x03)

With this command the host can set the timeout between polling reads.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x03	Polling timeout subcommand
Timeout	2	X	Timeout value in milliseconds as unsigned 16bit value
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x03	Polling timeout subcommand

Example set command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x03 - Polling timeout subcommand ID
 0xff 0x00 - Timeout set as 255ms

C2=>HOST: 0x00 - ACK byte
 0x16 - related command code POLLING_SETUP
 0x03 - Polling timeout subcommand ID

Example get command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x03 - Polling timeout subcommand

C2=>HOST: 0x00 - ACK byte
 0x16 - POLLING_SETUP
 0x03 - Polling timeout subcommand
 0xff 0x00 - timeout value

4.1.18.4 Ignore timeout (0x04)

With this command, the host can set the ignore timeout for the last detected tag. This timer starts counting when the tag is removed from the antenna field.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x04	Ignore timeout subcommand
Timeout	2	X	Timeout value in milliseconds as unsigned 16bit value
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x04	Ignore timeout subcommand

Example set command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x04 - Ignore timeout subcommand ID
 0xff 0x00 - Timeout set as 255ms

C2=>HOST: 0x00 - ACK byte
 0x16 - related command code POLLING_SETUP
 0x04 - Ignore timeout subcommand ID

Example get command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x04 - Ignore timeout subcommand

C2=>HOST: 0x00 - ACK byte
 0x16 - POLLING_SETUP
 0x04 - Ignore timeout subcommand
 0xff 0x00 - timeout value

4.1.18.5 Polling event packet (0x06)

With this command, the host can set up an asynchronous packet sent to the host for every UID event. An extra argument describes type of the UID know (saved on the known tag list) or unknown.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x06	Polling event packet subcommand
Known/Unknow flag	1	X	0 – Known flag 1 – Unknown flag
Type of the packet	1	X	0 – None 1 – Binary frame 2 – Plain text 3 – JSON packet 4 – Custom text format frame
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x06	Polling event packet subcommand

Example set command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x06 - Polling event packet subcommand
          0x01 - Unknown flag
          0x01 - Binary frame format

C2=>HOST: 0x00 - ACK byte
          0x16 - related command code POLLING_SETUP
          0x06 - Polling event packet subcommand

```

Example get command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x06 - Polling event packet subcommand

C2=>HOST: 0x00 - ACK byte
          0x16 - POLLING_SETUP
          0x06 - Polling event packet subcommand
          0x01 - binary frame sent when known tag is detected
          0x02 - plain text frame sent when unknown tag is detected

```

4.1.18.6 Polling LED event (0x07)

With this command, the host can set up an LED colour for every UID event. An extra argument describes type of the UID know (saved on the known tag list) or unknown.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x07	Polling LED event subcommand
Known/Unknow flag	1	X	0 – Known flag 1 – Unknown flag
LED colour	1	X	0 – None 1 – Red 2 – Green 3 – Blue 4 – White
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x07	Polling LED event subcommand

Example set command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x07 - Polling LED event subcommand
          0x01 - Unknown flag
          0x01 - Red colour
C2=>HOST: 0x00 - ACK byte
          0x16 - related command code POLLING_SETUP
          0x07 - Polling LED event subcommand

```

Example get command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x07 - Polling LED event subcommand
C2=>HOST: 0x00 - ACK byte
          0x16 - POLLING_SETUP
          0x07 - Polling LED event subcommand
          0x01 - Red led for known tag
          0x02 - Green led for unknown tag

```

4.1.18.7 Polling GPIO event (0x08)

With this command, the host can set up an GPIO event for every UID event. An extra argument describes type of the UID know (saved on the known tag list) or unknown.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x08	Polling GPIO event subcommand

Known/Unknow flag	1	X	0 – Known flag 1 – Unknow flag
GPIO number	1	X	GPIO number on the J1 header (in HEX format)
GPIO event type	1	X	0 – toggle low, 1 – toggle high
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x08	Polling GPIO event subcommand

Example set command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x08 - Polling GPIO event subcommand
          0x01 - Unknown flag
          0x01 - GPIO 1
          0x00 - Toggle low
C2=>HOST: 0x00 - ACK byte
          0x16 - related command code POLLING_SETUP
          0x08 - Polling GPIO event subcommand

```

Example get command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x08 - Polling GPIO event subcommand
C2=>HOST: 0x00 - ACK byte
          0x16 - POLLING_SETUP
          0x08 - Polling GPIO event subcommand
          0x02 - Toggle GPIO 2 for known tag
          0x00 - Toggle low
          0x03 - Toggle GPIO 3 for known tag
          0x01 - Toggle high

```

4.1.18.8 Event duration (0x09)

With this command, the host can set the time for known or unknown LED and GPIO event.

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x09	Polling event duration
Known/Unknow flag	1	X	0 – Known flag 1 – Unknow flag
Timeout	2	X	Timeout value in milliseconds as unsigned 16bit value
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x09	Polling event duration

Example set command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x09 - Polling event duration subcommand ID
          0xff 0x00 - Timeout set as 255ms

C2=>HOST: 0x00 - ACK byte
          0x16 - related command code POLLING_SETUP
          0x09 - Polling event duration subcommand ID

```

Example get command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x09 - Polling event duration subcommand ID

C2=>HOST: 0x00 - ACK byte
          0x16 - POLLING_SETUP
          0x09 - Polling event duration subcommand ID
          0xff 0x00 - timeout value for known event
          0xff 0x00 - timeout value for unknown event

```

4.1.18.9 Polling event custom text format (0x0A)

With this command, the host can set custom frame text format for known and unknown events. The format has to be transferred as ASCII bytes as part of the setup frame (see examples below). If the command is executed without arguments then the device returns current setup for known and unknown frames separated with byte 0x00. More information about custom frame format are described in the chapter 2.1.5

Command description			
Argument	Size	Value	Description
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x0A	Custom frame format subcommand
Known/Unknow flag	1	X	0 – Known flag 1 – Unknown flag
Type of the packet	1	X	Custom frame format as ASCII bytes
Response description			
ACK	1	0x00	
Command ID	1	0x16	POLLING_SETUP
Subcommand ID	1	0x0A	Custom frame format subcommand

Example set command:

```

HOST=>C2: 0x16 - POLLING_SETUP
          0x0A - Custom frame format subcommand
          0x01 - Unknown flag
          0x55 0x49 0x44 0x3A 0x20 0x25 0x75 0x3C
          0x43 0x52 0x3E 0x3C 0x4C 0x46 0x3E - Custom frame format
                                               bytes 'UID: %u<CR><LF>'

C2=>HOST: 0x00 - ACK byte
          0x16 - related command code POLLING_SETUP
          0x0A - Custom frame format subcommand

