Pepper C1 MUX User Manual

Manual version: V2.5[1]
23/09/2022

Table of Contents

[1] The newest User manual can be found on our website: https://eccel.co.uk/wp-content/downloads/Pepper_C1/C1_MUX_User_manual.pdf

1. Introduction

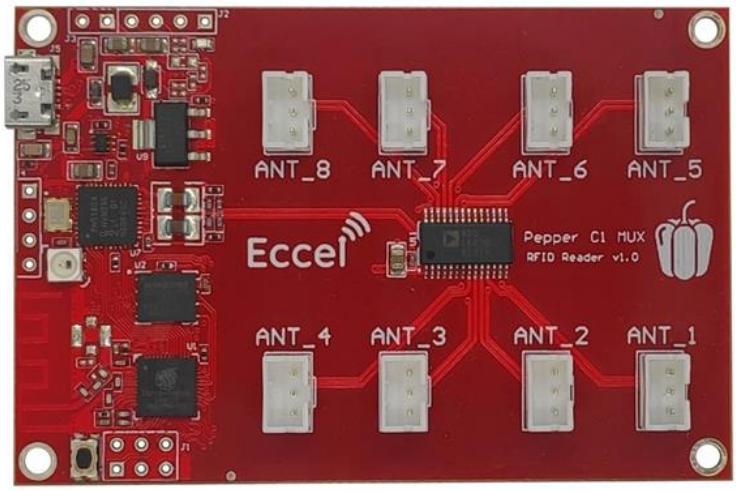
1.1 Device Overview

Features

- Low cost RFID Reader with MIFARE® Classic® in 1K, 4K memory, ICODE, MIFARE Ultralight®, MIFARE DESFire® EV1/EV2, MIFARE Plus® support
- Wireless connectivity:
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth SPP profile, BLE HID, and custom BLE service
- Built in Web Interface
- Over-the-Air lifetime updates
- Command interface via UART and TCP sockets with optional AES-128 encryption
- UART baud rate up to 921600 bps
- Configurable RGB LED indicator for RFID or Wi-Fi events
- Stand-alone mode (polling) up to 8 external antennas
- Antenna selection by one simple command
- IoT interfaces: MQTT, WebSocket
- High transponder read and write speed
- -25°C to 85°C operating range
- Multiple internal reference voltages
- RoHS compliant
- CE (RED) and UKCA compliant. FCC/ISED/PSE and other approvals easily obtained (see section 15)

Applications

- Access control
- Monitoring goods
- Approval and monitoring consumables
- Pre-payment systems
- Managing resources
- Contact-less data storage systems
- Evaluation and development of RFID systems



Description

The Pepper C1 MUX module is the multiplexed version of the Pepper C1 – the first Eccel Technology Ltd product with wireless connectivity connectivity by Wi-Fi 802.11b/g/n and Bluetooth SPP profile. The user can connect up to 8 external RFID antennas. Thanks to the wireless connectivity, the customer receives free lifetime Over-the-Air updates, and of course the communication protocol can be used over TCP instead of the traditional UART/USB interface. Combining these features with standalone mode provides a "straight out of the box" ready to use device for many application. In standalone mode, the module can also send a tag UID over MQTT or WebSockets, making it easy to integrate with IoT systems.

So, this is an ideal design choice if the user wishes to add RFID capability to their design quickly and without requiring extensive RFID and embedded software expertise and time. An advanced and powerful 32-bit microcontroller handles the RFID configuration setup and provides the user with a powerful yet simple command interface. This facilitates fast and easy read/write access to the memory and features of the various transponders supported by this module.

2. Electrical specification

2.1 Absolute maximum ratings

Stresses beyond the absolute maximum ratings listed in the table below may cause permanent damage to the device. These are stress ratings only, and do not refer to the functional operation of the device that should follow the recommended operating conditions.

Symbol	Parameter	Min	Max	Unit
T_S	Storage temperature	-40	+125	°C
V_{DDMAX}	Supply voltage (USB or J4 header)	3	5.5	V

Table 2-1. Absolute maximum ratings

2.2 Operating conditions

Symbol	Parameter	Min	Typ	Max	Unit
T_{Op}	Operating temperature	-25	25	+85	°C
H	Humidity	5	60	95	%
V_{DD}	Supply voltage (USB or J4 header)	3	5	5.5	V

Table 2-2. Operating conditions

2.3 DC characteristics (V_{DD} = 5 V, T_S = 25 °C)

Symbol	Parameter	Min	Typ	Max	Unit
V_{OUT}	Output voltage (regulator output, 3V3 pin on the J1 header)	3.23	3.3	3.37	V
V_{IH}	High-level input voltage (J1 header)	0.75 x V_{OUT}	-	V_{OUT} + 0.3	V
V_{IL}	Low-level input voltage (J1 header)	0	-	0.3 x V_{OUT}	V
V_{OH}	High-level output voltage (J1 header)	0.8 x V_{OUT}	-	-	V
V_{OL}	Low-level output voltage (J1 header)	-	-	0.3 x V_{OUT}	V
V_{ORS232}	V output RS232 (J2 header, RS232_TX pin)	-	5	5.4	V
V_{IRS232}	V input RS232 (J2 header, RS232_RX pin)	-25	-	+25	V

Table 2-3. DC characteristics

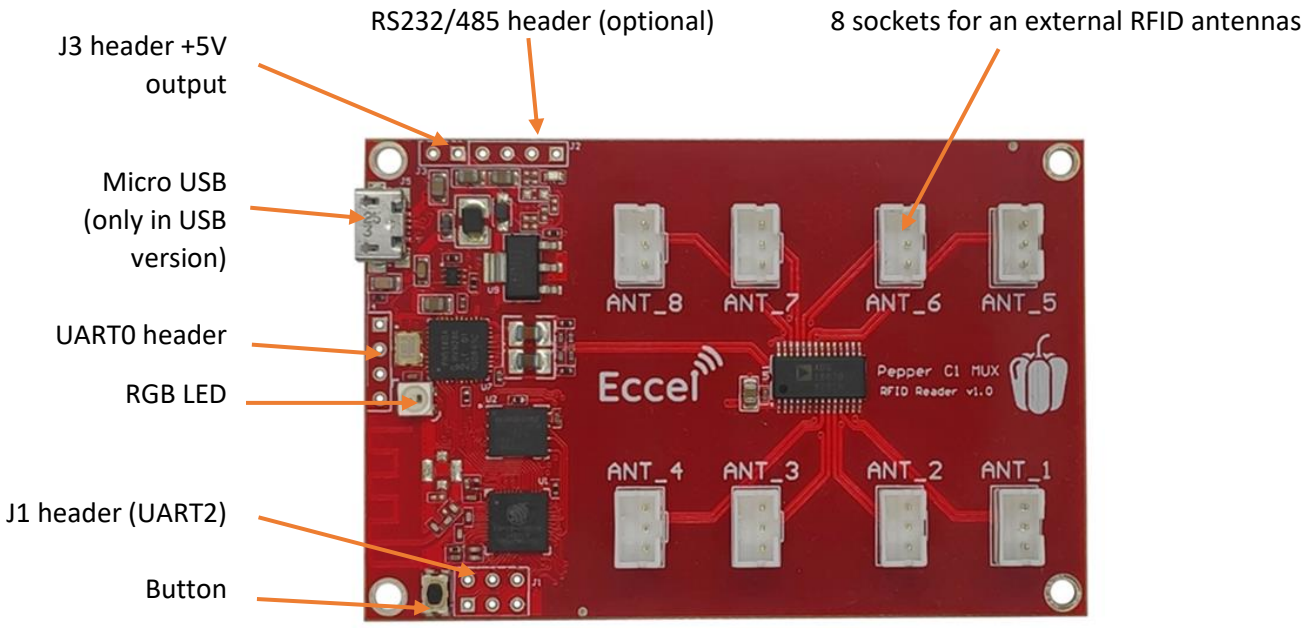
2.4 Current consumption (5V input)

Symbol			Parameter	Typ	Max	Unit
Wi-Fi enabled	Access Point mode	$I_{PN\_RFOFF\_AP}$	RF field off (AP)	150	170	mA
		$I_{PN\_RFON\_AP}$	RF field on (AP)	190	210	mA
	Station mode	$I_{PN\_RFOFF\_STA}$	RF field off (STA)	75	95	mA
		$I_{PN\_RFON\_STA}$	RF field on (STA)	130	150	mA
Wi-Fi Off	I_{PN_RFOFF}		RF field off	65	70	mA
	I_{PN_RFON}		RF field on	120	140	mA

Table 2-4. Current consumption

3. Getting started

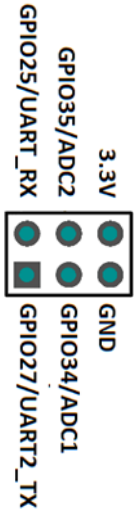
3.1 IO and peripherals



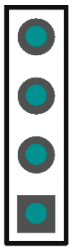
Micro USB – *only in USB version. Connected to the built in USB to TTL converter. This converter is routed to the UART0 header.*
RS232 header – *this connection is for optional built in RS232 converter. This option is available here:* *https://eccel.co.uk/product/pepper-c1-mux-rs232/*

3.1.1 J1 header description

- **GND** – Ground
- **3.3V** – Output
- **GPIOx** – general-purpose input/output
- **UART2_RX/UART2_TX** – UART2 in TTL standard with 3.3V levels



3.1.2 J2 header description (RS232/RS485 version only)



RS232_TX (from C1 MUX to host, max output voltage level ±5V)

RS232_RX (from host to C1 MUX, max input voltage level ±25V)

B (C1 MUX RS485 version)

A (C1 MUX RS485 version)

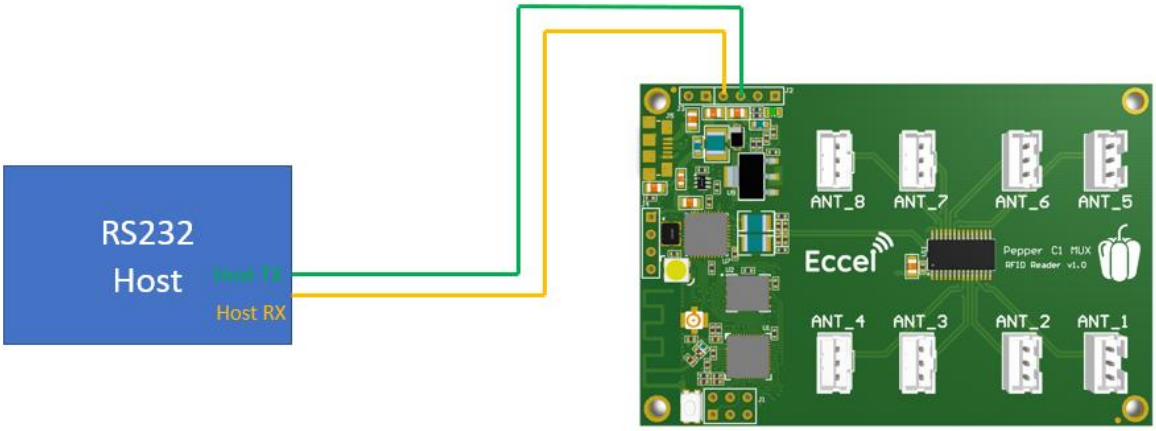


Figure 1 RS232 connection (Pepper C1 MUX RS232 version)

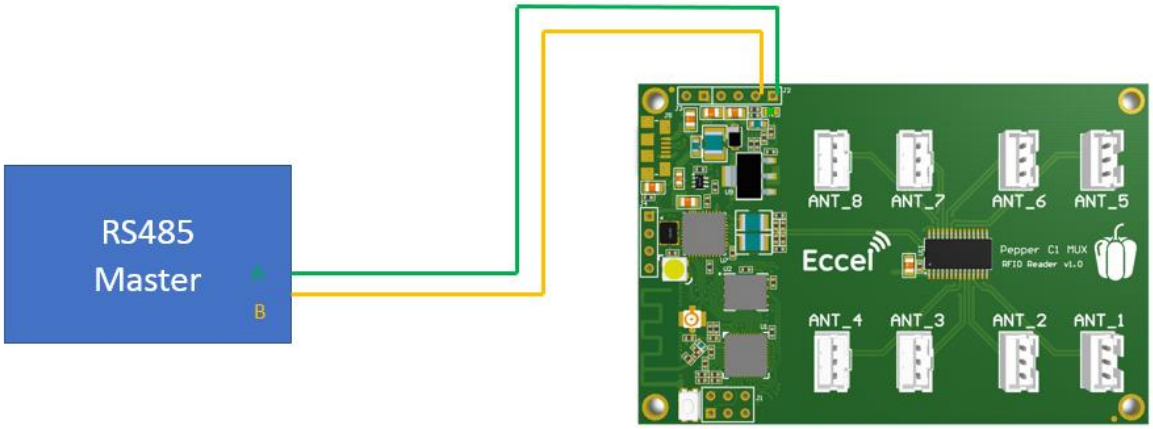
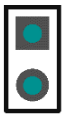


Figure 2 RS485 connection (Pepper C1 MUX RS485 version)

3.1.3 J3 header description

The J3 header is an additional power supply output socket. The maximum output current depends on the power supply connected to the J4 Vin pin, and is estimated as 50mA.



GND

+5V output

3.1.4 J4 UART0 header

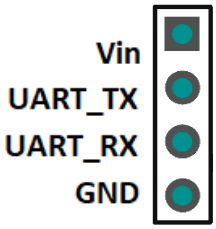
This is the UART0 header in the TTL standard with 3.3V levels. This is the same UART as it available on the USB port in the USB version.

Vin – Power supply, 3-5.5Voltage

UART0 TX – UART TX data from the module

UART0 RX – UART RX data to the module

GND – ground

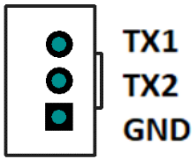


3.1.5 J6 External antenna header

The user has the possibility to work with up to 8 external RFID antennas simultaneously. Eccel Technology Ltd provides a variety of RFID antennas which the user can use together with this device: https://eccel.co.uk/product-category/antennas/

TX1, TX2 - Antenna driver output

GND – ground



3.2 Typical connection

The Pepper C1 MUX device can be connected to a host computer using a standard USB Micro cable. In the same way it can be powered to operate as a standalone device by using power sources such as a USB charger or power bank.

The computer operating system should recognize this device as a USB to TTL bridge or a USB to Serial port converter and it should appear in Windows device manager as a COM port. By default this COM port can be used for communication using the binary protocol described below.

The Reader also has the UART connector (J1 header) where the user can view output logs which contain additional information about temporary executing commands. The default configuration: baud: 115200, Data: 8 bit, Parity: none, Stop bits: 1 bit, Flow Control: none.