```

Example get command:

HOST=>C2: 0x16 - POLLING_SETUP
 0x0A - Custom frame format subcommand

C2=>HOST: 0x00 - ACK byte
 0x16 - POLLING_SETUP
 0x0A - Custom frame format subcommand

 0x4B 0x4E 0x4F 0x57 0x4E 0x3A 0x25 0x75
 0x3C 0x43 0x52 0x3E 0x3C 0x4C 0x46 0x3E
 0x00 0x55 0x4E 0x4B 0x4E 0x4F 0x57 0x4E
 0x3A 0x25 0x75 0x3C 0x43 0x52 0x3E 0x3C
 0x4C 0x46 0x3E - ASCII bytes for known and unknow format separated
with byte 0x00. In this example converted to ASCII bytes we have
KNOWN:%u<CR><LF> for known tag event and UNKNOWN:%u<CR><LF> for
unknown format frame

4.2 MIFARE Classics commands

This set of commands should be performed on MIFARE Classics tags.

4.2.1 Read block (0x20)

The read block command should be used to read data from the tag. It takes as arguments the block number of the first block to read, the number of blocks to read, the key A or B parameter, and the key number in key storage. The returned ACK answer contains data read from the specified tag memory. The number of bytes of this data is MIFARE Classic block size (16) multiplied by the number of blocks to be read.

Command description			
Argument	Size	Value	Description
Command ID	1	0x20	MF_READ_BLOCK
Block number	1	X	
Number of blocks	1	Y	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x20	MF_READ_BLOCK
Read data	Y*16	XXX	Bytes read from the tag. Number of bytes is number of requested blocks multiplied by 16.

Example:

```

HOST=>C2: 0x20 – MF_READ_BLOCK
          0x02 – block number 2
          0x02 – two blocks to read
          0x0A – key A should be selected from key storage
          0x00 – first key should be selected from key storage

C2=>HOST: 0x00 – ACK byte
          0x20 – related command code MF_READ_BLOCK

          0x01 0x2e 0x41 0x22 0x43 0x11 0x8e 0x20
          0x31 0x38 0x20 0x32 0x30 0x31 0x39 0x41
          0x81 0x23 0x42 0x28 0x33 0x01 0x8e 0x72
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 – 32 bytes result
  
```

4.2.2 Write block (0x21)

The write block command should be used to write data to the tag. It takes as arguments the block number of the first block to write, the number of blocks to write, the key A or B parameter, the key number in key storage, and the bytes to be written. The number of bytes to be written must be exactly the number of blocks to write multiplied by 16.

Command description			
Argument	Size	Value	Description
Command ID	1	0x21	MF_WRITE_BLOCK
Block number	1	X	
Number of blocks	1	Y	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Bytes to write	Y*16	XXX	Bytes to write. Number of this bytes must be number of requested blocks multiplied by 16.
Response description			
ACK	1	0x00	
Command ID	1	0x21	MF_WRITE_BLOCK

Example:

```

HOST=>C2: 0x21 - MF_WRITE_BLOCK
          0x02 - block number 2
          0x02 - two blocks to write
          0x0A - key A should be selected from key storage
          0x00 - first key should be selected from key storage

          0x01 0x2e 0x41 0x22 0x43 0x11 0x8e 0x20
          0x31 0x38 0x20 0x32 0x30 0x31 0x39 0x41
          0x81 0x23 0x42 0x28 0x33 0x01 0x8e 0x72
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 - 32 bytes to write

C2=>HOST: 0x00 - ACK byte
          0x21 - related command code MF_WRITE_BLOCK
  
```

4.2.3 Read value (0x22)

This command should be used to read a value from the tag. It takes as arguments the block number where the value is stored, the key A or B parameter, and the key number in key storage. The returned ACK response contains a value as a signed 32-bit value (LSB first) and an address byte as an unsigned 8bit value.

Command description			
Argument	Size	Value	Description
Command ID	1	0x22	MF_READ_VALUE
Block number	1	X	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x22	MF_READ_VALUE
Value	4	X	Signed 32-bit value (LSB first)
Address	1	X	Address byte

Example:

```

HOST=>C2: 0x22 - MF_READ_VALUE
           0x02 - block number 2
           0x0A - key A should be selected from key storage
           0x00 - first key should be selected from key storage

C2=>HOST: 0x00 - ACK byte
           0x22 - related command code MF_READ_BLOCK
           0x00 0x00 0x00 0x01 - value
           0x01 - address byte

```

4.2.4 Write value (0x23)

This command should be used to write a value to the tag. It takes as arguments the block number where the value should be stored, the key A or B parameter, the key number in key storage, a value (signed 32-bit LSB first) as 4 bytes, and an address byte (unsigned 8-bit value).

Command description			
Argument	Size	Value	Description
Command ID	1	0x23	MF_WRITE_VALUE
Block number	1	X	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Value	4	X	Signed 32-bit value (LSB first)
Address	1	X	Address byte
Response description			
ACK	1	0x00	
Command ID	1	0x23	MF_WRITE_VALUE

Example:

```

HOST=>C2: 0x23 - MF_WRITE_VALUE
           0x02 - block number 2
           0x0A - key A should be selected from key storage
           0x00 - first key should be selected from key storage
           0x00 0x00 0x00 0x01 - value
           0x01 - address byte

C2=>HOST: 0x00 - ACK byte
           0x23 - related command code MF_WRITE_BLOCK

```

4.2.5 Increment/decrement value (0x24)

This command should be used to increment or decrement a value stored in the tag memory. It takes as arguments the block number where the value is stored, the key A or B parameter, the key number in key storage, value (signed 32-bit LSB first) as 4 bytes to increment or decrement, and the increment/decrement flag.

Command description			
Argument	Size	Value	Description
Command ID	1	0x24	MF_INCREMENT_VALUE
Block number	1	X	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Delta value	4	X	Signed 32-bit value (LSB first)
Increment/Decrement	1	X	0x00 – Decrement by delta value 0x01 – Increment by delta value
Response description			
ACK	1	0x00	
Command ID	1	0x24	MF_INCREMENT_VALUE

Example:

```

HOST=>C2: 0x24 – MF_INCREMENT_VALUE
          0x02 – block number 2
          0x0A – key A should be selected from key storage
          0x00 – first key should be selected from key storage
          0x00 0x00 0x00 0x01 – delta value
          0x01 – increment flag

C2=>HOST: 0x00 – ACK byte
          0x24 – related command code MF_INCREMENT_BLOCK
  
```

4.2.6 Transfer value (0x25)

This command should be used to transfer a value from a volatile register on the tag to the block being addressed. It takes as arguments the block number where the value should be stored, the key A or B parameter, the key number in key storage.

Command description			
Argument	Size	Value	Description
Command ID	1	0x25	MF_TRANSFER_VALUE
Block number	1	X	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x25	MF_TRANSFER_VALUE

Example:

```

HOST=>C2: 0x25 – MF_TRANSFER_VALUE
          0x02 – block number 2
          0x0A – key A should be selected from key storage
          0x00 – first key should be selected from key storage
  
```

C2=>HOST: 0x00 – ACK byte
 0x25 – related command code MF_TRANSFER_BLOCK

4.2.7 Restore value (0x26)

This command should be used to restore a value to a volatile register on the tag from the block being addressed. It takes as arguments the block number where the value is stored, the key A or B parameter, key number in key storage.

Command description			
Argument	Size	Value	Description
Command ID	1	0x26	MF_RESTORE_VALUE
Block number	1	X	
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x26	MF_RESTORE_VALUE

Example:

HOST=>C2: 0x26 – MF_RESTORE_VALUE
 0x02 – block number 2
 0x0A – key A should be selected from key storage
 0x00 – first key should be selected from key storage

C2=>HOST: 0x00 – ACK byte
 0x26 – related command code MF_RESTORE_BLOCK

4.2.8 Transfer-Restore value (0x27)

This command performs a Restore-Transfer command sequence on the tag. It takes as arguments the block number to be decremented, the block number to be transferred to, the key A or B parameter, the key number in key storage. This command has the same functionality as the read value command, except that it can be used on a block which is corrupted – it tries to recover data from a corrupted block. The format of a value-type block allows for some bits to be corrupted and it still be possible to read and recover the proper value

Command description			
Argument	Size	Value	Description
Command ID	1	0x27	MF_TRANSFER_RESTORE_VALUE
Source block number	1	X	Block number to be decremented
Destination block number	1	X	Block number to be transferred to
Key A/B parameter	1	X	0x0A – Key A should be selected from key storage 0x0B – Key B should be selected from key storage
Key number	1	0-4	Key number in key storage

Response description			
ACK	1	0x00	
Command ID	1	0x27	MF_TRANSFER_RESTORE_VALUE

Example:

HOST=>C2: 0x27 - MF_TRANSFER_RESTORE_VALUE
 0x02 - source block number 2
 0x03 - destination block number 3
 0x0A - key A should be selected from key storage
 0x00 - first key should be selected from key storage

C2=>HOST: 0x00 - ACK byte
 0x27 - related command code MF_TRANSFER_RESTORE_BLOCK

4.3 MIFARE Ultralight commands

This set of commands should be performed on MIFARE Ultralight tags.

4.3.1 Read page (0x40)

The read page command should be used to read data stored in tag pages. It takes as arguments the page number of the first page to be read, and the number of pages to be read. The returned ACK answer contains data read from the specified tag memory. The number of bytes of this data is MIFARE Ultralight page size (4) multiplied by the number of pages to be read.

Command description			
Argument	Size	Value	Description
Command ID	1	0x40	MFU_READ_PAGE
Page number	1	X	
Number of pages	1	Y	
Response description			
ACK	1	0x00	
Command ID	1	0x40	MFU_READ_PAGE
Read data	Y*4	XXX	Bytes read from the tag. Number of bytes is number of requested pages multiplied by 4.

Example:

HOST=>C2: 0x40 - MFU_READ_PAGE
 0x02 - page number 2
 0x02 - two pages to read

C2=>HOST: 0x00 - ACK byte
 0x40 - related command code MFU_READ_PAGE
 0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 - 8 bytes result

4.3.2 Write page (0x41)

The write page command should be used to write data to the tag. It takes as arguments the page number of the first page to write, the number of pages to write, and the bytes to be written. The number of bytes to be written must be exactly the number of pages to write multiplied by 4.

Command description			
Argument	Size	Value	Description
Command ID	1	0x41	MFU_WRITE_PAGE
Page number	1	X	
Number of pages	1	Y	
Bytes to write	Y*4	XXX	Bytes to write. Number of this bytes must be number of requested pages multiplied by 4.
Response description			
ACK	1	0x00	
Command ID	1	0x41	MFU_WRITE_PAGE

Example:

```

HOST=>C2: 0x41 - MFU_WRITE_PAGE
          0x02 - page number 2
          0x02 - two pages to write
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 - 32 bytes to write

C2=>HOST: 0x00 - ACK byte
          0x41 - related command code MFU_WRITE_PAGE
  
```

4.3.3 Get version (0x42)

This command requests a version string from the TAG. The returned ACK answer consists of 8-bytes containing the version information defined by the NXP standard. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x42	MFU_GET_VERSION
Response description			
ACK	1	0x00	
Command ID	1	0x42	MFU_GET_VERSION
Version bytes	8	X	Version bytes from the TAG

Example:

```

HOST=>C2: 0x42 - MFU_GET_VERSION

C2=>HOST: 0x00 - ACK byte
          0x42 - related command code MFU_GET_VERSION
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 - version bytes
  
```

4.3.4 Read signature (0x43)

This command requests a version string from the device. The returned ACK answer contains 32-bytes with ECC signature defined by the NXP standard. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x43	MFU_READ_SIGNATURE
Response description			
ACK	1	0x00	
Command ID	1	0x43	MFU_READ_SIGNATURE
Version bytes	32	X	Signature bytes from the TAG

Example:

```

HOST=>C2: 0x43 - MFU_READ_SIGNATURE
C2=>HOST: 0x00 - ACK byte
          0x43 - related command code MFU_READ_SIGNATURE
          0x01 0x2e 0x41 0x22 0x43 0x11 0x8e 0x20
          0x31 0x38 0x20 0x32 0x30 0x31 0x39 0x41
          0x81 0x23 0x42 0x28 0x33 0x01 0x8e 0x72
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 - signature bytes
  
```

4.3.5 Write signature (0x44)

This command writes the signature information to the MIFARE Ultralight Nano TAG. It takes as arguments relative page location of the signature part to be written and four bytes of signature value to be written.

Command description			
Argument	Size	Value	Description
Command ID	1	0x44	MFU_WRITE_SIGNATURE
Relative page address	1	X	Relative page location of the signature part to be written
Bytes to write	4	XXX	Bytes of signature value to be written to the specified relative page address
Response description			
ACK	1	0x00	
Command ID	1	0x44	MFU_WRITE_SIGNATURE

Example:

```

HOST=>C2: 0x44 - MFU_WRITE_SIGNATURE
          0x00 - relative page number 0
          0x35 0x3a 0x30 0x33 - 4 bytes to write

C2=>HOST: 0x00 - ACK byte
          0x44 - related command code MFU_WRITE_SIGNATURE
  
```

4.3.6 Lock signature (0x45)

This command locks the signature temporarily or permanently based on the information provided in the API. The locking and unlocking of the signature can be performed using this command if the signature is not locked or temporary locked. If the signature is permanently locked, then unlocking can't be done.