Hint – If you don't have your own USB to TTL converter to connect to the log console available on the UART2 header, you can temporary change the communication method in the Communication tab to UART2 or TCP, then console logs should be available on the USB port.

4. Configuration – Web interface

The reader has Wi-Fi functionality and can be configured through the Web interface. The Pepper C1 MUX can work in either station mode or client mode. The default mode is station mode. The user can login using the web interface and set a SSID and a password for their Wi-Fi network.
The Web interface is divided into several sections: The Network configuration, RFID, Communication, IOT, Status and Upgrade. All sections are described below.

4.1 Network Configuration

The very first use of the Pepper C1 MUX Reader Web interface:

1. Connect your PC to the Wi-Fi Access Point named: Pepper_C1-MUX-XXXXXX, where XXXXXX is the last three bytes of the MAC address, e.g. Peeper_C1-MUX-567801.
2. Open your web browser and enter http://192.168.100.1
3. Enter the default username: admin, and the default password: admin.

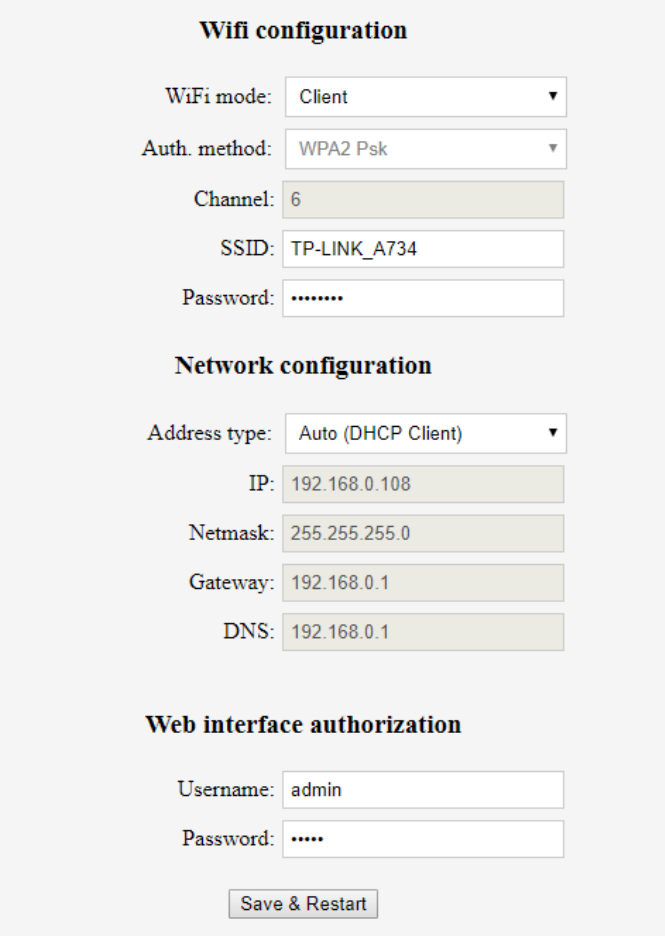


Figure 4-1 Web interface. Network configuration - Access point.

At this stage, change the Wi-Fi mode to Client, enter your SSID and the password. Change the IP, or set the Address type to Auto (DHCP Client). Optionally the user can change the Username and Password for the Web Interface. At the end of this process above, the Save & Restart button should be pressed. If you setup automatic IP and you don't know what IP is assigned by the DHCP server, you can browse the device logs to find this information.

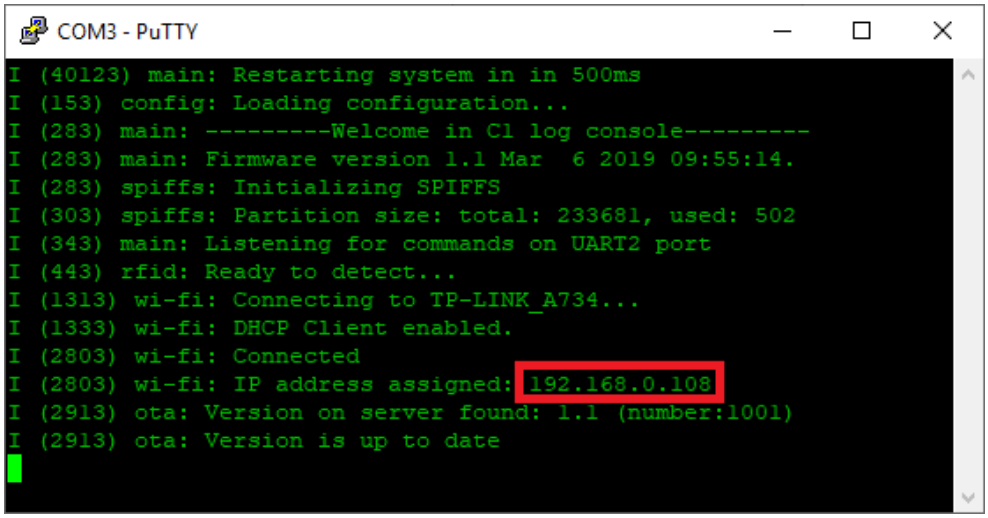


Figure 4-2 Output console. New IP address in the client mode.

The Pepper C1 MUX is now configured as a client, connected to a TP-LINK_A734. The automatic generated IP number is 192.168.0.108.

4.2 RFID

In this tab the user can change configuration for the default RFID behavior. This tab has three subcategories relating to RFID functionality and built in polling options:

- Polling
- Known UIDs
- RFID keys

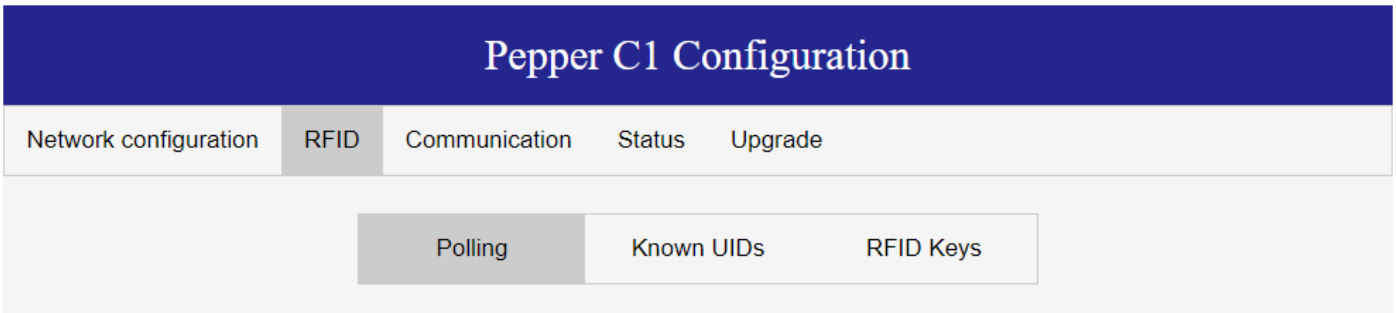


Figure 4-3 Configuration tabs for RFID

More information about this functionality is provided in the Polling mode section in this document.

4.3 Communication interfaces

4.3.1 General configuration

On this tab we can configure general options for the device.

- **MDNS service** – when this option is enabled this option device will announce its own name over this service. You can also query for _pepperc1._tcp.local to search all devices in the network. This option is enabled by default.
- **UDP discovery** – this is our custom UDP broadcast service listening on port 63311. To search for a devices in the network host have to send string "P_C1:SCAN" as broadcast message to the network and all devices should send response in format P_C1:<device name>:<version>. eg: P_C1:Pepper_C1-1A64D4:2.0
- **Device name** – this name will be used in all services, included in JSON frames etc.
- **Protocol password** – this is optional password needed for wireless connections like TCP client/server and BLE service.

4.3.2 UART configuration

On this configuration tab the user can select what will be provided on the UARTs available on the Pepper C1. Two UARTs are available

- **UART0/USB** – this UART port is accessible over USB connection for boards with USB port, or on the J4 port if boards don't have USB port.
- **UART2** – this UART port is available on the J1 port

On these ports we can select different protocols:

- **Binary protocol** – this is the standard protocol described in section 8.
- **Console logs** – with this option selected the reader sends internal logs to the user.
- **Modbus/ RS485 binary** – this protocol is only available on the UART2 port, this should be used on the boards with a RS485 converter.
- **UART Passthru** – this option should be enabled if you want to use other external devices over this UART port.

4.3.3 TCP client/server

These services provide communication using TCP connection. Frames should be sent in binary format. The user can configure a port for this service, timeout and server address for TCP client

4.3.4 Bluetooth

Three options are available for Bluetooth communication:

- Bluetooth SPP - Serial Port Profile
- BLE service – this is a custom Bluetooth Low Energy service. More details about this profile can be found in the Bluetooth Interface section in this document
- BLE HID – this profile can be used to emulate a BLE HID keyboard

Warning!
Bluetooth services use a lot of module memory, so in some cases Bluetooth service is not enabled at startup. The reader waits one minute at startup and if no activity is detected on the web interface, then the web service is disabled to release memory needed for the Bluetooth service. During this period, the module is blinks blue every 3 seconds.

4.3.5 MQTT

The device has a built in MQTT client and this tab is used to configure parameters needed for this communication. When the MQTT service is enabled **and the built in polling is enabled,** JSON frames with basic information about the tag is sent to the MQTT server. Please read MQTT interface description for more information about this interface and frame format.

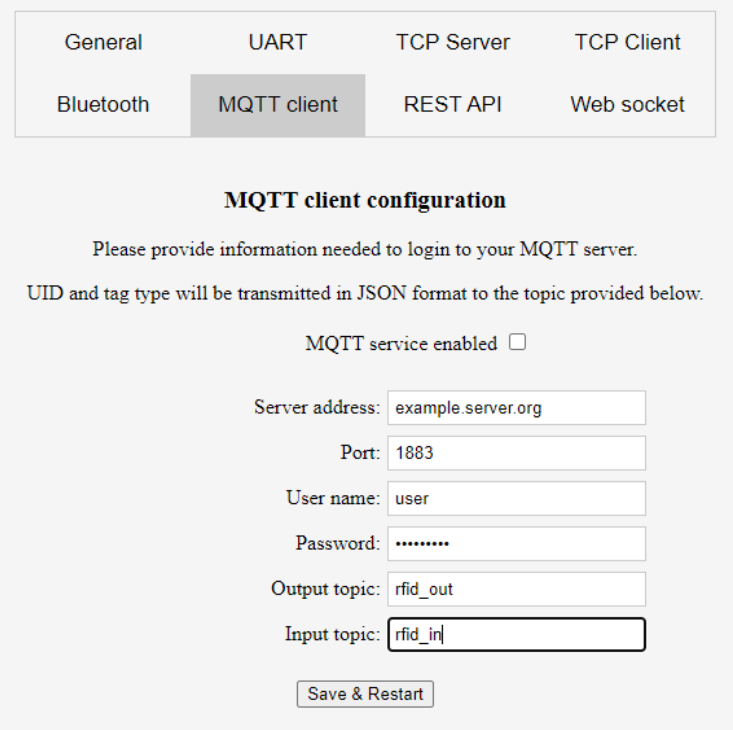


Figure 4-4 Web interface - the MQTT client configuration tab.

The picture below shows an example of a JSON frame received in a Node-RED system.

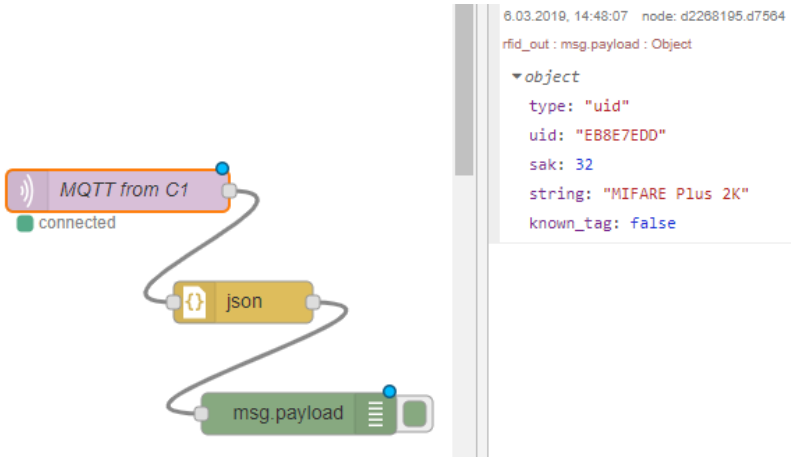


Figure 4-5 Node-Red – the MQTT client + JSON frame example

4.3.6 REST API interface

The device can also send frames in JSON format over REST API using the POST method. The user has to setup URL and authorization details if needed. This service also needs to have built in polling mode enabled. HTTPS protocol is also available but not recommended because of device performance. If it is used it is recommended to setup polling delay when the tag is detected.

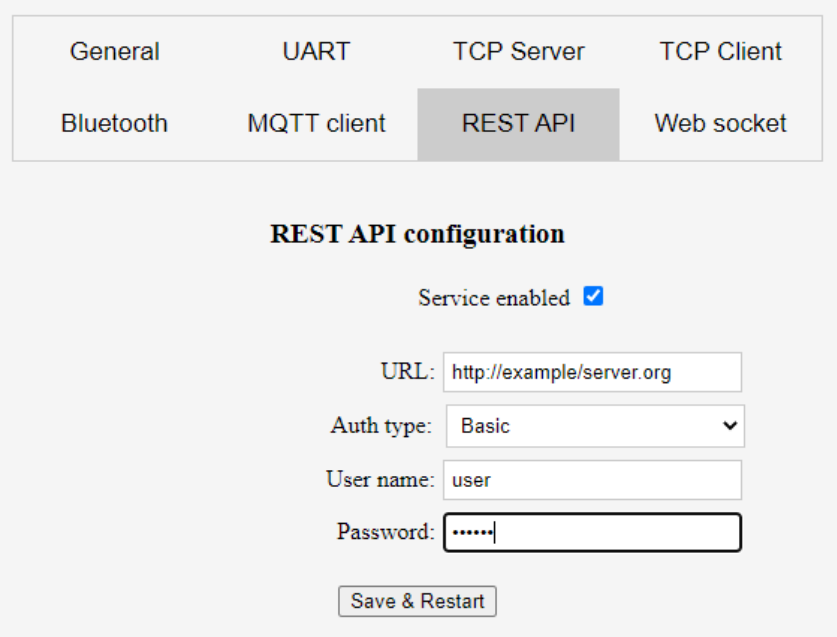


Figure 4-7 REST API configuration tab

4.3.7 Web sockets

In a similar way to the MQTT protocol, the device can send JSON messages over Web Sockets. If this service is enabled **and built in polling is enabled,** JSON frames can be handled using a Web socket with address ws://<device ip address>/<web socket name> eg. ws://172.16.16.62/wscomm.cgi.