Command description			
Argument	Size	Value	Description
Command ID	1	0x45	MFU_LOCK_SIGNATURE
Lock mode	1	X	0x00 – Unlock 0x01 – Lock 0x02 – Permanent lock
Response description			
ACK	1	0x00	
Command ID	1	0x45	MFU_LOCK_SIGNATURE

Example:

```

HOST=>C2: 0x45 – MFU_LOCK_SIGNATURE
          0x02 – permanent lock

C2=>HOST: 0x00 – ACK byte
          0x45 – related command code MFU_LOCK_SIGNATURE
  
```

4.3.7 Read counter (0x46)

This command should be used to read a counter from the TAG. It takes as arguments the counter number. The returned ACK response contains a value as a signed 24-bit value (LSB first).

Command description			
Argument	Size	Value	Description
Command ID	1	0x46	MFU_READ_COUNTER
Counter number	1	0-2	Counter number
Response description			
ACK	1	0x00	
Command ID	1	0x46	MFU_READ_COUNTER
Counter value	3	X	Unsigned 24-bit value, LSB first

Example:

```

HOST=>C2: 0x46 – MFU_READ_COUNTER
          0x01 – counter number

C2=>HOST: 0x00 – ACK byte
          0x46 – related command code MFU_READ_COUNTER
          0x00 0x00 0x01 – value
  
```

4.3.8 Increment counter (0x47)

This command should be used to increment a counter stored in the tag memory. It takes as arguments the counter number and increment value (24-bit value LSB first) as 3 bytes.

Command description			
Argument	Size	Value	Description
Command ID	1	0x47	MFU_INCREMENT_COUNTER
Counter number	1	0-2	Counter number
Increment value	3	X	Unsigned 24-bit value (LSB first)
Response description			
ACK	1	0x00	
Command ID	1	0x47	MFU_INCREMENT_COUNTER

Example:

```

HOST=>C2: 0x47 - MFU_INCREMENT_COUNTER
          0x02 - block number 2
          0x00 0x00 0x01 - increment value

C2=>HOST: 0x00 - ACK byte
          0x47 - related command code MFU_INCREMENT_COUNTER
  
```

4.3.9 Password auth (0x48)

This command tries to authenticate the tag using the chosen password. It takes as an argument a password as four bytes. The returned ACK response contains two bytes of password acknowledge (PACK).

Command description			
Argument	Size	Value	Description
Command ID	1	0x48	MFU_PASSWORD_AUTH
Counter number	4	X	4-bytes password
Response description			
ACK	1	0x00	
Command ID	1	0x48	MFU_PASSWORD_AUTH
PACK	2	X	Password acknowledge bytes

Example:

```

HOST=>C2: 0x48 - MFU_PASSWORD_AUTH
          0x00 0x00 0x00 0x00 - password

C2=>HOST: 0x00 - ACK byte
          0x48 - related command code MFU_PASSWORD_AUTH
          0x00 0x00 - password acknowledge bytes
  
```

4.3.10 Check Tearing Event (0x4A)

The Check Tearing Event command takes as arguments one byte with the counter number. This command checks whether there was a tearing event in the counter. The returned ACK response contains result byte. The value '0x00' is returned if there has been no tearing event, and '0x01' is returned if a tearing event occurred. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x49	MFU_CHECKEVENT
Counter number	1	0-2	Counter number
Response description			
ACK	1	0x00	
Command ID	1	0x49	MFU_CHECKEVENT

Example:

```

HOST=>C2: 0x49 - MFU_CHECKEVENT
          0x00 - counter number

C2=>HOST: 0x00 - ACK byte
          0x49 - related command code MFU_CHECKEVENT
          0x01 - tearing event occurred

```

4.4 ICODE (ISO15693) commands

This set of commands should be performed on ICODE (ISO15693) TAGs.

4.4.1 Inventory start (0x90)

This command starts the inventory procedure on ISO 15693 TAGs. It activates the first TAG detected during collision resolution. If no TAGs are detected, then an error with a timeout flag is returned. This command takes one argument AFI - Application Family Identifier. Please refer to the NXP documentation for more information.

If any TAG(s) is/are detected, then the command returns an ACK message containing the UID (8-bytes), a DSFID byte, and 1-byte which contains information about any other tags detected in the field that are available to be read.

Because GET_TAG_COUNT command is limited to 5 tags only, ICODE_INVENTORY_START/ICODE_INVENTORY_NEXT commands should be used to detect all ICODE tags within range of the antenna.

Command description			
Argument	Size	Value	Description
Command ID	1	0x90	ICODE_INVENTORY_START
AFI	1	X	Application Family Identifier
Response description			
ACK	1	0x00	
Command ID	1	0x90	ICODE_INVENTORY_START
UID	8	XXX	Unique identifier, inverted order
DSFID	1	X	Data Storage Format Identifier
More cards flag	1	X	0x00 – no more cards in range of antenna 0x01 – more cards in range of antenna

Example:

```

HOST=>C2: 0x90 - ICODE_INVENTORY_START
          0x00 - Application Family Identifier

C2=>HOST: 0x00 - ACK byte
          0x90 - related command code ICODE_INVENTORY_START
          0x04 0x8F 0x7F 0x0A 0x01 0x24 0x16 0xE0 - UID
          0x00 - DSFID
          0x01 - more cards in range of antenna

```

4.4.2 Inventory next (0x91)

This command should be used to continue the inventory procedure on ISO 15693 TAGs. It activates the next TAG that was detected during the collision resolution. It takes one argument, AFI - Application Family Identifier. Please refer to the NXP documentation for more information. If a TAG or multiple tags is/are detected, then this command returns an ACK message containing the UID (8-bytes), a DSFID byte, and 1-byte which contains information about any other tags detected in the field that are available to be read.

Command description			
Argument	Size	Value	Description
Command ID	1	0x91	ICODE_INVENTORY_NEXT
AFI	1	X	Application Family Identifier
Response description			
ACK	1	0x00	
Command ID	1	0x91	ICODE_INVENTORY_NEXT
UID	8	XXX	Unique identifier
DSFID	1	X	Data Storage Format Identifier
More cards flag	1	X	0x00 – no more cards in range of antenna 0x01 – more cards in range of antenna

Example:

```

HOST=>C2: 0x91 - ICODE_INVENTORY_NEXT
          0x00 - Application Family Identifier

C2=>HOST: 0x00 - ACK byte
          0x91 - related command code ICODE_INVENTORY_NEXT
          0x04 0x8F 0x7F 0x0A 0x01 0x24 0x16 0xE0 - UID
          0x00 - DSFID
          0x00 - no more cards available for reading

```

4.4.3 Read block (0x93)

The read block command should be used to read data stored in TAG blocks. It takes as arguments the block number of the first block to be read, and the number of blocks to be read. The returned ACK answer contains data read from the specified tag memory. The number of bytes of this data is ICODE block size (4) multiplied by the number of blocks to be read.

Command description			
Argument	Size	Value	Description
Command ID	1	0x93	ICODE_READ_BLOCK
Block number	1	X	
Block count	1	N	Number of block to read
Response description			
ACK	1	0x00	
Command ID	1	0x93	ICODE_READ_BLOCK
Read data	4*N	XXX	Bytes read from the tag.

Example:

```

HOST=>C2: 0x93 - ICODE_READ_BLOCK
          0x02 - block number 2
          0x01 - 1 block to read

C2=>HOST: 0x00 - ACK byte
          0x93 - related command code ICODE_READ_BLOCK
          0x35 0x3a 0x30 0x33 - 4 bytes block data

```

4.4.4 Write block (0x94)

The write block command should be used to write data to the tag. It takes as arguments the block number of the first block to write, the number of blocks to write, and the bytes to be written. The number of bytes to be written must be exactly the number of blocks to write multiplied by 4.

Command description			
Argument	Size	Value	Description
Command ID	1	0x94	ICODE_WRITE_BLOCK
Block number	1	X	
Block count	1	N	
Data to write	4*N	X	4-bytes data to write
Response description			
ACK	1	0x00	
Command ID	1	0x94	ICODE_WRITE_BLOCK

Example:

```

HOST=>C2: 0x94 - ICODE_WRITE_BLOCK
          0x02 - block number 2
          0x01 - block count 1
          0x35 0x3a 0x30 0x33 - 4 bytes to write

C2=>HOST: 0x00 - ACK byte
          0x94 - related command code ICODE_WRITE_BLOCK
  
```

4.4.5 Lock block (0x95)

This command performs a lock block command. Once it receives the lock block command, the TAG permanently locks the requested block. The command takes a one-byte argument representing the block number to be locked.

Command description			
Argument	Size	Value	Description
Command ID	1	0x95	ICODE_LOCK_BLOCK
Block number	1	X	
Response description			
ACK	1	0x00	
Command ID	1	0x95	ICODE_LOCK_BLOCK

Example:

```

HOST=>C2: 0x95 - ICODE_LOCK_BLOCK
          0x02 - block number 2

C2=>HOST: 0x00 - ACK byte
          0x95 - related command code ICODE_LOCK_BLOCK
  
```

4.4.6 Write AFI (0x96)

This command performs a write to Application Family Identifier value inside the TAG memory. The command takes a one-byte argument representing the AFI value.

Command description			
Argument	Size	Value	Description
Command ID	1	0x96	ICODE_WRITE_AFI
AFI value	1	X	
Response description			
ACK	1	0x00	
Command ID	1	0x96	ICODE_WRITE_AFI

Example:

```

HOST=>C2: 0x96 - ICODE_WRITE_AFI
          0xAA - new Application Family Identifier value

C2=>HOST: 0x00 - ACK byte
          0x96 - related command code ICODE_WRITE_AFI
  
```

4.4.7 Lock AFI (0x97)

This command performs a Lock AFI command on the TAG. When it receives the lock AFI request, the TAG locks the AFI value permanently into its memory.

Command description			
Argument	Size	Value	Description
Command ID	1	0x97	ICODE_LOCK_AFI
Response description			
ACK	1	0x00	
Command ID	1	0x97	ICODE_LOCK_AFI

Example:

```

HOST=>C2: 0x96 - ICODE_LOCK_AFI

C2=>HOST: 0x00 - ACK byte
          0x96 - related command code ICODE_LOCK_AFI
  
```

4.4.8 Write DSFID (0x98)

This command performs a write to Data Storage Format Identifier value inside the TAG memory. This command takes a one-byte argument representing the DSFID value.

Command description			
Argument	Size	Value	Description
Command ID	1	0x98	ICODE_WRITE_DSFID
DSFID value	1	X	
Response description			

ACK	1	0x00	
Command ID	1	0x98	ICODE_WRITE_DSIFID

Example:

HOST=>C2: 0x98 – ICODE_WRITE_DSIFID
 0xAA – new Data Storage Format Identifier value

C2=>HOST: 0x00 – ACK byte
 0x98 – related command code ICODE_WRITE_DSIFID

4.4.9 Lock DSIFID (0x99)

This command performs a Lock DSIFID command on the TAG. When it receives the lock DSIFID request, the TAG locks the DSIFID value permanently into its memory.

Command description			
Argument	Size	Value	Description
Command ID	1	0x99	ICODE_LOCK_DSIFID
Response description			
ACK	1	0x00	
Command ID	1	0x99	ICODE_LOCK_DSIFID

Example:

HOST=>C2: 0x99 – ICODE_LOCK_DSIFID

C2=>HOST: 0x00 – ACK byte
 0x99 – related command code ICODE_LOCK_DSIFID

4.4.10 Get System Information (0x9A)

This command performs get system information command on the TAG. No arguments are required. The ACK response contains bytes with system information. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x9A	ICODE_GET_SYSTEM_INFORMATION
Response description			
ACK	1	0x00	
Command ID	1	0x9A	ICODE_GET_SYSTEM_INFORMATION
System information	X	XXX	System information bytes

Example:

```

HOST=>C2: 0x9A - ICODE_GET_SYSTEM_INFORMATION

C2=>HOST: 0x00 - ACK byte
          0x9A - related command code ICODE_GET_SYSTEM_INFORMATION
          0x0F 0x04 0x8F 0x7F 0x0A 0x01 0x24
          0x16 0xE0 0x00 0x00 0x33 0x03 0x02 - result bytes
    
```

4.4.11 Get multiple BSS (0x9B)

This command performs get multiple block security status command on the TAG. It takes as arguments the block number for which the status should be returned and the number of blocks to be used for returning the status. The ACK response contains bytes with block security status information. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x9B	ICODE_GET_MULTIPLE_BSS
First block number	1	X	
Number of blocks	1	N	
Response description			
ACK	1	0x00	
Command ID	1	0x9B	ICODE_GET_MULTIPLE_BSS
BSS information	N	X	Blocks security status information

Example:

```

HOST=>C2: 0x9B - ICODE_GET_MULTIPLE_BSS
          0x00 - starting block number
          0x08 - number of BSS to read

C2=>HOST: 0x00 - ACK byte
          0x9B - related command code ICODE_GET_MULTIPLE_BSS
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 - result bytes
    
```

4.4.12 Password protect AFI (0x9C)

This command enables the password protection for AFI. The AFI password has to be transmitted before with ICODE_SET_PASSWORD command.