Figure 4-6 Web interface – the Web socket configuration

4.4 Status

This page provides information about the current firmware version, and basic information about the TAGs in range of the antenna. Keep in mind that built in polling must be enabled to get information from the tags. The clear page button will clear all readings. On status page you can also check information about memory available in the reader.

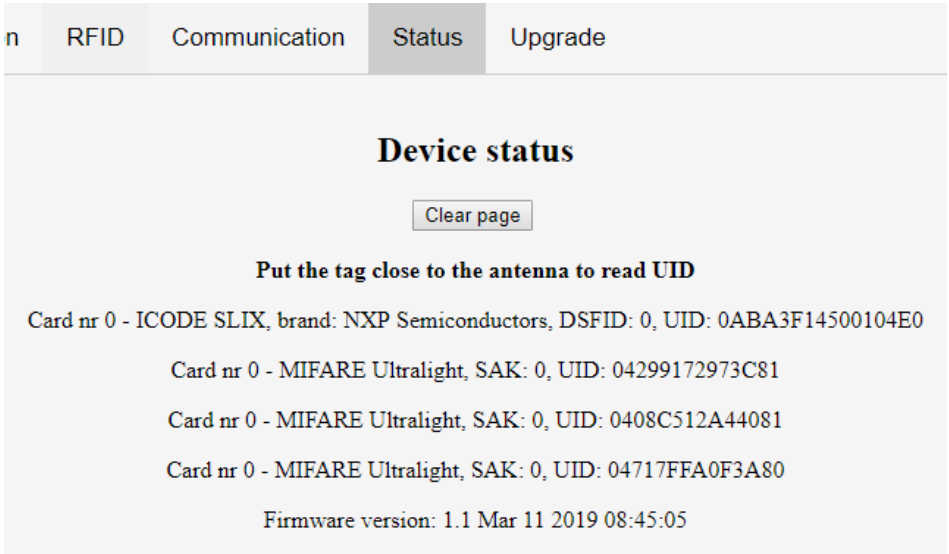


Figure 4-7 Web interface – the Status page

4.5 Firmware upgrade

In the Upgrade tab, the user is able to upgrade the reader firmware. There are two options: select the binary file to upload, or make an OTA Upgrade (Over The Air), which is a powerful feature of the Pepper C1. By clicking the OTA Upgrade button, the firmware file will be downloaded directly from our website www.eccel.co.uk to the reader flash memory and a firmware update will be performed. Each time the user visits the Upgrade tab, they will see information about the availability of the latest firmware version.

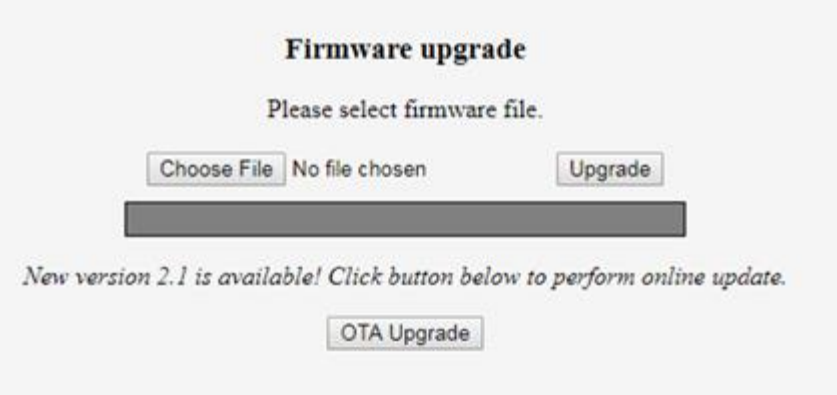


Figure 4-8 Web interface – the Firmware upgrade tab

4.6 Backup & Restore

In this tab the user can backup settings to the JSON file. This is a human readable format and therefore can be modified by the user. The backup file can be uploaded to any device with firmware higher than 2.0. and overwrites current settings in the device.

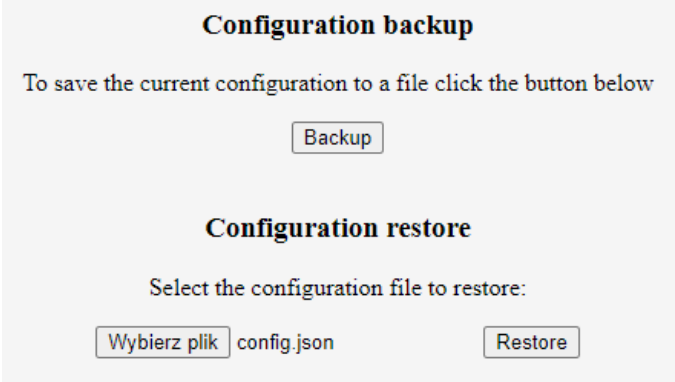


Figure 4-10 Backup & Restore tab

5. Rescue mode and factory reset

If the user forgets the password to the module or if the settings for the Wi-Fi need to be updated, the Pepper C1 MUX device provides two modes to resolve this situation: the rescue mode and factory defaults reset.

5.1 Rescue mode

This mode is dedicated specifically to update Wi-Fi connection settings or to access the web interface when the Wi-Fi is disabled. To enable this mode please follow this steps:

- Power up device
- Press the button and hold it for about 5 seconds – device blinks red every 1 second, release the button when device blinks white. **Do not hold the button longer if you don't want to perform full factory reset**
- The device should be available as an Access Point with the name Pepper_C1-MUX-xxxxxx. If the user has already provided a password for Wi-Fi connection, then this password needs to be entered in order to access the device. If a password has not yet been inputted by the user, then the device will be open and will not require any password for access

5.2 Automatic rescue mode

From firmware version 1.5 onwards, the Pepper C1 family is able to detect some faulty configurations and software problems automatically. If the device is not able to run for more than 15 seconds with the selected settings and keeps restarting, it runs in safe mode with all services turned off with only Wi-Fi and web interface running (if enabled in the configuration - if not the user can enable it by holding the button for three seconds.). The user will be informed about this situation by a message in the browser when the web interface is launched.

5.3 Resetting module to factory defaults

If the user wants to erase all settings stored in the device to factory defaults including Wi-Fi settings, communication settings and known UIDs, then the steps below need to be followed:

- Power up the device
- Press the button and hold it for about 10 seconds
- Release the button when the device blinks green
- The device should reboot itself and should be available for the user with default settings

6. MQTT interface

When MQTT client is configured in the web configurator and it is connected to the server the Pepper C1 MUX can send and receive frames in JSON format as described below.

6.1 Status frame

This frame is sent by the device to the server about the current status of the device. Currently it is only one frame with status startup.

Example:

```
{
      "type":          "startup",
      "device-name":   "Pepper_C1-MUX-1A64D5"
}
```

6.2 RFID frame

When RFID polling is enabled, the device sends information about the currently detected TAG.

Example:

```
{
      "type":          "uid",
      "uid":           "D89A7424",
      "sak":           8,
      "string":        "MIFARE Classic 1k/Plus 2k",
      "device-name":   "Pepper_C1-MUX-1A64D5",
      "memory":        "00112233445566770011223344556677",
      "known_tag":     false,
      "antenna":       2,
}
```

6.3 UART passthru frame

When passthru mode for UART2 is active, the device sends data received from the UART port to the server using frame with type set to “uart”. This method of communication can be used to transmit only text frames. If the host wants to use binary over UART2, then binary communication protocol should be used instead.

Example:

```
{
      "type":          "uart",
      "device-name":   "Pepper_C1-MUX-1A64D5",
      "msg":           "Hello world!!!"
}
```

7. Communication interface – binary interface

7.1 Overview

The Pepper C1 MUX module can be controlled using a simple binary protocol available over USB (using the built in USB-TTL converter), the UART2 header, or a TCP IP socket. 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 Web interface from 9600 up to 921600. The same settings can be applied when communication is switched to UART2.

When communication is set to TCP, the device's built in internet protocol socket module acts as a TCP server and listens for connection by default on port 1234. Only one active TCP connection is allowed to the module. The module has a built in 15 second timeout for connection, so if the host doesn't send any frame for this period, the connection will be closed on the server side. To avoid this, the user should send any frame to the module (e.g. DUMMY_COMMAND).