Command description			
Argument	Size	Value	Description
Command ID	1	0x9C	ICODE_PASSWORD_PROTECT_AFI
Response description			
ACK	1	0x00	
Command ID	1	0x9C	ICODE_PASSWORD_PROTECT_AFI

Example:

```

HOST=>C2: 0x9C – ICODE_PASSWORD_PROTECT_AFI

C2=>HOST: 0x00 – ACK byte
          0x9C – related command code ICODE_PASSWORD_PROTECT_AFI
  
```

4.4.13 Read EPC (0x9D)

This command reads EPC data from the TAG. The ACK response contains 12-bytes of EPC data. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x9D	ICODE_READ_EPC
Response description			
ACK	1	0x00	
Command ID	1	0x9D	ICODE_READ_EPC
EPC information	12	X	Please refer to the NXP documentation for more information.

Example:

```

HOST=>C2: 0x9D – ICODE_READ_EPC

C2=>HOST: 0x00 – ACK byte
          0x9D – related command code ICODE_READ_EPC
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 – result bytes
  
```

4.4.14 Get NXP System Information (0x9E)

This command retrieves the NXP system information value from the TAG. No arguments are required. The ACK response contains bytes with the NXP system information. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0x9E	ICODE_GET_NXP_SYSTEM_INFORMATION
Response description			
ACK	1	0x00	
Command ID	1	0x9E	ICODE_GET_NXP_SYSTEM_INFORMATION
System information	X	XXX	System information bytes

Example:

```

HOST=>C2: 0x9E – ICODE_GET_NXP_SYSTEM_INFORMATION

C2=>HOST: 0x00 – ACK byte
          0x9E – related command code ICODE_GET_NXP_SYSTEM_INFORMATION
          0x0F 0x04 0x8F 0x7F 0x0A 0x01 0x24
          0x16 0xE0 0x00 0x00 0x33 0x03 0x02 – result bytes
  
```

4.4.15 Get random number (0x9F)

This command requests a random number from the ICODE TAG. No arguments are required. The ACK response contains a 16-bit random number. This value should be used with ICODE_SET_PASSWORD command.

Command description			
Argument	Size	Value	Description
Command ID	1	0x9F	ICODE_GET_RANDOM_NUMBER
Response description			
ACK	1	0x00	
Command ID	1	0x9F	ICODE_GET_RANDOM_NUMBER
Random number	2	XXX	16-bits random number

Example:

HOST=>C2: 0x9F - ICODE_GET_RANDOM_NUMBER

C2=>HOST: 0x00 - ACK byte
 0x9F - related command code ICODE_GET_RANDOM_NUMBER
 0x7F 0x14 - result bytes

4.4.16 Set password (0xA0)

This command sets the password for the selected identifier. This command has to be executed just once for the related passwords if the TAG is powered. The password is calculated as XOR with the random number returned by the previously executed command ICODE_GET_RANDOM_NUMBER.

Here is an example how to calculate XOR password:

```
xorPassword[0] = password[0] ^ rnd[0];
xorPassword[1] = password[1] ^ rnd[1];
xorPassword[2] = password[2] ^ rnd[0];
xorPassword[3] = password[3] ^ rnd[1];
```

Command description			
Argument	Size	Value	Description
Command ID	1	0xA0	ICODE_SET_PASSWORD
Password Identifier	1	X	0x01 - Read password 0x02 - Write password 0x04 - Privacy password 0x08 - Destroy password 0x10 - EAS/AFI password
XOR Password	4	X	
Response description			
ACK	1	0x00	
Command ID	1	0xA0	ICODE_SET_PASSWORD

Example:

```

HOST=>C2: 0xA0 - ICODE_SET_PASSWORD
          0x02 - write password
          0x34 0x76 0x39 0x64 - calculated XOR password
C2=>HOST: 0x00 - ACK byte
          0xA0 - related command code ICODE_SET_PASSWORD

```

4.4.17 Write password (0xA1)

This command writes a new password to a selected identifier. With this command, a new password is written into the related memory. Note that the old password has to be transmitted before with ICODE_SET_PASSWORD. The new password takes effect immediately which means that the new password has to be transmitted with ICODE_SET_PASSWORD to get access to the protected blocks/pages. It takes as arguments the password identifier byte and the plain password 4-bytes long.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA1	ICODE_WRITE_PASSWORD
Password Identifier	1	X	0x01 – Read password 0x02 – Write password 0x04 – Privacy password 0x08 – Destroy password 0x10 - EAS/AFI password
Password	4	X	Plain password
Response description			
ACK	1	0x00	
Command ID	1	0xA1	ICODE_WRITE_PASSWORD

Example:

```

HOST=>C2: 0xA1 - ICODE_WRITE_PASSWORD
          0x02 - write password
          0x34 0x76 0x39 0x64 - Plain password

C2=>HOST: 0x00 - ACK byte
          0xA1 - related command code ICODE_WRITE_PASSWORD

```

4.4.18 Lock password (0xA2)

This command locks the addressed password. Note that the addressed password has to be transmitted before with ICODE_SET_PASSWORD. A locked password can no longer be changed.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA2	ICODE_LOCK_PASSWORD
Password Identifier	1	X	0x01 – Read password

			0x02 – Write password 0x04 – Privacy password 0x08 – Destroy password 0x10 - EAS/AFI password
Response description			
ACK	1	0x00	
Command ID	1	0xA2	ICODE_LOCK_PASSWORD

Example:

HOST=>C2: 0xA2 – ICODE_LOCK_PASSWORD
0x02 – write password

C2=>HOST: 0x00 – ACK byte
0xA2 – related command code ICODE_LOCK_PASSWORD

4.4.19 Protect page (0xA3)

This command changes the protection status of a page. Note that the related passwords have to be transmitted before with ICODE_SET_PASSWORD if the page is not public. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA3	ICODE_PAGE_PROTECT
Page address	1	X	<ul style="list-style-type: none"> Page number to be protected in case of products that do not have pages characterized as high and Low. Block number to be protected in case of products that have pages characterized as high and Low.
Protection status	1	X	<ul style="list-style-type: none"> Protection status options for the products that do not have pages characterized as high and Low: 0x00: ICODE_PROTECT_PAGE_PUBLIC 0x01: ICODE_PROTECT_PAGE_READ_WRITE_READ_PASSWORD 0x10: ICODE_PROTECT_PAGE_WRITE_PASSWORD 0x11: ICODE_PROTECT_PAGE_READ_WRITE_PASSWORD_SEPERATE Extended Protection status options for the products that have pages characterized as high and Low: 0x01: ICODE_PROTECT_PAGE_READ_LOW 0x02: ICODE_PROTECT_PAGE_WRITE_LOW 0x10: ICODE_PROTECT_PAGE_READ_HIGH 0x20: ICODE_PROTECT_PAGE_WRITE_HIGH
Response description			
ACK	1	0x00	
Command ID	1	0xA2	ICODE_PAGE_PROTECT