7.2 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

7.3 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, 0xcbdc,
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, 0xefbe, 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;
}
```

8. Bluetooth interface

8.1 Bluetooth Serial Port Profile

The Pepper C1 MUX is able to work over Bluetooth using Serial Port Profile. This protocol has been available since firmware version 1.3. However in version firmware version 1.4 onwards we have changed how SPP is enabled. From firmware version 1.4 onwards, this protocol can be enabled only in the web interface on the communication tab.

The default PIN is '0000'. The communication protocol, frame format, and commands are exactly the same as for the other communication interfaces. Wi-Fi interface is no not accessible in this mode.

8.2 Bluetooth GATT service

From firmware version 1.4 onwards, the Pepper C1 also supports the Bluetooth Low Energy standard over GATT services. For this purpose, a special custom service is available with two characteristics, one for write and one for read with notification when new data is available.

- Service: `f03c26b1-3fb1-4d67-912e-4ae31159aef0`
- Write characteristics: `f03c26b2-3fb1-4d67-912e-4ae31159aef0`
- Read characteristics: `f03c26b3-3fb1-4d67-912e-4ae31159aef0`

This communication method can be enabled in the web interface or temporarily in order to configure the device using the dedicated 'Pepper C1 configurator' application available in the Google Play store or in the Apple App Store. To enable this mode without web interface, the user can press the built-in button three time quickly and then the device will switch temporarily to this mode until there is a power cycle. This temporary mode is confirmed by one blue blink on the built-in LED. Because of the slow speed, this is not a recommended method to upgrade the firmware.

To enable Wi-Fi mode please hold built-in button for 3 seconds.

8.2.1 Using Android and iOS based smartphones as a virtual RFID TAG over BLE

From firmware version 1.7 onwards, the Pepper C1 supports virtual RFID TAG emulation based not on RFID technology but using BLE which is available in most smartphones on the market now. Thanks to this, users can use their own smartphones as an alternative to RFID TAGS when the application needs only a UID. Every phone sends a unique 8-bytes long UID to the reader. To make it work, users have to install our free application called "Pepper C1 BLE card" available in Google Play and App Store.

This application also has another useful feature - PIN based application authentication instead of UID. When the user selects this method in the phone application, a frame with PIN is sent to the reader. Both virtual UID and PIN are handled by the device as a UID, so it can be received as an asynchronous frame sent over binary protocol, plain text or MQTT just like a normal RFID tag UID received by the reader. More details about frame format for this UID can be found in dedicated sections in this manual describing GET_UID frame or MQTT.

This feature needs to have BLE service enabled as an additional interface in the web configuration.

8.2.2 Bluetooth Low Energy GATT as an additional interface

If an application needs to combine the BLE feature with other communication methods like UART or TCP communication, then the user can set up the reader to use BLE interface at the same time when other services are running. But because of memory limits, some features may not be available at the same time. To make it possible, the web interface is shut down 1 minute after boot up if it is not used within this time. After the first minute and after the web interface is disabled, the BLE service becomes available. The device will blink blue every 3 seconds to show that the BLE service is waiting for the web interface to become disabled.

8.3 Bluetooth LE HID profile

From firmware version 1.4 onwards, the Pepper C1 also provides Bluetooth Low Energy HID support. Thanks to this profile, the user can pair the Pepper C1 to a PC or smartphone like one would a normal keyboard and, if polling mode is enabled, the reader will send a key sequence corresponding to the UID (unique serial number) read from the TAG. On the configuration page, the user can also enable an extra ENTER key after each UID sent to the host to separate a string of UID reads to make it more legible.

To enable Wi-Fi mode please hold the built-in button for at least 3 seconds.

9. RS-485 Communication

From firmware version 1.4 onwards, two new protocols are available dedicated specially for RS-485 connection: Modbus and extended binary protocol. Both are available only on the Pepper Wireless C1 MUX RS-485 hardware.

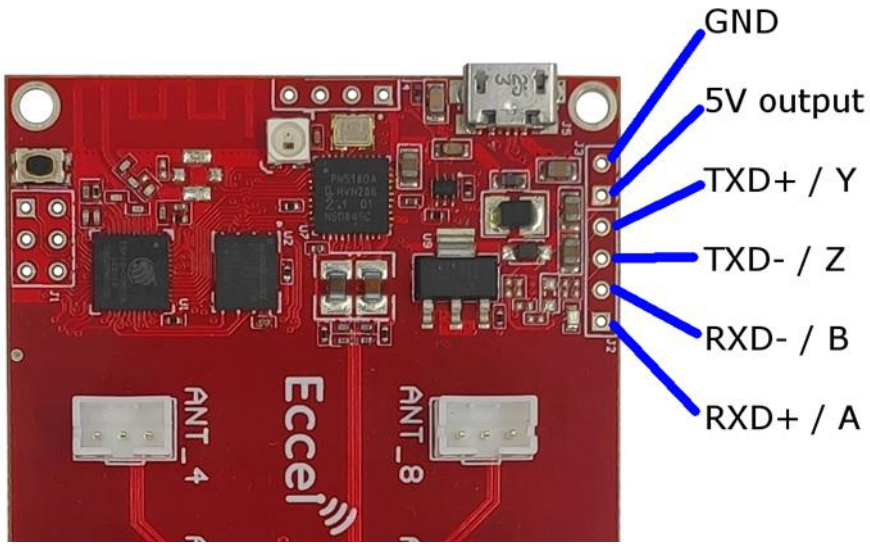


Figure 9 RS-485 pinout

9.1 Modbus RTU

If this communication is selected in the web interface, the device can be connected to a Modbus network as a slave device with an address that is also configured in the web interface. Because Modbus communication API is different to the default Pepper C1 protocol, special registers and commands are used to communicate with the reader. But the command and response format is exactly the same as described in the protocol description of this manual.

Function	Command	Address range
Request to the reader	Write Holding register (0x10)	0-127
Response from the reader	Read Input register (0x04)	0-127 0 – response length 1-n – response bytes
Get polling UID	Read Input Register (0x04)	128 – 139 128 – UID len 129 – 139 - UID
Antenna idx (only for multiplexer version, from firmware version 1.7)	Read Input Register (0x04)	138

The Write Holding Register (0x10) is used to write a command to the device. For example, if the host wants to write the command GET_UID, (one byte 0x02) then they must execute the command Write Holding Register to address 0x00, value 0x02 with length 1. The device sends a confirmation indicating the success or otherwise of the write operation. Then the host system should Read Input Register (command 0x04) at address 0x00 to get the length of the response and then read the response from address 0x01. If the length value is 0, then the response is not yet ready.

The Holding and input registers are 16-bit registers words, but every register stores only one byte from the command and response.

To optimize communication, one special register is created at address 128. It is a 9-bytes long register containing current length + UID of the TAG placed within range of the reader's antenna when internal polling is enabled. So, if the host application wants just to read the RFID tag UID, then this register should be checked to get valid values.

Example below demonstrate scenario described above including all bytes included in the Modbus protocol.

HOST => Write Holding Register, command GET_UID 0x02:

0x01 – Slave address
0x10 – Write Holding Register command
0x00 0x00 – Write address
0x00 0x01 – quantity of registers (every register is 16bits long)
0x02 – bytes count
0x00 0x02 - data to write – GET UID command
0x27 0x91 - Modbus CRC

READER => Write Holding Register confirmation

0x01 - slave address
0x10 - Write Holding Register command
0x00 0x00 - Write address
0x00 0x01 - quantity of registers (every register is 16bits long)
0x01 0xC9 - Modbus CRC

HOST=> Read Input Register (reading response length + response body in one read)

0x01 - slave address
0x04 - Read Input Registers command
0x00 0x00 – Start Address
0x00 0x04 - quantity of registers (every register is 16bits long)
0xF1 0xC9 - Modbus CRC

READER=> Response length + body

0x01 - slave address
0x04 - Read Input Registers command
0x08 - 8 bytes response (4 registers, 16bits each)
0x00 0x03 - reader response length
0x00 0x00 0x00 0x02 0x00 0x01 - three bytes of response stored in 16bits registers
0x00 0x00 - ACK
0x00 0x02 - GET_UID response
0x00 0x01 - 1 tag found
0x77 0x0D - Modbus CRC

9.2 Binary protocol over RS-485

Because in some cases the binary protocol can be more convenient to use since firmware v1.4 device supports binary protocol extended with address byte. Thanks to this the host can use normal binary protocol but keep the addressing option like in the Modbus protocol. The address of the device is the first byte in the command body.

The length of the command is the sum of the Address byte + Command body + 2 bytes CRC. See table below.

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

10. 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 HTTP interface 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.

The key storage can be also managed in the web interface under RFID->Key storage tab.

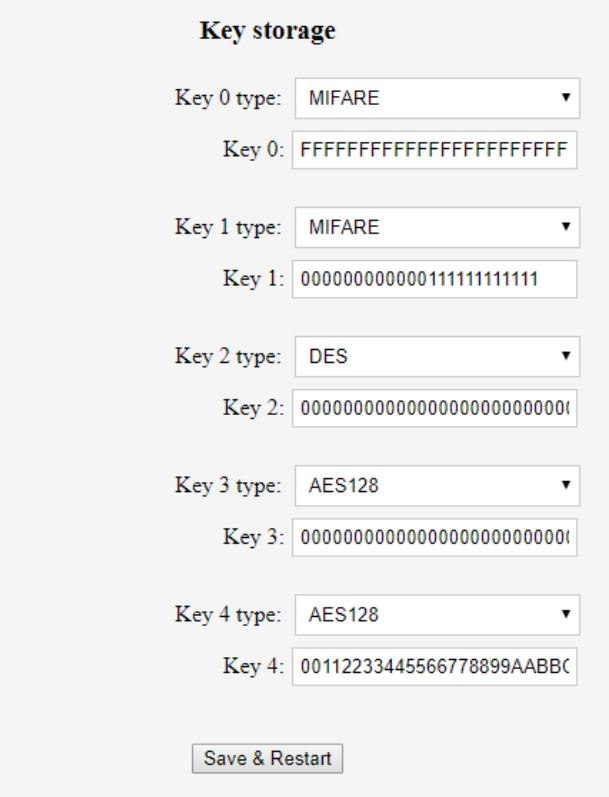


Figure 10-1 Web interface – Key storage TAB

11. Polling mode

In this mode the Pepper C1 MUX device executes the continuous repeated enumerate tags UID command. Depending upon the polling settings in the web interface, 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)

This mode needs to also be activated in order to send frames using the MQTT client and to the WebSocket interface. These modes are enabled in the Web Interface.

11.1 Web configuration for polling mode

All feature related with polling can be configured in Web interface under RFID->Polling tab.

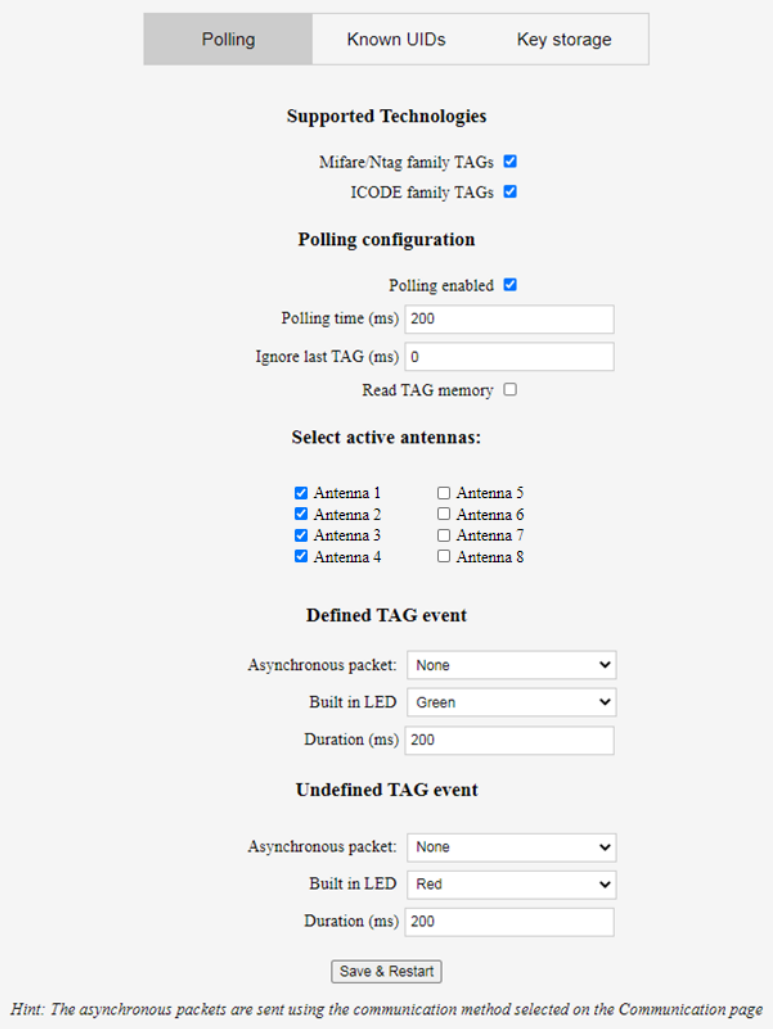


Figure 11-1 Web interface – polling configuration tab

As shown in Figure 11-1 above, you can configure different actions for a defined tag (stored in device memory) and undefined. Both actions have five parameters to configure:

11.1.1 Supported technologies

From version 1.5 onwards, 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.

11.1.2 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. From version 1.5 onwards, the user can specify “Ignore timeout” parameter. Thanks to this timeout when the same TAG is detected in rage 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.

11.1.3 Read memory settings

From firmware version 1.5 onwards, the Pepper C1 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 two ways now:

- When Asynchronous packet is selected to Plain text or JSON format
- Attached to JSON frames sent over MQTT and Web sockets.

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.

11.1.4 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:

- **Asynchronous packet** – the device can send packets over the communication protocol selected in the communication tab. Three packet options are available:
    - Binary packet format – with these settings, the module sends the Get tag UID (0x03) frame but with ASYNC flag instead of ACK. This is the best method if the user already uses binary protocol as the selected communication method. Here is an example:

```
C1=>HOST: 0xFE - ASYNC byte
          0x03 - related command code GET_TAG_UID
          0x01 – MIFARE tag type
          0x20 – tag parameter
          0x74 0x54 0x12 0x65 – tag UID bytes
```

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

```
Card nr 0 - MIFARE Ultralight, SAK: 0, UID: 0408C512A4408
```

- 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

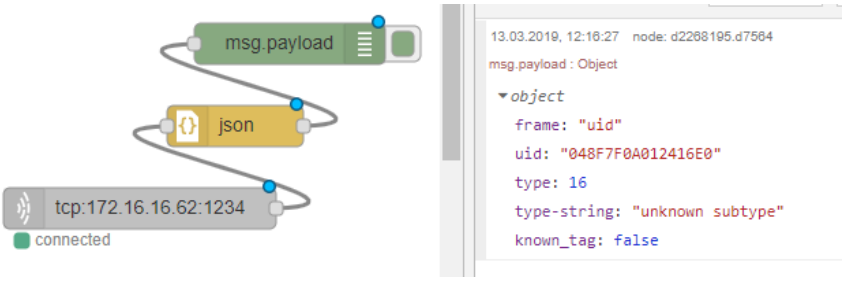


Figure 11-2 JSON frame example

- **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

Warning!

In the above configuration, the user has selected 1 active antenna: Antenna. **Only this active antennas will take part in the polling routine.** After communicating with the last active antenna the device will wait for 200 ms (polling timeout) before starting the next polling round.

11.2 Known UID list

This tab in the web interface is used to manage known UIDs stored in the device memory. Thanks to this, in standalone mode, the Pepper C1 MUX can perform different actions for known and unknown UIDs.

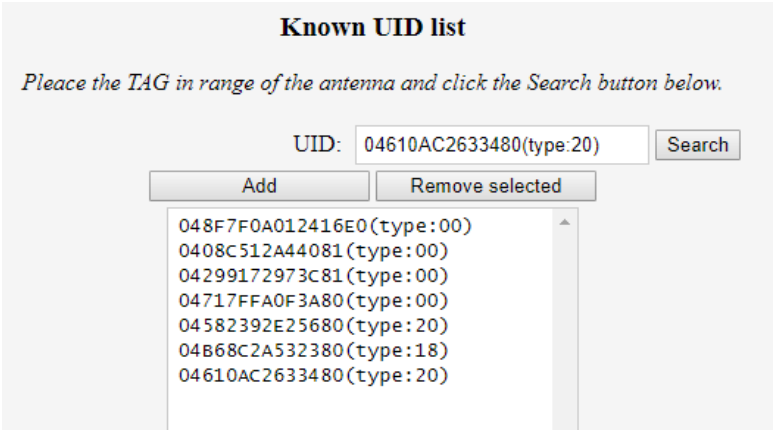


Figure 11-3 Web interface – know UID list

12. 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 1-1024 bytes.

12.1 Generic commands

12.1.1 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=>C1: 0x02 – GET_TAG_COUNT command

C1=>HOST: 0x00 - ACK byte
          0x02 - related command code GET_TAG_COUNT
          0x01 – argument for GET_TAG_COUNT – 0x01 – one tag detected
```

12.1.2 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**
ACK	1	0x00	
Command ID	1	0x01	DUMMY_COMMAND

Example:

```
C1=>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:

MIFARE DESFire errors – layer byte 0x19

Error byte:
0x80 - MF DF Response - No changes done to backup files
0x81 - MF DF Response - Insufficient NV-Memory
0x82 - MF DF Invalid key number specified
0x83 - MF DF Current configuration/status does not allow the requested command
0x84 - MF DF Requested AID not found on PICC
0x85 - MF DF Attempt to read/write data from/to beyond the files/record's limits
0x86 - MF DF Previous cmd not fully completed. Not all frames were requested or provided by the PCD
0x87 - MF DF Num. of applns limited to 28. No additional applications possible
0x88 - MF DF File/Application with same number already exists
0x89 - MF DF Specified file number does not exist
0x8A - MF DF Crypto error returned by PICC
0x8B - MF DF Parameter value error returned by PICC
0x8C - MF DF DESFire Generic error. Check additional Info
0x8D - MF DF ISO 7816 Generic error. Check Additional Info

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)

12.1.3 Dummy command (0x01)

This command takes no arguments. It is used to check that the module 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=>C1: 0x01 –DUMMY_COMMAND

C1=>HOST: 0x00 - ACK byte
          0x01 - related command code DUMMY_COMMAND
```

12.1.4 Get tag count (0x02)

The command send 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=>C1: 0x02 – GET_TAG_COUNT

C1=>HOST: 0x00 - ACK byte
          0x02 - related command code GET_TAG_COUNT
          0x01 – number of tags in range
```

12.1.5 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 me 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

			0x0S - 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=>C1: 0x03 – GET_TAG_UID
          0x00 – TAG idx

C1=>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
```

12.1.6 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 me less than number of tags reported by GET_TAG_COUNT command
Response description			
ACK	1	0x00	
Command ID	1	0x04	ACTIVATE_TAG

Example:

```
HOST=>C1: 0x04 – ACTIVATE_TAG
          0x00 – TAG idx
```

```
C1=>HOST: 0x00 - ACK byte
          0x04 - related command code ACTIVATE_TAG
```

12.1.7 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=>C1: 0x05 – HALT

C1=>HOST: 0x00 - ACK byte
          0x05 - related command code HALT
```

12.1.8 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 Web interface.

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=>C1: 0x06 – SET_POLLING
          0x00 – Stop polling temporary

C1=>HOST: 0x00 - ACK byte
          0x06 - related command code SET_POLLING
```

12.1.9 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=>C1: 0x07 – SET_KEY
          0x00 – Key number
          0x06 – MIFARE key type
          0x00 0x00 0x00 0x00 0x00 0x00
          0xFF 0xFF 0xFF 0xFF 0xFF 0xFF – Key bytes

C1=>HOST: 0x00 - ACK byte
          0x07 - related command code SET_KEY
```

12.1.10 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=>C1: 0x08 – SAVE_KEYS

C1=>HOST: 0x00 - ACK byte
          0x08 - related command code SAVE_KEYS
```

12.1.11 Network config (0x09)

This command should be used to setup or read network parameters. Depending upon the second byte of the command, different parameters of the network configuration can be changed. Below is the full list of possible network parameters. Also, the ACK response contains a byte detailing the parameters that have been set.

To read current settings the host should send the request without parameters, the ACK response contains current settings of this requested field.