Example:

HOST=>C2: 0xA3 – ICODE_PAGE_PROTECT
0x02 – second block selected

0x01 - ICODE_PROTECT_PAGE_READ_LOW flag selected

C2=>HOST: 0x00 - ACK byte
0xA3 - related command code ICODE_PAGE_PROTECT

4.4.20 Lock page protection (0xA4)

This command permanently locks the protection status of a page. Note that the related passwords have to be transmitted before with ref ICODE_SET_PASSWORD if the page is not public.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA4	ICODE_LOCK_PAGE_PROTECTION
Page number	1	X	
Response description			
ACK	1	0x00	
Command ID	1	0xA4	ICODE_LOCK_PAGE_PROTECTION

Example:

HOST=>C2: 0xA4 - ICODE_LOCK_PAGE_PROTECTION
0x02 - page number
C2=>HOST: 0x00 - ACK byte
0xA4 - related command code ICODE_LOCK_PAGE_PROTECTION

4.4.21 Get multiple block protection status (0xA5)

This instructs the label to return the block protection status of the requested blocks. It takes as arguments the first block number to get the block protection status and the number of blocks.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA5	ICODE_GET_MULTIPLE_BPS
First block number	1	X	
Number of blocks	1	N	
Response description			
ACK	1	0x00	
Command ID	1	0xA5	ICODE_GET_MULTIPLE_BPS
BSS information	N	X	Blocks protection status information

Example:

HOST=>C2: 0xA5 - ICODE_GET_MULTIPLE_BPS
0x00 - starting block number
0x08 - number of BSS to read
C2=>HOST: 0x00 - ACK byte

0xA5 - related command code ICODE_GET_MULTIPLE_BPS
 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 - result bytes

4.4.22 Destroy (0xA6)

This command permanently destroys the label (tag). The destroy password has to be transmitted before with ICODE_SET_PASSWORD. This command is irreversible and the label will never respond to any command again. This command can take the XOR password argument for the ICODE products that requires this argument. The XOR password calculation method is described in the ICODE_SET_PASSWORD description.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA6	ICODE_DESTROY
XOR password	4	X	Optional XOR password
Response description			
ACK	1	0x00	
Command ID	1	0xA6	ICODE_DESTROY

Example:

```
HOST=>C2: 0xA6 - ICODE_DESTROY
C2=>HOST: 0x00 - ACK byte
          0xA6 - related command code ICODE_DESTROY
```

4.4.23 Enable privacy (0xA7)

This command instructs the label to enter privacy mode. In privacy mode, the label will only respond to ICODE_GET_RANDOM_NUMBER and ICODE_SET_PASSWORD commands. To get out of the privacy mode, the Privacy password has to be transmitted before with ICODE_SET_PASSWORD.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA7	ICODE_ENABLE_PRIVACY
XOR password	4	X	Optional XOR password
Response description			
ACK	1	0x00	
Command ID	1	0xA7	ICODE_ENABLE_PRIVACY

Example:

```
HOST=>C2: 0xA7 - ICODE_ENABLE_PRIVACY
C2=>HOST: 0x00 - ACK byte
          0xA7 - related command code ICODE_ENABLE_PRIVACY
```

4.4.24 Enable 64-bit password (0xA8)

This instructs the label that both Read and Write passwords are required for protected access. Note that both the Read and Write passwords have to be transmitted before with ICODE_SET_PASSWORD.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA8	ICODE_ENABLE_64BIT_PASSWORD
Response description			
ACK	1	0x00	
Command ID	1	0xA8	ICODE_ENABLE_64BIT_PASSWORD

Example:

```
HOST=>C2: 0xA8 - ICODE_ENABLE_64BIT_PASSWORD
C2=>HOST: 0x00 - ACK byte
          0xA8 - related command code ICODE_ENABLE_64BIT_PASSWORD
```

4.4.25 Read signature (0xA9)

This command reads the signature bytes from the TAG. No arguments are required. The ACK response contains bytes containing the signature bytes. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0xA9	ICODE_READ_SIGNATURE
Response description			
ACK	1	0x00	
Command ID	1	0xA9	ICODE_READ_SIGNATURE
Signature bytes	X	XXX	Signature bytes

Example:

```
HOST=>C2: 0xA9 - ICODE_READ_SIGNATURE
C2=>HOST: 0x00 - ACK byte
          0xA9 - related command code ICODE_READ_SIGNATURE
          0x0F 0x04 0x8F 0x7F 0x0A 0x01 0x24
          0x16 0xE0 0x00 0x00 0x33 0x03 0x02 - result bytes
```

4.4.26 Read config (0xAA)

This command reads multiple 4-byte data chunks from the selected configuration block address. It takes two arguments, the first block number and the number of blocks to read the configuration data.

Command description			
Argument	Size	Value	Description
Command ID	1	0xAA	ICODE_READ_CONFIG
First block number	1	X	
Number of blocks	1	N	
Response description			
ACK	1	0x00	
Command ID	1	0xAA	ICODE_READ_CONFIG
Configuration bytes	N*4	X	

Example:

```

HOST=>C2: 0xAA - ICODE_READ_CONFIG
          0x00 - starting block number
          0x02 - number of blocks to read

C2=>HOST: 0x00 - ACK byte
          0xAA - related command code ICODE_READ_CONFIG
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 - result bytes

```

4.4.27 Write config (0xAB)

This command writes configuration bytes to addressed block data from the selected configuration block address. It takes three arguments: the option byte, the block number and the configuration bytes. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0xAB	ICODE_WRITE_CONFIG
Option byte	1	X	0x01 – Enable option 0x00 – Disable option
Block number	1	X	
Configuration bytes	4	X	
Response description			
ACK	1	0x00	
Command ID	1	0xAB	ICODE_WRITE_CONFIG

Example:

```

HOST=>C2: 0xAB - ICODE_WRITE_CONFIG
          0x01 - option byte
          0x00 - block number
          0x00 0x00 0x00 0x00 - config bytes

C2=>HOST: 0x00 - ACK byte
          0xAB - related command code ICODE_WRITE_CONFIG

```

4.4.28 Pick random ID (0xAC)

This command enables the random ID generation in the tag. This interface is used to instruct the tag to generate a random number in privacy mode. Please refer to the NXP documentation for more information.