12.1.11.1 Setting Wi-Fi mode

This command has one argument to setup Wi-Fi adapter mode to: Access Point, Client or Off. In the case of the Wi-Fi adapter being disabled, the user needs to use this command again with different settings to enable it again or just perform a factory reset.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x00	Wi-Fi mode subcommand
Mode (optional)	1	X	0x00 – Access Point 0x01 – Client 0x02 – Wi-Fi adapter off
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x00	Wi-Fi mode subcommand
Mode (optional)	1	X	Same as for request

Example1 – set mode:

```
HOST=>C1: 0x09 – NET_CFG
          0x00 – Wi-Fi mode subcommand
          0x01 – Client mode

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x00 – Wi-Fi mode subcommand
```

Example2 – get mode:

```
HOST=>C1: 0x09 – SET_NET_CFG
          0x00 – Wi-Fi mode subcommand

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x00 – Wi-Fi mode subcommand
          0x01 – Client mode
```

12.1.11.2 Wi-Fi authorization mode

This command gets one argument to setup Wi-Fi authorization mode. This setting is only applied in Access Point mode. In client mode authorization is automatically detected.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x01	Wi-Fi authorization mode subcommand
Mode	1	X	0x00 – Open 0x01 – WEP 0x02 – WPA PSK 0x03 – WPA2_PSK 0x04 - WPA_WPA2_PSK 0x05 - WPA2_ENTERPRISE
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x01	Wi-Fi authorization mode subcommand
Mode	1	X	Same as for request

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x01 – Wi-Fi authorization mode subcommand
          0x03 – WPA2_PSK

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x01 – Wi-Fi authorization mode subcommand
```

12.1.11.3 Wi-Fi channel

This command gets one argument to setup the Wi-Fi channel. This setting is only applied in Access Point mode. In client mode, the channel is automatically detected.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x02	Wi-Fi channel subcommand
Channel (optional)	1	1-13	Channel number
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x02	Wi-Fi channel subcommand
Channel (optional)	1	1-13	Channel number

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x02 – Wi-Fi channel mode
          0x05 – channel number

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x02 – Wi-Fi channel mode
```

12.1.11.4 Wi-Fi network SSID

This command sets/gets the SSID for the Wi-Fi adapter. Depending upon mode configuration, this setting will be applied to Access Point or Client.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x03	Wi-Fi SSID subcommand
Channel(optional)	1-32	X	SSID - network name
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x03	Wi-Fi SSID subcommand
Channel(optional)	1-32	X	SSID - network name

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x03 – Wi-Fi SSID subcommand
          0x50 0x65 0x65 0x70 0x65 0x72 0x5f 0x43 0x31  - network SSID

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x03 – Wi-Fi SSID subcommand
```

12.1.11.5 Wi-Fi network password

This command sets/gets the password for the Wi-Fi network. Depending upon mode configuration, this setting will be applied to Access Point or Client.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x04	Wi-Fi SSID network password
Password(optional)	1-32	X	Password
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x04	Wi-Fi SSID network password
Password (optional)	1-32	X	Password

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x04 – Wi-Fi password subcommand
          0x61 0x64 0x6d 0x69 0x6e - network password

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x04 – Wi-Fi password subcommand
```

12.1.11.6 Network IP address mode

This command gets one argument to setup network address mode: DHCP client or static IP address. In the case of static IP being selected, the user needs to provide IP addresses for the module IP, netmask, gateway and DNS.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x05	IP address mode subcommand
Network address mode(optional)	1	X	0x00 – DHCP client 0x01 – Static IP
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x05	IP address mode subcommand
Network address mode(optional)	1	X	0x00 – DHCP client 0x01 – Static IP

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x05 – IP address mode subcommand
          0x00 – Static IP address mode

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x05 – IP address mode subcommand
```

12.1.11.7 Network IP addresses

These four subcommands should be used to setup: IP address, netmask, gateway and DNS. If a DHCP client is enabled with the command described above these settings are ignored.

Command description			
Argument	Size	Value	Description
Command ID	1	0x09	NET_CFG
Subcommand ID	1	X	0x06 – IP address

			0x07 – netmask address 0x08 – gateway address 0x09 – DNS address
Address (optional)	4	X	Address bytes
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	0x06-0x09	Address mode subcommand
Address (optional)	4	X	Address bytes

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x06 – IP address subcommand
          0xC0 0xA8 0x00 0x02 – IP address 192.168.0.2

C1=>HOST: 0x00 - ACK byte
          0x09 - related command code NET_CFG
          0x06 – IP address subcommand
```

12.1.11.8 Web interface user name and password (0x09)

This command should be used to setup the username and password needed to access the web interface. Default settings for the username and password are admin/admin.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x09	NET_CFG
Subcommand ID	1	X	0x0A – User name subcommand 0x0B – password subcommand
User/password (optional)	1-32	X	Username/password bytes
Response description			
ACK	1	0x00	
Command ID	1	0x09	NET_CFG
Subcommand ID	1	X	0x0A – User name subcommand 0x0B – password subcommand
User/password (optional)	1-32	X	Username/password bytes

Example:

```
HOST=>C1: 0x09 – NET_CFG
          0x0B – Web password subcommand
          0x61 0x64 0x6d 0x69 0x6e – Web interface password
```

```
C1=>HOST: 0x00 – ACK byte
          0x09 – related command code NET_CFG
          0x0B – Web password subcommand
```

12.1.12 Reboot (0x0A)

This command requests a software reboot for the Pepper C1 MUX module. After this command the device will not accept any protocol commands for 1 second. In case of communication over Wi-Fi this time can be longer and depends upon network configuration.

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

Example:

```
HOST=>C1: 0x0A – REBOOT

C1=>HOST: 0x00 – ACK byte
          0x0A – related command code REBOOT
```

12.1.13 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=>C1: 0x0B – GET_VERSION

C1=>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
```

12.1.14 UART passthru (0x0C)

This command is used to transmit and receive data to the UART2 port using binary protocol. Thanks to this the host application can communicate with an external device attached to the UART2 port. This command works only when TCP server on the client is selected as the main communication interface. This option can be really useful when an application requires communication with an external device, and thanks to the built in Wi-Fi interface, the Pepper C1 MUX can act as a bidirectional Wi-Fi to UART bridge.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x0C	UART_PASSTHRU
Response description			
ACK	1	0x00	
Command ID	1	0x0C	UART_PASSTHRU
Data	X	X	Data transmitted or received over UART2 port

Example:

```
HOST=>C1: 0x0C – UART_PASSTHRU
          0x31 0x2e 0x31 0x20 0x4a 0x61 – data bytes

C1=>HOST: 0x0C – UART_PASSTHRU
          0x34 0x2e 0x35 0x20 0x4b 0x60 – data bytes
```

12.1.15 Set active antenna (0x0F)

This command sets the active antenna number. Available numbers are from 1 to 8.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x0F	SET_ACTIVE_ANTENNA
Antenna number	1	X	Number from 1 to 8
Response description			
ACK	1	0x00	
Command ID	1	0x0F	SET_ACTIVE_ANTENNA
Antenna number	1	X	Currently set antenna number

Example:

```
HOST=>C1: 0x0F – SET_ACTIVE_ANTENNA
          0x02 – select the antenna number 2

C1=>HOST: 0x00 – ACK byte
          0x0F – related command code SET_ACTIVE_ANTENNA
          0x02 – Currently set antenna number
```

12.1.16 Bluetooth pin command (0x10)

This command should be used to setup the PIN for the Bluetooth interface. Default PIN is '0000'. If you call this command without any PIN parameter, then the device sends a response containing the current PIN settings.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x10	BLUETOOTH_PIN
PIN	4	X	Four digits pin number (optional)
Response description			
ACK	1	0x00	
Command ID	1	0x10	BLUETOOTH_PIN
Current PIN	4	X	

Example1 – setup new PIN:

```
HOST=>C1: 0x10 – BLUETOOTH_PIN
          0x31 0x32 0x33 0x34 – New pin value '1234'

C1=>HOST: 0x00 – ACK byte
     0x10 – related command code BLUETOOTH_PIN
```

Example2 – read current PIN:

```
HOST=>C1: 0x10 – BLUETOOTH_PIN

C1=>HOST: 0x00 – ACK byte
          0x10 – related command code BLUETOOTH_PIN
          0x31 0x32 0x33 0x34 – Pin value '1234'
```

12.1.17 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

Example1 – setup new PIN:

```
HOST=>C1: 0x11 – FACTORY_RESET
          0x01 0x02 0x03 0x04 – Extra parameters
```

```
C1=>HOST: 0x00 – ACK byte
     0x11 – related command code FACTORY_RESET
```

12.1.18 Protocol authorization (0x12)

From firmware version 1.7 onwards, the Pepper C1 reader supports protocol authorization for wireless interfaces like BLE service and TCP client and server. This option helps to protect these interfaces from unauthorized access. If this password is set in the configuration, then the user has to use this command every time in order to establish and authorize a new connection with the reader, before executing other commands. Two commands are available for executing without authorization "Dummy command" and "Get version". The Password can be set using this command or by using the web interface.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x12	PROTOCOL_AUTH
Option	1	X	0x00 – login 0x01 – modify password 0x02 – query for password
Password	1-32	X	Password for login or modify option
Response description			
ACK	1	0x00	
Command ID	1	0x12	PROTOCOL_AUTH
Password	1-32	x	Optional password for option 0x02

Example – login procedure:

```
HOST=>C1: 0x12 – PROTOCOL_AUTH
          0x00 – login option
          0x31 0x32 0x33 0x34 0x35 0x36 0x37 – password bytes

C1=>HOST: 0x00 – ACK byte
          0x12 – related command code PROTOCOL_AUTH
```

Example – query for password:

```
HOST=>C1: 0x12 – PROTOCOL_AUTH
          0x02 – query for password


C1=>HOST: 0x00 – ACK byte
          0x12 – related command code PROTOCOL_AUTH
          0x31 0x32 0x33 0x34 0x35 0x36 0x37 – password bytes
```

12.1.19 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.

12.1.19.1 General settings

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	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x00	General subcommand ID
MDNS service	1	X	0x00 – disabled, 0x01 – enabled
UDP discovery service	1	X	0x00 – disabled, 0x01 – enabled
Device name length	1	X	Length of the device name
Device name	X	X	Device name as ASCII bytes
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x00	General subcommand ID

Example – setup general settings procedure:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x00 – general subcommand
          0x01 – MDNS service enabled
          0x01 – UDP service enabled
          0x10 – device name length
          0x50 0x65 0x70 0x70 0x65 0x72 0x5f 0x43
          0x31 0x2d 0x31 0x41 0x36 0x34 0x44 0x34 – device name bytes

C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x00 – general subcommand ID
```

Example – query for password:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x00 – query for general config


C1=>HOST: 0x00 – ACK byte
          0x13 - PROTOCOL_CONFIG
          0x00 – general settings subcommand
          0x00 – MDNS disabled
          0x01 – UDP discovery enabled
          0x10 – device name length
          0x50 0x65 0x70 0x70 0x65 0x72 0x5f 0x43
          0x31 0x2d 0x31 0x41 0x36 0x34 0x44 0x34 – device name bytes
```

12.1.19.2 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 – 9600bps 0x01 – 19200bps 0x02 – 38400bps 0x03 – 57600bps 0x04 – 115200bps 0x05 – 230400bps 0x06 – 460800bps 0x07 – 921600bps
UART2 protocol	1	X	0x00 – Binary protocol 0x01 – Console logs 0x02 – Modbus 0x03 – RS485 binary protocol 0x04 – Passthru mode
UART2 baud	1	X	0x00 – 9600bps 0x01 – 19200bps 0x02 – 38400bps 0x03 – 57600bps 0x04 – 115200bps 0x05 – 230400bps 0x06 – 460800bps 0x07 – 921600bps
Option bytes	X	X	Option bytes described below
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x01	

Option bytes description			
Protocol	**Size**	**Value**	**Description**
Modbus/ RS485 binary	1	X	Device address on RS485
Passthru	1 X 1 X	X X bytes X X bytes	Passthru Wifi connected frame length Passthru Wifi connected frame bytes Passthru Wifi disconnected frame length Passthru Wifi disconnected frame bytes

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x01 – UART subcommand
          0x01 – Console logs on UART0
          0x04 – 115200 baud
          0x04 – Uart passthru mode on UART2
          0x04 - 115200 baud
          0x00 – UART passthru Wifi connected frame length, no data bytes
          0x03 – UART passthru Wifi disconnected frame length
          0x50 0x65 0x70 – data bytes

C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x01 – UART subcommand ID
```

12.1.19.3 TCP server settings

This command should be used to setup TCP server parameters.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x02	TCP server subcommand ID
Service enabled	1	X	0x00 – disabled, 0x01 – enabled
TCP server port	2	X	Port two bytes LSB first
TCP server timeout	2	X	Timeout in seconds, LSB first
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x02	TCP server subcommand ID

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x02 – TCP subcommand ID
          0x01 – service enabled
          0xD2 0x04 – TCP port 1234
          0x00 0x00 - timeout

C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x02 – general subcommand ID
```

12.1.19.4 TCP client settings

This command should be used to setup TCP client parameters.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x03	TCP client subcommand ID
Service enabled	1	X	0x00 – disabled, 0x01 – enabled
TCP port	2	X	Port two bytes LSB first
TCP client timeout	2	X	Timeout in seconds, LSB first
TCP server address	X	X	server address as ASCII bytes
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x03	TCP client subcommand ID

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x03 – TCP server subcommand ID
          0x01 – service enabled
          0xD2 0x04 – TCP port 1234
          0x00 0x00 – timeout
          0x65 0x78 0x61 0x6d 0x70 0x6c 0x65
          0x2e 0x63 0x6f 0x6d – server address bytes example.com

C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x03 – TCP server subcommand ID
```

12.1.19.5 Bluetooth settings

This command should be used to setup Bluetooth interface parameters

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x04	Bluetooth subcommand ID
Selected service	1	X	0x00 – disabled 0x01 – Bluetooth SPP 0x02 – Bluetooth Low Energy service 0x03 – Bluetooth HID service
Optional parameters	X	- SPP service - HID service	- 4 bytes Bluetooth PIN - Send ENTER after UID 0x00 – disabled, 01-enabled
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x03	Bluetooth subcommand ID

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x04 – Bluetooth subcommand ID
          0x01 – SPP service enabled
          0x31 0x32 0x33 0x34 – SPP pin '1234'
C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x04 – Bluetooth subcommand ID
```

12.1.19.6 MQTT client settings

This command should be used to setup MQTT parameters.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x05	MQTT subcommand ID
Service enabled	1	X	0x00 – disabled, 0x01 – enabled
Port	2	X	MQTT server port, LSB first
Server length	1	X	Server name length
Server name	X	X	Server name as ASCII bytes
User name length	1	X	User name length
User name	X	X	User name ASCII bytes
Password length	1	X	Password name length
Password	X	X	Password ASCII bytes
Out topic length	1	X	Out topic name length
Out topic	X	X	Out topic ASCII bytes
In topic length	1	X	In topic name length
In topic	X	X	In topic ASCII bytes
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x05	MQTT subcommand ID

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x05 – MQTT subcommand ID
          0x01 – MQTT service enabled
          0x5B 0x07 – port 1883
          0x0B 0x65 0x78 0x61 0x6D 0x70
          0x6C 0x65 0x2E 0x63 0x6F 0x6D -server length + server bytes
          0x04 0x75 0x73 0x65 0x72  - user name length + username bytes
          0x08 0x70 0x61 0x73 0x73 0x77 0x6F 0x72 0x64
                              – password length byte + password bytes
          0x08 0x72 0x66 0x69 0x64 0x5F 0x6F 0x75 0x74
                              – out topic length byte + out topic bytes
          0x07 0x72 0x66 0x69 0x64 0x5F 0x69 0x6E
                              – in topic length byte + in topic bytes
```

```
C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x05 – MQTT subcommand ID
```

12.1.19.7 REST API settings

This command should be used to set up the REST API parameters. REST API can work over HTTP and HTTPS but the secure version needs a lot of memory and therefore may not work with Bluetooth services enabled at the same time. It is also recommended to set up "Ignore the last TAG" in the RFID polling configuration to a value higher than 1000ms.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x06	REST API subcommand ID
Service enabled	1	X	0x00 – disabled, 0x01 – enabled
Authorization type	1	X	0x00 – disabled 0x01 – Basic 0x02 - Digest
URL length	1	X	URL name length
URL name	X	X	URL name as ASCII bytes
User name length	1	X	User name length
User name	X	X	User name ASCII bytes
Password length	1	X	Password name length
Password	X	X	Password ASCII bytes
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x06	REST API subcommand ID

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x06 – REST API subcommand ID
          0x01 – service enabled
          0x01 – Authorization type set to Basic
          0x0B 0x65 0x78 0x61 0x6D 0x70
          0x6C 0x65 0x2E 0x63 0x6F 0x6D - URL length + URL bytes
          0x04 0x75 0x73 0x65 0x72  - user name length + username bytes
          0x08 0x70 0x61 0x73 0x73 0x77 0x6F 0x72 0x64
                              – password length byte + password bytes

C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x05 – REST API subcommand ID
```

12.1.19.8 Web socket settings

This command should be used to setup Web server settings.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x07	Web socket subcommand ID
Service enabled	1	X	0x00 – disabled, 0x01 – enabled
URL length	1	X	URL name length
URL name	X	X	URL name as ASCII bytes
Response description			
ACK	1	0x00	
Command ID	1	0x13	PROTOCOL_CONFIG
Subcommand ID	1	0x07	Web socket subcommand ID

Example:

```
HOST=>C1: 0x13 – PROTOCOL_CONFIG
          0x07 – Web service subcommand ID
          0x01 – service enabled
          0x0B 0x65 0x78 0x61 0x6D 0x70
          0x6C 0x65 0x2E 0x63 0x6F 0x6D - URL length + URL bytes
C1=>HOST: 0x00 – ACK byte
          0x13 – related command code PROTOCOL_CONFIG
          0x05 – Web service subcommand ID
```

.1.1 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 with LSB order.
Response description			
ACK	1	0x00	
Command ID	1	0x14	LED command

Example:

```
HOST=>C1: 0x14 – LED command
          0xFF 0x00 0x00 – set red colour
          0x64 0x00 – timeout 100ms

C1=>HOST: 0x00 – ACK byte
          0x0E – related command code LED
```

12.2 MIFARE Classics commands

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

12.2.1 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=>C1: 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

C1=>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
```

12.2.2 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=>C1: 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

C1=>HOST: 0x00 – ACK byte
          0x21 – related command code MF_WRITE_BLOCK
```

12.2.3 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=>C1: 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

C1=>HOST: 0x00 – ACK byte
          0x22 – related command code MF_READ_BLOCK
          0x00 0x00 0x00 0x01 – value
          0x01 – address byte
```

12.2.4 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=>C1: 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

C1=>HOST: 0x00 – ACK byte
          0x23 – related command code MF_WRITE_BLOCK
```

12.2.5 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=>C1: 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

C1=>HOST: 0x00 – ACK byte
          0x24 – related command code MF_INCREMENT_BLOCK
```

12.2.6 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=>C1: 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
```

```
C1=>HOST: 0x00 – ACK byte
          0x25 – related command code MF_TRANSFER_BLOCK
```

12.2.7 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=>C1: 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


C1=>HOST: 0x00 – ACK byte
          0x26 – related command code MF_RESTORE_BLOCK
```

12.2.8 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=>C1: 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

C1=>HOST: 0x00 – ACK byte
          0x27 – related command code MF_TRANSFER_RESTORE_BLOCK
```

12.3 MIFARE Ultralight commands

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

12.3.1 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=>C1: 0x40 – MFU_READ_PAGE
          0x02 – page number 2
          0x02 – two pages to read

C1=>HOST: 0x00 – ACK byte
          0x40 – related command code MFU_READ_PAGE
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 – 8 bytes result
```

12.3.2 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=>C1: 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

C1=>HOST: 0x00 – ACK byte
          0x41 – related command code MFU_WRITE_PAGE
```

12.3.3 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=>C1: 0x42 – MFU_GET_VERSION

C1=>HOST: 0x00 – ACK byte
          0x42 – related command code MFU_GET_VERSION
          0x31 0x35 0x3a 0x33 0x35 0x3a 0x30 0x33 – version bytes
```

12.3.4 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=>C1: 0x43 – MFU_READ_SIGNATURE

C1=>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
```

12.3.5 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=>C1: 0x44 – MFU_WRITE_SIGNATURE
          0x00 – relative page number 0
          0x35 0x3a 0x30 0x33 – 4 bytes to write

C1=>HOST: 0x00 – ACK byte
          0x44 – related command code MFU_WRITE_SIGNATURE
```

12.3.6 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=>C1: 0x45 – MFU_LOCK_SIGNATURE
          0x02 – permanent lock
```

```
C1=>HOST: 0x00 – ACK byte
          0x45 – related command code MFU_LOCK_SIGNATURE
```

12.3.7 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=>C1: 0x46 – MFU_READ_COUNTER
          0x01 – counter number

C1=>HOST: 0x00 – ACK byte
          0x46 – related command code MFU_READ_COUNTER
          0x00 0x00 0x01 – value
```

12.3.8 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=>C1: 0x47 – MFU_INCREMENT_COUNTER
          0x02 – block number 2
          0x00 0x00 0x01 – increment value

C1=>HOST: 0x00 – ACK byte
          0x47 – related command code MFU_INCREMENT_COUNTER
```

12.3.9 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=>C1: 0x48 – MFU_PASSWORD_AUTH
          0x00 0x00 0x00 0x00 – password

C1=>HOST: 0x00 – ACK byte
          0x48 – related command code MFU_PASSWORD_AUTH
          0x00 0x00 – password acknowledge bytes
```

12.3.10 Ultralight-C authenticate (0x49)

This command tries to authenticate the MIFARE Ultralight-C tag using the password stored in the key storage. It takes as an argument one byte with the key number in the key storage.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x49	MFUC_AUTHENTICATE
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x49	MFUC_AUTHENTICATE

Example:

```
HOST=>C1: 0x49 – MFUC_AUTHENTICATE
          0x00 – key number

C1=>HOST: 0x00 – ACK byte
          0x49 – related command code MFUC_AUTHENTICATE
```

12.3.11 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
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x49	MFU_CHECKEVENT

Example:

```
HOST=>C1: 0x49 – MFU_CHECKEVENT
          0x00 – counter number

C1=>HOST: 0x00 – ACK byte
          0x49 – related command code MFU_CHECKEVENT
          0x01 – tearing event occurred
```

12.4 MIFARE DESFire commands

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

12.4.1 Get version (0x60)

This command requests version information from the tag. The returned ACK answer contains 28-bytes with version information.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x60	MFDF_GET_VERSION
Response description			
ACK	1	0x00	
Command ID	1	0x60	MFDF_GET_VERSION
Read data	28	XXX	Version bytes read from the tag

Example:

```
HOST=>C1: 0x60 – MFDF_GET_VERSION

C1=>HOST: 0x00 – ACK byte
          0x60 – related command code MFDF_GET_VERSION

          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 – 28 bytes result
```

12.4.2 Select application (0x61)

This command requests select application operation on the tag. Takes as argument 3-byes containing AID.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x61	MFDF_GET_VERSION
AID	3	X	Application ID
Response description			
ACK	1	0x00	
Command ID	1	0x61	MFDF_GET_VERSION

Example:

```
HOST=>C1: 0x61 – MFDF_SELECT_APP
          0x01 0x02 0x03 – 3 bytes AID

C1=>HOST: 0x00 – ACK byte
          0x61 – related command code MFDF_SELECT_APP
```

12.4.3 List application IDs (0x62)

This command requests lists application IDs from the TAG. The returned ACK answer contains the bytes with application IDs. Every ID is 3-bytes long.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x62	MFDF_LIST_APP_IDS
Response description			
ACK	1	0x00	
Command ID	1	0x62	MFDF_LIST_APP_IDS
Application IDs	X*3	X	Bytes with applications IDs

Example:

```
HOST=>C1: 0x62 – MFDF_LIST_APP_IDS

C1=>HOST: 0x00 – ACK byte
          0x62 – related command code MFDF_LIST_APP_IDS
          0x00 0x00 0x01 – first AID
          0xAA 0xBB 0xCC – second AID
          0x55 0x55 0x55 – third AID
          ...
```

12.4.4 List files IDs (0x63)

This command returns the file IDs of all active files within the currently selected application. The returned ACK answer contains the bytes with file IDs. Every file ID is 3-bytes long.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x63	MFDF_LIST_FILE_IDS
Response description			
ACK	1	0x00	
Command ID	1	0x63	MFDF_LIST_FILE_IDS
Application IDs	X*3	X	Bytes with files IDs

Example:

```
HOST=>C1: 0x63 – MFDF_LIST_FILE_IDS

C1=>HOST: 0x00 – ACK byte
          0x63 – related command code MFDF_LIST_FILE_IDS
          0x00 0x00 0x01 – first file ID
          0xAA 0xBB 0xCC – second file ID
          0x55 0x55 0x55 – third file ID
          ...
```

12.4.5 Authenticate (0x64)

This command tries to authenticate the MIFARE DESFire using the password stored in the key storage. It takes as an argument one byte with the key number in the key storage, and one byte with the key number on the card. This command can be used with DES and 2K3DES keys.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x64	MFDF_AUTHENTICATE
Key number in storage	1	0-4	Key number in key storage
Key number on card	1	x	Key number on card
Response description			
ACK	1	0x00	
Command ID	1	0x64	MFDF_AUTHENTICATE

Example:

```
HOST=>C1: 0x64 – MFDF_AUTHENTICATE
          0x00 – key number

C1=>HOST: 0x00 – ACK byte
          0x64 – related command code MFDF_AUTHENTICATE
```

12.4.6 Authenticate ISO (0x65)

This command tries to authenticate the MIFARE DESFire tag in ISO CBS send mode using the key stored in the key storage. It takes as an argument one byte with the key number in the key storage, and one byte with the key number on the card. This command can be used with DES, 3DES and 3K3DES keys.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x65	MFDF_AUTHENTICATE_ISO
Key number	1	0-4	Key number in key storage
Key number on card	1	X	Key number on card
Response description			
ACK	1	0x00	
Command ID	1	0x65	MFDF_AUTHENTICATE_ISO

Example:

```
HOST=>C1: 0x65 – MFDF_AUTHENTICATE_ISO
          0x00 – key number

C1=>HOST: 0x00 – ACK byte
          0x65 – related command code MFDF_AUTHENTICATE_ISO
```

12.4.7 Authenticate AES (0x66)

This command tries to authenticate the MIFARE DESFire using the key stored in the key storage, and one byte with the key number on the card. It takes as an argument one byte with the key number in the key storage. This command can be used with AES128 keys.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x66	MFDF_AUTHENTICATE_ISO
Key number	1	0-4	Key number in key storage
Response description			
ACK	1	0x00	
Command ID	1	0x66	MFDF_AUTHENTICATE_ISO

Example:

```
HOST=>C1: 0x66 – MFDF_AUTHENTICATE_AES
          0x00 – key number

C1=>HOST: 0x00 – ACK byte
          0x66 – related command code MFDF_AUTHENTICATE_AES
```

12.4.8 Create application (0x67)

This command tries to create application on the tag. It takes three arguments: 3-bytes of application ID, the keySettings1 byte and the keySettings2 byte. Please refer to the NXP documentation for more information about key settings bytes.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x67	MFDF_CREATE_APP
Application ID	3	X	Application ID bytes
Key settings 1	1	X	Please refer to the NXP documentation for more information
Key settings 2	1	X	Please refer to the NXP documentation for more information
Response description			
ACK	1	0x00	
Command ID	1	0x67	MFDF_CREATE_APP

Example:

```
HOST=>C1: 0x67 – MFDF_CREATE_APP
          0x00 – key number
          0x01 0x02 0x03 – application ID
          0xED 0x84 – key settings bytes
C1=>HOST: 0x00 – ACK byte
          0x67 – related command code MFDF_CREATE_APP
```

12.4.9 Delete application (0x68)

This command tries to delete an application from the tag. It takes one argument with the application ID.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x68	MFDF_DELETE_APP
Application ID	3	X	Application ID bytes
Response description			
ACK	1	0x00	
Command ID	1	0x68	MFDF_DELETE_APP

Example:

```
HOST=>C1: 0x68 – MFDF_DELETE_APP
          0x01 0x02 0x03 – application ID

C1=>HOST: 0x00 – ACK byte
          0x68 – related command code MFDF_DELETE_APP
```

12.4.10 Change key (0x69)

This command tries to change the key for the selected application. It takes three arguments: the old key number from key storage, the new key number in the key storage and the key number on the card. The key type of the application keys cannot be changed.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x69	MFDF_CHANGE_KEY
Old key number	1	0-4	Key number in key storage
New key number	1	0-4	Key number in key storage
Key number on card	1	X	Key number on the card
Response description			
ACK	1	0x00	
Command ID	1	0x69	MFDF_CHANGE_KEY

Example:

```
HOST=>C1: 0x69 – MFDF_CHANGE_APP
          0x00 – old key number
          0x01 – new key number
          0x00 – key number

C1=>HOST: 0x00 – ACK byte
          0x69 – related command code MFDF_CHANGE_APP
```