Command description			
Argument	Size	Value	Description
Command ID	1	0xAC	ICODE_PICK_RANDOM_ID
Response description			
ACK	1	0x00	
Command ID	1	0xAC	ICODE_PICK_RANDOM_ID

Example:

```
HOST=>C2: 0xAB - ICODE_PICK_RANDOM_ID

C2=>HOST: 0x00 - ACK byte
          0xAB - related command code ICODE_PICK_RANDOM_ID
```

4.4.29 Extended read block (0xB3)

The extended read block command should be used to read data stored in TAG blocks but only if the tag supports this command – if you are not sure please use ICODE_READ_BLOCK command. It takes as arguments the block number of the first block to be read, and the number of blocks to be read. The returned ACK answer contains data read from the specified tag memory. The number of bytes of this data is ICODE block size (4) multiplied by the number of blocks to be read.

Command description			
Argument	Size	Value	Description
Command ID	1	0xB3	ICODE_EXT_READ_BLOCK
Block number	2	X	Unsigned 16bit value with LSB order.
Block count	1	N	Number of block to read
Response description			
ACK	1	0x00	
Command ID	1	0xB3	ICODE_EXT_READ_BLOCK
Read data	4*N	XXX	Bytes read from the tag.

Example:

```
HOST=>C2: 0xB3 - ICODE_EXT_READ_BLOCK
          0x02 0x00 - block number 2
          0x01 - 1 block to read

C2=>HOST: 0x00 - ACK byte
          0xB3 - related command code ICODE_EXT_READ_BLOCK
          0x35 0x3a 0x30 0x33 - 4 bytes block data
```

4.4.30 Extended write block (0xB4)

The extended write block command should be used to write data to the tag but only if the tag supports this command – if you are not sure please use ICODE_WRITE_BLOCK command. It takes as arguments the block number of the first block to write, the number of blocks to write, and the bytes to be written. The number of bytes to be written must be exactly the number of blocks to write multiplied by 4.

Command description			
Argument	Size	Value	Description
Command ID	1	0xB4	ICODE_EXT_WRITE_BLOCK
Block number	2	X	Unsigned 16bit value with LSB order.
Block count	1	N	
Data to write	4*N	X	4-bytes data to write
Response description			
ACK	1	0x00	
Command ID	1	0xB4	ICODE_EXT_WRITE_BLOCK

Example:

```

HOST=>C2: 0xB4 - ICODE_EXT_WRITE_BLOCK
          0x02 0x00 - block number 2
          0x01 - block count 1
          0x35 0x3a 0x30 0x33 - 4 bytes to write

C2=>HOST: 0x00 - ACK byte
          0xB4 - related command code ICODE_WRITE_BLOCK

```

4.4.31 ICODE custom command (15693) (0xBF)

The reader is capable to send custom commands over ISO 15693 protocol. The device adds SOF, EOF, and CRC16 automatically so the host has to prepare only the content of the frame. The ACK frame contains bytes received from the tag including response flags and all bytes except SOF, EOF, CRC16.

This command can be useful if you want to execute non-standard commands to tags like ST25. The first execution of the command enables the RF field. It can be mixed with standard commands, so the host software can execute the inventory command first and then send a custom command to do non-standard operations on the TAG.

Command description			
Argument	Size	Value	Description
Command ID	1	0xBF	ICODE_CUSTOM_COMMAND
DATA	X	X	Custom data send to the TAG
Response description			
ACK	1	0x00	
Command ID	1	0xBF	ICODE_CUSTOM_COMMAND
ACK data	x	x	Bytes received from the TAG

Example inventory command:

HOST=>C2: 0xBF - ICODE_CUSTOM_COMMAND
 26 01 00 - inventory command bytes

C2=>HOST: 0x00 - ACK byte
 0xBF - related command code ICODE_CUSTOM_COMMAND
 00 00 0F B0 30 02 00 39 02 E0 - tag response with response
 flags bytes

4.5 OTA upgrade

The commands listed below can be used to perform an OTA upgrade.

4.5.1 OTA begin (0xF0)

This command must be executed to start the OTA upgrade process. The device responds with an ACK frame when the command is finished.

Command description			
Argument	Size	Value	Description
Command ID	1	0x0F0	OTA begin
Response description			
ACK	1	0x00	
Command ID	1	0xF0	OTA begin

Example:

HOST=>C2: 0xF0 - OTA begin

C2=>HOST: 0x00 - ACK byte
 0xF0 - related command code OTA begin

4.5.2 OTA firmware frame (0xF1)

When the OTA begin frame has already been executed, the host application can upload binary firmware file in chunks that are 128 bytes long (the last frame may be smaller).

Command description			
Argument	Size	Value	Description
Command ID	1	0x0F1	OTA frame
Firmware bytes	Max. 128		Firmware bytes in chunks 128bytes long.
Response description			
ACK	1	0x00	
Command ID	1	0xF1	OTA frame

Example:

```

HOST=>C2: 0xF1 - OTA frame
          0x34 0x67 ... 0x45 - firmware bytes

C2=>HOST: 0x00 - ACK byte
          0xF1 - related command code OTA frame
    
```

4.5.3 OTA finish (0xF2)

The command must be executed after all firmware frames are written to the device. The bootloader application checks the integrity of the application. After this step the host can send the REBOOT command to reboot the device and run the new firmware. If there is a problem with communication after a device upgrade, please perform a factory reset.

Command description			
Argument	Size	Value	Description
Command ID	1	0x0F2	OTA finish
Response description			
ACK	1	0x00	
Command ID	1	0xF2	OTA finish

Example:

```

HOST=>C2: 0xF4 - OTA finish

C2=>HOST: 0x00 - ACK byte
          0xF4 - related command code OTA finish
    
```

5. Revision history

Revision	Date	Changes
1.0	30-Jul-2025	First release
1.1	3-Dec-2025	Sleep command added
1.2	15-May-2026	Section about GPIO command updated

MIFARE, MIFARE Ultralight, MIFARE Plus, MIFARE Classic, and MIFARE DESFire are trademarks of NXP B.V.

No responsibility is taken for the method of integration or final use of the Pepper C2 readers.

More information about the Pepper C2 family and other products can be found at the Internet site:

www.eccel.co.uk

or alternatively contact ECCEL Technology by e-mail at:

sales@eccel.co.uk