12.4.11 Get key settings (0x6A)

This command gets the key settings bytes from the tag. This command does not require any arguments but an application must be selected and authorized.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x6A	MFDF_GET_KEY_SETTINGS
Response description			
ACK	1	0x00	
Command ID	1	0x6A	MFDF_GET_KEY_SETTINGS
Key settings	2	X	Key settings bytes

Example:

```
HOST=>C1: 0x6A – MFDF_GET_KEY_SETTINGS

C1=>HOST: 0x00 – ACK byte
          0x6A – related command code MFDF_GET_KEY_SETTINGS
          0x01 0x02 – key settings bytes
```

12.4.12 Change key settings (0x6B)

This command changes the key settings bytes for the selected and authorized application. It takes one argument, 2-byes long with key settings.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x6B	MFDF_CHANGE_KEY_SETTINGS
New key settings	2	X	Key settings bytes
Response description			
ACK	1	0x00	
Command ID	1	0x6B	MFDF_CHANGE_KEY_SETTINGS

Example:

```
HOST=>C1: 0x6B – MFDF_GET_KEY_SETTINGS
          0x01 0x02 – key settings bytes

C1=>HOST: 0x00 – ACK byte
          0x6B – related command code MFDF_GET_KEY_SETTINGS
```

12.4.13 Create standard or backup data file (0x6C)

This command creates a file for the storage of plain unformatted user data within the selected application. It takes four arguments listed in the table below.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x6C	MFDF_CREATE_DATA_FILE
File number	1	X	File number inside application
Access rights	2	X	Please refer to the NXP documentation for more information
File size	3	X	file size, LSB first
Backup file	1	X	0x00 – Standard file 0x01 – Backup file

Response description			
ACK	1	0x00	
Command ID	1	0x6B	MFDF_CREATE_DATA_FILE

Example:

```
HOST=>C1: 0x6C – MFDF_CREATE_DATA_FILE
          0x01 – file number
          0xEE 0xEE – access rights
          0x40 0x00 0x00 – file 64-bytes long
          0x01 – backup file

C1=>HOST: 0x00 – ACK byte
          0x6C – related command code MFDF_CREATE_DATA_FILE
```

12.4.14 Write data (0x6D)

This command writes data to standard data files or backup data files. It takes three arguments: the file number, the offset in the file where data should be stored, and the data bytes to be written. To store data on the TAG, a commit transaction command is required.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x6D	MFDF_WRITE_DATA
File number	1	X	File number inside application
File offset	3	X	file offset, 3-bytes LSB value
Data	N	X	Data bytes to write
Response description			
ACK	1	0x00	
Command ID	1	0x6D	MFDF_WRITE_DATA

Example:

```
HOST=>C1: 0x6D – MFDF_WRITE_DATA
          0x01 – file number
          0x00 0x00 0x00 – zero offset
          0x01 0x02 0x03 0x04 0x05 0x06 0x07 - data
C1=>HOST: 0x00 – ACK byte
          0x6D – related command code MFDF_WRITE_DATA
```

12.4.15 Read data (0x6E)

This command reads data from standard data files or backup data files. It takes three arguments: the file number, the offset in the file where data is stored, and the number of bytes to be read. The returned ACK response contains the data that has been read.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x6E	MFDF_READ_DATA
File number	1	X	File number inside application

File offset	3	X	file offset, 3-bytes LSB value
Data length	3	X	Read data length, 3-bytes LSB value
Response description			
ACK	1	0x00	
Command ID	1	0x6E	MFDF_READ_DATA

Example:

```
HOST=>C1: 0x6E – MFDF_READ_DATA
          0x01 – file number
          0x00 0x00 0x00 – zero offset
          0x07 0x00 0x00 – seven bytes to read
C1=>HOST: 0x00 – ACK byte
          0x6E – related command code MFDF_READ_DATA
          0x01 0x02 0x03 0x04 0x05 0x06 0x07 - data
```

12.4.16 Create value file (0x6F)

This command creates files for the storage and manipulation of 32bit signed integer values within an existing application on the TAG. It takes seven arguments listed in the table below.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x6F	MFDF_CREATE_VALUE_FILE
File number	1	X	File number inside application
Access rights	2	X	Please refer to the NXP documentation for more information
Low limit	4	X	Low limit as 4-bytes signed value, LSB first
Up limit	4	X	Up limit as 4-bytes signed value, LSB first
Initial value	4	X	Initial value as 4-bytes signed value, LSB first
Get free enabled	1	X	Please refer to the NXP documentation for more information
Limit credited	1	X	Please refer to the NXP documentation for more information
Response description			
ACK	1	0x00	
Command ID	1	0x6F	MFDF_CREATE_VALUE_FILE

Example:

```
HOST=>C1: 0x6F – MFDF_CREATE_VALUE_FILE
          0x02 – file number
          0xEE 0xEE – access rights
          0x00 0x00 0x00 0x00 – low limit
          0x80 0x00 0x00 0x00 – up limit
          0x00 0x00 0x00 0x00 – initial value
          0x01 – get free enabled
          0x01 – limited credit

C1=>HOST: 0x00 – ACK byte
          0x6F – related command code MFDF_CREATE_VALUE_FILE
```

12.4.17 Get value (0x70)

This command returns the value stored in a value file on the TAG. The returned ACK response contains 4 bytes of signed value, LSB-first.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x70	MFDF_GET_VALUE
File number	1	X	File number inside application
Response description			
ACK	1	0x00	
Command ID	1	0x70	MFDF_GET_VALUE
Value	4	X	4 bytes signed value, LSB first

Example:

```
HOST=>C1: 0x70 – MFDF_GET_VALUE
          0x02 – file number

C1=>HOST: 0x00 – ACK byte
          0x70 – related command code MFDF_GET_VALUE
          0x05 0x00 0x00 0x00 – 4 bytes signed value, LSB first
```

12.4.18 Credit file (0x71)

This command increases a value stored in a value file on the TAG.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x71	MFDF_CREDIT
File number	1	X	File number inside application
Credit value	4	X	4 bytes signed value, LSB first
Response description			
ACK	1	0x00	
Command ID	1	0x71	MFDF_CREDIT

Example:

```
HOST=>C1: 0x71 – MFDF_CREDIT
          0x02 – file number
          0x05 0x00 0x00 0x00 – 4 bytes signed value, LSB first

C1=>HOST: 0x00 – ACK byte
          0x71 – related command code MFDF_CREDIT
```

12.4.19 Limited credit (0x72)

This command allows a limited increase of a value stored in a value file without having full credit permissions to the file. Please refer to the NXP documentation for more information.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x72	MFDF_LIMITED_CREDIT
File number	1	X	File number inside application
Credit value	4	X	4 bytes signed value, LSB first
Response description			
ACK	1	0x00	
Command ID	1	0x72	MFDF_ LIMITED_CREDIT

Example:

```
HOST=>C1: 0x72 – MFDF_ LIMITED_CREDIT
          0x02 – file number
          0x05 0x00 0x00 0x00 – 4 bytes signed value, LSB first

C1=>HOST: 0x00 – ACK byte
          0x72 – related command code MFDF_ LIMITED_CREDIT
```

12.4.20 Debit file (0x73)

This command decreases a value stored in a value file on the TAG.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x73	MFDF_DEBIT
File number	1	X	File number inside application
Credit value	4	X	4 bytes signed value, LSB first
Response description			
ACK	1	0x00	
Command ID	1	0x73	MFDF_DEBIT

Example:

```
HOST=>C1: 0x73 – MFDF_DEBIT
          0x02 – file number
          0x05 0x00 0x00 0x00 – 4 bytes signed value, LSB first

C1=>HOST: 0x00 – ACK byte
          0x73 – related command code MFDF_DEBIT
```

12.4.21 Create record file (0x74)

This command creates files for multiple storage of structurally similar data within an existing application. If the cyclic flag is 0x00, then further writing is not possible unless it is cleared. If the cyclic flag is set to 0x01, then the new record overwrites the oldest record.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x74	MFDF_CREATE_RECORD_FILE
File number	1	X	File number inside application
Access rights	2	X	Please refer to the NXP documentation for more information
Record size	2	X	Record size, 16-bits LSB value
Number of records	2	X	Number of records, 16-bits LSB value
Cyclic flag	1	X	If cyclic file is full: 0x00 - further writing is not possible unless it is cleared 0x01 - the new record overwrites oldest record
Response description			
ACK	1	0x00	
Command ID	1	0x74	MFDF_CREATE_RECORD_FILE

Example:

```
HOST=>C1: 0x74 – MFDF_CREATE_RECORD_FILE
          0x03 – file number
          0xEE 0xEE – access rights
          0x08 0x00 – 8-bytes for every record
          0x40 0x00 – 64 records
          0x01 – cyclic flag

C1=>HOST: 0x00 – ACK byte
          0x74 – related command code MFDF_CREATE_VALUE_FILE
```

12.4.22 Write record (0x75)

This command writes data to a record file. It takes two arguments: the file number and the data bytes to be written. To store data on the TAG, a commit transaction command is required.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x75	MFDF_WRITE_RECORD_DATA
File number	1	X	File number inside application
Data	N	X	Data bytes to write
Response description			
ACK	1	0x00	
Command ID	1	0x75	MFDF_WRITE_DATA

Example:

```
HOST=>C1: 0x75 – MFDF_WRITE_DATA
          0x01 – file number
          0x01 0x02 0x03 0x04 0x05 0x06 0x07 - data
C1=>HOST: 0x00 – ACK byte
          0x75 – related command code MFDF_WRITE_RECORD_DATA
```

12.4.23 Read record (0x76)

This command reads data from a record file. It takes three arguments: the file number, the record number, and the number of bytes to be read. The returned ACK response contains the data that has been read.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x76	MFDF_READ_RECORD
File number	1	X	File number inside application
Record number	2	X	Record number, 2-bytes LSB value
Data length	2	X	Read data length, 2-bytes LSB value
Response description			
ACK	1	0x00	
Command ID	1	0x76	MFDF_READ_RECORD

Example:

```
HOST=>C1: 0x76 – MFDF_READ_RECORD
          0x01 – file number
          0x00 0x01 – record number
          0x08 0x00 – eighth bytes to read
C1=>HOST: 0x00 – ACK byte
          0x76 – related command code MFDF_READ_RECORD
          0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 - data
```

12.4.24 Clear records (0x77)

This command resets cyclic or lineal record files. It takes as an argument the file number.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x77	MFDF_CLEAR_RECORDS
File number	1	X	File number inside application
Response description			
ACK	1	0x00	
Command ID	1	0x77	MFDF_CLEAR_RECORDS

Example:

```
HOST=>C1: 0x77 – MFDF_CLEAR_RECORDS
          0x01 – file number

C1=>HOST: 0x00 – ACK byte
          0x77 – related command code MFDF_CLEAR_RECORDS
```

12.4.25 Delete file (0x78)

This command permanently deactivates a file within the file directory of the currently selected application. It takes as an argument the file number.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x78	MFDF_DELETE_FILE
File number	1	X	File number inside application
Response description			
ACK	1	0x00	
Command ID	1	0x78	MFDF_DELETE_FILE

Example:

```
HOST=>C1: 0x78 – MFDF_DELETE_FILE
          0x01 – file number

C1=>HOST: 0x00 – ACK byte
          0x78 – related command code MFDF_DELETE_FILE
```

12.4.26 Get free memory (0x79)

This command returns a value corresponding to the amount of free memory available on the TAG. No arguments are required. The available memory is returned as a 4 byte unsigned LSB value.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x79	MFDF_GET_FREE_MEM
Response description			
ACK	1	0x00	
Command ID	1	0x79	MFDF_GET_FREE_MEM
Free memory	4	X	Free memory, 4-bytes, LSB first

Example:

```
HOST=>C1: 0x79 – MFDF_GET_FREE_MEM

C1=>HOST: 0x00 – ACK byte
          0x79 – related command code MFDF_GET_FREE_MEM
          0x00 0x08 0x00 0x00 – free memory
```

12.4.27 Format memory (0x7A)

This command releases user memory in the TAG. No arguments are required.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x7A	MFDF_FORMAT
Response description			
ACK	1	0x00	
Command ID	1	0x7A	MFDF_FORMAT

Example:

```
HOST=>C1: 0x7A – MFDF_FORMAT

C1=>HOST: 0x00 – ACK byte
          0x7A – related command code MFDF_FORMAT
```

12.4.28 Commit transaction (0x7B)

This command validates all previous write access on backup data files, value files and record files within one application. No arguments are required.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x7B	MFDF_COMMIT_TRANSACTION
Response description			
ACK	1	0x00	
Command ID	1	0x7B	MFDF_COMMIT_TRANSACTION

Example:

```
HOST=>C1: 0x7B – MFDF_COMMIT_TRANSACTION

C1=>HOST: 0x00 – ACK byte
          0x7B – related command code MFDF_COMMIT_TRANSACTION
```

12.4.29 Abort transaction (0x7C)

This command invalidates all previous write access on backup data files, value files and record files within one application. No arguments are required.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x7C	MFDF_ABORT_TRANSACTION
Response description			
ACK	1	0x00	
Command ID	1	0x7C	MFDF_ABORT_TRANSACTION

Example:

```
HOST=>C1: 0x7C – MFDF_ABORT_TRANSACTION

C1=>HOST: 0x00 – ACK byte
          0x7C – related command code MFDF_ABORT_TRANSACTION
```

12.4.30 Get file settings file (0x7D)

This command gets settings for the selected file. The format of the settings bytes depends on the file type.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x7D	MFDF_GET_FILE_SETTINGS
File number	1	X	File number inside application
Response description			
ACK	1	0x00	
Command ID	1	0x7D	MFDF_GET_FILE_SETTINGS
File type	1	X	0x00 – data file 0x01 – backup file 0x02 – credit file 0x03 – record file 0x04 – cyclic file
Access rights	2	X	Please refer to the NXP documentation for more information
Settings bytes data file	3		3 bytes - file size, LSB first
value file	10		4 bytes – lower limit, LSB first 4 bytes – upper limit, LSB first 1 byte – get free enabled 1 byte – limited credit enabled
record or cyclic files	9		3 bytes – record size 3 bytes – max number of records 3 bytes – current number of records

Example:

```
HOST=>C1: 0x7D – MFDF_GET_FILE_SETTINGS
          0x01 – file number

C1=>HOST: 0x00 – ACK byte
          0x7D – related command code MFDF_GET_FILE_SETTINGS
          0x00 – data file type
          0xEE 0xEE – access rights
          0x20 0x00 0x00 – file size 32 bytes, LSB first
```

12.4.31 Set file settings (0x7E)

This command sets new access rights for the selected file.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x7E	MFDF_SET_FILE_SETTINGS
File number	1	X	File number inside application
New access rights	2	X	Please refer to the NXP documentation for more information

Response description			
ACK	1	0x00	
Command ID	1	0x7E	MFDF_WRITE_DATA

Example:

```
HOST=>C1: 0x7E – MFDF_SET_FILE_SETTINGS
          0x01 – file number
          0xEE 0xEE- new access rights bytes
C1=>HOST: 0x00 – ACK byte
          0x7E – related command code MFDF_SET_FILE_SETTINGS
```

12.5 ICODE (ISO15693) commands

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

12.5.1 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
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=>C1: 0x90 – ICODE_INVENTORY_START
          0x00 – Application Family Identifier

C1=>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
```

12.5.2 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=>C1: 0x91 – ICODE_INVENTORY_NEXT
          0x00 – Application Family Identifier

C1=>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
```

12.5.3 Stay quiet (0x92)

This command performs an ISO15693 Stay Quiet command to the selected TAG. When the tag receives the Stay quiet command, it enters the quiet state and will not send back a response. The TAG exits the quiet state upon the execution of a reset (power off) or the command ICODE_INVENTORY_START. Please refer to the NXP documentation for more information.

Command description			
Argument	**Size**	**Value**	**Description**
Command ID	1	0x92	ICODE_STAY_QUIET
Response description			
ACK	1	0x00	
Command ID	1	0x92	ICODE_STAY_QUIET

Example:

```
HOST=>C1: 0x92 – ICODE_STAY_QUIET

C1=>HOST: 0x00 – ACK byte
          0x92 – related command code ICODE_STAY_QUIET
```

12.5.4 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=>C1: 0x93 – ICODE_READ_BLOCK
          0x02 – block number 2
          0x01 – 1 block to read

C1=>HOST: 0x00 – ACK byte
          0x93 – related command code ICODE_READ_BLOCK
          0x35 0x3a 0x30 0x33 – 4 bytes block data
```

12.5.5 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=>C1: 0x94 – ICODE_WRITE_BLOCK
          0x02 – block number 2
          0x01 – block count 1
          0x35 0x3a 0x30 0x33 – 4 bytes to write

C1=>HOST: 0x00 – ACK byte
          0x94 – related command code ICODE_WRITE_BLOCK
```

12.5.6 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=>C1: 0x95 – ICODE_LOCK_BLOCK
          0x02 – block number 2

C1=>HOST: 0x00 – ACK byte
          0x95 – related command code ICODE_LOCK_BLOCK
```

12.5.7 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=>C1: 0x96 – ICODE_WRITE_AFI
          0xAA – new Application Family Identifier value

C1=>HOST: 0x00 – ACK byte
          0x96 – related command code ICODE_WRITE_AFI
```

12.5.8 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=>C1: 0x96 – ICODE_LOCK_AFI

C1=>HOST: 0x00 – ACK byte
          0x96 – related command code ICODE_LOCK_AFI
```

12.5.9 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_DSFID

Example:

```
HOST=>C1: 0x98 – ICODE_WRITE_DSFID
          0xAA – new Data Storage Format Identifier value

C1=>HOST: 0x00 – ACK byte
          0x98 – related command code ICODE_WRITE_DSFID
```

12.5.10 Lock DSFID (0x99)

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

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

Example:

```
HOST=>C1: 0x99 – ICODE_LOCK_DSFID

C1=>HOST: 0x00 – ACK byte
          0x99 – related command code ICODE_LOCK_DSFID
```

12.5.11 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=>C1: 0x9A – ICODE_GET_SYSTEM_INFORMATION

C1=>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
```

12.5.12 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=>C1: 0x9B – ICODE_GET_MULTIPLE_BSS
          0x00 – starting block number
          0x08 – number of BSS to read

C1=>HOST: 0x00 – ACK byte
          0x9B – related command code ICODE_GET_MULTIPLE_BSS
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 – result bytes
```

12.5.13 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=>C1: 0x9C – ICODE_PASSWORD_PROTECT_AFI

C1=>HOST: 0x00 – ACK byte
          0x9C – related command code ICODE_PASSWORD_PROTECT_AFI
```

12.5.14 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=>C1: 0x9D – ICODE_READ_EPC

C1=>HOST: 0x00 – ACK byte
          0x9D – related command code ICODE_READ_EPC
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 – result bytes
```

12.5.15 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=>C1: 0x9E – ICODE_GET_NXP_SYSTEM_INFORMATION

C1=>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
```

12.5.16 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=>C1: 0x9F – ICODE_GET_RANDOM_NUMBER

C1=>HOST: 0x00 – ACK byte
          0x9F – related command code ICODE_GET_RANDOM_NUMBER
          0x7F 0x14 – result bytes
```

12.5.17 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
XOR Password	4	X	
Response description			
ACK	1	0x00	
Command ID	1	0xA0	ICODE_SET_PASSWORD

Example:

```
HOST=>C1: 0xA0 – ICODE_SET_PASSWORD
          0x02 – write password
          0x34 0x76 0x39 0x64 – calculated XOR password

C1=>HOST: 0x00 – ACK byte
          0xA0 – related command code ICODE_SET_PASSWORD
```

12.5.18 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
Password	4	x	Plain password
Response description			
ACK	1	0x00	
Command ID	1	0xA1	ICODE_WRITE_PASSWORD

Example:

```
HOST=>C1: 0xA1 – ICODE_WRITE_PASSWORD
          0x02 – write password
          0x34 0x76 0x39 0x64 – Plain password

C1=>HOST: 0x00 – ACK byte
          0xA1 – related command code ICODE_WRITE_PASSWORD
```

12.5.19 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
Response description			
ACK	1	0x00	
Command ID	1	0xA2	ICODE_LOCK_PASSWORD

Example:

```
HOST=>C1: 0xA2 – ICODE_LOCK_PASSWORD
          0x02 – write password

C1=>HOST: 0x00 – ACK byte
          0xA2 – related command code ICODE_LOCK_PASSWORD
```

12.5.20 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	• 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	• 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=>C1: 0xA3 – ICODE_PAGE_PROTECT
          0x02 – second block selected
          0x01 - ICODE_PROTECT_PAGE_READ_LOW flag selected

C1=>HOST: 0x00 – ACK byte
          0xA3 – related command code ICODE_PAGE_PROTECT
```

12.5.21 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=>C1: 0xA4 – ICODE_LOCK_PAGE_PROTECTION
          0x02 – page number
C1=>HOST: 0x00 – ACK byte
          0xA4 – related command code ICODE_LOCK_PAGE_PROTECTION
```

12.5.22 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=>C1: 0xA5 – ICODE_GET_MULTIPLE_BPS
          0x00 – starting block number
          0x08 – number of BSS to read

C1=>HOST: 0x00 – ACK byte
          0xA5 – related command code ICODE_GET_MULTIPLE_BPS
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 – result bytes
```

12.5.23 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=>C1: 0xA6 – ICODE_DESTROY

C1=>HOST: 0x00 – ACK byte
          0xA6 – related command code ICODE_DESTROY
```

12.5.24 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=>C1: 0xA7 – ICODE_ENABLE_PRIVACY

C1=>HOST: 0x00 – ACK byte
          0xA7 – related command code ICODE_ENABLE_PRIVACY
```

12.5.25 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=>C1: 0xA8 – ICODE_ENABLE_64BIT_PASSWORD

C1=>HOST: 0x00 – ACK byte
          0xA8 – related command code ICODE_ENABLE_64BIT_PASSWORD
```

12.5.26 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=>C1: 0xA9 – ICODE_READ_SIGNATURE

C1=>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
```

12.5.27 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=>C1: 0xAA – ICODE_READ_CONFIG
          0x00 – starting block number
          0x02 – number of blocks to read

C1=>HOST: 0x00 – ACK byte
          0xAA – related command code ICODE_READ_CONFIG
          0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 – result bytes
```

12.5.28 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=>C1: 0xAB – ICODE_WRITE_CONFIG
          0x01 – option byte
          0x00 – block number
          0x00 0x00 0x00 0x00 – config bytes
C1=>HOST: 0x00 – ACK byte
          0xAB – related command code ICODE_WRITE_CONFIG
```

12.5.29 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=>C1: 0xAB – ICODE_PICK_RANDOM_ID

C1=>HOST: 0x00 – ACK byte
          0xAB – related command code ICODE_PICK_RANDOM_ID
```

13. OTA upgrade

The commands listed below can be used to perform an OTA upgrade. The latest OTA file is always available here: http://eccel.co.uk/wp-content/downloads/Pepper_C1/Pepper_C1.bin

13.1.1 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=>READER: 0xF0 – OTA begin

READER=>HOST: 0x00 – ACK byte
              0xF0 – related command code OTA begin
```

13.1.2 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 can 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=>READER:  0xF1 – OTA frame
               0x34 0x67 … 0x45 – firmware bytes

READER=>HOST:  0x00 – ACK byte
               0xF1 – related command code OTA frame
```

13.1.3 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=>READER: 0xF4 – OTA finish

READER=>HOST: 0x00 – ACK byte
              0xF4 – related command code OTA finish
```

14. Mechanical dimension

All dimensions are in mm.

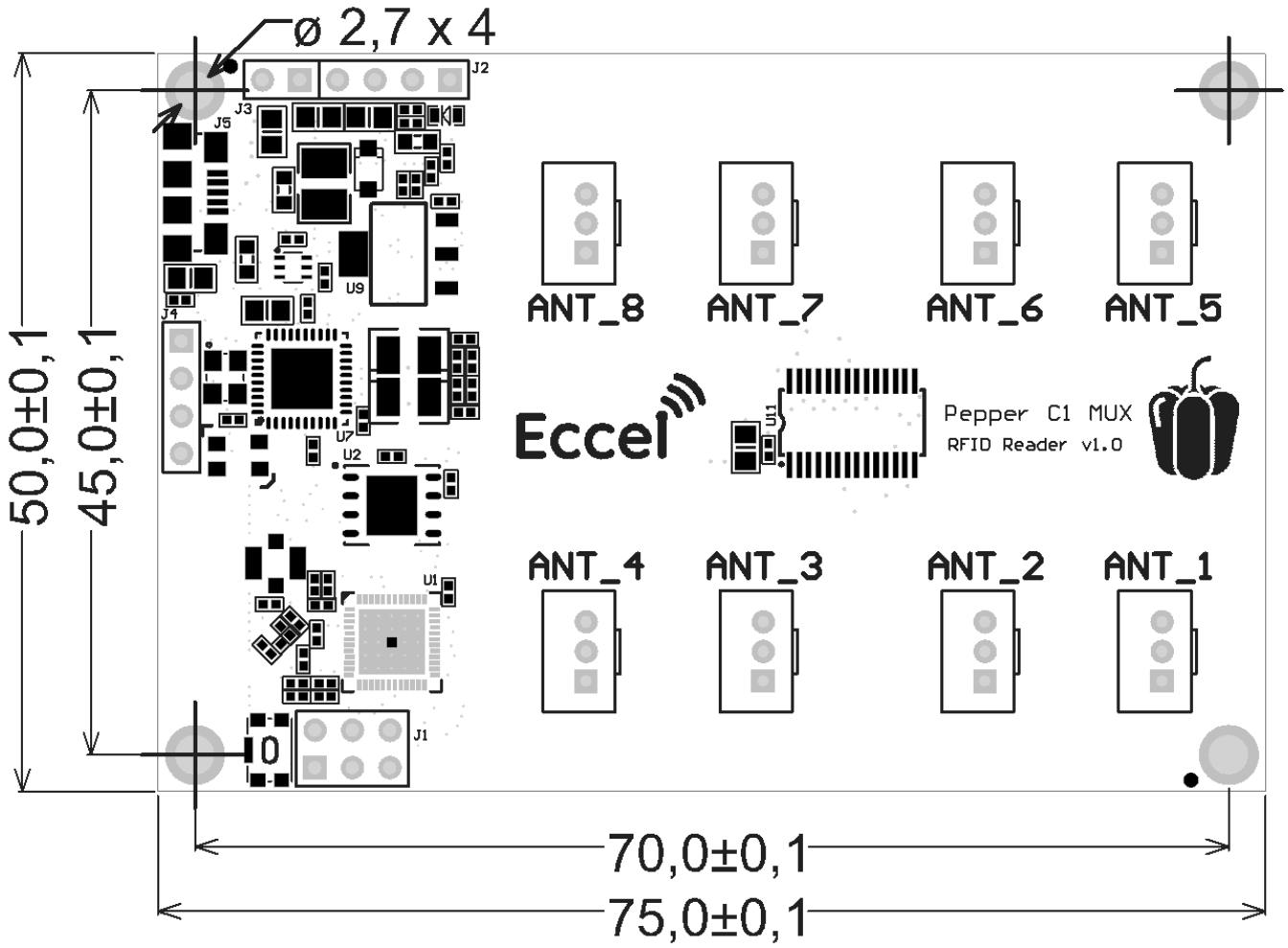


Figure 14-1

15. Design guidelines

- Use noise free power supply to power the C1 MUX Reader
  Insufficient power supply may reduce RFID performance
- Keep wires for power lines as short as possible or add high value electrolytic capacitor near J4 header
- Ferrite on cables should be considered in noisy environment, especially where long signal and power lines are being used
- Place The C1 MUX Reader board away from any source of noise.
- Keep antenna cables away from each other and other cables in your device.
- Do not route antenna cables in parallel to other signal cables.
  Antenna cables should cross other signals with 90° angle
- Do not make loops from too long antenna cables
- Provide enough spacing between active antennas. Distance equal to the antenna size should be considered as minimum and increased if necessary.
- RFID tag may be discovered by all active antennas placed too close to each other even if it’s located above just one of them.

16. RF Emissions and Susceptibility Approvals

Eccel have tested and declare that this product meets all the requirements of the relevant RF directives (RED) to be declared CE (European Union) and UKCA (United Kingdom) compliant. Please see our declaration of conformity for this on the downloads tab of the product webpage.

This product is designed to be incorporated into products easily and quickly such that those products can pass any national or regional statutory RF requirements and certifications such as FCC (USA), ISED (Canada) and PSE (Japan) for example.

This product is designed to meet all statutory RF requirements applicable worldwide using the most cost effective but robust design methodology.

Eccel is pleased to offer customers very cost-effective certification for their end equipment that incorporate this product. Prices start from £3K per approval/ certification. Please contact us for more details at sales@eccel.co.uk .

No responsibility is taken for the method of integration or final use of the C1 modules

More information about the C1 module and other products can be found at the Internet site:

http://www.eccel.co.uk

or alternatively contact ECCEL Technology (IB Technology) by e-mail at:

sales@eccel.co.uk